Last updated: June 11, 2013

COMPUTER NETWORKS

Performance and Quality of Service

IVAN MARSIC

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Rutgers University, New Brunswick, New Jersey

Author's address:

Ivan Marsic
Rutgers University
Department of Electrical and Computer Engineering
94 Brett Road
Piscataway, New Jersey 08854
marsic@ece.rutgers.edu

Book website: http://www.ece.rutgers.edu/~marsic/books/CN/

Preface

This book reviews modern computer networks with a focus on performance and quality of service. Wired and wireless/mobile networks today are working in concert and need to be studied together, rather than separately. Multimedia applications are ever more important and need to be given adequate representation relative to traditional data applications. This book covers these topics with an emphasis on underlying principles and core concepts.

Audience

This book is designed for upper-division undergraduate and graduate courses in computer networking. By focusing on explanations rather than collections of facts, it is intended primarily for learning, rather than reference. The book's focus on basic concepts should also be appealing to practitioners interested in the "whys" behind the commonly encountered networking technologies. The reader should have basic knowledge of probability and statistics (reviewed in the Appendix). Most concepts do not require mathematical sophistication beyond a first undergraduate course. I tried to present a serious material in a fun way so the reader may take pleasure in learning something nontrivial. It is never easy, but I hope it will be worth your effort.

Approach and Organization

In structuring the text, I faced the choice between logically grouping the topics vs. gradually identifying and addressing issues. The former creates a neater structure and the latter is more suitable for teaching new material. I compromised by opportunistically adopting both approaches.

Chapter 1 reviews essential networking technologies. It is condensed but more technical than many current networking books. I tried to give an engineering overview that is sufficiently detailed but not too long. This chapter serves as the basis for the rest of the book. I believe that it may also suffice for an introductory course on computer networks.

Chapter 2 reviews end-to-end mechanisms for congestion management in data networks. Most of these mechanisms are implemented in different variants of Transmission Control Protocol (TCP), which is the most popular Internet protocol.

Chapter 3 reviews the requirements and solutions for multimedia networking. It describes desirable network behaviors, rather than detailed mechanisms for implementing such behaviors. Such mechanisms are described in Chapter 5, after the operation of network routers is detailed in Chapter 4.

Chapter 4 describes how network routers forward data packets. It also describes simple techniques for modeling queuing delays in routers.

Chapter 5 describes router techniques for reducing or redistributing queuing delays across data packets belonging to different flows. These include scheduling and policing the network traffic. This chapter also reviews the MPLS protocol that is particularly important to network operators.

Chapter 6 describes wireless networks, focusing on the network and link layers, rather than on physical layer issues.

Chapter 7 reviews network security issues.

Chapter 8 describes network measurement techniques.

Chapter 9 reviews major protocols used in the Internet that are either not essential or are specific implementations of generic protocols presented in earlier chapters. The most essential Internet protocols, such as TCP and IP are presented in earlier chapters.

The Appendix provides a brief review of probability and statistics.

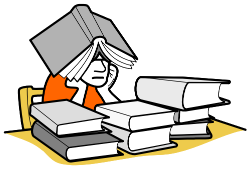
Bibliographic references are mostly excluded from the main text. The reader will find extensive literature review at the end of each chapter, in sections Summary and Bibliographical Notes.

Solved Problems

This book puts great emphasis on exercise problems for two reasons. First, I believe that specific problems are the best way to explain difficult concepts. I should say that these are *worked examples* with detailed explanation of the solution, which is much more than showing a concise result for a problem. Second, I wanted to keep the main text relatively short and focused on the key concepts. Therefore, I use problems to illustrate less important or advanced topics. Every chapter is accompanied with a set of problems. Solutions for most of the problems can be found at the back of the text, starting on page 459.

Additional Materials

Lecture slides, programming projects and Wireshark labs, and online links to related topics are available at the book website. There is also a simple TCP protocol simulator written in the Java programming language. The website is: http://www.ece.rutgers.edu/~marsic/books/CN/.



Contents at a Glance

Table of Contents

Chapter 1
Introduction to Computer Networks

Contents

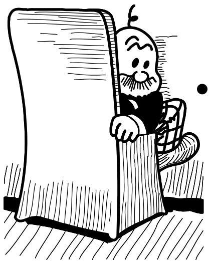
1.1 Introduction

A network is a set of devices (often referred to as nodes) connected by communication links that are built using different physical media. A node can be a computer, telephone, or any other device capable of sending and receiving messages. The communication medium is the physical path by which message travels from sender to receiver. Example media include fiber-optic cable, copper wire, or air carrying radio waves.

1.1.1 The Networking Problem

Networking is about transmitting messages from senders to receivers (over a "communication channel"). Key issues we encounter include:

- "Noise" damages (corrupts) the messages; we would like to be able to *communicate reliably* in the presence of noise
- Establishing and maintaining physical communication lines is costly; we would like to be able to *connect arbitrary senders and receivers* while keeping the economic cost of network resources to a minimum



- Time is always an issue in information systems as is generally in life; we would like to be able to provide *expedited delivery* particularly for messages that have short deadlines

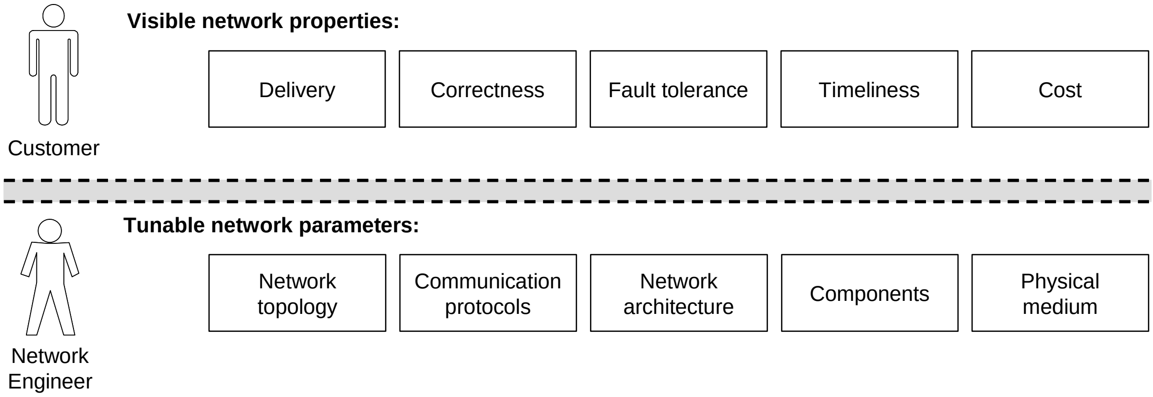


Figure 1-1: The customer cares about the visible network properties that can be controlled by the adjusting the network parameters.

Figure 1-1 illustrates what the customer usually cares about and what the network engineer can do about it. The visible network variables ("symptoms"), easily understood by a non-technical person include:

Delivery: The network must deliver data to the correct destination(s). Data must be received only by the intended recipients and not by others.

Correctness: Data must be delivered accurately, because distorted data is generally unusable.

Timeliness: Data must be delivered before they need to be put to use; else, they would be useless.

Fault tolerance and cost effectiveness are important characteristics of networks. For some of these parameters, the acceptable value is a matter of degree, judged subjectively. Our focus will be on network performance (objectively measurable characteristics) and quality of service (psychological determinants).

Limited resources can become overbooked, resulting in message loss. A network should be able to deliver messages even if some links experience outages.

The tunable parameters (or "knobs") for a network include: network topology, communication protocols, architecture, components, and the physical medium (connection lines) over which the signal is transmitted.

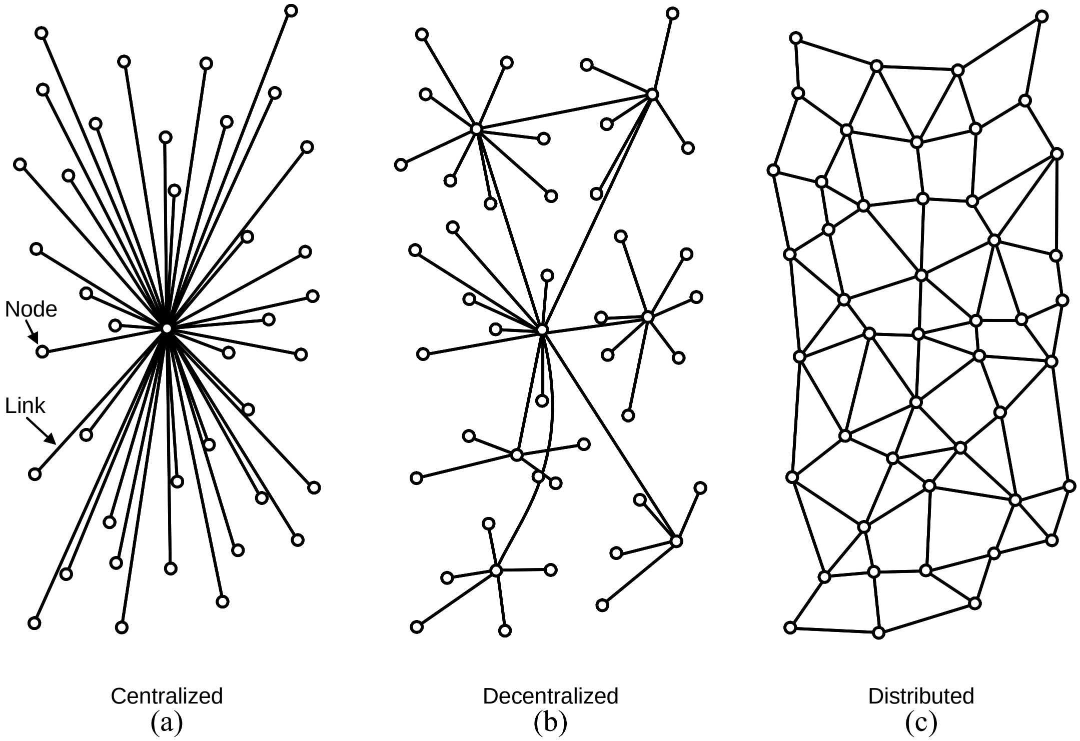


Figure 1-2: Different network topologies have different robustness characteristics relative to the failures of network elements.

- Connection topology: completely connected graph compared to link sharing with multiplexing and demultiplexing. Paul Baran considered in 1964 theoretically best architecture for survivability of data networks (Figure 1-2). He considered only network graph topology and assigned no qualities to its nodes and links[1]. He found that the distributed-topology network which resembles a fisherman's net, Figure 1-2(c), has the greatest resilience to element (node or link) failures. Figure 1-3 shows the actual topology of the entire Internet (in 1999). This topology evolved over several decades by incremental contributions from many independent organizations, without a "grand plan" to guide the overall design. In a sense, one could say that the Internet topology evolved in a "self-organizing" manner. Interestingly, it resembles more the decentralized-topology network with many hubs (Figure 1-2(b)), and to a lesser extent the distributed topology (Figure 1-2(c)). A possible reason may be that, because groups of nodes belong to different organizations ("administrative domains"), it is easier to manage cross-border issues through one or few designated nodes. If the whole Internet developed organically, all nodes had about equal capabilities, and there were enough human resources to manage each node, then one could speculate that the distributed model would have prevailed.

[1] When discussing computer networks, the term "host" is usually reserved for communication endpoints and "node" is used for intermediary computing nodes that relay messages on their way to the destination.

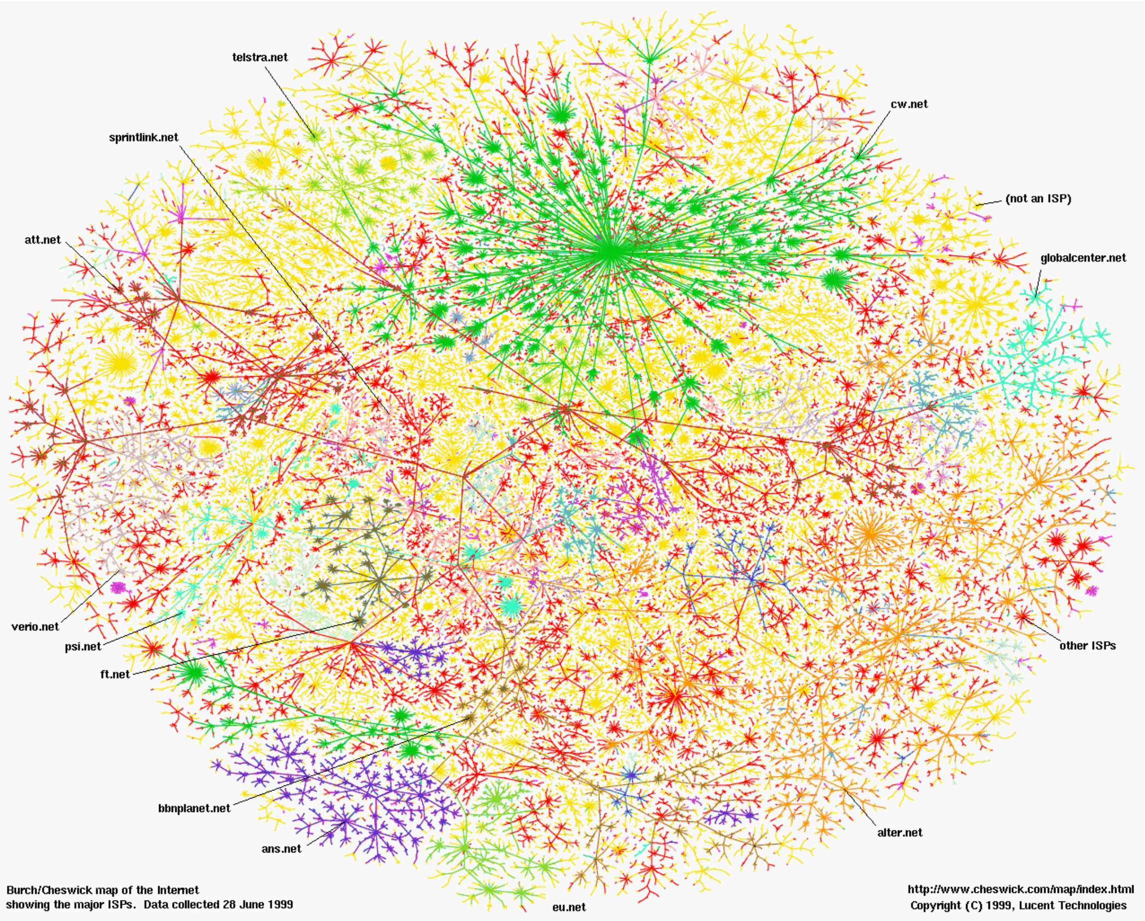

telstra.net
cw.net
sprintlink.net
(not an ISP)
att.net
globalcenter.net
verio.net
psi.net
ft.net
other ISPs
bbnplanet.net
ans.net
alter.net
eu.net
Burch/Cheswick map of the Internet
showing the major ISPs. Data collected 28 June 1999
http://www.cheswick.com/map/index.html
Copyright (C) 1999, Lucent Technologies

Figure 1-3. The map of the connections between the major Internet Service Providers (ISPs). [From the Internet mapping project: http://www.cheswick.com/]

- Network architecture: what part of the network is a fixed infrastructure as opposed to being ad hoc built for a temporary need

- Component characteristics: reliability and performance of individual hardware components (nodes and links). Faster and more reliable components are also more costly. When a network node (called switch or router) relays messages from a faster to a slower link, a congestion and a waiting-queue buildup may occur under a heavy traffic. In practice, all queues have limited capacity of their "waiting room," so the loss occurs when messages arrive at a full queue. A lost message will never reach its destination; if the message is important, countermeasures must be taken to prevent or mitigate the loss (Sections 1.2 and 1.3)

- Performance metrics: success rate of transmitted messages (or, message loss rate), average delay of message delivery, and delay variability (also known as "jitter")

- Different applications (data/voice/multimedia) have different requirements for network service in order to be useful to human users: some applications are impaired by message loss and others are impaired by delay or jitter

There are some major problems faced by network engineers when building a large-scale network, such as the Internet that is now available worldwide. Some of these problems are non-technical:

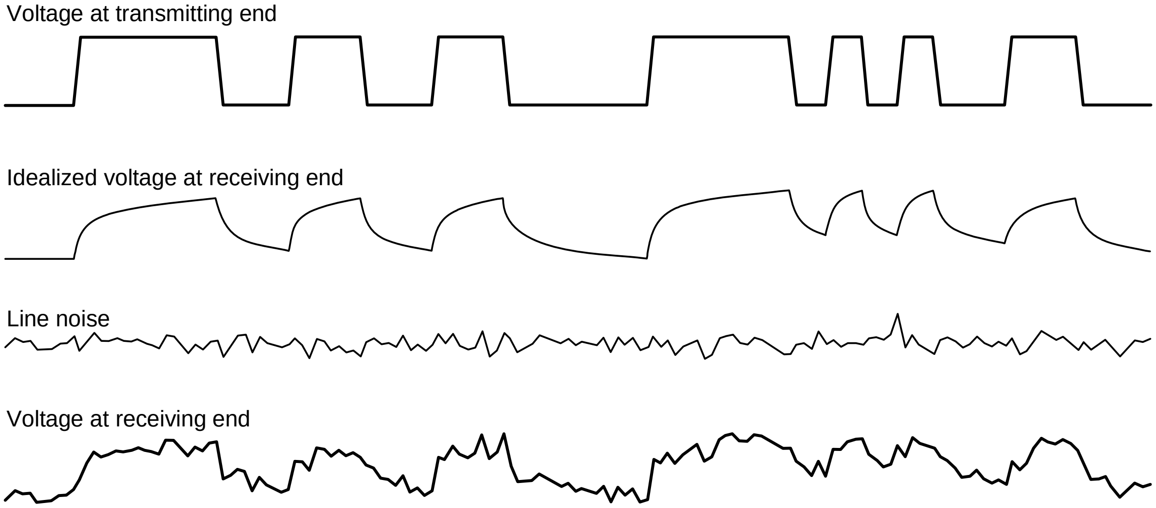
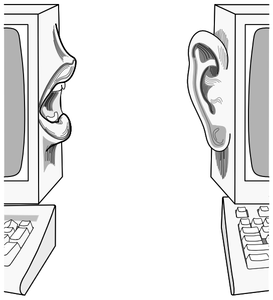


Figure 1-4: Digital signal distortion in transmission due to transients and noise associated with physical lines.

- *Heterogeneity*: Diverse software and hardware of network components need to coexist and interoperate. The diversity results from different user needs and their economic capabilities, as well as because installed infrastructure tends to live long enough to become mixed with several new generations of technologies.

- *Autonomy*: Different parts of the Internet are controlled by independent organizations. Even a sub-network controlled by the same multinational organization, such as IBM or Coca Cola, may cross many state borders. These independent organizations are generally in competition with each other and do not necessarily provide one another the most accurate information about their own networks. The implication is that the network engineer can effectively control only a small part of the global network. As for the rest, the engineer will be able to receive only limited information about the characteristics of others' autonomous sub-networks. Any local solutions must be developed based on that limited information about the rest of the global network.

- *Scalability*: Although a global network like the Internet consists of many autonomous domains, there is a need for standards that prevent the network from becoming fragmented into many non-interoperable pieces ("islands"). Solutions are needed that will ensure smooth growth of the network as many new devices and autonomous domains are added. Again, information about available network resources is either impossible to obtain in real time, or may be proprietary to the domain operator.



1.1.2 Communication Links

There are many phenomena that affect the transmitted signal, some of which are illustrated in Figure 1-4. Although the effects of transients and noise are exaggerated, they illustrate an important point. The input pulses must be well separated because too short pulses will be "smeared" together. This can be observed for the short-duration pulses at the right-hand side of the pulse train. Obviously, the receiver of the signal in the bottom row of Figure 1-4 will have great difficulty figuring out whether or not there were pulses in the transmitted signal. You can also see that longer pulses are better

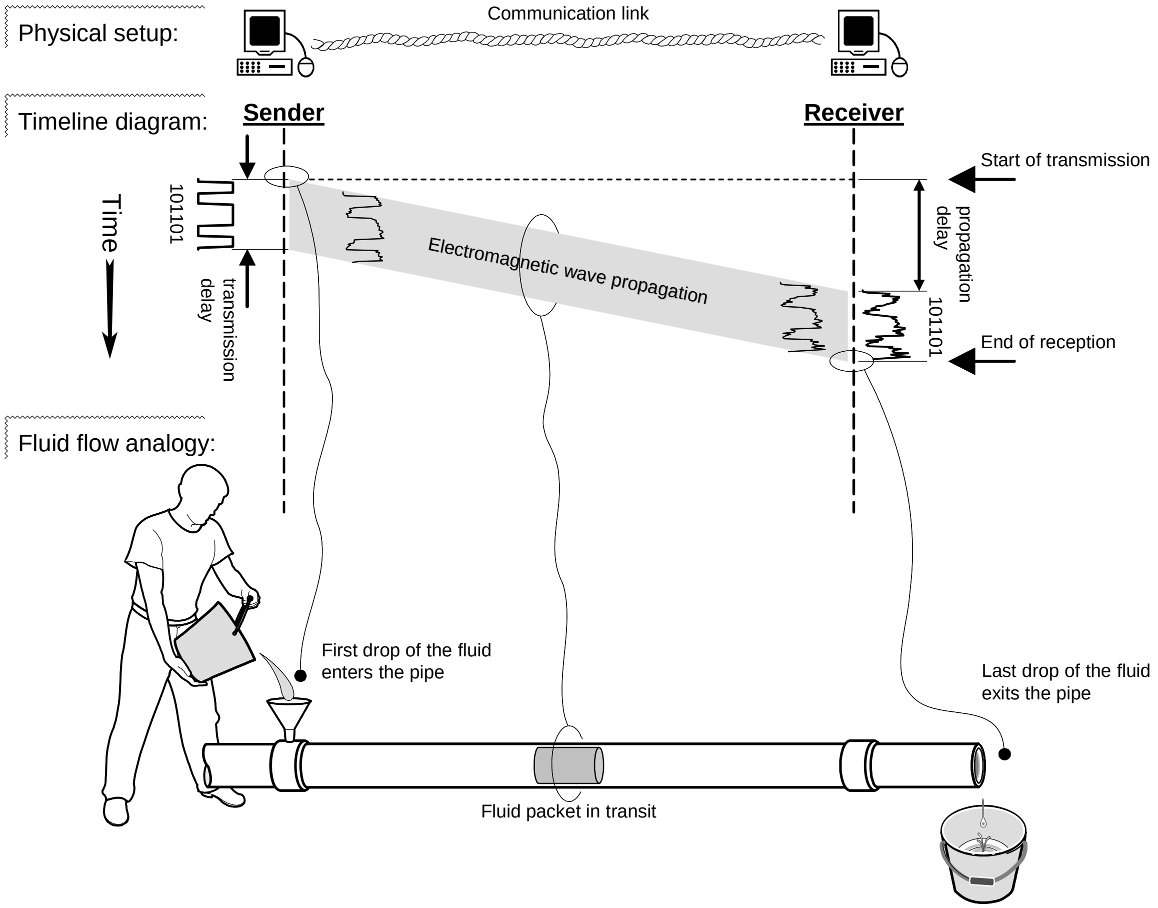


Figure 1-5: Timeline diagram for data transmission from sender to receiver. Time increases down the page.

separated and easier to recognize in the distorted signal. The minimum tolerable separation depends on the physical characteristics of a transmission line (e.g., copper vs. optical fiber). If each pulse corresponds to a single bit of information, then the minimum tolerable separation of pulses determines the maximum number of bits that can be transmitted over a particular transmission line. This number limits the **data rate** of a communication link (or transmission rate, or transmission capacity), which is the rate at which the network interface accepts data bits. The data rate of a link is measured in the units of *bits per second* (bps).

It is common to represent data transmissions on a timeline diagram as shown in Figure 1-5. This figure also illustrates delays associated with data transmission. Although information bits are not necessarily transmitted as rectangular pulses of voltage, all transmission lines are conceptually equivalent, as represented in Figure 1-6, because the transmission capacity for every line is expressed in bits/sec or bps. The time required to transmit a single bit on a given link is known as **bit time**. In this book, we will always visualize transmitted data as a train of digital pulses. The reader interested in physical methods of signal transmission should consult a communications-engineering textbook, such as [Haykin, 2009].

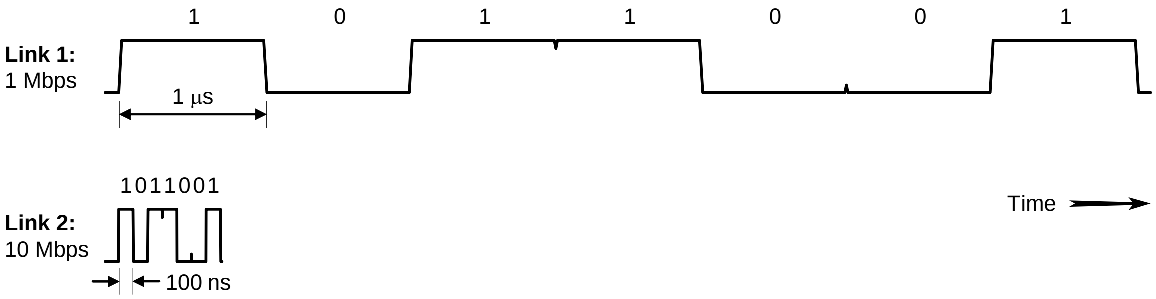


Figure 1-6: Transmission link capacity determines the speed at which the link can transmit data. In this example, each bit on Link 1 is 1 μs wide, while on Link 2 each bit is 100 ns wide. Hence, Link 2 can transmit ten times more data than Link 1 in the same time interval.

A common characterization of noise on transmission lines is **bit error rate** (BER): the fraction of bits received in error relative to the total number of bits received in transmission. Given a packet *n* bits long and assuming that bit errors occur independently of each other, a simple approximation for the packet error rate is:

$$PER = 1 - (1 - BER)^n \approx 1 - e^{-n \cdot BER} \tag{1.1}$$

An important attribute of a communication link is how many bitstreams can be transmitted on it at the same time. If a link allows transmitting only a single bitstream at a time, then the nodes connected to the link must coordinate their transmissions to avoid different bitstreams corrupting each other (known as data collision). Such links are known as *broadcast links* or *multiple-access links*. Point-to-point links often support data transmissions in both directions simultaneously. This kind of a link is said to be **full duplex**. A point-to-point link that supports data flowing in only one direction at a time is called **half duplex**. In other words, the nodes on each end of this kind of a link can both transmit and receive, but not at the same time—they only can do it by taking turns. It is like a one-lane road with bidirectional traffic. We will assume that all point-to-point links are full duplex, unless stated otherwise. A full-duplex link can be implemented in two ways: either the link must contain two physically separate transmission paths, one for sending and one for receiving, or the capacity of the communication channel is divided between signals traveling in opposite directions. The latter is usually achieved by *time division multiplexing* (TDM) or *frequency division multiplexing* (FDM).

In TDM or FDM, the communication link transmission capacity is subdivided into *m* portions and each portions is assigned to a different logical stream of bits. In FDM, the channel bandwidth *W* is subdivided into *m* channels, each with bandwidth *W*/*m* (actually slightly less, to allow for guard bands between channels). If the original channel could transmit at data rate *R* bits/sec, each of the *m* sub-channels would transmit at *R*/*m* bits/sec. Therefore, the transmission of a given packet would last *m* times longer on an FDM sub-channel.

In TDM, allocation is done by dividing the time axis into slots of fixed length, such as one bit long, or one byte long, or one packet long (assuming that all packets are equally long). As with FDM, conceptually we can view the link as consisting of *m* logically separate links or sub-channels. Again, the transmission of a given packet would last *m* times longer on a TDM sub-channel than it would on the original channel.

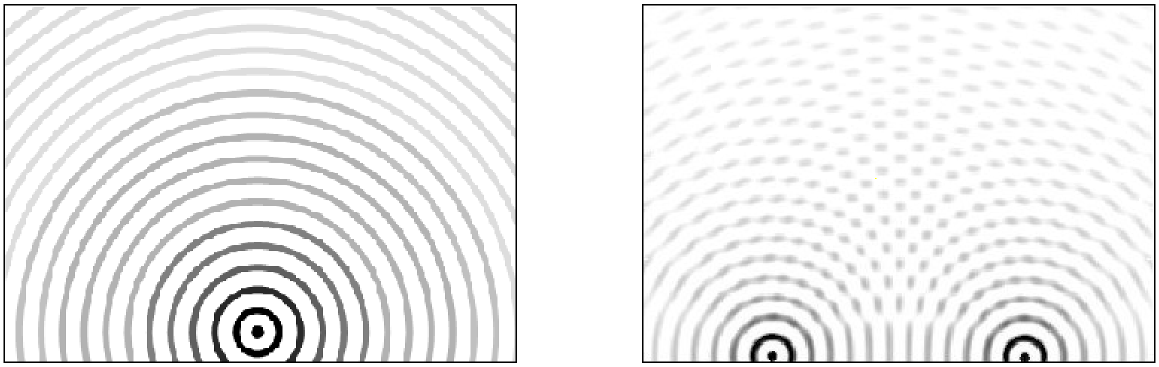


Figure 1-7: Wireless transmission. Left: single point source. Right: interference of two point sources.

Wireless Link

Consider a simple case of a point source radiating electromagnetic waves in all directions (Figure 1-7, left). As the reader will recall from basic physics, all waves (acoustic, light, radio) behave in the same way, so it is useful to keep in mind some visible waves, such as water surface waves, while thinking about radio waves. A receiver senses the amplitude of a received wave signal to decode the message. The received signal strength decreases exponentially with the sender-receiver distance. As any other communication medium, the wireless channel is subject to thermal noise, which distorts the signal randomly according to a Gaussian distribution of noise amplitudes. As the distance between a transmitter and receiver increases, the received signal strength decreases to levels close to the background noise floor. At a certain distance from the sender, the signal strengths will become so weak that the receiver will not be able reliably to discern signal from noise. This distance, known as **transmission range**, is decided arbitrarily, depending on what is considered acceptable bit error rate. For example, we can define the transmission range as the sender-to-receiver distance for which the packet error rate is less than 10 %.

In addition to thermal noise, the received signal may be distorted by parallel transmissions from other sources (Figure 1-7, right). This phenomenon is known as **interference**. Because this normally happens only when both sources are trying to transmit data (unknowingly of each other’s parallel transmissions), this scenario is called **packet collision**. A key observation is that *collisions occur at the receiver*—the sender is not disturbed by concurrent transmissions, but receiver cannot correctly decode sender’s message if it is combined with an interfering signal. If the source and receiver nodes are far away from the interfering source, the interference effect at the receiver will be a slight increase in the error rate. If the increased error rate is negligible, the source and receiver will be able to carry out their communication despite the interference. Note, however, that the interference of simultaneously transmitting sources never disappears—it only is reduced exponentially with an increasing mutual distance (Figure 1-8). The minimum distance (relative to the receiver) at which interferer’s effect can be considered negligible is called **interference range**. In Figure 1-8, node *D* is within the interference range of receiver *B*. Nodes *C* and *E* are outside the interference range. However, although outside the interference range defined for a single interferer, if nodes *C* and *E* are transmitting simultaneously their combined

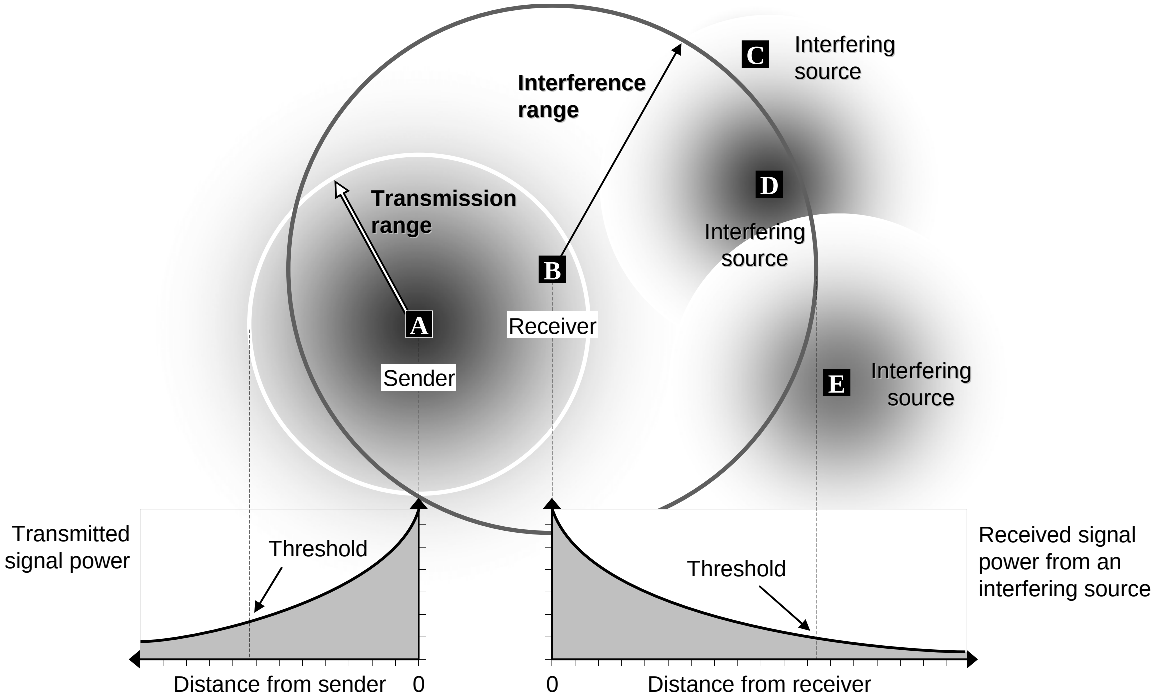


Figure 1-8: Transmission range and interference range for wireless links.

interference at *B* may be sufficiently high to cause as great or greater number of errors as a single interferer within the interference range.

1.1.3 Packets and Statistical Multiplexing

The communication channel essentially provides an abstraction of a continuous stream of symbols transmitted that are subject to a certain error probability. When interacting with another person, whether face-to-face or over the telephone, we think of units of communication in terms of conversational turns: first one person takes a turn and delivers their message, then the other person takes a turn, and so on. *Messages* could be thought of as units of communication exchanged by two (or more) interacting persons. We note that there are benefits of slicing a long oration into a sequence of smaller units of discourse. This slicing into messages gives the other person chance to clarify a misunderstanding or give a targeted response to a specific item.

In computer communication networks, messages are represented as strings of binary symbols (0 or 1), known as bits. Generally, messages are of variable length and some of them may still be considered too long for practical network transmission. There are several reasons for imposing a limit on message length. One is that longer messages stand a higher chance of being corrupted by an error (see Equation (1.1)). Another reason is to avoid the situation where a sending application seizes the link for itself by sending very long messages while other applications must wait for a long time. Therefore, messages are broken into shorter bit strings known as **packets**. These packets are then transmitted independently and reassembled into messages at the destination. This allows individual packets to opportunistically take *alternate routes* to the destination and *interleave* the network usage by multiple sources, thus avoiding inordinate waiting periods for some sources to transmit their information.

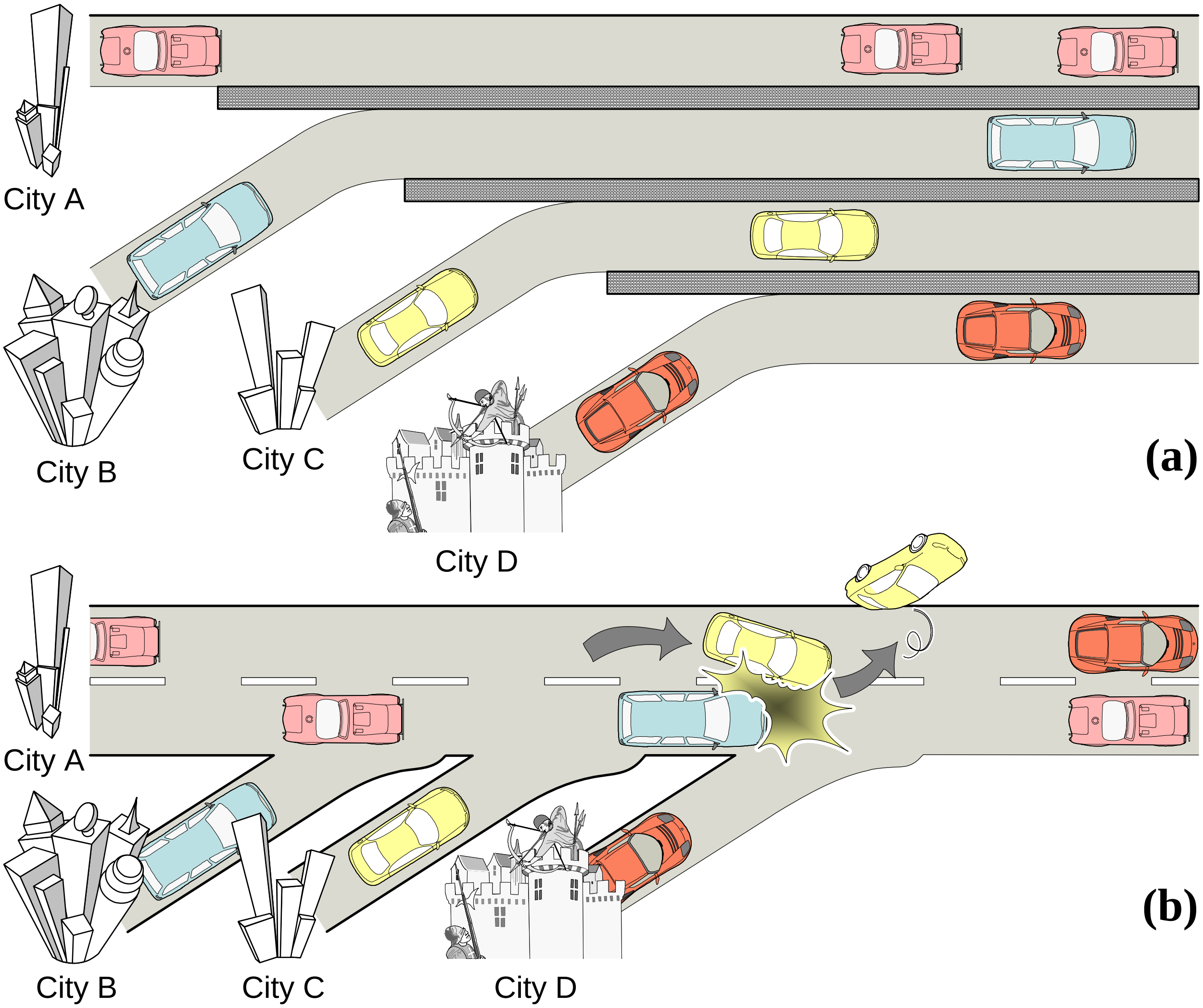


Figure 1-9: An analogy illustrating dedicated lines (a) compared to statistical maultiplexing (b).

Different network technologies impose different limits on the size of data blocks they can handle, which is known as the **maximum transmission unit** (MTU). For example, a regular Ethernet frame uses a frame format that limits the size of the payload it sends to 1,500 bytes. Note that the MTU value specifies the maximum payload size and does not include the header size of the header that is prepended to the payload of a packet.

Statistical Multiplexing

In Section 1.1.2 we mentioned time division multiplexing (TDM) and frequency division multiplexing (FDM) as example schemes for link sharing among multiple data streams. However, although their rigid rules for subdividing the link capacity ensure orderly and predictable data delivery, they may also incur inefficiencies. Consider the analogy in Figure 1-9 which shows traffic in multi-lane highways. In the top figure, cars are not allowed to cross over to other lanes, which corresponds to TDM or FDM schemes. An advantage is that there are no delays introduced by merging traffic or unpredictable outcomes of switching the lanes. A drawback is that some lanes may be empty and go unused, while others may be full and cars may be prevented from entering. Restricting crossover increases travel time because some lanes may not beutilized while others are momentarily stressed. Unlike this, in the bottom of Figure 1-9 cars can change lanes, with the advantage that overall the highway is utilized more efficiently. Of course, there are disadvantages, as well, and they include delays at merging points and potential collisions. This second scenario represents **statistical multiplexing** using packet multiplexers

Real-world systems are designed with sub-peak capacity for economic reasons. As a result, they experience congestion and delays during peak usage periods. Highways experience traffic slowdown during rush hours; restaurants or theaters experience crowds during weekend evenings; etc. Designing these systems to support peak usage without delays would not be economically feasible—most of the time they would be underutilized.

1.1.4 Communication Protocols

A **protocol** is a *set of rules* agreed-upon by interacting entities, e.g., computing devices, that govern their communicative exchanges. In real world, protocols serve as mechanisms by which organizations or individuals manage the complexity of articulating their work. A protocol defines how and in what order each task must be performed. It is hard to imagine accomplishing any task involving multiple entities without relying on a protocol. For example, one could codify the rules for how a customer (C) purchases goods from a merchant (M) as follows:

1. C→M Request catalog of products
2. C←M Respond catalog
3. C→M Make selections
4. C←M Deliver selections
5. C→M Confirm delivery
6. C←M Issue bill
7. C→M Make payment
8. C←M Issue confirmation

The customer and merchant may be located remote from each other and using other entities to help accomplish the purchasing task, such as a bank for credit-card transactions, or a postal service for parcel delivery.

Figure 1-10

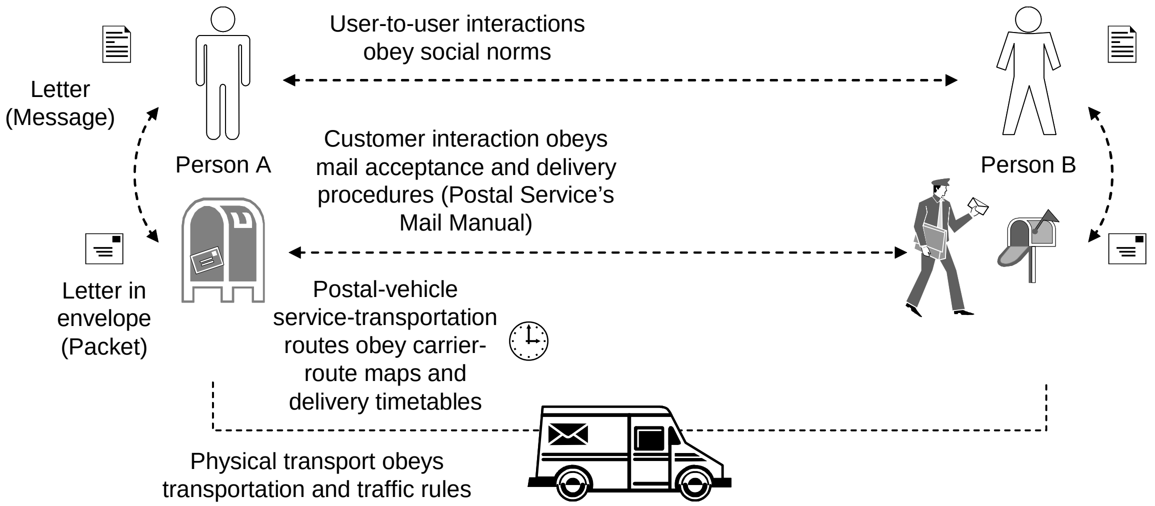


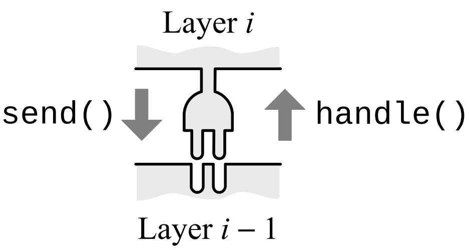
Figure 1-10: Protocol layers for conventional mail transport. Note the "layered design"—the division of labor between different layers of the "protocol stack."

An important characteristic of protocols is that the units of communication are **data packets**. Each data packet consists of a **header** that contains the packet guidance information to help guide the packet from its source to its destination, and the **payload**, which is the user information to be delivered to the destination address. A packet is delimited by the length information in its header, or by a special ending pattern of bits. (Of course, if the ending pattern is corrupted, the receiver will not be able to recognize the end of the packet.) In packet-switched networks, packets are transmitted at random times, and the receiver at the end of a communication link must have a means to distinguish an arriving packet from random noise on the line. For this purpose, each packet is preceded with a special sequence of bits that mark the start of the packet. This special bit pattern is usually called the **preamble**. Each receiver is continuously hunting for the preamble to catch the arriving packet. If the preamble is corrupted by random noise, the packet will be lost (i.e., unnoticed by the receiver).

Communication in computer networks is very complex. One effective means of dealing with complexity is using the division of labor, known as *modular design* with *separation of concerns*. In this approach, the system is split into modules and each module is assigned separate responsibilities ("concerns"). Network designers usually adopt a restricted version of modular design, know as **layered design**. Each **layer** defines a collection of conceptually similar functions (or, services) distinct from those of the other layers. The restriction in layered design is that a module in a given layer provides services to the layer just above it and receives services from the layer just below it. Layers are ordered in that each layer can use services only of lower-level layers and not of the layers above it. The layering approach forbids the modules from using services from (or providing to) non-adjacent layers. The bottom layer does not depend on any other layer and the top layer is not used by any other layer.

Each layer of the layered architecture is responsible for performing a well-defined function. A layer contains one or more software *modules* that offer services characteristic for this layer. Each module is called **protocol**. A protocol defines two application-programming interfaces (APIs):

1. *Service interface* to the protocols in the layer above this layer. The upper-layer protocols use this interface to “plug into” this layer and hand it data packets to `send()`. Each layer also defines a `handle()` callback operation through which the lower layer calls this layer to handle an incoming packet.
2. *Peer interface* to the counterpart protocol on a remote machine. This interface defines the format and meaning of data packets exchanged between peer protocols to support communication.



There are many advantages of layered design, primarily because it decomposes the problem of building a network into more manageable components. Each component can be developed independently and used interchangeably with any other component that complies with its service interface. However, there are some disadvantages, as well. For example, when a layer needs to make a decision about how to handle a data packet, it would be helpful to know what kind of information is inside the packet. Because of strict separation of concerns, particularly between the non-adjacent layers, this information is not available, so a more intelligent decision cannot be made. This is the reason why recently **cross-layered designs** are being adopted, particularly for wireless networks (see Chapter 6).

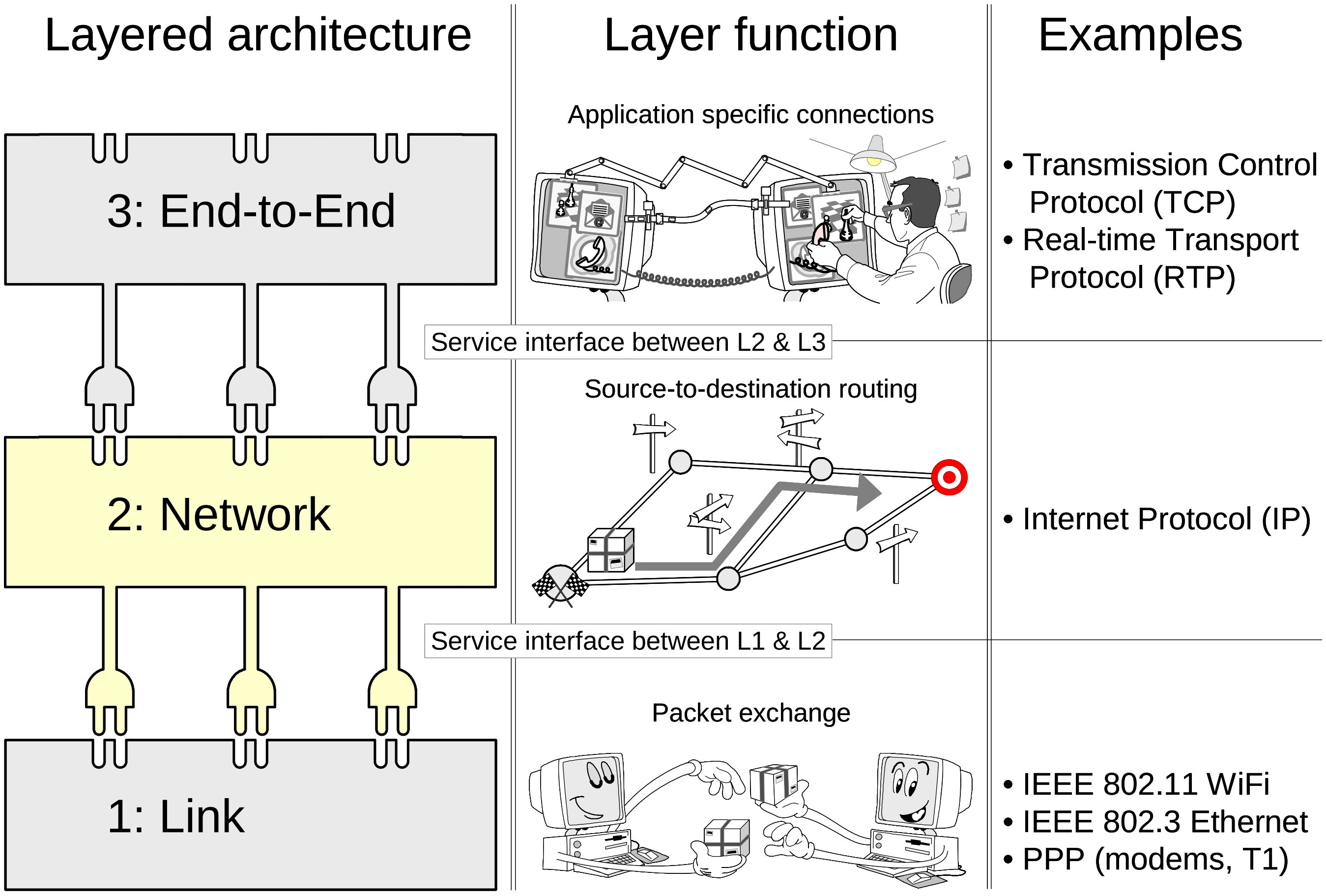
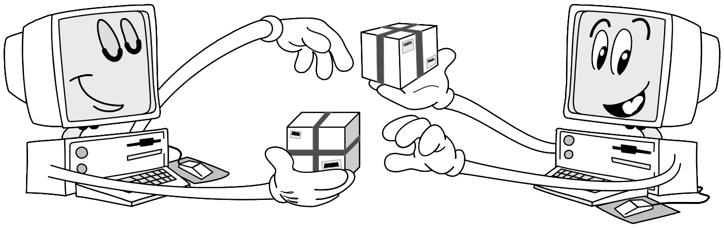

Layered architecture
Layer function
Examples
3: End-to-End
Application specific connections
• Transmission Control Protocol (TCP)
• Real-time Transport Protocol (RTP)
Service interface between L2 & L3
2: Network
Source-to-destination routing
• Internet Protocol (IP)
Service interface between L1 & L2
1: Link
Packet exchange
• IEEE 802.11 WiFi
• IEEE 802.3 Ethernet
• PPP (modems, T1)

Figure 1-11: Three-layer reference architecture for communication protocol layering.

Three-Layer Model

In recent years, the three-layer model (Figure 1-11) has emerged as reference architecture of computer networking protocols.

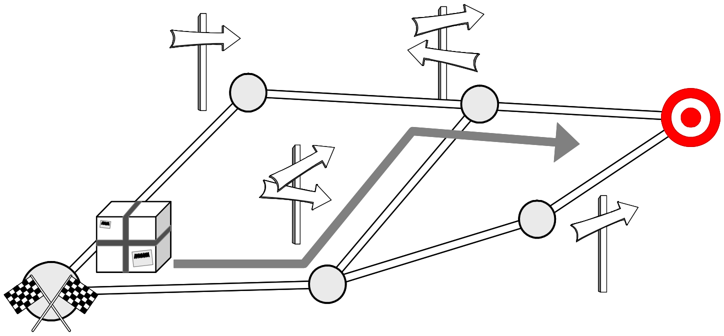


LAYER-1 – Link layer: is at the bottom of the protocol stack and implements a packet delivery service between nodes that are attached to the same physical link (or, physical medium). The physical link may be point-to-point from one transmitter to a receiver, or it may be shared by a number of transmitters and receivers (known as "broadcast link," Section 1.3.3).

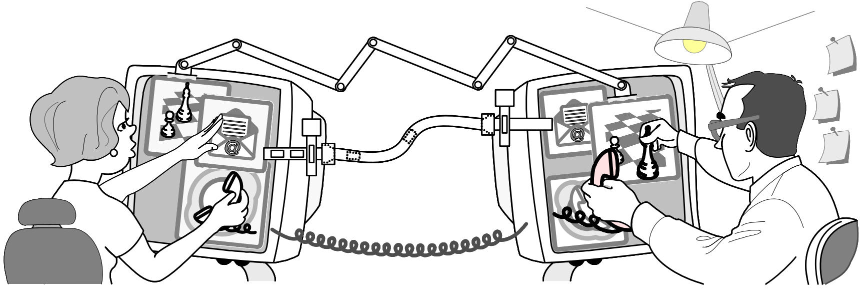
There is the "physical layer," which implements a digital transmission system that delivers bits. But, you would not know it because it is usually tightly coupled with the link layer by the link technology standard. Link and physical layers are usually standardized together and technology vendors package them together, as will be seen later in Section 1.5.

In wireless networks, physical communication is much more complex than in wired networks. Therefore, it may be justifiable to distinguish the physical and link layers, to keep both manageable. Because this book is mainly about protocols and not about physical communication, I will consider both together as a single, Link layer.

The link layer is not concerned with bridging end hosts across (many) intermediate links; this is why we need the network layer.



LAYER-2 – Network layer: provides concatenation of links to connect arbitrary end hosts. It will be elaborated in Section 1.4 where we describe the most popular network layer protocol: the Internet Protocol (IP). However, host computers are not the endpoints of communication—application programs running on these hosts are the actual endpoints of communication! The network layer is not concerned with application requirements. It may provide a range of choices for an upper layer to select from. For example, the network layer may support "quality of service" through "service level agreements," "resource reservation," but it does not know which one of these choices is the best for a particular application program; this is why we need the end-to-end layer.



LAYER-3 – End-to-end layer: this layer brings together (encapsulates) all communication-specific features of an application program. Here is the first time that we are concerned with application requirements.

The figure on the right is meant to illustrate that different applications need different type of connection for optimal performance.

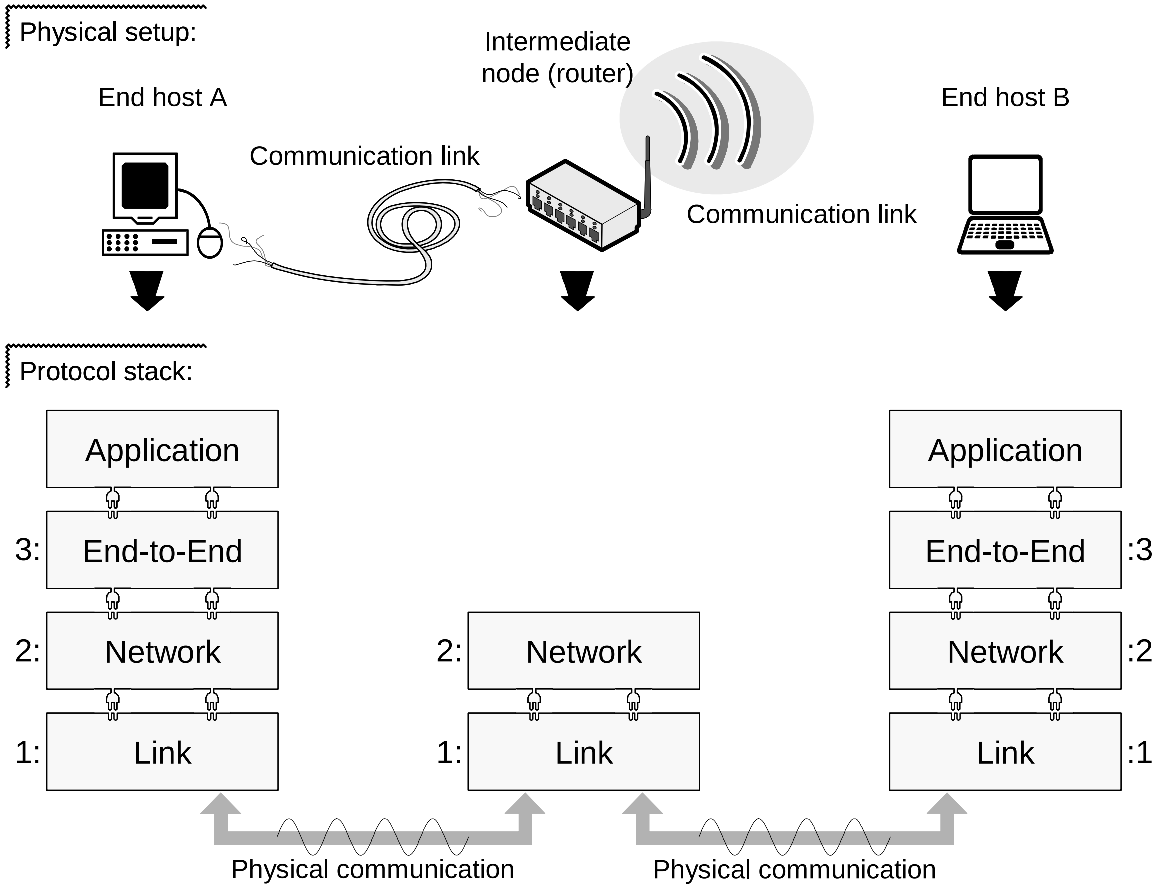


Figure 1-12: Protocol layering in end hosts and intermediate nodes (switches or routers).

For example, manipulation-based applications (such as video games) require an equivalent of mechanical links to convey user's action. Telephony applications need an equivalent of a telephone wire to carry user's voice, etc. A most prominent example of an end-to-end protocol is TCP, described in Chapter 2.

A fundamental design principle of network protocols and distributed systems in general is the **end-to-end principle**. The principle states that, whenever possible, communications protocol operations should occur at the end-points of a communications system, or as close as possible to the resource being controlled. According to the end-to-end principle, protocol features are only justified in the lower layers of a system if they are a performance optimization.

Figure 1-12 shows the layers involved when a message is sent from an application running on one host to another running on a different host. The application on host *A* hands the message to the end-to-end layer, which passes it down the protocol stack on the same host machine. Every layer accepts the payload handed to it and processes it to add its characteristic information in the form of an additional header (Figure 1-13). The link layer transmits the message over the physical medium. As the message travels from *A* to *B*, it may pass through many intermediate nodes, known as *switches* or *routers*. In every receiving node (including the intermediate ones), the message is received by the bottommost (or, link) layer and passed up through their protocol stack. Because intermediate nodes are not the final destination (or, end point), they do not have the

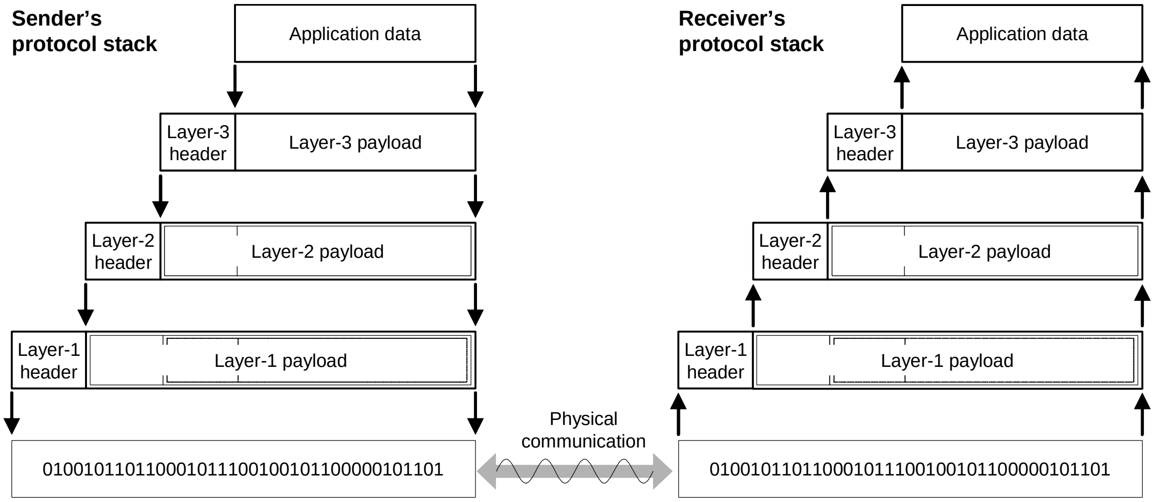


Figure 1-13: Packet nesting across the protocol stack: an entire packet of an upper layer becomes the data payload of a lower layer. Also see Figure 1-14.

complete protocol stack, but rather only the two bottommost layers: link and network layers (see Figure 1-12).

Pseudo code of a generic protocol module in layer *i* is given in Listing 1-1.

Listing 1-1: Pseudo code of a protocol module in layer *i*.

```
// Definition of packet format for layer i.
// Implementing the java.io.Externalizable interface makes possible to serialize
// a Packet object to a byte stream (which becomes the payload for the lower-layer protocol).
public class PacketLayer_i implements java.io.Externalizable {
    // packet header
    private String sourceAddress;
    private String receiverAddress;
    private String packetID;  // this packet's identifier
    private String receivingProtocol; // upper layer protocol at receiver

    // packet payload
    private byte[] payload;

    // constructor
    public PacketLayer_i(
        byte[] data, String recvAddr, String upperProtocol
    ) {
        payload = data;
        sourceAddress = address of my host computer;
        receiverAddress = recvAddr;
        packetID = generate unique identifier for this packet;
        receivingProtocol = upperProtocol;
    }

    public void writeExternal(ObjectOutput out) {
        // Packet implements java.io.Externalizable instead of java.io.Serializable
        // to be able to control how the serialization stream is written, because
        // it must follow the standard packet format for the given protocol.
    }
    public void readExternal(ObjectOutput out) {
        // reconstruct a Packet from the received bytestream
    }
}
```

```
// Definition of a generic protocol module in layer i.

public class ProtocolLayer_i {
    // maximum number of outstanding packets at sender (zero, if NOT a persistent sender)
    public static final int N; // (N is also called the sliding window size

    // lower layer protocol that provides services to this protocol
    private ProtocolLayer_iDOWN lowerLayerProtocol;

    // look-up table of upper layer protocols that use services of this protocol
    private HashMap upperLayerProtocols;

    // look-up table of next-receiver node addresses based on final destination addresses
    //        (this object is shared with the routing protocol, shown in Listing 1-2)
    private HashMap forwardingTable;

    // list of unacknowledged packets that may need to be retransmitted
    //       (maintained only for persistent senders that provide reliable transmission)
    private ArrayList unacknowledgedPackets = new ArrayList();

    // constructor
    public ProtocolLayer_i(
        ProtocolLayer_iDOWN lowerLayerProtocol
    ) {
        this.lowerLayerProtocol = lowerLayerProtocol;
    }

    // sending service offered to the upper layer protocols, called in a top-layer thread
    public void send(
        byte[] data, String destinationAddr,
        ProtocolLayer_iUP upperProtocol
    ) throws Exception {
        // if persistent sender and window of unacknowledged packets full, then do nothing
        if ((N > 0) && (N - unacknowledgedPackets.size() <= 0)) ) {
           throw exception: admission refused by overbooked sender;
        }

        // create the packet to send
        PacketLayer_i outgoingPacket =
           new PacketLayer_i(data, destinationAddr, upperProtocol);

        // serialize the packet object into a byte-stream (payload for lower-layer protocol)
        java.io.ByteArrayOutputStream bout =
            new ByteArrayOutputStream();
        java.io.ObjectOutputStream outstr =
            new ObjectOutputStream(bout);
        outstr.writeObject(outgoingPacket);

        // look-up the receiving node of this packet based on the destination address
        //      (requires synchronized access because the forwarding table is shared
        //      with the routing protocol)
        synchronized (forwardingTable) { // critical region
            String recvAddr = forwardingTable.get(destinationAddr);
        } // end of the critical region

        // hand the packet as a byte-array down the protocol stack for transmission
        lowerLayerProtocol.send(
             bout.toByteArray(), recvAddr, this
        );
    }

    // upcall method, called from the layer below this one, when data arrives
    //    from a remote peer (executes in a bottom-layer thread!)
    public void handle(byte[] data) {

        // reconstruct a Packet object from the received data byte-stream
        ObjectInputStream instr = new ObjectInputStream(
            new ByteArrayInputStream(data)
        );
        PacketLayer_i receivedFrame =
            (PacketLayer_i) instr.readObject();

        // if this packet is addressed to me ... (on a broadcast medium)
        if (receivedFrame.getReceiverAddress() == my address) {
            // ...demultiplex to upper layer protocol assigned to handle this packet's payload
            synchronized (upperLayerProtocols) { // critical region
              ProtocolLayer_iUP upperProtocol = (ProtocolLayer_iUP)
                  upperLayerProtocols.get(
                      receivedFrame.getReceivingProtocol()
                  );
            } // end of the critical region

            // remove this protocol's header and
            //      hand the payload over to the upper layer protocol
            upperProtocol.handle(receivedFrame.getPayload());
        }
    }

    // Receiver endpoint stores here the reference to an upper-layer protocol that will be used
    // in the method handle() to demultiplex the receiving protocol to handle a packet
    public void setHandler(
        String receivingProtocol, ProtocolLayer_iUP upperProtocol
    ) {
        // add a <key, value> entry into the look-up table to demultiplex upper layer protocol
        upperLayerProtocols.put(receivingProtocol, upperProtocol);
    }

    // Method called by the routing protocol (running in a different thread or process)
    public void setReceiver(
        String destinationAddr, String receiverAddr
    ) {
        // add a <key, value> entry into the forwarding look-up table
        synchronized (forwardingTable) { // critical region
            forwardingTable.put(destinationAddr, receiverAddr);
```

```
61          } // end of the critical region
62      }
63 }
```

Here I provide only a brief description of the above pseudocode. We will encounter and explain the details later in this chapter, as new concepts are introduced. The attribute `upperProtocol` is used to decide to which upper-layer protocol to deliver a received packet's payload. This process is called *protocol multiplexing* and allows the protocol at a lower-level layer to serve different upper-layer protocols. The lower-layer sender protocol combines ("multiplexes") several data streams and its counterpart at the receiver separates ("demultiplexes") different streams for delivery to the correct upper-layer receiver protocol.

Some functionality is not shown in Listing 1-1 to keep the pseudo code manageable. For example, in case of a persistent sender in the method `send()` we should set the retransmission timer for the sent packet and store the packet in the `unacknowledgedPackets` list. Similarly, in the method `handle()` we should check what packet is acknowledged and remove the acknowledged packet(s) from the `unacknowledgedPackets` list. Also, the method `send()` is shown to check only the `forwardingTable` to determine the intermediary receiver (`recvAddr`) based on the final destination address. In addition, we will see that different protocol layers use different addresses for the same network node (Section 9.3.1). For this reason, it is necessary to perform address translation from the current-layer address (`recvAddr`) to the address of the lower layer before passing it as an argument in the `send()` call, in Line 33. (See Section 9.3 for more about address translation.)

The reader who carefully examined Listing 1-1 will have noticed that packets from higher layers become nested inside the packets of lower layers as they are passed down the protocol stack (Figure 1-13). The protocol at a lower layer is *not* aware of any structure in the data passed down from the upper layer, i.e., it does not know if the data can be partitioned to header and payload or where their boundary is—it simply considers the whole thing as an unstructured data payload.

It is important to keep in mind how protocol layering is reflected in the packet format (Figure 1-14). Each protocol has its competences and does not understand the header format of protocols in other layers. A link-layer protocol does not know how to interpret the network-layer header and vice versa. This compartmentalization of competences helps keep protocols reasonably simple. In addition, when a network node receives a packet, the node's protocols read the headers one by one, starting at the bottommost layer and going up the stack to the topmost layer. Each protocol removes its own header and passes the remaining packet up the protocol stack until the application data carried in the packet is handed over to the receiving application.

The generic protocol implementation in Listing 1-1 works for all protocols in the layer stack. However, each layer will require some layer-specific modifications. The protocol in Listing 1-1 best represents a Network layer protocol for source-to-destination packet delivery.

The Link layer is special because it is at the bottom of the protocol stack and cannot use services of any other layer. It runs the `receiveBits()` method in a continuous loop (perhaps in a separate thread or process) to hunt for arriving packets. This method in turn calls Link layer's `handle()`, which in turn calls the upper-layer (Network) method `handle()`. Link layer's `send()` method, instead of using a lower-layer service, itself does the sending by calling this layer's own method `sendBits()`.

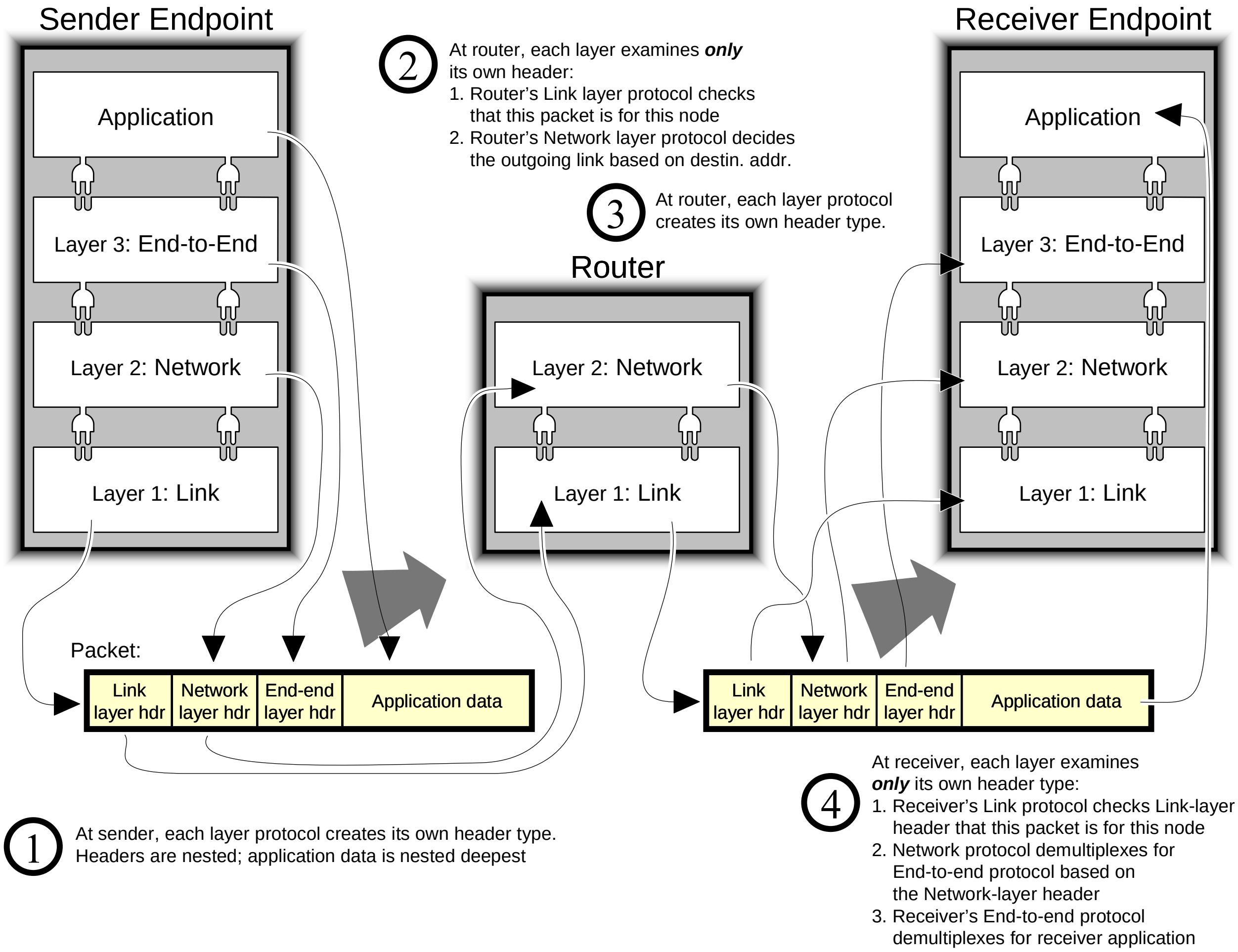

Sender Endpoint
Application
Layer 3: End-to-End
Layer 2: Network
Layer 1: Link
Packet:
Link layer hdr
Network layer hdr
End-end layer hdr
Application data
2
At router, each layer examines *only* its own header:
1. Router's Link layer protocol checks that this packet is for this node
2. Router's Network layer protocol decides the outgoing link based on destin. addr.
3
At router, each layer protocol creates its own header type.
Router
Layer 2: Network
Layer 1: Link
Receiver Endpoint
Application
Layer 3: End-to-End
Layer 2: Network
Layer 1: Link
Link layer hdr
Network layer hdr
End-end layer hdr
Application data
4
At receiver, each layer examines *only* its own header type:
1. Receiver's Link protocol checks Link-layer header that this packet is for this node
2. Network protocol demultiplexes for End-to-end protocol based on the Network-layer header
3. Receiver's End-to-end protocol demultiplexes for receiver application
1
At sender, each layer protocol creates its own header type. Headers are nested; application data is nested deepest

Figure 1-14: The protocol at layer *i* restricts its processing to the header that was created by the sender's layer *i* protocol and does not deal with any other headers in a packet.

An important feature of a link-layer protocol is **data transparency**, which means that it must carry any bit pattern in the data payload. An example of a special bit pattern is the packet preamble that helps the receiver to recognize the start of an arriving packet (mentioned at the start of this section). Data transparency means that the link layer must not forbid the upper-layer protocol to send data containing special bit patterns. To implement data transparency, link-layer protocol uses a technique known as **bit stuffing** (or, **byte stuffing**, depending on what the smallest unit for measuring the payload is). Bit stuffing defines a special control escape bit pattern, call it *ESC* (Figure 1-15). The method `sendBits()`examines the payload received from the upper-layer protocol. If it encounters a special control sequence, say preamble (call it *PRE*), then it "stuffs" (adds) a control escape sequence *ESC* into the transmitted data stream, before *PRE* (resulting in *ESC PRE*), to indicate that the following *PRE* is *not* a preamble but is, in fact, actual data. Similarly, if the control escape pattern *ESC* itself appears as actual data, it too must be preceeded by an *ESC*. The method `receiveBits()` removes any control escape patterns that it finds in the received packet before delivering it to the upper-layer method `handle()` (Figure 1-15).

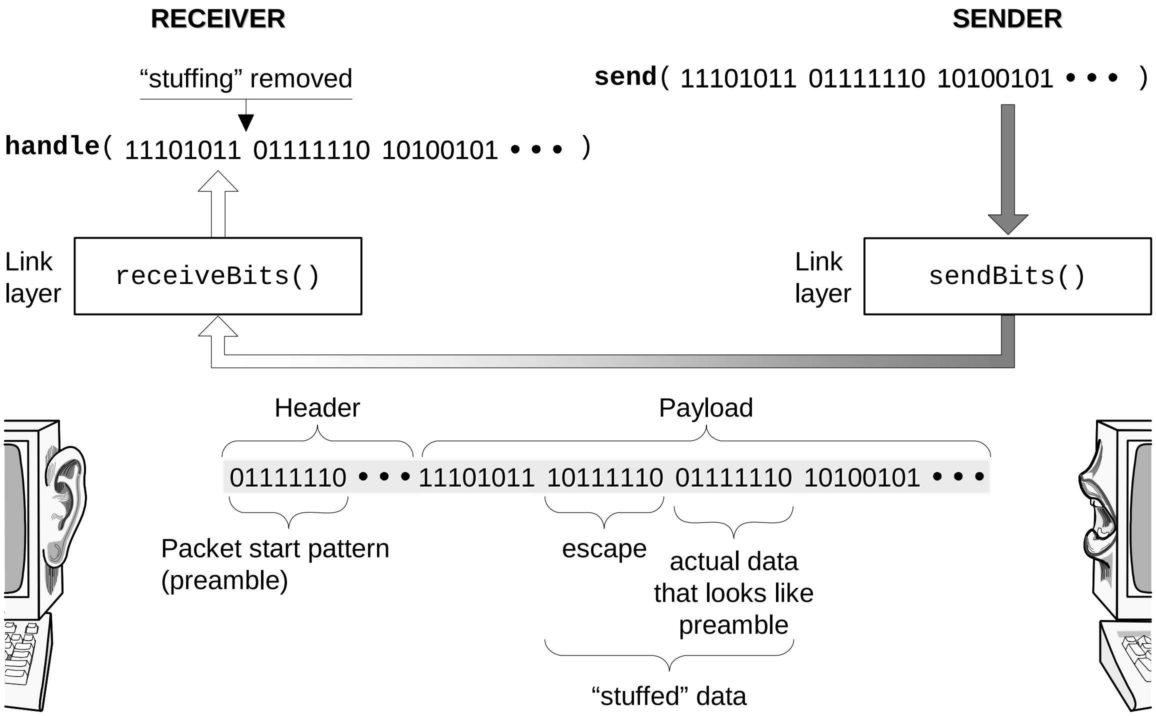


Figure 1-15: Bit stuffing to escape special control patterns in the frame data.

The pseudo code in Listing 1-1 is only meant to illustrate how one would write a protocol module. It is extremely simplified and certainly not optimized for performance. My main goal is to give the reader an idea about the issues involved in protocol design. We will customize the pseudo code from Listing 1-1 for different protocols, such as routing protocols in Listing 1-2 (Section 1.4), and TCP sender in Listing 2-1 (Section 2.1.2).

When Is a "Little in the Middle" OK? The Internet's End-to-End Principle Faces More Debate; by Gregory Goth -- http://ieeexplore.ieee.org/iel5/8968/28687/01285878.pdf?isnumber

Why it's time to let the OSI model die; by Steve Taylor and Jim Metzler, Network World, 09/23/2008 -- http://www.networkworld.com/newsletters/frame/2008/092208wan1.html

Open Systems Interconnection (OSI) Reference Model

The OSI model has seven layers (Figure 1-16). The layer functionality is as follows:

Layer 7 – Application: Its function is to provide application-specific services. Examples include call establishment and management for a telephony application (SIP protocol, Section 9.6.2), mail services for e-mail forwarding and storage (SMTP protocol), and directory services for looking up global information about various network objects and services (LDAP protocol). Note that the application layer is distinct from the application itself, which provides business logic and user interface.

Visit http://en.wikipedia.org/wiki/OSI_model for more details on the OSI Reference Architecture

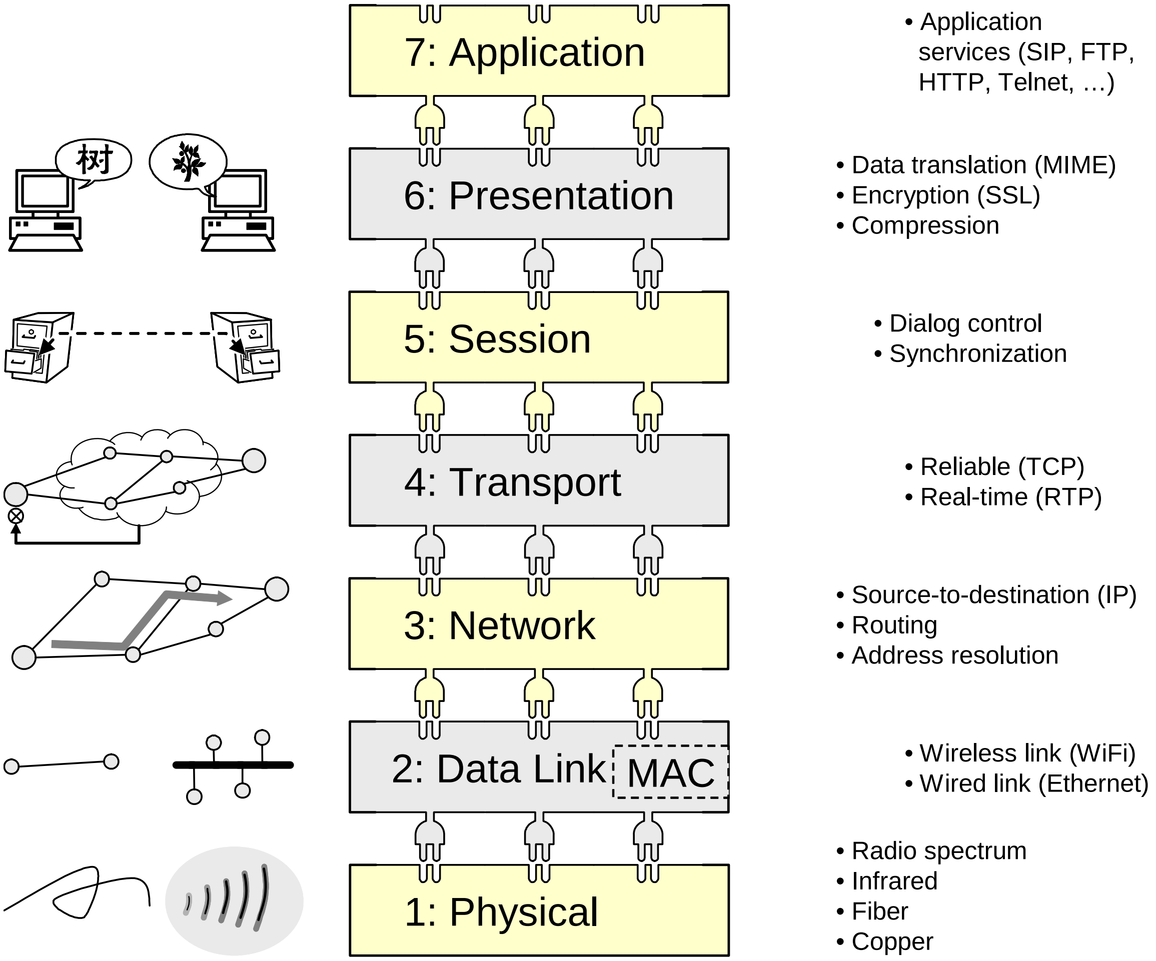


Figure 1-16: OSI reference architecture for communication protocol layering.

Layer 6 – Presentation: Its function is to "dress" the messages in a "standard" manner. It is sometimes called the *syntax layer* because it deals with the syntax and semantics of the data exchanged between the network nodes. This layer performs *translation* between data representations and formats to support interoperability between different encoding systems (ASCII vs. Unicode) or hardware architectures (the least significant bit of data representations may be the first or the last). It also performs *encryption* and *decryption* of sensitive information. Lastly, this layer also performs data *compression* to reduce the number of bits to be transmitted, which is particularly important for multimedia data (audio and video).

Layer 5 – Session: Its function is to maintain a "conversation" across multiple related message exchanges between two hosts (called *session*), to keep track of the progress of their communication. This layer establishes, authenticates, manages, and terminates sessions. Example services include keeping track of whose turn it is to transmit (dialog control) and checkpointing long conversations to allow them to resume after a crash.

Layer 4 – Transport: Its function is to provide reliable or expedient delivery of messages from end to end, or error recovery. (The Transport layer is the interface Unix programmers see when transmitting information over *TCP/IP sockets* between two Unix processes.)

Layer 3 – Network: Its function is to move packets from source to destination in an efficient manner (called *routing*), e.g., along the shortest path, and to provide internetworking of different network types (a key service is address resolution across different networks or network layers).

Layer 2 – Data Link: Its function is to organize bits into packets or frames, and to provide packet exchange between adjacent nodes. Figure 1-16 indicates the MAC (medium access control) sub-layer of the Data Link layer. MAC protocol is needed for broadcast links to coordinate the link use by multiple senders (Section 1.3.3).

Layer 1 – Physical: Its function is to transmit bits over a physical link, such as copper wire or air, and to provide mechanical and electrical specifications.

The seven layers can be conceptually organized to three subgroups. First, Layers 1, 2, and 3—physical, link, and network—are the *network support layers*. They deal with the physical aspects of moving data from one device to another, such as electrical specifications, physical connections, physical addressing, etc. Second, Layers 5, 6, and 7—session, presentation, and application—can be thought of as *user support layers*. They allow interoperability among unrelated software systems. Third, Layer 4—the transport layer—ensures end-to-end reliable data transmission (although Layer 2 may additionally ensure reliable data transmission on individual links).

When compared to the three-layer model (Figure 1-11), OSI layers 1, 2 correspond to layer 1, the Link layer, in the three-layer model. OSI layer 3 corresponds to layer 2, the Network layer, in the three-layer model. Finally, OSI layers 4, 5, 6, and 7 correspond to layer 3, the End-to-end layer, in the three-layer model.

The OSI model serves mainly as a reference for thinking about protocol architecture issues. There are no actual protocol implementations that follow the OSI model. Because it is dated, the rest of this book mainly relies on the three-layer model.

1.2 Reliable Transmission via Redundancy

To counter the line noise, a common technique is to add redundancy or context to the message. Imagine you are transmitting a message to a friend. If your friend already knows the message contents, then there may be no point in transmitting it. At a minimum, your friend needs to know how to interpret the individual symbols of a message. Assume that the message carries the outcomes of a sequence of coin tosses (head/tail). The simplest approach is to transmit "1" for head and "0" for tail. However, a bit flipped by noise would result in a catastrophic loss of information. Instead of transmitting "1" or "0", for every head you can transmit "111" and for every tail you can transmit "000." The advantage of sending two redundant symbols is that if one of the original symbols flips, say "000" is sent and "001" is received, the receiver simply guess that the original message is "000," which represents a tail outcome. This guessing is aided by two assumptions. First, the sender and receiver have agreed on a dictionary of permissible "code words," which in this case are "111" and "000." Upon receiving "001," your friend realizes that this is not a permissible word. Second, the receiver assumes that the original code word that was sent is the closest word in the dictionary, which is "000." Of course, if two symbols become flipped, catastrophically, so “000” turns to “011,” then the receiver would erroneously infer that the original is “head.” We can make messages more robust to noise by adding greater redundancy. Therefore, instead of two redundant symbols, we could have ten: for every “head” you could transmit “1111111111” and for every “tail” you could transmit “0000000000.” The probability that the message will be corrupted by noise catastrophically becomes progressively lower with more redundancy. However, there is an associated penalty: the economic cost of transmitting the longer message grows higher because every communication line can transmit only a limited number of bits per unit of time. Transmitting redundant information takes away from the capacity for transmitting true information. Finding the right *tradeoff* between robustness and cost requires the knowledge of the physical characteristics of the transmission line as well as the knowledge about the importance of the message to the receiver (and the sender).

Relying on a dictionary of permissible code words is a common technique for correcting errors. For example, when you are typing the word “information” and by mistake you enter “inrtormation,” the spell-checking software quickly figures out that the intended word is “information,” because this is the closest meaningful (i.e., permissible) word in the dictionary. In this case, the redundancy is present in the fact that all other words differ by many letters from the word “information.” A contrasting example is the word “than” that can quickly become “thin” or “then” and the dictionary-based approach would not help. (What may help is knowing the larger context of the sentence or paragraph.) One could design a more economic dictionary than the dictionaries of human languages, by using much fewer letters per word. The advantage would be, for example, that this book would have much fewer pages of text. However, the disadvantage of removing redundancy is that it becomes difficult or impossible to correct spelling errors.

Example of adding redundancy to make messages more robust will be seen in Internet telephony (VoIP), where forward error correction (FEC) is used to counter the noise effects.

It is always easier to detect errors than to (detect and) correct errors. If damage or loss can be detected, then an option is to request retransmission but, the request and retransmission take time, which results in increased response latency. FEC is better but incurs overhead.

1.2.1 Error Detection and Correction by Channel Coding

To bring the message home, here is a very simplified example for the above discussion. Note that this oversimplifies many aspects of error coding to get down to the essence. Assume that you need to transmit 5 different messages, each message containing a single integer number between 1 – 5. You are allowed to “encode” the messages by mapping each message to a number between 1 – 100. Assume that the noise amplitude is distributed according to the normal distribution, as shown in [Figure X]. What are the best choices for the codebook?

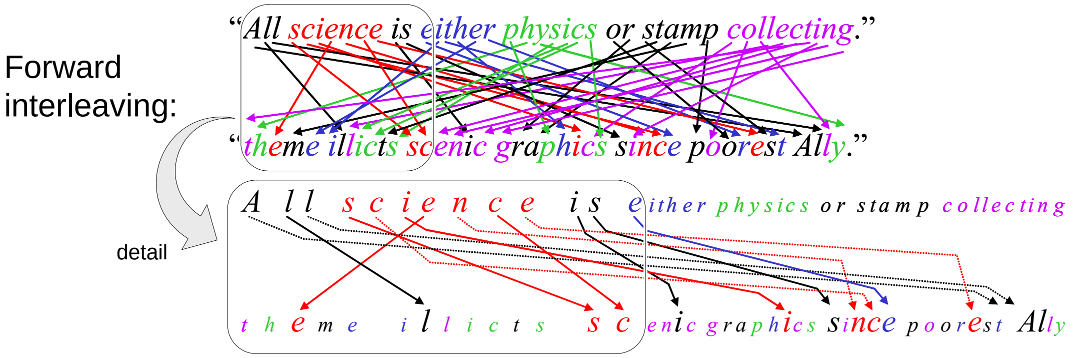
Note: this really represents a *continuous* case, not digital, because numbers are not binary and errors are not binary. But just for the sake of simplicity…

1.2.2 Interleaving

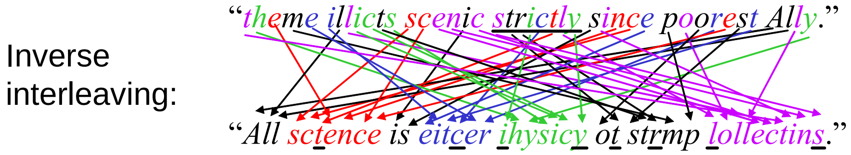
Redundancy and error-correcting codes are useful when errors are randomly distributed. If errors are clustered, they are not effective. Consider the following example. Say you want to send the following message to a friend: “*All science is either physics or stamp collecting*.”[2] A random noise in the communication channel may result in the following distorted message received by your friend: “*All scidnce is eitjer physocs or statp colletting*.” By simply using a spelling checker, your friend may easily recover the original message. One the other hand, if the errors were clustered, the received message may appear as: “*All science is either checker or stamp collecting*.” Obviously, it is impossible to guess the original message unless you already know what Rutherford said.

This kind of clustered error is usually caused by a *jamming source*. It may not necessarily be a hostile adversary trying to prevent the communication, but it could be a passive narrow-band jamming source, such as microwave owen, which operates in the same frequency range as Wi-Fi wireless networking technology (Section 1.5.3).

To recover from such errors, one can use **interleaving**. Let us assume that instead of sending the original message as-is, you first scramble the letters and obtain the following message:


Forward interleaving:
“All science is either physics or stamp collecting.”
“theme illicts scenic graphics since poorest Ally.”
detail
A l l s c i e n c e i s either physics or stamp collecting
t h e m e i l l i c t s s c enic graphics since poorest Ally

Now you transmit the message “*theme illicts scenic graphics since poorest Ally*.” Again, the jamming source inflicts a cluster of errors, so the word “graphics” turns into “strictly,” and your friend receives the following message: “*theme illicts scenic strictly since poorest Ally.*” Your friend must know how to unscramble the message by applying an inverse mapping to obtain:


Inverse interleaving:
“theme illicts scenic strictly since poorest Ally.”
“All sctence is eitcer ihysicy ot strmp lollectins.”

Therefore, with interleaving, the receiver will obtain a message with errors randomly distributed, rather than missing a complete word. By applying a spelling checker, your friend will recover the original message.

[2] Ernest Rutherford, in J. B. Birks, “Rutherford at Manchester,” 1962.

1.3 Reliable Transmission by Retransmission

We introduced channel encoding as a method for dealing with errors (Section 1.2). However, encoding provides only probabilistic guarantees about the error rates—it can *reduce* the number errors to an arbitrarily small amount, but it cannot *eliminate* them. When error is detected that cannot be corrected, it may be remedied by repeated transmission. This is the task for Automatic Repeat Request (ARQ) protocols. In case retransmission fails, the sender should *persist* with repeated retransmissions until it succeeds or decides to give up. Of course, even ARQ retransmission is a probabilistic way of ensuring reliability and the sender should not persist forever with retransmissions. After all, the link to the receiver may be broken, or the receiver may be dead. There is no absolutely certain way to guarantee reliable transmission.

Failed transmissions manifest themselves in two ways:

- Packet error: Receiver receives the packet and discovers error via error control
- Packet loss: Receiver never receives the packet (or fails to recognize it as such)

If the former, the receiver can request retransmission. If the latter, the sender must detect the loss by the lack of confirmation from the receiver within a given amount of time.

Common requirements for a reliable protocol are that: (1) it delivers one and only one copy of a given packet to the receiving application; and, (2) all packets are delivered in the same order they are presented to the sender. “Good” protocol:

- Delivers a single copy of every packet to the receiver application
- Delivers the packets in the order they were presented to the sender

A lost or damaged packet should be retransmitted. A **persistent sender** is a protocol participant that tries to ensure that at least one copy of each packet successfully reaches the receiver protocol, by sending repeatedly until it receives an acknowledgment. The receiving protocol may receive duplicate packets, but must deliver no more than a single copy to the receiving application. To make retransmission possible, a copy is kept in the transmit buffer (temporary local storage) until the sender receives the acknowledgment that the packet is successfully received by the receiver. Buffering generally uses the fastest memory chips and circuits and, therefore, the most expensive memory, which means that the buffering space is scarce. Disk storage is cheap but not practical for packet buffering because it provides relatively slow data access.

On their way through the network, different packets can take different routes to the destination, and thus arrive in a different order than sent. The receiver may temporarily store (buffer) the out-of-order packets until the missing packets arrive. Different ARQ protocols are designed by making different choices for the following issues:

- Where to buffer: at sender only, or both sender and receiver?
- What is the maximum allowed number of outstanding packets, waiting to be acknowledged?

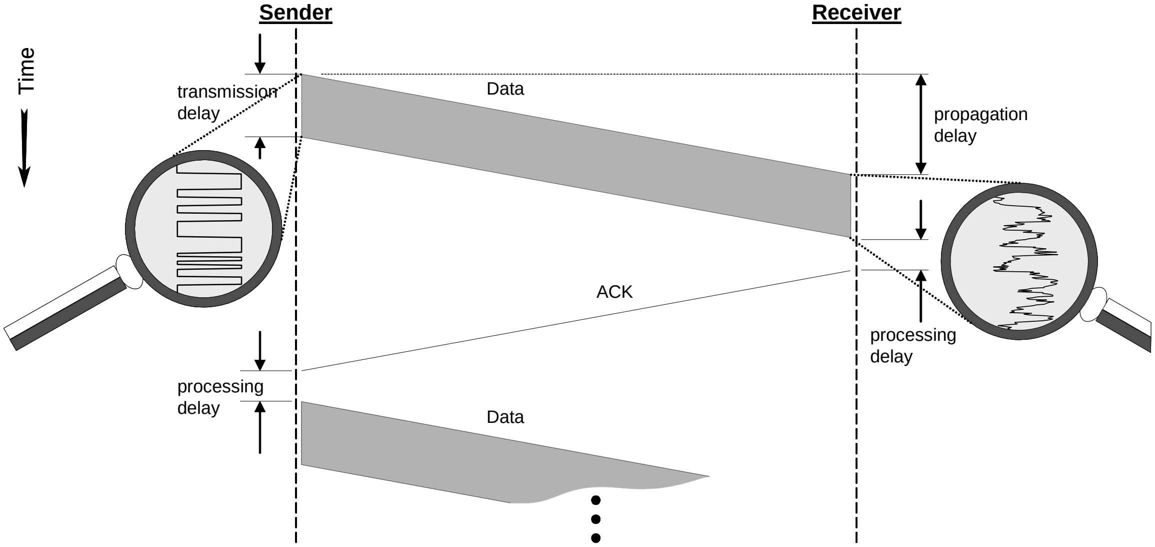


Figure 1-17: Timeline diagram for reliable data transmission with acknowledgments.

- How is a packet loss detected: a timer expires, or the receiver explicitly sends a “negative acknowledgment” (NAK)? (Assuming that the receiver is able to detect a damaged or missing packet.)

Another parameter needed for reliable communication is a **sequence number** in the packet header to identify successive packets. A sequence number allows the receiver to detect missing packets lost in transit, or recognize duplicate packets that it has already received.

The **sender utilization** of an ARQ connection is defined as the fraction of time that the sender is busy sending data (regardless of whether they are successfully received by the receiver).

The **throughput** of an ARQ connection is defined as the average rate of successful message delivery.

The **goodput** of an ARQ connection is defined as the rate at which data are sent uniquely, i.e., this rate does not include error-free data that reach the receiver as duplicates. In other words, the goodput is the fraction of time that the receiver is receiving data that it has not received before.

The transmissions of packets between a sender and a receiver are usually illustrated on a timeline as in Figure 1-17. There are several types of delay associated with packet transmissions. To illustrate, here is an analogy: you are in your office, plan to go home, and on your way home you will stop at the bank to deposit your paycheck. From the moment you start, you will get down to the garage (“transmission delay”), drive to the bank (“propagation delay”), wait in the line (“queuing delay”), get served at teller’s window (“processing delay” or “service delay”), and drive home (additional “propagation delay”).

The first delay type is **transmission delay**, which is the time that takes the sender to place the data bits of a packet onto the transmission medium. In other words, transmission delay is measured at the sender from when the first bit of a packet enters a link until the last bit of that same packet enters the link. This delay depends on the transmission rate *R* offered by the medium (in bits per second or bps), which determines how many bits (or pulses) can be generated per unit of time at the transmitter (Section 1.1.2). It also depends on the length *L* of the packet (in bits). Hence, the transmission delay is:

$$t_x = \frac{\text{packet length}}{\text{bandwidth}} = \frac{L \text{ (bits)}}{R \text{ (bits per second)}} \tag{1.2}$$

Propagation delay is defined as the time elapsed between when a bit is sent at the sender and when it is received at the receiver. This delay depends on the distance *d* between the sender and the receiver and the velocity *v* of electromagnetic waves in the transmission medium, which is proportional to the speed of light in vacuum ($c \approx 3{\times}10^8$ m/s), $v = c/n$, where *n* is the index of refraction of the medium. Both in copper wire and glass fiber or optical fiber $n \approx 3/2$, so $v \approx 2 \times 10^8$ m/s. The index of refraction for dry air is approximately equal to 1. The propagation delay is:

$$t_p = \frac{\text{distance}}{\text{velocity}} = \frac{d \text{ (m)}}{v \text{ (m/s)}} \tag{1.3}$$

Processing delay is the time needed for processing a received packet. At the sender side, the packet may be received from an upper-layer protocol or from the application. At the receiver side, the packet is received from the network or from a lower-layer protocol. Examples of processing include conversion of a stream of bytes to frames or packets (known as *framing* or *packetization*), data compression, encryption, relaying at routers, etc. Processing delays usually can be ignored when looking from an end-host's viewpoint. However, processing delay is very critical for routers in the network core that need to relay a huge number of packets per unit of time, as will be seen later in Section 1.4.4.

Another important parameter is the **round-trip time** (or **RTT**), which is the time a bit of information takes from departing until arriving back to the sender if it is immediately bounced back at the receiver. This time on a single transmission link is often assumed to equal RTT = $2 \times t_p$. Determining the RTT is much more complex if the sender and receiver are connected over a network where multiple alternative paths exist, as will be seen later in Section 2.1.3. However, even on a single link, the notion of RTT is much more complex than just double the propagation delay. To better understand RTT, we need to consider what it is used for and how it is measured. RTT is most often used by a sender to set up its retransmission timer in case a packet is lost. Obviously, network nodes do not send individual bits; they send packets. RTT is measured by recording the time when a packet is sent, reading out the time when the acknowledgment is received, and subtracting these two values:

$$\text{RTT} = (\text{time when the acknowledgment is received}) - (\text{time when the packet is sent}) \tag{1.4}$$

To understand what contributes to RTT, we need to look at how packets travel through the network. First, acknowledgments may be *piggybacked* on data packets coming back for the receiver. Therefore, even if the transmission delay is not included at the sender side, receiver's transmission delay does contribute to the RTT. (However, when an acknowledgment is piggybacked on a regular data packet from the receiver to the sender, the transmission time of this packet must be taken into account.)

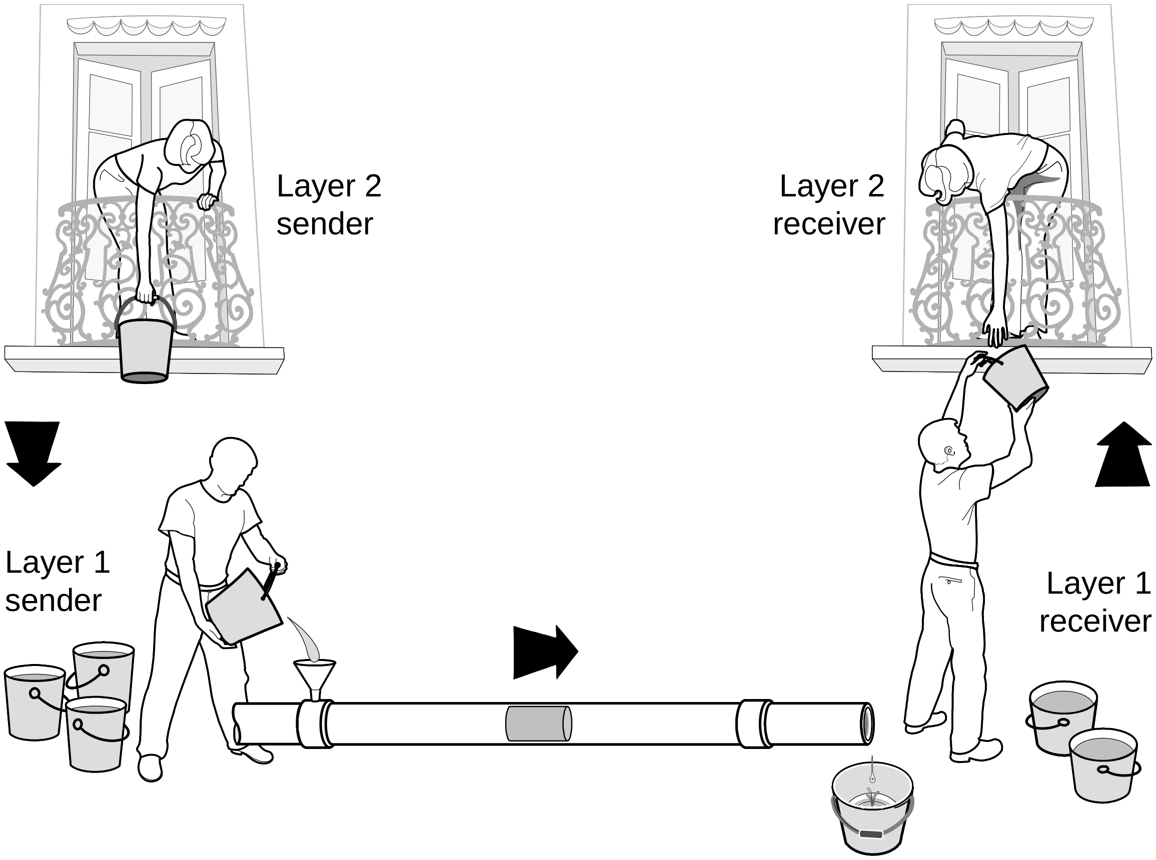


Figure 1-18: Fluid flow analogy for delays in packet delivery between the protocol layers. The key point of this illustration is to highlight the passage of time. In real world, everything takes time, and so does in the digital network world.

Second, we have to remember that network nodes use layered protocols (Section 1.1.4). Continuing with the fluid flow analogy from Figure 1-5, we illustrate in Figure 1-18 how delays are introduced between the protocol layers. If the ultimate sender is at Layer 2 or above, then the RTT calculation includes the transmission time at the physical-layer (Layer 1)—emptying a bucket in Figure 1-18. The physical-layer receiver waits until a bucket is full (i.e., the whole packet is received) before it delivers it to the upper layer (Layer 2).

The delay components for a single link and a three-layer protocol are illustrated in Figure 1-19. Sender's transmission delay will not be included in the measured RTT only if the sender operates at the link/physical layer. A sender operating at any higher layer (e.g., network or transport layers), cannot avoid having the transmission delay included in the measured RTT, because it cannot know when the packet transmission on the physical medium will actually start or end.

Third, lower layers of sender's protocol stack may incur significant *processing delays*. Suppose that the sender is at the OSI Transport layer and it measures the RTT to receive the acknowledgment from the receiver, which is also at the OSI Transport layer (Figure 1-16). When a lower layer receives a packet from a higher layer, the lower layer may not forward the packet immediately, because it may be busy with sending some other packets. Also, if the lower layer uses error control, it will incur processing delay while calculating the checksum or some other type of error-control code. Later we will learn about other types of processing delays, such as

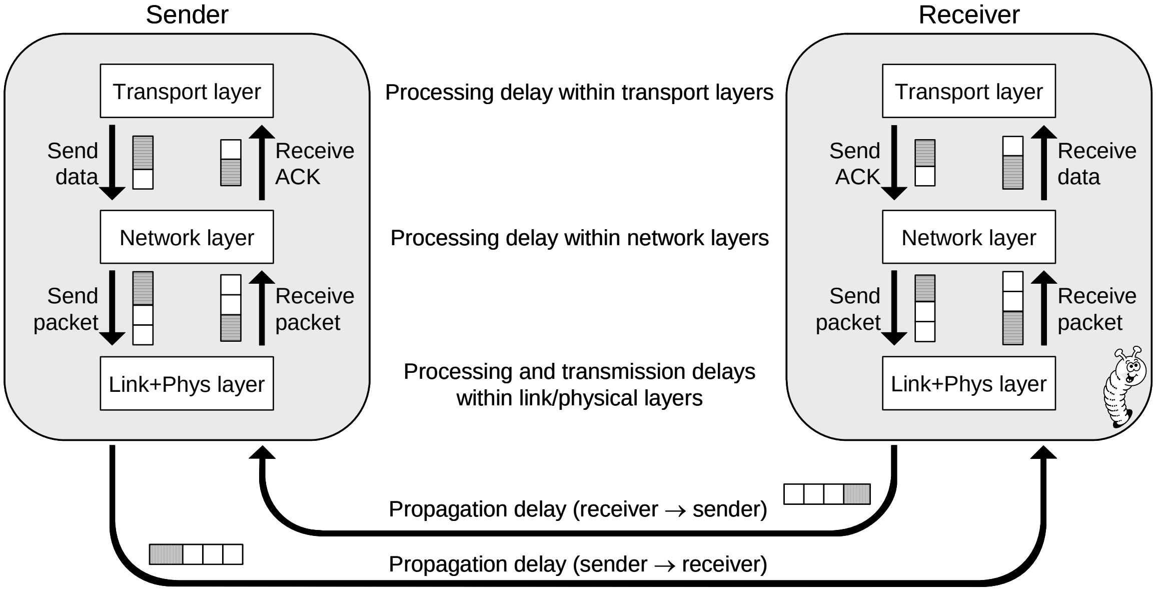

Sender
Receiver
Transport layer
Send data
Receive ACK
Network layer
Send packet
Receive packet
Link+Phys layer
Transport layer
Send ACK
Receive data
Network layer
Send packet
Receive packet
Link+Phys layer
Processing delay within transport layers
Processing delay within network layers
Processing and transmission delays within link/physical layers
Propagation delay (receiver → sender)
Propagation delay (sender → receiver)

Figure 1-19: Delay components that contribute to round-trip time (RTT).

time spent looking up forwarding tables in routers (Section 1.4), time spent dividing a long message into fragments and later reassembling it (Section 1.4.1), time spent compressing data, time spent encrypting and decrypting message contents, etc.

Fourth, lower layers may implement their own reliable transmission service, which is transparent to the higher layer. An example are broadcast links (Section 1.3.3), which keep retransmitting lost packets until reaching the limit on the number of attempts. The question, then, is, What counts as the transmission delay for a packet sent by a higher layer and transmitted by a lower layer, which included several retransmissions? Should we count only the successful transmission (the last one), or the preceding unsuccessful transmissions, as well?

In summary, the reader should be aware that RTT estimation is a complex issue even for a scenario of a single communication link connecting the sender and receiver. Although RTT is often approximated as double the propagation delay, this may be grossly inaccurate and the reader should examine the feasibility of this approximation individually for each scenario.

Mechanisms needed for reliable transmission by retransmission:

- Error detection for received packets, e.g., by checksum
- Receiver feedback to the sender, via acknowledgment or negative acknowledgment
- Retransmission of a failed packet, which requires storing the packet at the sender until the sender obtains a positive acknowledgment that the packet reached the receiver error-free
- Sequence numbers, so the receiver can distinguish duplicate packets
- Retransmission timer, if packet loss on the channel is possible (not only error corruption), so that the sender can detect the loss

Several popular ARQ (Automatic Repeat Request) protocols are described next.

1.3.1 Stop-and-Wait

Problems related to this section: Problem 1.2 → Problem 1.4; also see Problem 1.12

The simplest retransmission strategy is *stop-and-wait*. In this protocol, the sender buffers only a single packet at the sender and does not send the next packet before ensuring that the current packet is correctly received (Figure 1-17). A timer is also set when a packet is transmitted packet to detect packet loss by the timer expiration.

When the sender receives a corrupted ACK/NAK or detects a gap in the sequence numbers of received packets, it could send back to the receiver a NAK (negative acknowledgment). For pragmatic reasons (to keep the receiver software simple), the receiver does nothing and the sender just re-sends the packet when its retransmission timer expires.

Assuming error-free communication, the utilization of a Stop-and-wait sender is determined as follows. The entire cycle to transport a single packet takes a total of $(t_x + 2 \times t_p)$ time. (We assume that the acknowledgment packets are tiny, so their transmission time is negligible.) Of this time, the sender is busy t_x time. Therefore:

$$U_{sender}^{S\&W} = \frac{t_x}{t_x + 2 \cdot t_p} \tag{1.5}$$

Given a probability of packet transmission error p_e, which can be computed using Eq. (1.1), we can determine *how many times*, on average, a packet will be (re-)transmitted until successfully received and acknowledged. This value is known as the **expected number of transmissions** per packet. Our simplifying assumption is that error occurrences in successively transmitted packets are independent events[3]. A successful transmission in one round requires error-free transmission of two packets: forward data and feedback acknowledgment. We again assume that these are independent events, so the joint probability of success is:

$$p_{\text{succ}} = \left(1 - p_e^{\text{DATA}}\right) \cdot \left(1 - p_e^{\text{ACK}}\right) \tag{1.6}$$

The probability of a failed transmission in one round is $p_{\text{fail}} = 1 - p_{\text{succ}}$. Then, the number of attempts K needed to transmit successfully a packet is a *geometric random variable*. The probability that the first k attempts will fail and the $(k+1)^{\text{st}}$ attempt will succeed equals:

$$\Pr(K = k) = \left(1 - p_{\text{succ}}\right)^k \cdot p_{\text{succ}} = p_{\text{fail}}^k \cdot p_{\text{succ}} \tag{1.7}$$

where $k = 0, 1, 2, \ldots$. The round in which the packet is successfully transmitted is a random variable N, with the probability distribution function given by (1.7). Its expected value is the summation of each random variable value multiplied by the probability of that value:

$$E\{N\} = \sum_{k=0}^{\infty} k \cdot \Pr(K = k) = \sum_{k=0}^{\infty} k \cdot p_{\text{fail}}^k \cdot p_{\text{succ}} = p_{\text{succ}} \cdot \left(\sum_{k=0}^{\infty} p_{\text{fail}}^k + \sum_{k-0}^{\infty} k \cdot p_{\text{fail}}^k \right)$$

Recall that the summation formula for the geometric series is:

[3] This assumption is valid only if we assume that thermal noise alone affects packet errors. However, the independence assumption will not be valid for temporary interference in the environment, such as a microwave oven interference on the wireless channel.

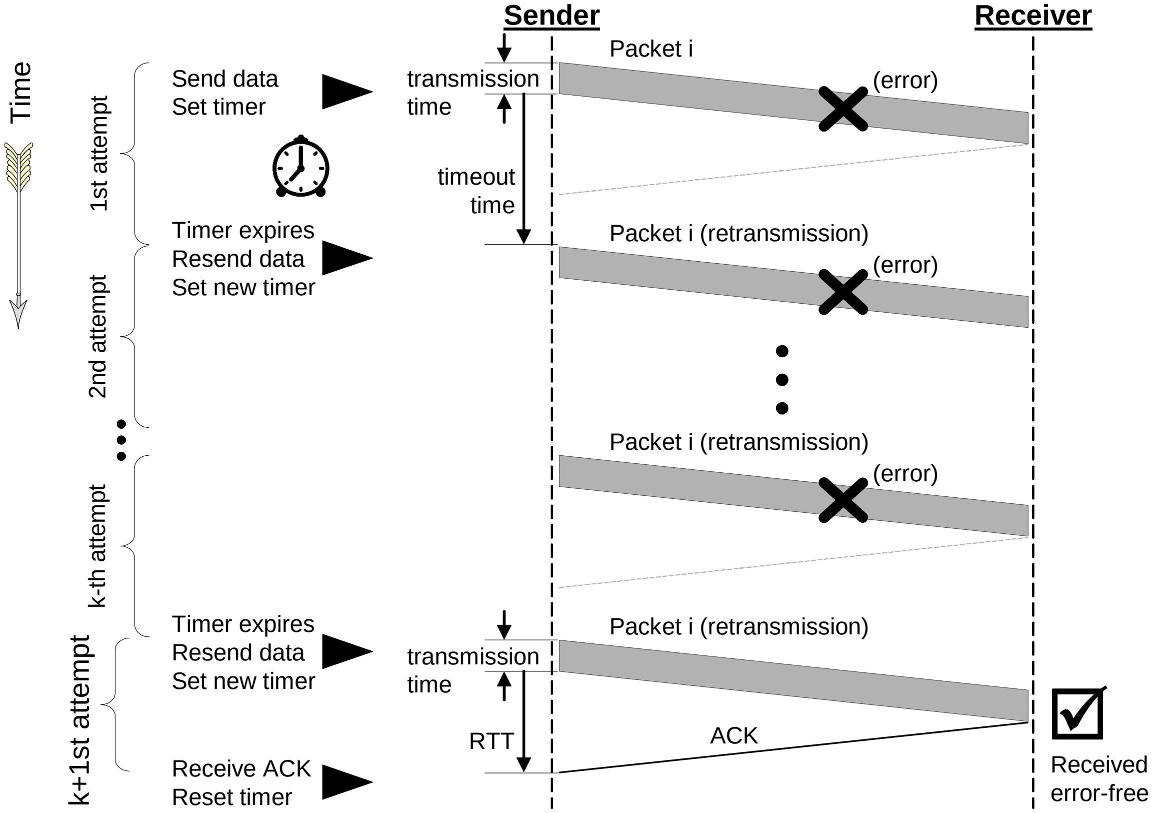


Figure 1-20: Stop-and-Wait with errors. The transmission succeeds after *k* failed attempts.

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}, \qquad \sum_{k=0}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2}$$

Therefore, we obtain the expected number of transmissions (recall that $p_{\text{fail}} = 1 - p_{\text{succ}}$):

$$E\{N\} = p_{\text{succ}} \cdot \left(\frac{1}{1-p_{\text{fail}}} + \frac{p_{\text{fail}}}{(1-p_{\text{fail}})^2} \right) = p_{\text{succ}} \cdot \frac{1}{(1-p_{\text{fail}})^2} = \frac{1}{p_{\text{succ}}} \tag{1.8}$$

We can also determine the **expected delay per packet** as follows. Successful transmission of one packet takes a total of $t_{\text{succ}} = t_x + 2 \times t_p$, assuming that transmission time for the acknowledgment packet can be ignored. A single failed packet transmission takes a total of $t_{\text{fail}} = t_x + t_{\text{out}}$, where t_{out} is retransmission timer's countdown time. If a packet is successfully transmitted after k failed attempts, then its total transmission time equals: $T_{k+1}^{\text{total}} = k \cdot t_{\text{fail}} + t_{\text{succ}}$, where k = 0, 1, 2, … (see Figure 1-20). The total transmission time for a packet is a random variable T_{k+1}^{total}, with the probability distribution function given by (1.7). Its expected value is:

$$E\{T^{\text{total}}\} = \sum_{k=0}^{\infty} \underbrace{\left(k \cdot t_{\text{fail}} + t_{\text{succ}}\right)}_{\text{rnd.var. value}} \cdot \underbrace{p_{\text{fail}}^k \cdot p_{\text{succ}}}_{\text{probability}} = p_{\text{succ}} \cdot \left(t_{\text{succ}} \sum_{k=0}^{\infty} p_{\text{fail}}^k + t_{\text{fail}} \sum_{k=0}^{\infty} k \cdot p_{\text{fail}}^k \right)$$

Following a derivation similar as for Eq. (1.8), we obtain the expected delay per packet:

$$E\{T^{\text{total}}\} = p_{\text{succ}} \cdot \left(\frac{t_{\text{succ}}}{1-p_{\text{fail}}} + \frac{p_{\text{fail}} \cdot t_{\text{fail}}}{\left(1-p_{\text{fail}}\right)^2} \right) = t_{\text{succ}} + \frac{p_{\text{fail}}}{p_{\text{succ}}} \cdot t_{\text{fail}} \tag{1.9}$$

The expected sender utilization in case of a noisy link is:

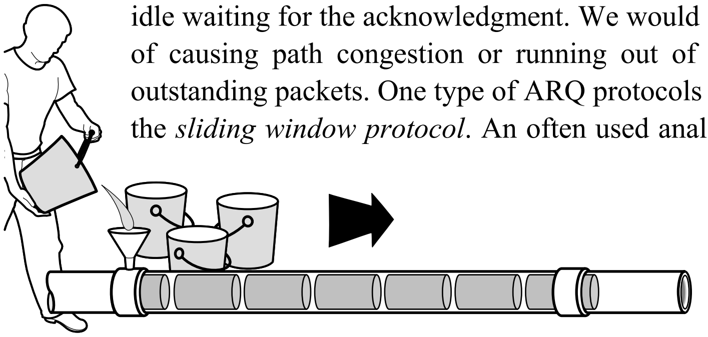
$$E\left\{U^{S\&W}_{sender}\right\} = \frac{t_x \cdot E\{N\}}{E\{T^{\text{total}}\}} = \frac{t_x}{p_{\text{succ}} \cdot t_{\text{succ}} + p_{\text{fail}} \cdot t_{\text{fail}}} \tag{1.10}$$

Here, we are considering the expected fraction of time the sender will be busy out of the total expected time to transmit a packet successfully. That is, ($t_x \cdot E\{N\}$) includes both unsuccessful attempted transmissions and the successful transmission (the last one).

1.3.2 Sliding-Window Protocols

Problems related to this section: Problem 1.5 → Problem 1.12

Stop-and-wait is very simple but also very inefficient, because the sender spends most of the time idle waiting for the acknowledgment. We would like the sender to send as much as possible, short of causing path congestion or running out of the memory space for buffering copies of the outstanding packets. One type of ARQ protocols that offer higher efficiency than Stop-and-wait is the *sliding window protocol*. An often used analogy for sender utilization is that of "keeping the pipe full." Sliding window protocols are better at keeping the pipe full and the sender utilized. This is desirable because the sender should be busy sending as long as it has packets ready to send.



The **sender window size** N is a measure of the maximum number of outstanding (i.e., unacknowledged) packets in the network. Figure 1-21 shows the operation of sliding window protocols in case of no errors in communication. The **receiver window size** W gives the upper bound on the number of out-of-order packets that the receiver is willing to accept. The receiver temporarily stores these packets until the missing packets arrive and fill the gap, so it can deliver all packets in-order to the receiving application. In the case shown in Figure 1-21, both sender and receiver have the same window size. The invariant for window-based protocols is that $N \leq W$.

A sliding-window sender should keep in a temporary memory ("buffer") all outstanding packets for which the acknowledgment has not yet arrived. Therefore, the send buffer size should be N packets large. The sent-but-unacknowledged packets are called "in-flight" packets or "in-transit" packets. The sender must ensure that the number of in-flight packets is always $\leq N$.

The sliding window protocol is actually a family of protocols that have some characteristics in common and others different. Next, we review two popular types of sliding window protocols: Go-back-*N* (GBN) and Selective Repeat (SR). The TCP protocol described in Chapter 2 is another example of a sliding window protocol. The key difference between GBN and SR is in the way they manage communication errors.

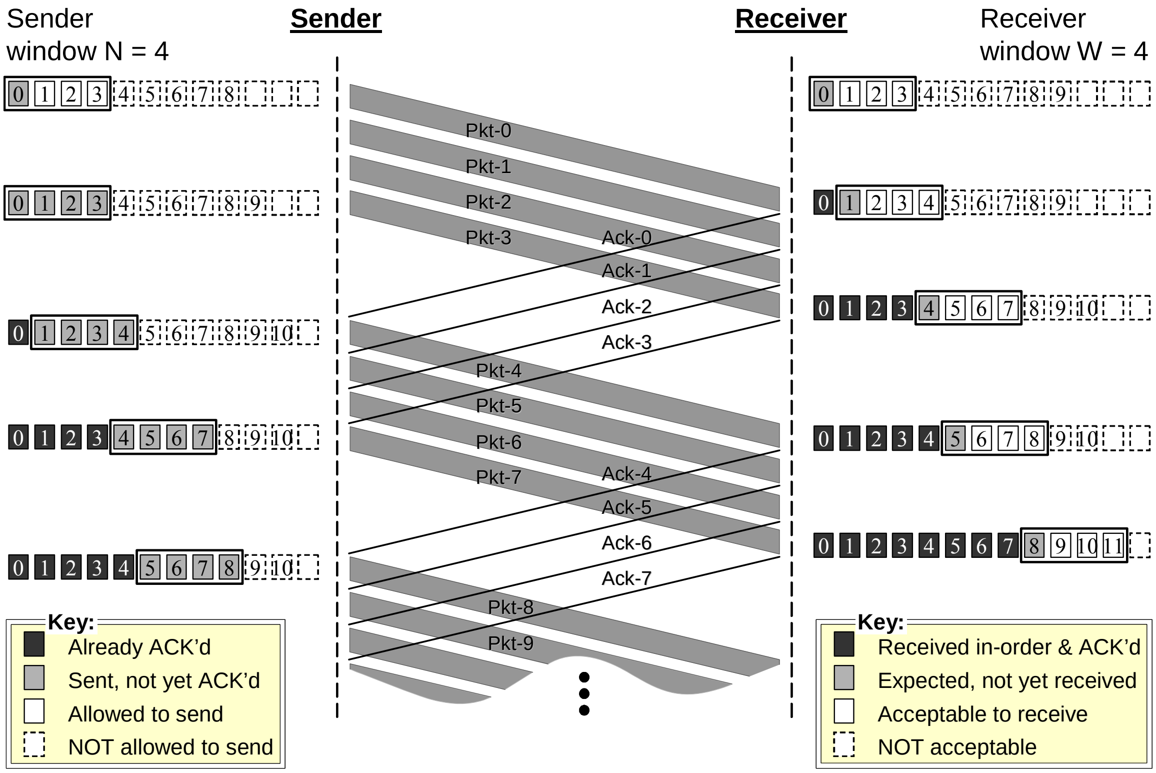


Figure 1-21: Sliding window protocol in operation over an error-free link.

Go-back-*N*

The key idea of the Go-back-*N* protocol is to keep the receiver design simple. This means that the receiver accepts only the next expected packet and immediately discards any packets received out-of-order. Hence, the receiver needs only to memorize the sequence number of the next expected packet (single variable), and does not have any memory to buffer out-of-order packets.

As with all sliding-window protocols, the Go-back-*N* sender should be able to buffer up to *N* outstanding packets. Buffered packets are retransmitted in case of suspected packet loss.

The operation of Go-back-*N* is illustrated in Figure 1-22, which shows a scenario where the second data packet and the fifth acknowledgment packet are lost. The sender sends sender-window-size packets ($N = 3$) and stops, waiting for acknowledgments to arrive. When Ack-0 arrives, acknowledging the first packet (Pkt-0), the sender slides its window by one and sends the next available packet (Pkt-3). The sender stops again and waits for the next acknowledgment.

Because Pkt-1 is lost, it will never be acknowledged. Although Pkt-2 made it through error-free the first time, it was discarded, because the Go-back-*N* receiver has no memory to buffer out-of-order packets. The sender will wait for an acknowledgment of Pkt-1 until its associated retransmission timer expires. Upon timeout, the Go-back-*N* sender resends *all* packets that have been previously sent but have not yet been acknowledged, that is, all "in-flight" packets. This is where this protocol's name comes from. Because the sender will usually have *N* in-flight packets, a timeout will cause it to go back by *N* and resend the *N* outstanding packets. The rationale for

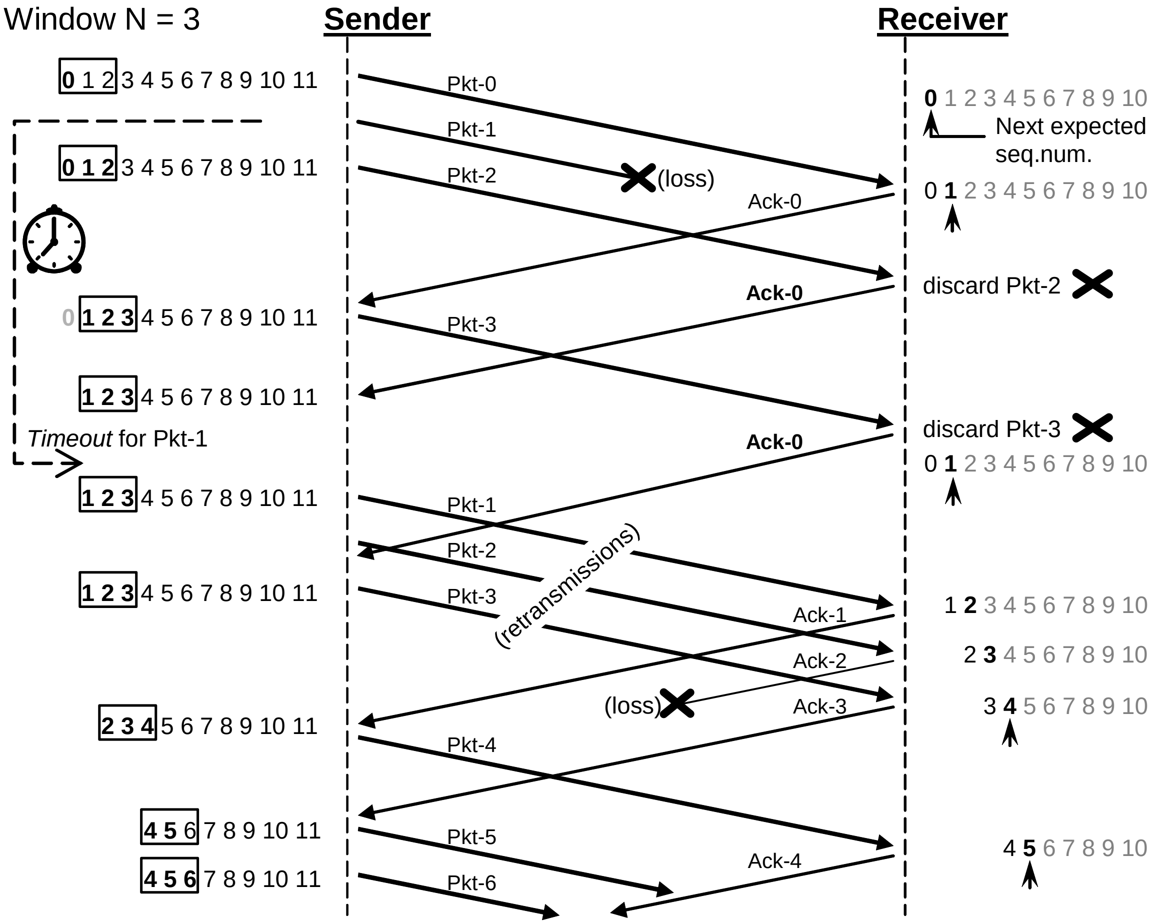


Figure 1-22: Go-back-N protocol in operation under communication errors.

this behavior is that if the oldest outstanding packet is lost, then all the subsequent packets are lost as well, because the Go-back-*N* receiver automatically discards out-of-order packets.

As mentioned, the Go-back-*N* receiver memorizes a single variable—the sequence number of the next expected packet. The receiver considers a packet *correctly received* if and only if:

1. The received packet is error-free
2. The received packet arrived in-order, i.e., its sequence number equals next-expected-sequence-number.

In the example shown in Figure 1-22, Pkt-1 is lost, so Pkt-2 arrives out of order. Because the Go-back-*N* receiver discards any packets received out of order, Pkt-2 is automatically discarded. (If you question the wisdom of discarding correctly received out-of-order packets, wait until we describe the Selective Repeat protocol next. For now, accept this inefficiency as a tradeoff that keeps the receiver design simple.) One salient feature of Go-back-*N* is that the receiver sends **cumulative acknowledgments**, where an acknowledgment with sequence number *m* indicates that all packets with a sequence number up to and including *m* have been correctly received at the receiver. The receiver sends acknowledgment even for incorrectly received packets, but in this case, the previously correctly received packet is being acknowledged. In Figure 1-22, the receipt of Pkt-2 generates a duplicate acknowledgment Ack-0. Note also that when Ack-2 is lost, the

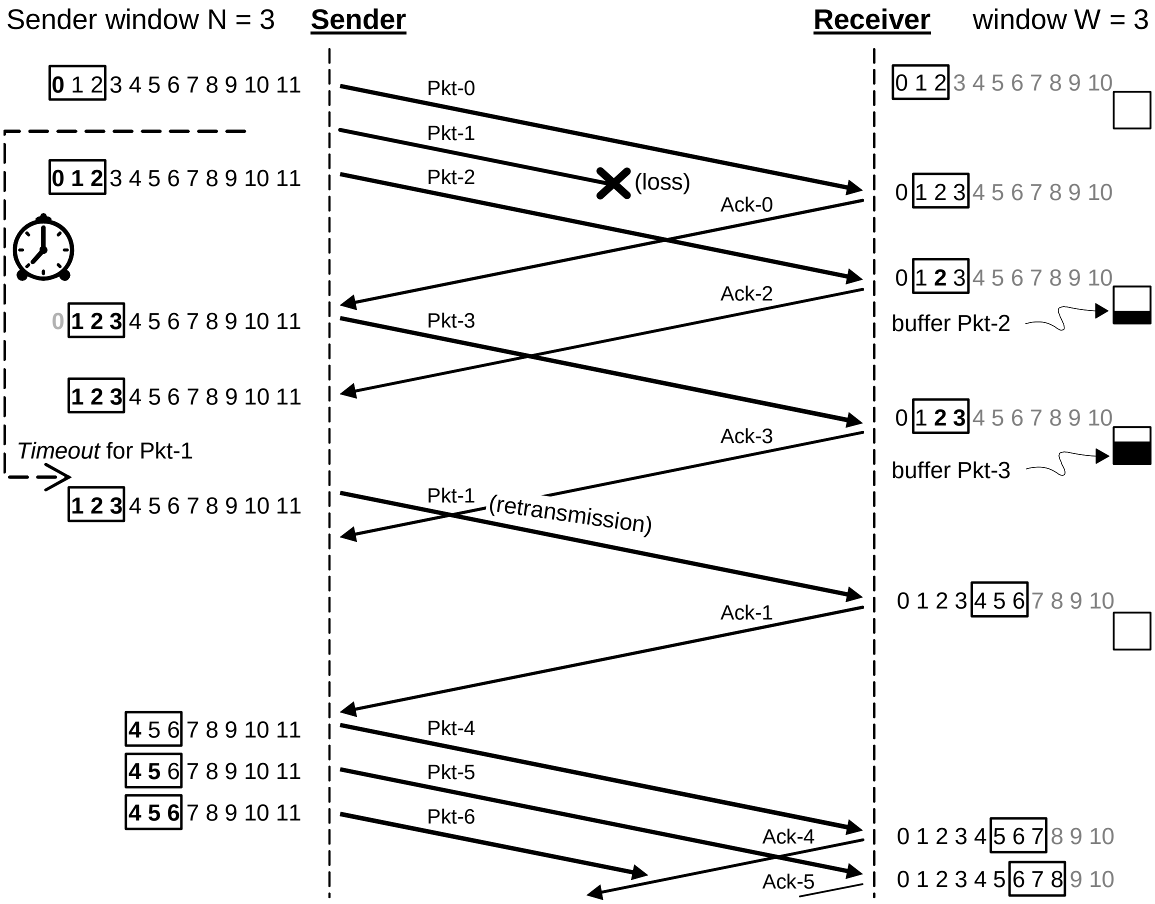


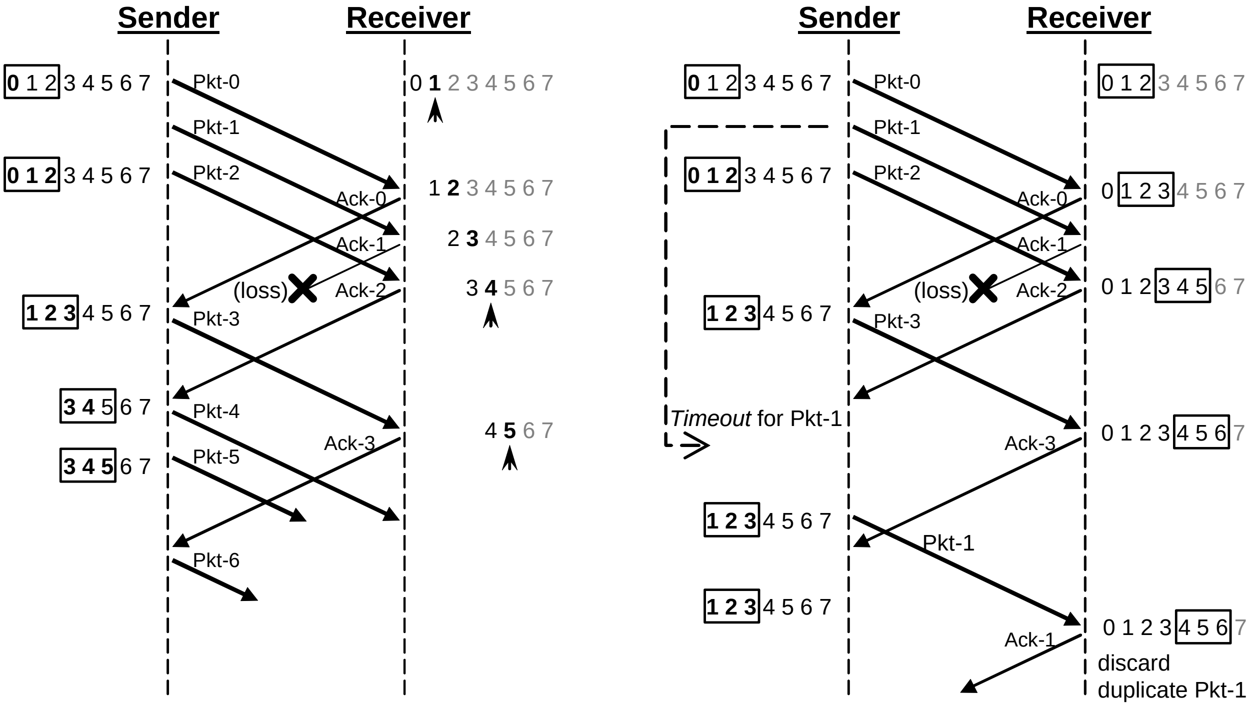
Figure 1-23: Selective Repeat protocol in operation under communication errors.

sender takes Ack-3 to acknowledge all previous packets, including Pkt-2. Hence, a lost acknowledgment does not need to be retransmitted as long as the acknowledgment acknowledging the following packet arrives before the retransmission timer expires. Performance of Go-back-*N* is considered in Problem 1.6, Problem 1.11 and in Example 4.2 (Section 4.2.4).

Selective Repeat (SR)

The key idea of the Selective Repeat protocol is to avoid discarding packets that are received error-free and, therefore, to avoid unnecessary retransmissions. Go-back-*N* suffers from performance problems, because a single packet error can cause Go-back-*N* sender to retransmit a large number of packets, many of them unnecessarily. Figure 1-23 illustrates the operation of the Selective Repeat protocol. Unlike a Go-back-*N* sender which retransmits all outstanding packets when a retransmission timer times out, a Selective Repeat sender retransmits only a single packet—the oldest outstanding one.

Unlike a Go-back-*N* receiver, a Selective Repeat receiver sends **individual acknowledgments**, where an acknowledgment with sequence number *m* indicates only that the packet with sequence number *m* has been correctly received. There is no requirement that packets are received in order. If a packet is received out of order but error-free, it will be buffered in the receiver's memory



(a) Go-back-N

(b) Selective Repeat

Figure 1-24: Comparison of Go-back-N and Selective-Repeat acknowledgments.

until the in-order missing packets are received. Then, all in-order packets are delivered to the receiving application.

Figure 1-24 illustrates the difference between the behaviors of Go-back-*N* (GBN) cumulative acknowledgments and Selective Repeat (SR) individual acknowledgments. Note that both protocols require the sender to acknowledge duplicate packets, which were received and acknowledged earlier. The reason for this requirement is that a duplicate packet usually indicates that the acknowledgment has been lost. Without an acknowledgment, the sender window would never move forward and the communication would come to a halt. (Note also that a packet can be retransmitted when its acknowledgment is delayed, so the timeout occurs before the acknowledgment arrives. In this case, the sender window would simply move forward.) Again, SR acknowledges only the last received (duplicate) packet, whereas GBN *cumulatively* acknowledges all the packets received up to and including the last one. In Figure 1-24(a), Ack-1 was lost, but when Ack-2 arrives it acknowledges all packets up to and including Pkt-2. This acknowledgment shifts the sender's window forward by 2 and the sender advances uninterrupted. Unlike this, in Figure 1-24(b) the SR sender needs to retransmit Pkt-1 because it never receives Ack-1 before its timeout expired.

SIDEBAR 1.1: The Many Faces of Acknowledgments

♦ The attentive reader may have noticed that acknowledgments are used for several purposes. For example, earlier we saw that a received ACK informs the sender that: (a) the corresponding data packet arrived at the receiver; (b) the sender may stop the retransmission-timer countdown for the corresponding data packet; (c) the sender may discard the copy of the acknowledged packet and release the memory buffer space; and, (d) the sender may send another data packet. By sending an ACK, the receiver is also implicitly asking for the next packet to be sent. Note that an acknowledgment usually only confirms that the data packet arrived error-free to the receiver, but it does not say whether the receiver acted upon the received data and completed the required processing. To obtain this knowledge, an additional acknowledgment at the application level is required. In other words, each protocol layer may use its own ACKs and not know about ACKs used by upper or lower layers. Later, in Chapter 2, we will learn about some additional uses of acknowledgments in the TCP protocol.

In practice, a combination of Selective Repeat and Go-back-*N* is used, as will be seen with TCP in Chapter 2.

1.3.3 Broadcast Links

Broadcast links allow connecting many network nodes over the same medium. Hence, when one node transmits, all or most other nodes on the link can hear the transmission. If two or more nodes are transmitting simultaneously, their signals will interfere with each other (see Figure 1-7 for interference on a wireless link). A receiver that receives the interference signal will not be able to decode either of the original signals; this is known as **packet collision**. Therefore, the nodes should take turns transmitting their packets. However, this is easier said than done: when a node has a packet ready for transmission, it does not know whether any other nodes are also about to transmit. A key technical problem for broadcast links is **coordination of transmissions**, to control collisions. This is the task for the medium access control (MAC) sub-layer in Figure 1-16.

The problem of coordination means that a station ready for transmission needs to know when it is safe to do so. In other words, it needs to know if other stations are also ready to send or have an arbiter just tell each station when to send. Generally, techniques for transmission coordination on broadcast links belong to two categories: (*i*) *managed access* to the channel, with transmission timetables exchanged ahead of time; or (*ii*) non-managed *random access* that needs to deal with collisions. Collisions could be prevented by designing strictly timed transmissions (using managed access); avoided by listening before speaking; or, detected after they happened and remedied by retransmitting the corrupted information. An example of preventing collisions by design is the TDMA (Time Division Multiple Access) protocol. It creates a timetable and assigns different time slots to different nodes. A node is allowed to transmit only within its designated time slot. After all nodes are given opportunity to transmit, the cycle is repeated. The problem with this approach is that if some nodes do not have data ready for transmission, their slot goes unused. Any other node that may wish to transmit is delayed and has to wait for its predetermined slot even though the link may currently be idle. This inefficiency is similar to that illustrated in Figure 1-9, which prompted us to introduce statistical multiplexing.

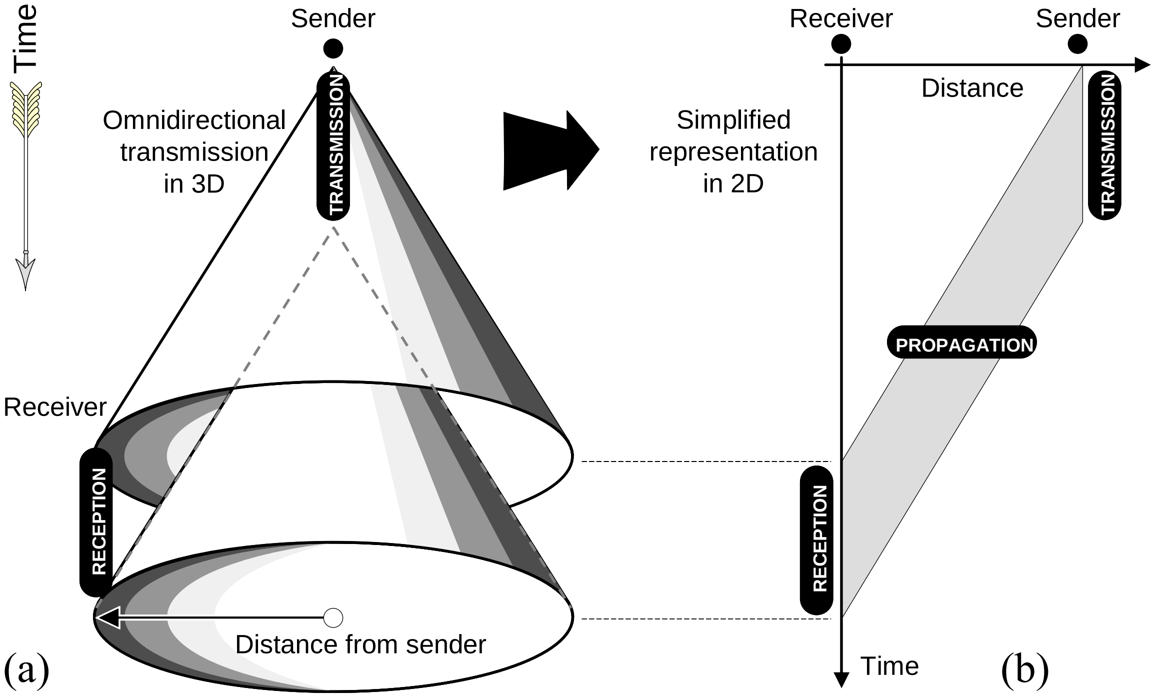


Figure 1-25: (a) Transmission cone formed in space-time as the radio wave propagates from the sender. (b) The transmission cone is usually shown in a two-dimensional projection as a parallelogram similar to transmission on wired links, Figure 1-5.

A popular class of protocols for broadcast links is **random-access protocols**, which are based on ARQ protocols (Section 1.3), with an addition of the backoff delay mechanism. Backoff mechanism is a key mechanism for coordination of multiple senders on a broadcast link. It assumes that a lost packet signifies a collision and inserts a random delay ("back off") before a retransmission is attempted. The **backoff interval** (or "backoff delay") is the amount of time the sender waits to reduce the probability of collision with another sender that also has a packet ready for transmission. A regular protocol over a wire line has no reason to back off when it detects a packet loss, because it assumes that the sender is *not* contending for the link against other senders and any loss is solely due to channel noise. On the other hand, a random-access protocol assumes that any packet loss is due to a collision of concurrent senders and it tries to prevent further collisions by introducing a random backoff before attempting a re-transmission. It is a way to provide stations with "polite behavior." This method is commonly used when multiple concurrent senders are competing for the same resource; another example will be seen at the End-to-end layer of protocol stack, for the TCP protocol (Section 2.1.3).

Wireless links are different from wired point-to-point links, as already discussed in Section 1.1.2. Figure 1-25 illustrates how each bit transmitted by an omnidirectional transmitter forms a **transmission cone** in space-time. This fact is important to keep in mind when we will consider packet collisions.

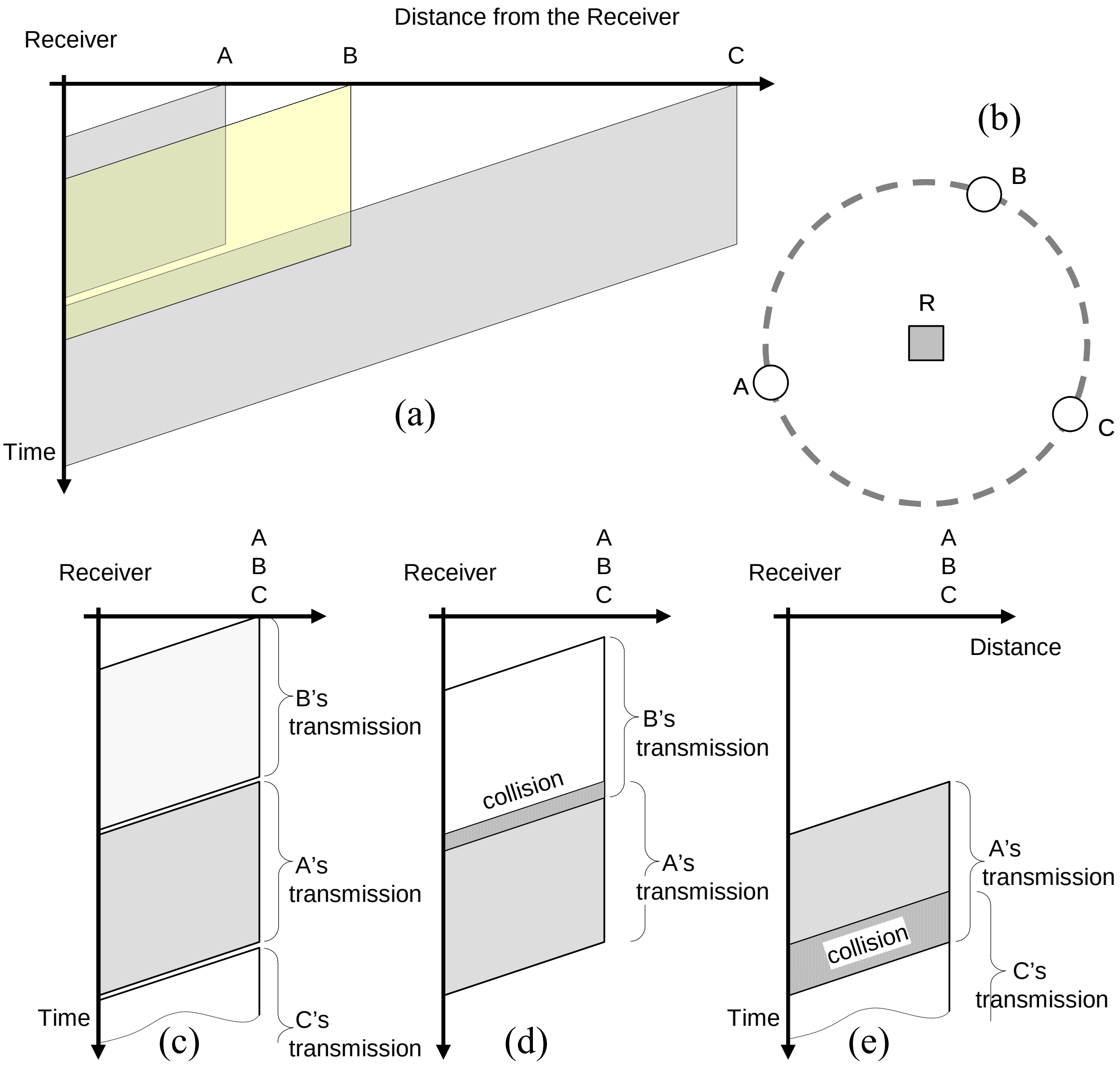


Figure 1-26: Packet collision occurs when transmission cones overlap. (a) Whether transmission cones will overlap depends on the transmission start times and sender distances. (b) Stations in real wireless networks are approximately equidistant. Transmission cones overlap (causing collision) if a station *B* starts transmitting less than one packet time before station *A* (d), or if a station *C* starts transmitting less than one packet time after station *A* (e).

Packet collisions occur when transmission cones of several stations overlap in space-time because the receiver will receive interference signal (Figure 1-26). Figure 1-26(a) illustrates a scenario where sufficiently distant stations *A* and *C* may transmit simultaneously and still avoid collision. However, actual wireless networks are usually designed as mobile stations organized around a central fixed station that relays their messages, known as "base station" or "access point." The mobile stations are approximately equidistant from the central base station, such as very close (in local-area networks) or very distant (in satellite networks). Therefore, a likely real-world scenario is illustrated in Figure 1-26(b). A transmission from station *A* will not suffer collision if no other station starts transmitting within less than one packet time before *A* started nor within less than

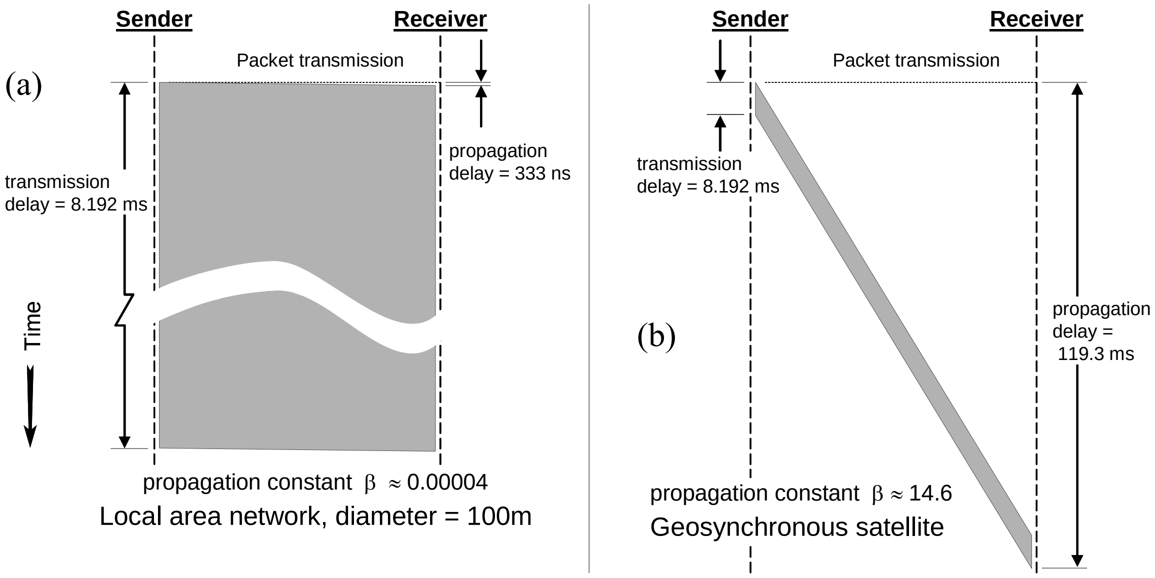

Sender
Receiver
Packet transmission
(a)
propagation delay = 333 ns
transmission delay = 8.192 ms
Time
propagation constant β ≈ 0.00004
Local area network, diameter = 100m
Sender
Receiver
Packet transmission
transmission delay = 8.192 ms
(b)
propagation delay = 119.3 ms
propagation constant β ≈ 14.6
Geosynchronous satellite

Figure 1-27: Propagation constant *β* for different wireless networks.

one packet time after *A* started, as shown in Figure 1-26(c). Otherwise, transmission cones will overlap and a collision will occur, as shown in Figure 1-26(d) and Figure 1-26(e).

A useful parameter for wireless networks is the propagation delay normalized to the packet transmission time. This is the time (in packet transmission units) required for all network nodes to detect a start of a new transmission or an idle channel after a transmission ended. Intuitively, the **parameter *β*** is the number of bits that a transmitting station can place on the medium before a most distant station within the transmission range receives the first bit of the first packet.

Recall that signal propagation time is t_p = distance/velocity, as given earlier by Eq. (1.3). The transmission delay is t_x = packet-length/bandwidth, as given by Eq. (1.2). The parameter β is calculated as

$$\beta = \frac{t_{p\,\max}}{t_x} = \frac{d_{\max} \cdot R}{v \cdot L} \tag{1.11}$$

If we normalize β for the unit packet length, then $\beta = t_{p\,\max}$. The velocity of electromagnetic waves in dry air equals $v \approx 3 \times 10^8$ m/s, and in copper or optical fiber it equals $v \approx 2 \times 10^8$ m/s. Therefore, propagation time is between 3.33 and 5 nanoseconds per meter (ns/m). Given a wireless local area network (W-LAN) where all stations are located within a 100 m diameter, the (maximum) propagation delay is $t_p \approx 333$ ns. If the bandwidth (or, data rate) of the same W-LAN is 1 Mbps, the transmission delay for a 1-Kbyte packet equals $t_x = 8.192$ ms. The relationship is illustrated in Figure 1-27(a). Recall from Figure 1-6 that on a 1 Mbps link, 1 bit is 1 μs wide, so the leading edge of the first bit will reach the receiver long before the sender finished the transmission of this bit. In other words, the propagation delay is practically negligible. On the other hand, the altitude of a geosynchronous satellite is 35,786 km above the Earth surface, so the propagation delay is $t_p \approx 119.3$ ms. As shown in Figure 1-27, the respective β parameters for these networks are $\beta_{\mathrm{LAN}} \approx 0.00004$ and $\beta_{\mathrm{GSS}} \approx 14.6$. The time taken by the electronics for signal detection should also be added to the propagation time when computing β, but it is often ignored as negligible. We will see later how parameter β plays an important role in network design.

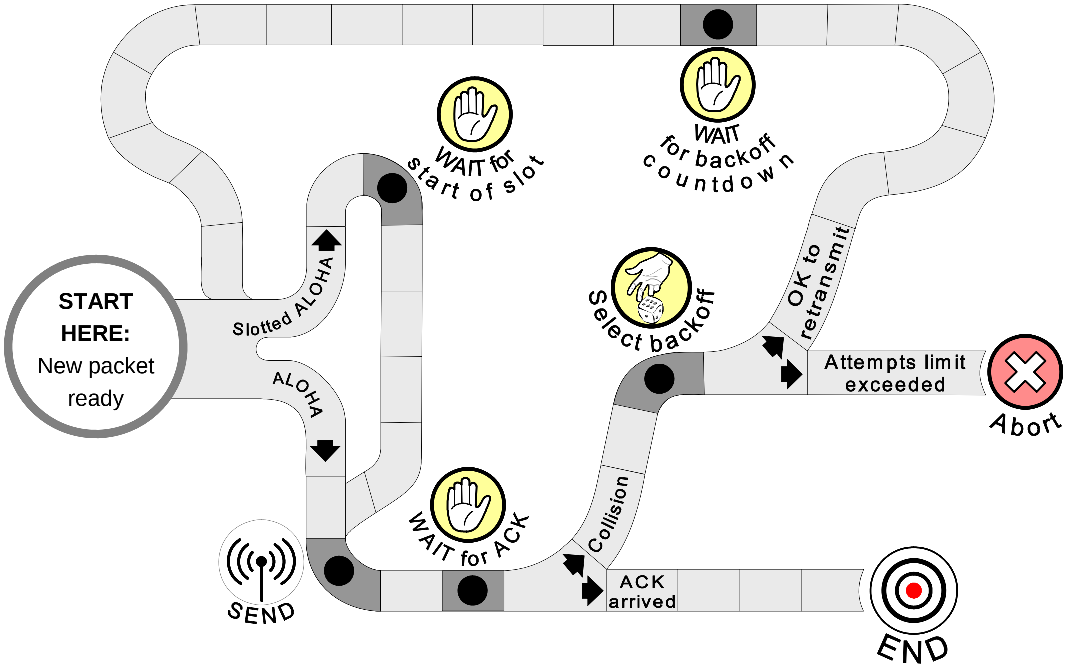


Figure 1-28: The sender's state diagram for ALOHA and Slotted ALOHA protocols.

Medium access control (MAC) protocols often implement reliable transmission by retransmission (an ARQ protocol). The reason for addressing reliability at the link layer is that wireless links are significantly more unreliable than wired links. Noise, interference, collisions, and other propagation effects result in the loss of a significant number of packets. Even with error-correction codes, a number of MAC packets may not successfully be received. This situation can be handled by reliability mechanisms at a higher layer, such as transport-layer protocol (OSI Layer 4). However, timers used for retransmission at higher layers (which control paths comprising many links) are typically on the order of seconds (see TCP timers in Section 2.1.3). To avoid unnecessary delays, it is therefore more efficient to deal with errors at the link level and rapidly retransmit the corrupted packets.

The ALOHA Protocol

Problems related to this section: Problem 1.13 → Problem 1.15

A simple random-access protocol for broadcast media is called ALOHA. There are two versions of ALOHA: *pure* or *plain ALOHA* (or *asynchronous* ALOHA) transmits packets as soon as they become ready, and *slotted ALOHA* (or *synchronous* ALOHA) which is allowed to transmit packets only at regular intervals. The state diagram for the sender side of both variations of the protocol is shown in Figure 1-28. Plain ALOHA sends the packet immediately as it becomes ready, while slotted ALOHA has to wait for the start of the next time interval (or, **slot**). In other words, in slotted ALOHA, all transmissions are strictly clocked at regular time intervals and all stations march forward in lockstep. After transmission, the sender stops-and-waits for the acknowledgment. If the acknowledgment arrives, this is the end of the current cycle, and the sender awaits the next packet for sending (when generated by the protocol's local user). If the acknowledgment does not arrive, the sender assumes that a collision occurred and it instantiates

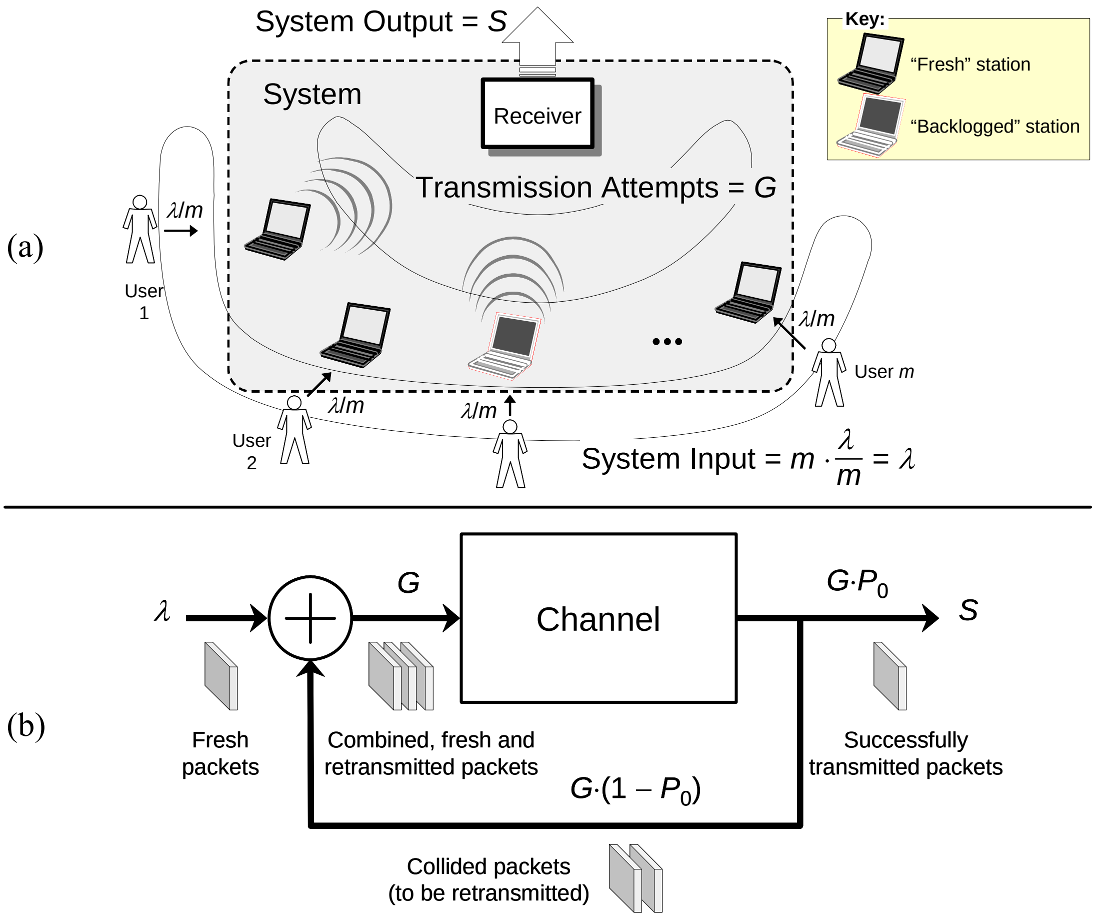

System Output = S
Key:
"Fresh" station
"Backlogged" station
System
Receiver
Transmission Attempts = G
(a)
λ/m
User 1
λ/m
User 2
λ/m
λ/m
User m
System Input = m · λ/m = λ
(b)
λ
G
Channel
G·P0
S
Fresh packets
Combined, fresh and retransmitted packets
G·(1 – P0)
Successfully transmitted packets
Collided packets (to be retransmitted)

Figure 1-29: (a) ALOHA system representation. (b) Modeled as a feedback system.

its backoff interval. The backoff interval is selected differently in pure versus slotted ALOHA. Pure ALOHA selects a random time duration, whereas slotted ALOHA selects a random number of slots. After waiting for backoff countdown to zero, the sender repeats the cycle and retransmits the packet. As we know from earlier discussion, the sender does not persist forever in resending the packet. If the number of attempts exceeds a given threshold, the sender gives up and aborts the retransmissions of this packet.

ALOHA is a very simple protocol, almost identical to Stop-and-Wait ARQ (Section 1.3.1), except that it also has the backoff mechanism. ALOHA similarly does not initiate transmission of the next packet before ensuring that the current packet is correctly received. To derive the throughput performance of ALOHA, we make the following assumptions (Figure 1-29(a)):

1) There are *m* wireless nodes (stations), each generating new packets for transmission according to a Poisson process (see Appendix) with the rate λ/m packets per slot (defined next).

2) All packets have the same length. The time needed for packet transmission (transmission delay t_x) is called *slot length*, and it is normalized to equal 1. The unit of time is one slot.

3) When a new packet is generated on a node, it is immediately transmitted (in pure ALOHA) and stored until an acknowledgment is received. While storing the packet, the node is said to

be *backlogged*. Because of Stop-and-Wait ARQ, a backlogged node must first is successfully transmit the current packet before working on the new packets generated after this one.

4) If only a single node transmits, the transmission is always successful (noiseless channel); if two or more nodes transmit simultaneously, there will be a collision and all collided packets will be lost.

5) All nodes receive *instantaneous feedback* about the success or failure of the transmission. In other words, an acknowledgment is received *immediately* upon packet transmission, without any propagation delays. This approximation is reasonable in a local area network with acknowledgment packets much smaller than data packets.

6) New packets will not experience delay waiting for transmission of previous packets on the same node (i.e., a new packet will be the first in line for transmission on its node). This assumption can be achieved in two ways:

   a) Either: each node is limited to hold a single packet at a time. If the node is backlogged, the newly generated packet is discarded. Although it is wasteful to discard newly generated packets, we expect that this would happen very rarely because for a large m and a small λ, only a small fraction of nodes will be backlogged and very few backlogged nodes will generate new packets. Therefore, very few packets would be discarded.

   b) Or: there is an infinite number m of nodes, which again with a relatively small rate λ implies that only a small fraction of nodes will be backlogged and new packets will almost never encounter a backlogged node.

Although these assumptions are not entirely realistic, they are a useful approximation of a common scenario. Under these assumptions, the ALOHA system can be modeled as in Figure 1-29(b). In addition to the new packets, the backlogged nodes generate retransmissions of the packets that previously suffered collisions. If the retransmissions are sufficiently randomized, it is plausible to approximate the total number of transmission attempts per slot (retransmissions and new transmissions combined) as a Poisson random variable with some parameter $G > \lambda$, where G is the total arrival rate on the channel (new and backlogged packets, combined). As shown in Figure 1-29(b), a fraction P_0 of these packets will be successfully transmitted and leave the system. This is the throughput S of the system:

$$\text{throughput} = \text{arrival rate} \times \text{probability of no collision}$$

The remaining fraction of $G\cdot(1-P_0)$ packets will suffer collision and will feed back for retransmission. Now we can explain why some of the above assumptions are necessary. If the number of stations m were small and the arrival rate λ of new packets sufficiently large, then a considerable number of stations would become backlogged and the arrival rate of new packets on the channel would dynamically diminish. The reason is that new packets that are delayed waiting for backlogged packets to be transmitted would not count as competing for the channel. The simple model in Figure 1-29(b) would not be valid any more.

The quantity P_0 is the probability that there will be no overlapping transmissions during a packet's transmission. From the Poisson distribution formula (see Appendix A), $P_0 = P\{\mathrm{A}(t + \tau) - A(t) = 0\} = e^{-G\tau}$. For slotted ALOHA, there will be no overlapping transmissions if no other station transmits during the same slot. Therefore, the interval susceptible to collision is one slot and $\tau = 1$ time unit, which is one slot. We have:

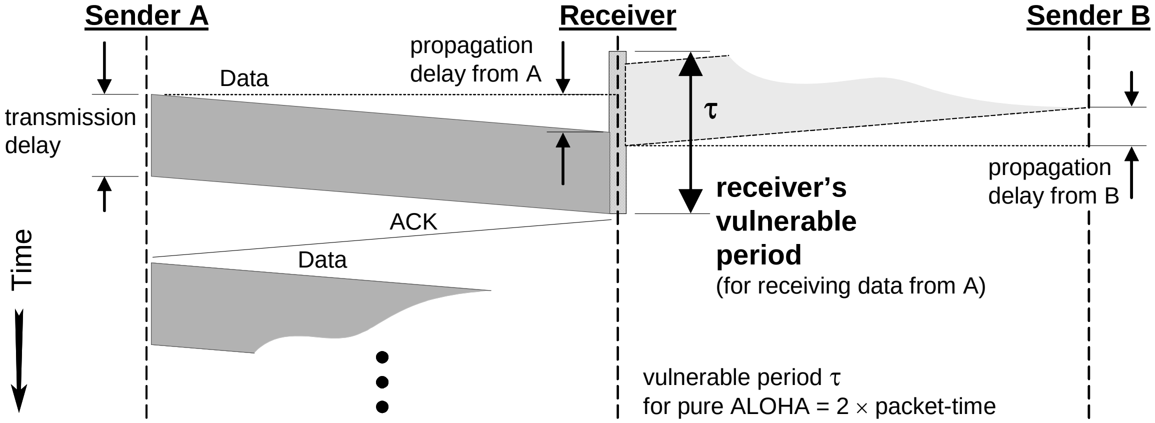


Figure 1-30: The receiver's *vulnerable period* τ is the time interval during which transmission cones from *A* and *B* can overlap (resulting in a collision). Also see Figure 1-26.

Slotted ALOHA throughput: $S = G \cdot P_0 = G \cdot P\{A(t+1) - A(t) = 0\} = G \cdot e^{-G}$ (1.12a)

For pure ALOHA, the interval susceptible to collision is different. We define receiver's **vulnerable period** (or "window of vulnerability") as the time during which transmission cones of other packets may overlap with the packet under consideration. For pure ALOHA, the vulnerable period is two slots long, $\tau = [t - 1, t + 1)$, where t is the start of reception of packet from A, as illustrated in Figure 1-30. Any transmission that started within less than one packet-time before this transmission or during this transmission will overlap with this transmission and result in a collision. Therefore, $\tau_{\text{pure ALOHA}} = 2$, and:

Pure ALOHA throughput: $S = G \cdot P_0 = G \cdot P\{A(t+\tau) - A(t) = 0\} = G \cdot e^{-2G}$ (1.12b)

To achieve a reasonable throughput, we would expect $0 < \lambda < 1$ on average, because the channel can successfully carry at most one packet per slot, i.e., only one node can successfully "talk" (or, transmit) at a time without collision. Also, for the system to function, in equilibrium the departure rate S of packets leaving the system should equal the arrival rate λ. In equilibrium, on one hand, the departure rate cannot physically be greater than the arrival rate. On the other hand, if it is smaller than the arrival rate, all the nodes will eventually become backlogged and no new packets will be admitted. In equilibrium, the arrival rate (system input), λ, should be the same as the departure rate (system output), $S = G{\cdot}e^{-G\tau}$. This relationship is illustrated in Figure 1-31.

We see that for slotted ALOHA, the maximum possible throughput of $1/e \approx 0.368$ occurs at $G = 1$. This result is reasonable, because if $G < 1$, too many slots are left idle, and if $G > 1$, too many collisions are generated. At $G = 1$, the arrival rate on channel (retransmissions and new transmissions combined) is one packet per packet time (or, per slot). Of the transmitted packets, the fraction $1/e$ will successfully reach the receiver, and the remaining $1 - \frac{1}{e}$ will be garbled in collision (in effect, waste).

Slotted ALOHA performs better than pure ALOHA because slotted ALOHA has twice shorter vulnerable period. The disadvantage is that slotted ALOHA is more complex to implement—synchronizing the transmitters for slotted arrival at the receiver is not trivial, but can be

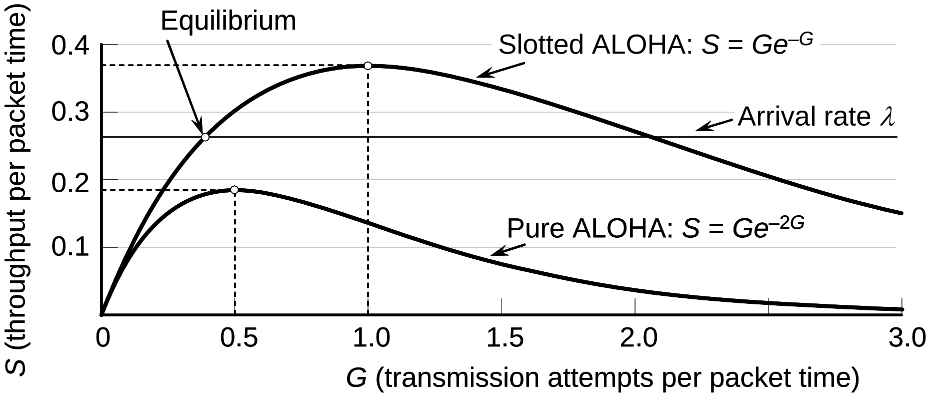


Figure 1-31: Efficiency of the ALOHA MAC protocol. (In the case of Slotted ALOHA, the packet time is equal to the *slot time*.)

accomplished with relatively stable clocks, a small amount feedback from the receiver, and some guard time between the end of a packet transmission and the beginning of the next slot.

Carrier Sense Multiple Access Protocols (CSMA)

Problems related to this section: Problem 1.17 → Problem 1.18

It is both advantage and weakness of the ALOHA protocol that it employs a very simple strategy for coordinating the transmissions: a node transmits a new packet as soon as it is created, and in case of collision, it waits for a random backoff time and retransmits.

An improved coordination strategy is to have the nodes "listen before they talk." That is, the sender listens to the channel before transmitting and transmits only if the channel is detected as idle. Listening to the channel is known as **carrier sense**, which is why this coordination strategy has the name *carrier sense multiple access* (CSMA).

The medium is decided *idle* if there are no transmissions for time duration the parameter β time units, because this is the propagation delay between the most distant stations in the network. The time taken by the electronics for signal detection should also be added to the propagation time when computing channel-sensing time, but it is usually ignored as negligible.

The key issues with a listen-before-talk approach are:

(1) When to listen and, in case the channel is found busy, whether to persist listening until it becomes idle or stop listening and try later

(2) Whether to transmit immediately upon finding the channel idle or randomly delay the transmission (because other nodes might have been waiting for idle channel, too)

Upon finding the channel busy because another node is transmitting, the node might persist listening until the ongoing transmission ends. Another option is to listen periodically (and do something else in between). Once the channel becomes idle, the node might transmit immediately, but there is a danger that some other nodes also waited ready for transmission, which would lead to a collision. Another option is, once the channel becomes idle, to defer the transmission briefly for a random amount of time, and only if the channel remains idle, start transmitting the packet. This reduces the chance of a collision significantly, although it does not

Table 1-1: Characteristics of three basic CSMA protocols when the channel is sensed idle or busy. If a transmission was unsuccessful, all three protocols perform backoff and repeat.

CSMA Protocol	Sender's listening-and-transmission rules
Nonpersistent	If medium is idle, transmit. If medium is busy, wait random amount of time and sense channel again.
1-persistent	If medium is idle, transmit (i.e., transmit with probability 1). If medium is busy, *continue sensing* until channel is idle; then transmit immediately (i.e., transmit with probability 1).
p-persistent	If medium is idle, transmit with probability p. If medium is busy, *continue sensing* until channel is idle; then transmit with probability p.

remove it, because both nodes might by chance select the same random interval to delay their transmissions. Several CSMA protocols that make different choices regarding the listening and transmission start are shown in Table 1-1. For all protocols in Table 1-1, when the sender discovers that a transmission was unsuccessful (by a retransmission timer timeout), the sender behaves the same way: it inserts a randomly distributed retransmission delay ("backoff interval") and repeats the listen-and-transmit procedure until the packet is acknowledged.

The efficiency of CSMA is better than that of ALOHA because of CSMA's shorter vulnerable period: The stations will not initiate transmission if they "hear" a transmission already in progress. To determine how long it takes a station to "hear" a transmission in progress (i.e., the length of the *vulnerable period for CSMA-based stations*), we observe it cannot be only the propagation delay. In wireless LANs, the propagation delay is negligible (Figure 1-27(a)). When a sender starts transmitting, the front wave will reach all other nodes in the W-LAN in no time, but that does not mean that they will "hear" it. The vulnerable period must also include the minimum time required for any other node to decide reliably that there is an ongoing transmission and that it should defer its own transmission. That is why there must be a minimum packet size, which corresponds to the time needed for busy-channel detection. Therefore, in CSMA the vulnerable period is defined as the normalized propagation delay (parameter β), where the normalization is performed over the *minimum* packet size (instead of any packet size).

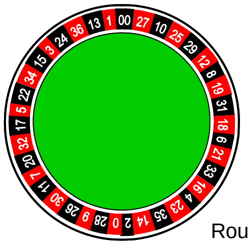
Note that nonpersistent CSMA is less greedy than 1-persistent CSMA in the sense that, upon observing a busy channel, it does not continually sense it with intention of seizing it immediately upon detecting the end of the previous transmission (Table 1-1). Instead, nonpersistent CSMA waits for a random period and then repeats the procedure. Consequently, this protocol leads to better channel utilization but longer average delays than 1-persistent CSMA.

SIDEBAR 1.2: Binary Exponential Backoff

♦ CSMA protocols introduce a variation on the backoff mechanism, known as binary exponential backoff. The sender using **binary exponential backoff** doubles the range of backoff delays for every failed transmission. The backoff range is also known as **contention window size**, which is the number of choices available for random backoff. For example, if a station rolled a dice, then the contention window size would be six. After *k* collisions, the backoff range is [0, $2^k - 1$] backoff slot times. Increasing the backoff range increases the number of choices for the random delay. This, in turn, makes it less likely that several stations will select the same delay value and, therefore, reduces the probability of repeated collisions. It is like using different random devices to decide how long to wait. So, the nodes that suffered a collision first select their backoff delay by tossing a coin (two choices: heads or tails); if multiple nodes make the same choice and again experience a collision, then they try by rolling a dice (six choices), etc. (These devices do not provide a binary exponential sequence of ranges, but they are used just for illustration.)

Coin
two choices
{0, 1}

Dice
six choices
{1, 2, 3, 4, 5, 6}

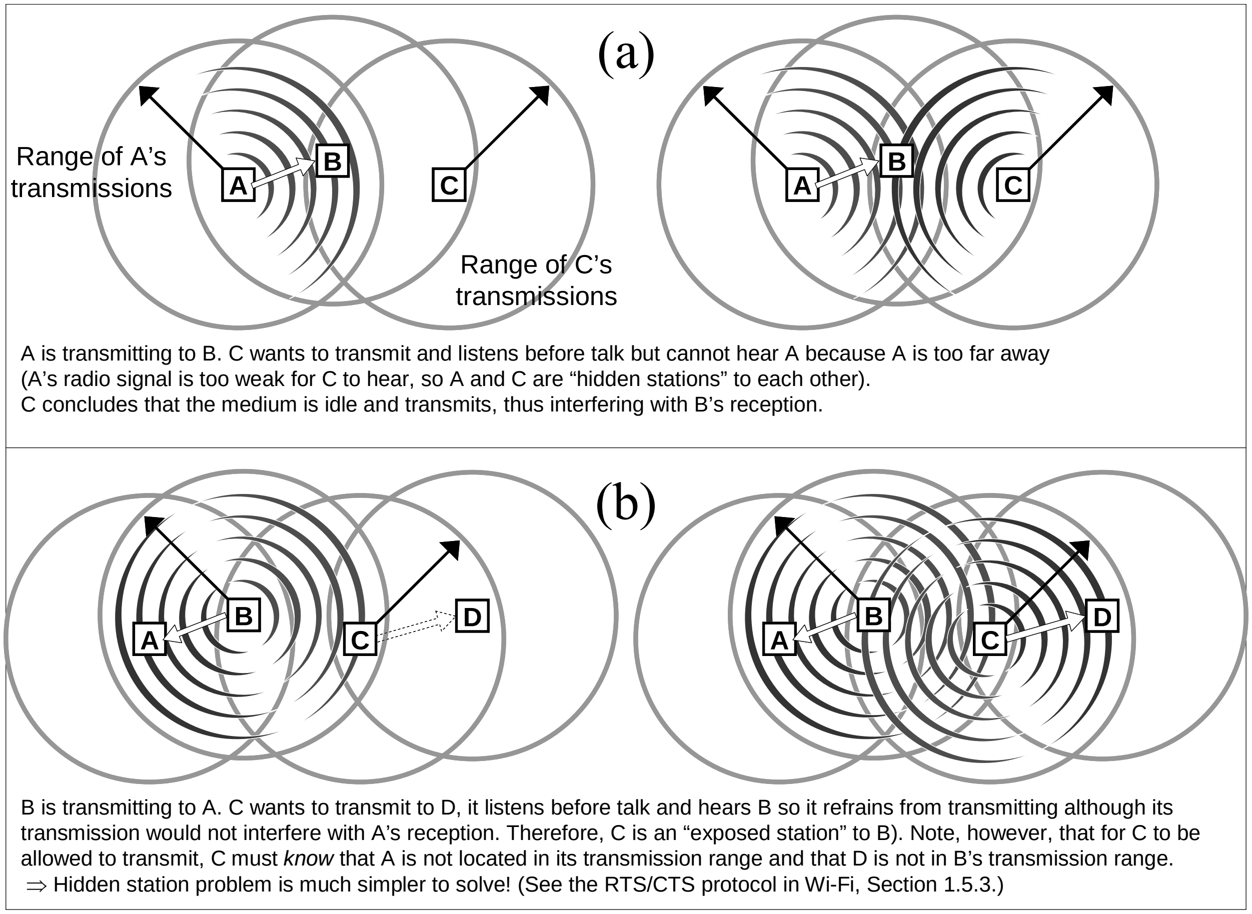


Roulette
38 choices (American)
{0, 00, 1, 2, 3, …, 36}

The backoff range is usually ***truncated***, which means that after a certain number of increases, the backoff range reaches a ceiling and the exponential growth stops. For example, if the ceiling is set at *k*=10, then the maximum delay is 1023 backoff slot times. In addition, the number of attempted retransmissions is limited, so that after the maximum allowed number of retransmissions the sender gives up and aborts the retransmission of this packet. The sender resets its backoff parameters and retransmission counters at the end of a successful transmission or if the transmission is aborted.

Note that ALOHA implements a backoff mechanism, but *not* an exponentially growing backoff—ALOHA backoff range remains constant. Exponential backoff is particularly useful when there are many stations and at any time an unknown number may transmit. When there are many potential senders, our backoff range naturally must be large to avoid having multiple stations select the same random delay. However, the stations cannot know how many *potential* senders are *actually* sending. A node that suffered collision does not want invariably to select from a large range and have to wait for too long before attempting retransmission, even if currently there were only few competing senders. On the other hand, it cannot know how many other nodes are currently competing for transmission, so it goes through a search process. Each collided node starts with a small range and, if unsuccessful, the nodes keep growing the range until they find a minimum range that will allow collision-free transmission. TCP protocol (Section 2.1.3) also implements binary exponential backoff for a similar reason.

Wireless broadcast networks show some phenomena not present in wireline broadcast networks. The air medium is partitioned into broadcast regions, rather than being a single broadcast medium. This partitioning is simply due to the exponential propagation decay of radio signal, as discussed earlier in Section 1.1.2. Broadcast region for a transmitter is delimited by the perimeter



A is transmitting to B. C wants to transmit and listens before talk but cannot hear A because A is too far away (A's radio signal is too weak for C to hear, so A and C are "hidden stations" to each other).
C concludes that the medium is idle and transmits, thus interfering with B's reception.

B is transmitting to A. C wants to transmit to D, it listens before talk and hears B so it refrains from transmitting although its transmission would not interfere with A's reception. Therefore, C is an "exposed station" to B). Note, however, that for C to be allowed to transmit, C must *know* that A is not located in its transmission range and that D is not in B's transmission range.
⇒ Hidden station problem is much simpler to solve! (See the RTS/CTS protocol in Wi-Fi, Section 1.5.3.)

Figure 1-32: (a) Hidden station problem form CSMA protocols: C cannot hear A's transmissions. (b) Exposed station problem: C defers transmission to D because it hears B's transmission.

within which its signal can be heard and decoded. As a result, two interesting phenomena arise: (*i*) not all stations within a partition can necessarily hear each other; and, (*ii*) the broadcast regions can overlap. The former causes the hidden station problem and the latter causes the exposed station problem, as described next.

Unlike the wireline broadcast medium, the transitivity of connectivity does not apply. In wireline broadcast networks, such as Ethernet, if station *A* can hear station *B* and station *B* can hear station *C*, then station *A* can hear station *C*. This is not always the case in wireless broadcast networks, as seen in Figure 1-32(a). In the **hidden station problem**, station *C* cannot hear station *A*'s transmissions and may mistakenly conclude that the medium is available. If *C* does start transmitting, it will interfere at *B*, garbling the packet from *A*. Generally, a station *X* is considered to be hidden from another station *Y* in the same receiver's area of coverage if the transmission coverages of the transceivers at *X* and *Y* do not overlap. A station that can sense the transmission from both the source and receiver nodes is called **covered station**.

Different air partitions can support multiple simultaneous transmissions, which are successful as long as each receiver hears at most one transmitter at a time. In the **exposed station problem** (Figure 1-32(b)), station *C* defers its transmission to *D* because it hears *B*'s transmission. If *C* senses the medium, it will hear an ongoing transmission (by *B*) and falsely conclude that it may not send to *D*, when in fact such a transmission would cause bad reception only in the zone between *B* and *C*, where neither of the intended receivers is located. Thus, the carrier sense mechanism is insufficient to detect all transmissions on the wireless medium. The exposed station problem is much more difficult to solve than the hidden station problem because, among others, the exposed station needs to know the locations of other stations, which is not trivial to find.

Hidden and exposed station problems arise only for CSMA-type protocols. ALOHA, for instance, does not suffer from such problems because it does not sense the channel before transmission (i.e., it does not listen before talking). Under the hidden stations scenario, the performance of CSMA degenerates to that of ALOHA, because the carrier-sensing mechanism ("listen-before-talk") essentially becomes useless. With exposed stations it is even worse because carrier sensing prevents the exposed stations from transmitting, where ALOHA would not mind the busy channel. Section 1.5.3 describes how the RTS/CTS protocol mitigates the hidden station problem.

CSMA/CD

Problems related to this section: Problem 1.19 → ?

Persistent and nonpersistent CSMA protocols are clearly an improvement over ALOHA because they ensure that no station begins to transmit if it senses the channel busy. Another improvement is for stations to abort their transmissions as soon as they detect a collision[4]. In both ALOHA and regular CSMA, collisions involve entire packet transmissions. Unlike this, quickly terminating damaged packets saves time and bandwidth. This modified CSMA protocol is known as *CSMA with Collision Detection*, or CSMA/CD, which is a variant of 1-persistent CSMA. A CSMA/CD sender transmits each packet as follows (Figure 1-33):

1. Wait persistently until the channel is idle.
2. When the channel is idle, transmit immediately *and* sense the carrier during the transmission (or, "listen *while* talking"—unlike listening *before* talking).
3. If you detect collision, abort the ongoing transmission, double the backoff range, choose a random amount of backoff delay, wait for this amount of delay, and go to step 1.

A given station can experience a collision during the initial part of its transmission (the **collision window** or vulnerable period, as defined earlier) until its transmitted signal has had time to propagate to all stations on the communication medium. Once the collision window has passed, a transmitting station is said to have *acquired the medium*; subsequent collisions are avoided because all other stations can be assumed to have detected the signal and are deferring to it. The time to acquire the medium is thus based on the round-trip propagation time. If the station transmits the complete packet successfully (without detecting a collision) and has additional data to transmit, it will again listen to the channel before attempting a transmission (Figure 1-33).

[4] Collisions can be detected because they are generally higher in signal amplitude than normal signals. In networks with wired media, the station compares the signal that it outputs on the wire with the one observed on the wire to detect collision. If these signals are not the same, a collision has occurred. In wireless networks, a protocol called CSMA/CN (collision notification) has been developed to approximate CSMA/CD [Sen, et al., 2010].

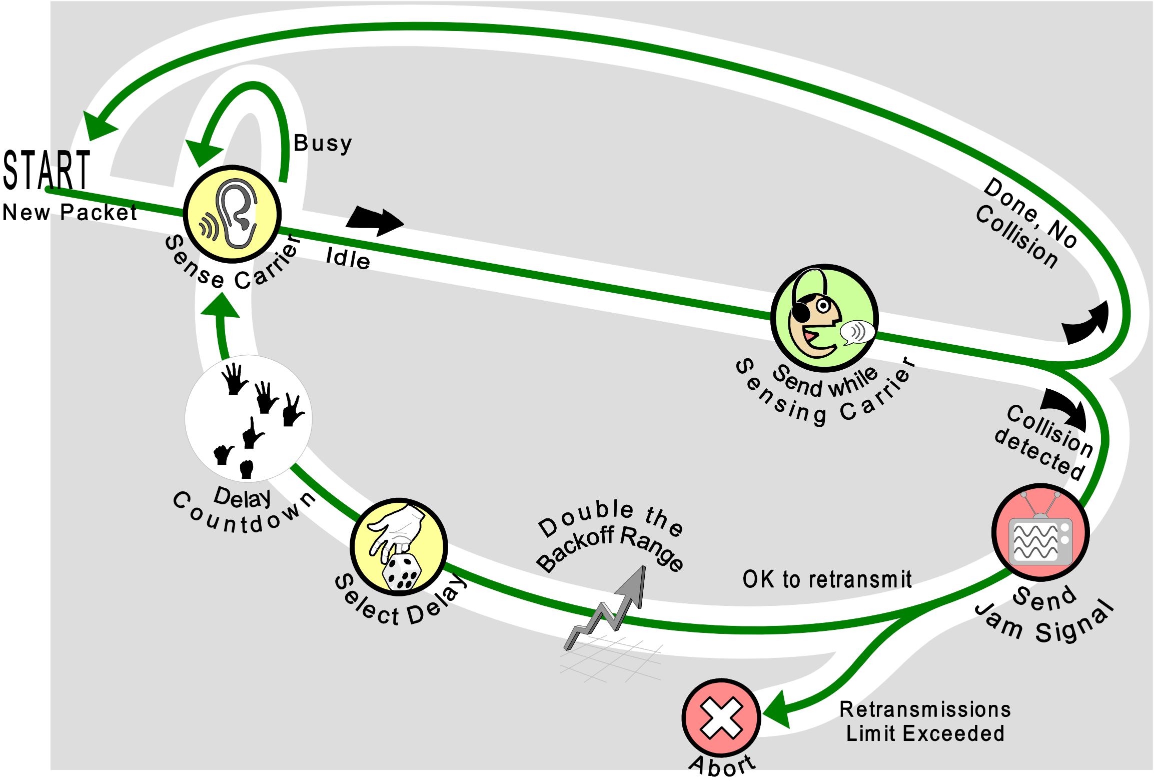


Figure 1-33: The sender's state diagram for CSMA/CD protocol.

The collision detection process is illustrated in Figure 1-34. At time t_1 both stations are listening ready to transmit. To maintain one collision domain, the CSMA/CD protocol requires that the transmitter detect the collision before it has finished transmitting its packet. Otherwise, the transmitter may assume that the packet reached the receiver successfully, without collision. For this reason, CSMA/CD defines the ***smallest valid packet size***. The transmission time of the smallest packet must be longer than one round-trip propagation time, i.e., 2β, where β is the propagation constant defined by Eq. (1.11). When CSMA/CD was invented in the 1970s, it was decided that the network range should be about 2500 meters. Given the link data rate of 10 Mbps, the *smallest valid packet* must be at least 64-bytes long, because we can transmit about 500 bits (rounded to $512=2^9$) within the round-trip propagation time on a copper wire = $2\times t_p = 2\times2500$ m / (2×10^8 m/s) = 50 μs at the data rate of 10 Mbps. This time 2β also represents one **contention slot time**[5] (or **backoff slot time**) that is used for backoff delay countdown (Figure 1-33).

Any station that detects collision continues transmitting a **jam signal**, which carries a special binary pattern to inform the other stations that a collision occurred. The jamming pattern is 32 to 48 bits long and ensures that the collision lasts long enough to be detected by all stations on the network. If STA 2 in Figure 1-34 happened to detect collision on the first bit of transmission, and just stopped transmitting the moment it detected this collision, then it might not have stayed on the medium long enough for STA 1 also to detect the collision during its transmission. (The

[5] Keep in mind that CSMA contention slots are different from ALOHA slots. Each slot in slotted ALOHA corresponds to one packet transmission time. Unlike this, a CSMA contention slot is the time a station listens to the channel before transmission, to detect if the channle is idle or busy.

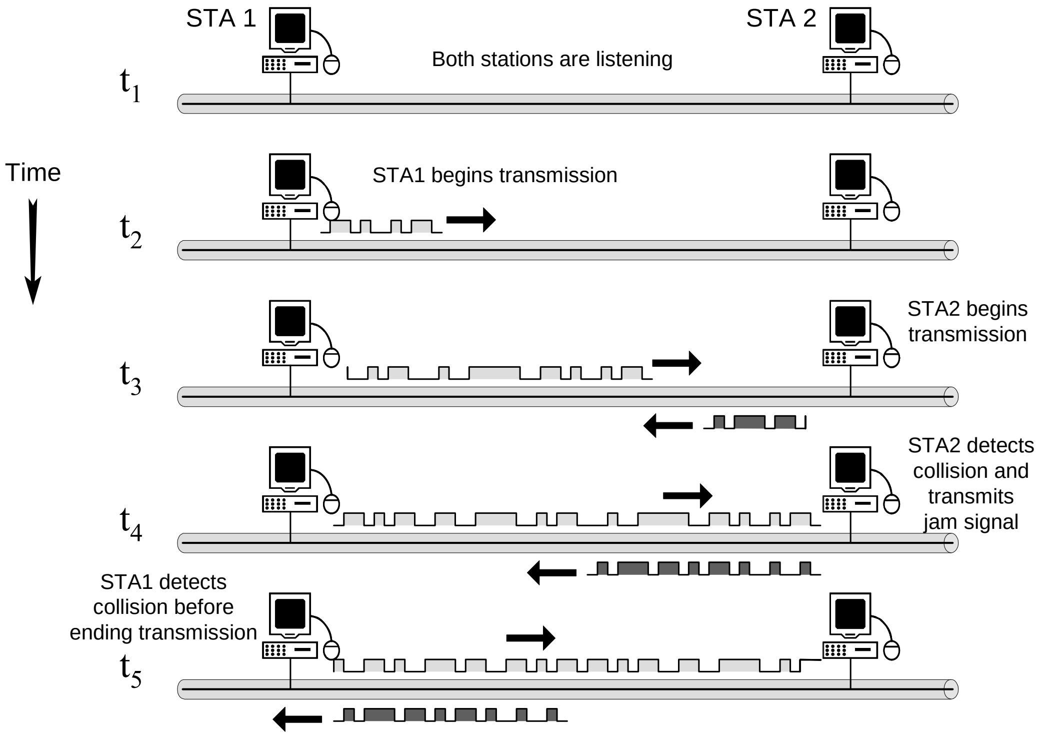

STA 1
STA 2
Both stations are listening
Time
t_1
STA1 begins transmission
t_2
STA2 begins transmission
t_3
STA2 detects collision and transmits jam signal
t_4
STA1 detects collision before ending transmission
t_5

Figure 1-34: Collision detection by CSMA/CD stations.

receiver is not guaranteed to detect the collision on the very first bit of the collision.) Therefore, rather than just stop transmitting, STA 2 transmits the jam signal so that it stays on the medium long enough to ensure that STA 1 realizes that a collision occurred.

It is important to realize that collision detection is an analog process. The station's hardware must listen to the cable while it is transmitting. If the signal it reads back is different from the signal it is putting out, it knows that a collision is occurring. The station checks for collision after each transmitted bit, although several successive bits may be needed to detect an ongoing collision. Recall from Sidebar 1.2 that after *k* collisions, a random number of slot times is chosen from the backoff range [0, 2^k – 1]. After the first collision, each sender might wait 0 or 1 slot times. After the second collision, the senders might wait 0, 1, 2, or 3 slot times, and so forth. In addition, as indicated in Figure 1-33, the number of attempted retransmissions is limited ("truncated binary exponential backoff"), so that after the maximum allowed number of retransmissions the sender aborts the retransmission of this packet. The CSMA/CD protocol usually limits the number of retransmission attempts to 15 times (16 if counting the original transmission). The sender resets its parameters after a successful transmission or an aborted transmission.

Note that CSMA/CD achieves semi-reliable transmission without acknowledgments as in regular ARQ protocols. If the sender does not detect collision, it simply assumes that the receiver received the packet error-free, and there is no need for an acknowledgment. If the sender detects collision during a transmission, it aborts the transmission and retransmits the packet after a backoff delay. The receiver, however, cannot sense collisions because only the sender can compare its output signal with than on the wire, to detect a collision. The receiver detects collisions by receiving the jam signal. In addition, each packet carries a *checksum* so that the receiver can detect errors due to channel noise. The receiver silently drops a corrupted packet,

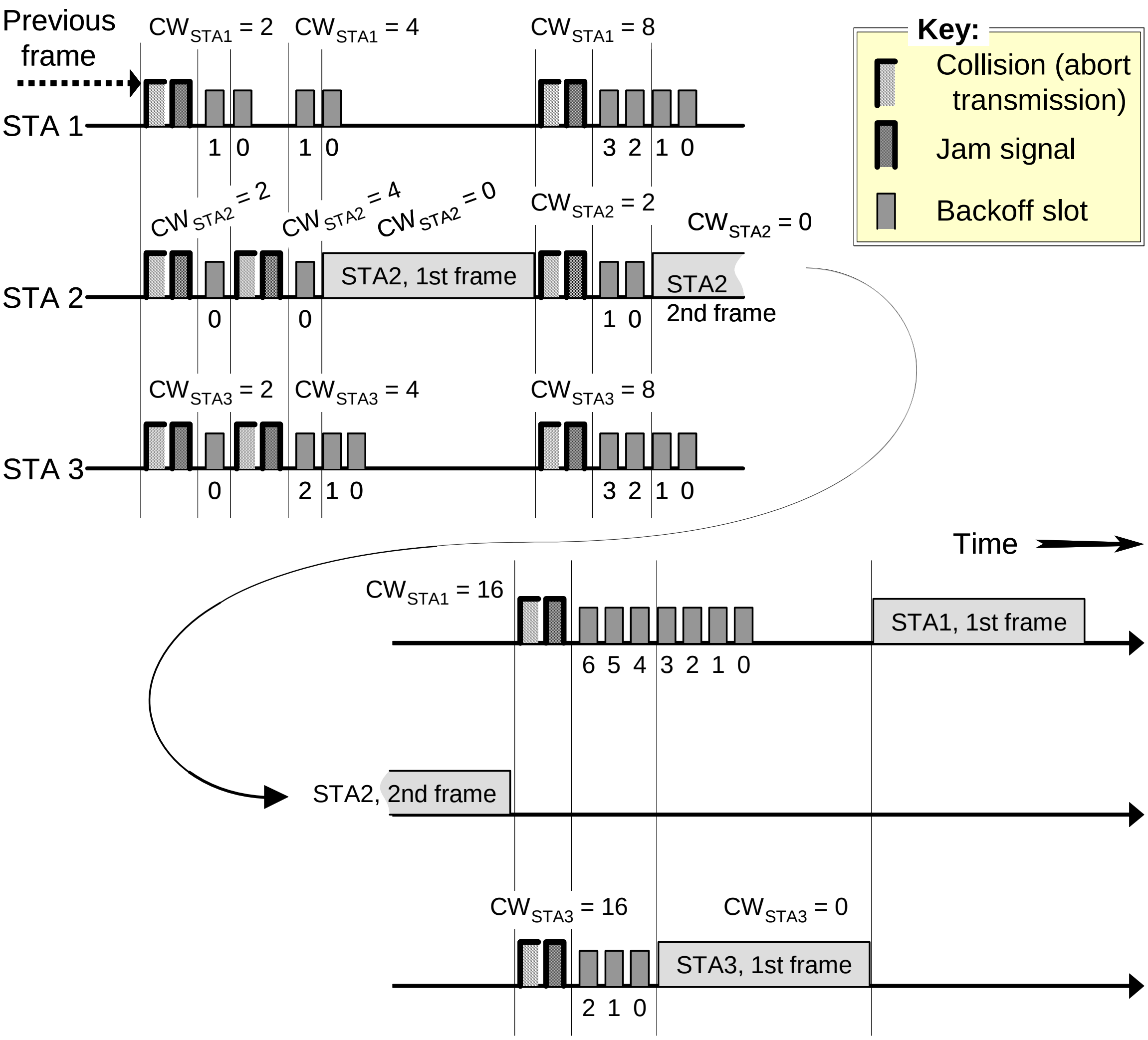


Figure 1-35: Example of three CSMA/CD stations transmission with collision and backoff.

and the CSMA/CD sender, not knowing about the loss, will not retransmit. If the packet is important, the sending application should detect the loss and retransmit.

Here is an example:

Example 1.1 Illustration of a Timing Diagram for CSMA/CD

Consider a local area network of three stations using the CSMA/CD protocol shown in Figure 1-33. At the end of a previous transmission, station-1 and station-3 each have one packet to transmit, while station-2 has two packets. Assume that all packets are of the same length. After the first collision assume that the randomly selected backoff values are: STA1 = 1; STA2 = 0; STA3=0. Next, after the second collision, the backoff values are: STA1 = 1; STA2 = 0; STA3=2. Then, after the third collision, the backoff values are: STA1 = 3; STA2 = 1; STA3=3. Finally, after the fourth collision the backoff values are: STA1 = 6; (STA2 is done by now); STA3=2. Draw the channel timing (not protocol timing) diagram and indicate the contention window (CW) sizes.

The solution is shown in Figure 1-35. Initially, all three stations attempt transmission and there is a collision. They all detect the collision, abort their transmissions in progress, and send the jam signal. After this, all three stations set their contention window (*CW*) size to 2 and randomly choose their delay periods from the set {0, …, *CW*} = {0, 1}. As given in the problem statement, station-1 chooses its backoff delay as 1, while stations 2 and 3 booth choose their backoff delay as 0. This leads to the second collision. After the second backoff delay, station-2 succeeds in transmitting its first packet and resets its backoff parameters (including the contention window *CW*) to their default values. The other two stations keep the larger ranges of the contention window because they have not successfully transmitted their packets yet. This gives station-2 an advantage after the third collision. Because it chooses the backoff delay from a shorter range of values (*CW*=2), it is more likely to select a small value and, therefore, again succeed in transmitting another packet, as shown in Figure 1-35. The effect is known as "Ethernet Capture" and shows that the medium sharing is not necessarily fair.

To derive the performance of the CSMA/CD protocol, we will assume a network of *m* stations with heavy and constant load, where all stations are always ready to transmit. We make a simplifying assumption that there is a constant retransmission probability in each contention slot (recall from footnote 5 on page 51 that CSMA slots are different from ALOHA slots). Suppose that each station transmits during a contention slot with probability *q*. The probability *A* that any one station will acquire the channel and successfully transmit in that slot is the probability that only one station will transmit:

$$p_{\text{succ}} = \binom{m}{1} \cdot (\text{one transmits, others don't}) = \quad m \cdot q \cdot (1-q)^{m-1}$$

To find the maximum achievable throughput on a CSMA/CD channel, we assume that the stations use the value of *q* that maximizes p_{succ}. By taking a derivative of p_{succ} with respect to *q* and setting it to zero, we find that p_{succ} is maximized when $q = 1/m$, with $p_{\text{succ}} \to 1/e$ as $m \to \infty$. Next, we calculate the average number of contention slots that a station wastes before it succeeds in transmitting a packet. The probability that a station will suffer collision $(j - 1)$ times and succeed on the j^{th} attempt (i.e., that the contention interval is exactly *j* slots long) is $p_j = p_{\text{succ}} \cdot (1 - p_{\text{succ}})^{j-1}$. The average number of slots per contention is the expected value:

$$\sum_{j=0}^{\infty} j \cdot p_j = \sum_{j=0}^{\infty} j \cdot p_{\text{succ}} \cdot (1 - p_{\text{succ}})^{j-1} = \frac{1}{p_{\text{succ}}}$$

Because each slot is $2 \cdot \beta$ long, the mean contention interval is $w = 2 \cdot \beta / p_{\text{succ}}$. Assuming the optimal *q*, the average number of contention slots is never more than *e*, so *w* is at most $2 \cdot \beta \cdot e \approx 5.4 \times \beta$. If an average packet takes $t_x = L/R$ seconds to transmit, then the channel efficiency is:

$$\eta_{\text{CSMA/CD}} = \frac{t_x}{t_x + w} = \frac{L/R}{L/R + 2 \cdot \beta / p_{succ}} = \frac{1}{1 + 2 \cdot \beta \cdot e \cdot R / L} \tag{1.13}$$

where p_{succ} is substituted with its optimal value $1/e$. The efficiency expression shows that CSMA/CD can achieve throughputs that are close to 1 when β is much smaller than 1.

We will see in Section 1.5.2 how IEEE 802.3 LAN, known as Ethernet, uses CSMA/CD.

CSMA/CA

Problems related to this section: Problem 1.20 → Problem 1.23

In wireless LANs, it is not practical to detect collisions because of two main reasons:

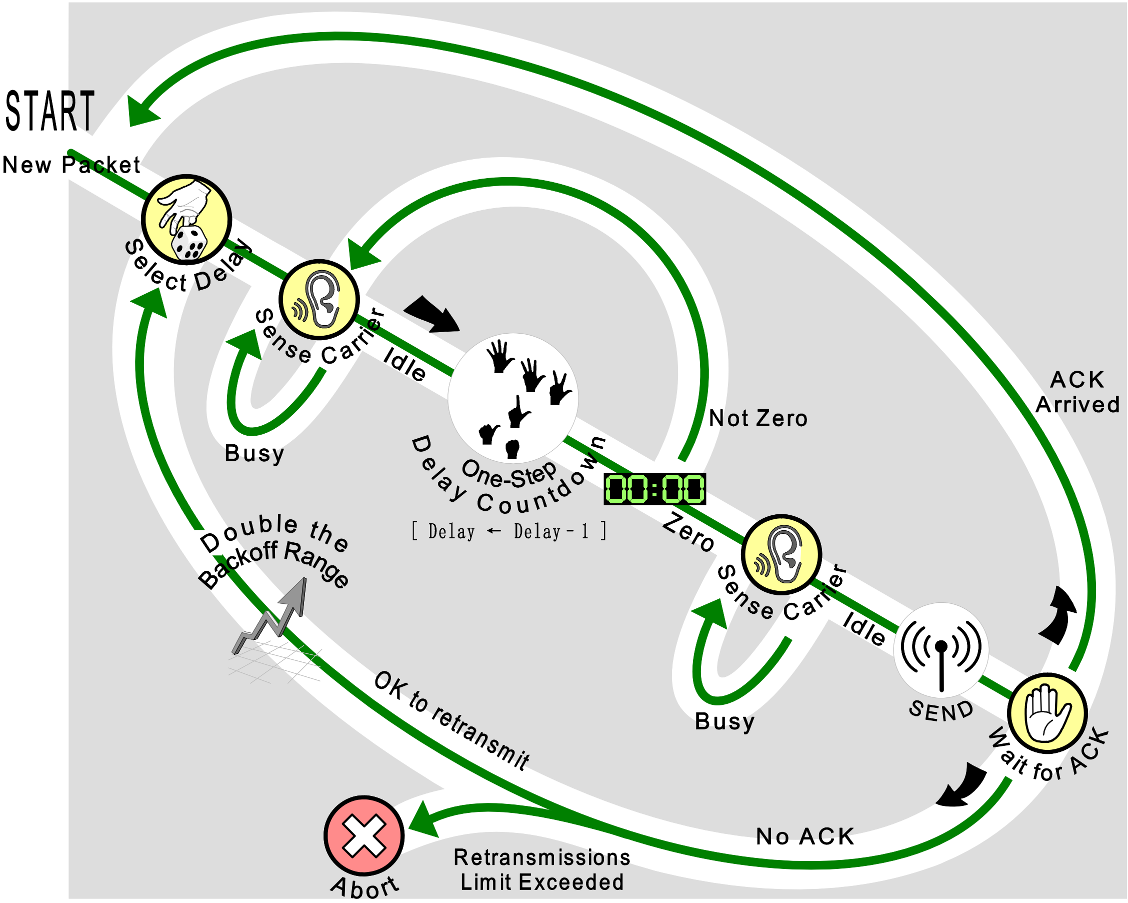


Figure 1-36: The sender's state diagram for CSMA/CA protocol.

1. Implementing a collision detection mechanism would require the implementation of a full duplex radio, capable of transmitting and receiving at once. Unlike wired LANs, where a transmitter can simultaneously monitor the medium for a collision, in wireless LANs the transmitter's power overwhelms a collocated receiver. The dynamic range of the signals on the medium is very large. This is mainly result of the propagation decay, where the signal decays exponentially from its source (recall Figure 1-7!). Thus, a transmitting station cannot effectively distinguish incoming weak signals from noise and the effects of its own transmission.

2. In a wireless environment, we cannot assume that all stations hear each other, which is the basic assumption of the collision detection scheme. Again, due to the propagation decay we have problems like the *hidden station problem* (Figure 1-32). That is, the fact that a transmitting station senses the medium idle does not necessarily mean that the medium is idle around the receiver area.

As a result, when a station transmits a packet, it has no idea whether the packet collided with another packet until it receives an acknowledgment from the receiver (or times out due to the lack of an acknowledgment). In this situation, collisions have a greater effect on performance than with CSMA/CD, where colliding packets can be quickly detected and aborted while the transmission is in progress. Thus, it makes sense to try to avoid collisions, if possible, and a popular scheme for this is *CSMA/Collision Avoidance*, or CSMA/CA. CSMA/CA is essentially *p*-persistence, with the twist that when the medium becomes idle, a station must wait for a time period to learn about the fate of the previous transmission before contending for the medium. Figure 1-36 shows sender's state diagram. After a packet is transmitted, the maximum time until a station detects a collision is twice the propagation time of a signal between the stations that are farthest apart plus the detection time. Thus, the station needs to listen for at least $2\times\beta$ to ensure that the station is always capable of determining if another station has accessed the medium at the start of the previous listening interval. The interval between packets needed for the carrier-sense mechanism to determine that the medium is idle and available for transmission is called a **contention slot** or **backoff slot** (we already introduced this term for CSMA/CD).

When a station wants to transmit a data packet, it first senses the medium whether it is busy. If the medium is busy, the station enters the **access deferral state**. The station continuously senses the medium, waiting for it to become idle. When the medium becomes idle, the station first sets a **contention timer** to a time interval randomly selected in the range [0, $CW-1$], where CW is a predefined **contention window** length. Note that unlike CSMA/CD (Figure 1-33), CSMA/CA station performs carrier sensing after every slot counted down, i.e., it is listening during the contention window (Figure 1-36). In other words, during the backoff procedure, if the station senses the channel as idle for the duration of a backoff slot, the station decrements the counter by one. If the channel is sensed as busy, the station freezes the countdown and waits for the channel to become idle. The station can transmit the packet after it counts down to zero.

After transmitting a packet, the station waits for the receiver to send an ACK. If no ACK is received, the packet is assumed lost to collision, and the source tries again, choosing a contention timer at random from an interval twice as long as the one before (*binary exponential backoff*, see Sidebar 1.2). The decrementing counter of the timer guarantees that the station will transmit, unlike a *p*-persistent approach where for every slot the decision of whether or not to transmit is based on a fixed probability of transmission (p) or retransmission (q_r). Thus regardless of the timer value a station starts at, it always counts down to zero. If the station senses that another station has begun transmission while it was waiting for the expiration of the contention timer, it does not reset its timer, but merely freezes it, and restarts the countdown when the other station completes its transmission. In this way, stations that happen to choose a longer timer value will likely resume its countdown with a small backoff. In effect, such stations get higher priority in the next round of contention.

As seen, CSMA/CA deliberately introduces random delay before transmission in order to avoid collision. If two or more stations transmit simultaneously, they will not be able to discover the collision until their retransmission timers expire for the lack of acknowledgment. Therefore, collision detection in CSMA/CA takes significantly longer than in CSMA/CD. Avoiding collisions increases the protocol efficiency in terms of the percentage of packets that get successfully transmitted (useful throughput). Note that the efficiency measures only the ratio of the successful transmission to the total number of transmissions. However, it does not specify the delays that result from the deferrals introduced to avoid the collisions. **Error! Reference source not found.** shows the qualitative relationship for the average packet delays, depending on the packet arrival rate.

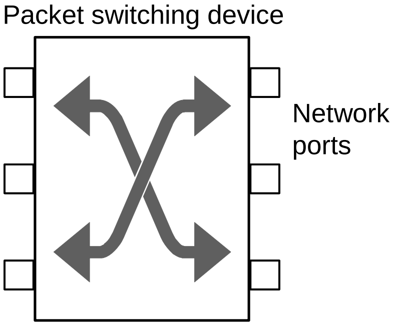
We will see in Section 1.5.3 how IEEE 802.11 wireless LAN, known as Wi-Fi, uses CSMA/CA.

1.4 Routing and Addressing

In general networks, arbitrary source-destination pairs communicate via intermediary network nodes. These intermediary nodes are called switches or routers and their main purpose is to bring packets to their destinations. A good routing protocol will also do it in an efficient way, meaning via the shortest path or a path that is in some sense desirable. The data-carrying capacity of the resulting source-to-destination path directly depends on the efficiency of the routing protocol employed. This section describes how network nodes forward packets towards their destinations.

Bridges, Switches, and Routers

A **packet switch** is a network device with several incoming and outgoing links that forwards packets from incoming to outgoing links. An attachment of a device to the network is called a **network interface** or **network port**. The main function of a switch is to move an incoming packet to an outgoing port and transmit it. The appropriate outgoing port is decided based on the packet's guidance information (contained in the packet header).

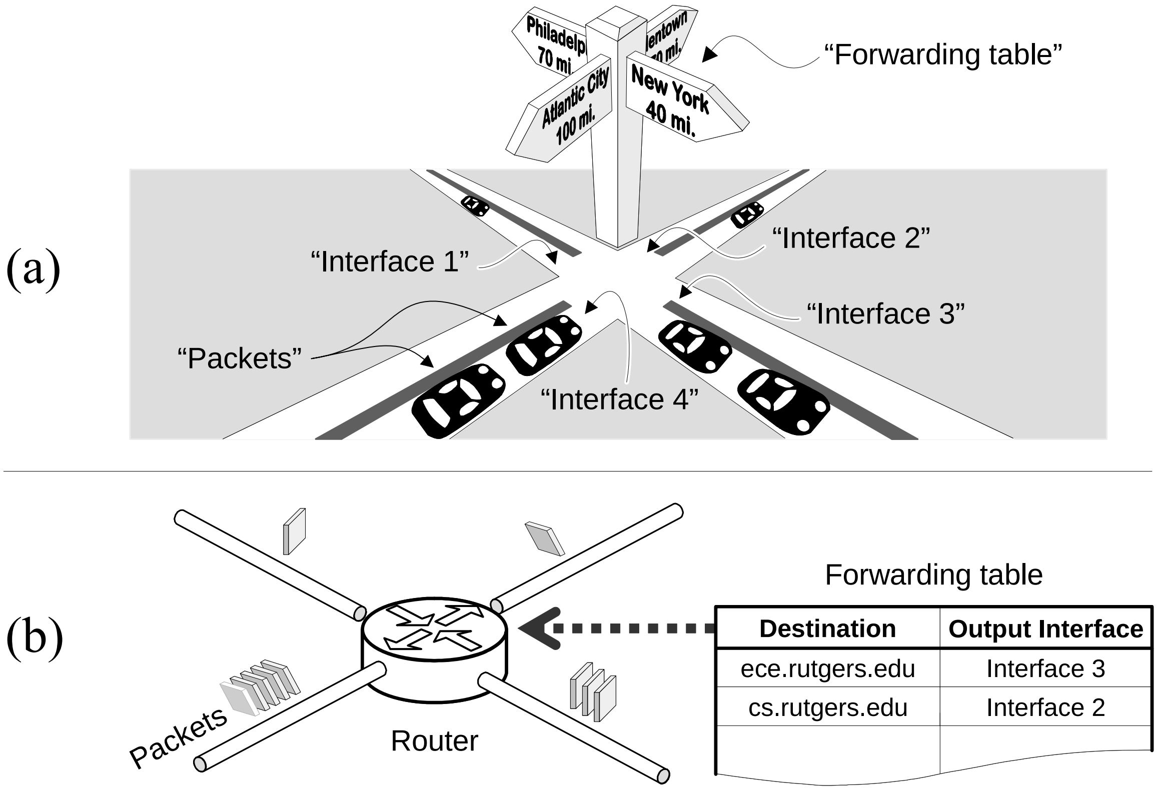


Network switching devices are called different names based on the layer of the protocol stack at which the packet switching takes place. The reader will encounter switching-device names such as *hubs*, *bridges*, *switches* or *routers*. A **hub** switches packets from incoming ports to outgoing ports at OSI Layer 1 (Physical Layer in Figure 1-16). A **bridge** switches packets at OSI Layer 2 (Data Link Layer) provided that both incoming and outgoing links use identical protocols for their physical and link layers of their protocol stack. Because bridged networks use the same protocols, the amount of processing required at the bridge is minimal. There are also more sophisticated bridges, which are capable of mapping from one link-layer protocol format to another. The terms "bridge" and "switch" are often used synonymously. We will continue the discussion of bridges and switches in Section 1.5.2.

Routers are general-purpose packet switches that can interconnect links using arbitrary protocols and maintain information about suitable network paths. A router has two important functions: (1) *routing*, which is the process of finding and maintaining optimal paths between any source and destination nodes in the network; and, (2) *forwarding* (or *switching*), which is the process of relaying incoming data packets along a routing path. A router is a switch that builds its forwarding table using routing algorithms. Differences and similarities between switches and routers will become more clear when we describe how switches work (Section 1.5.2) and how routers work (Section 4.1).

Routing often searches for the shortest path, which in abstract graphs is a graph distance between the endpoint nodes. Shortest path can be determined in different ways, such as:

- Knowing the graph topology, calculate the shortest path

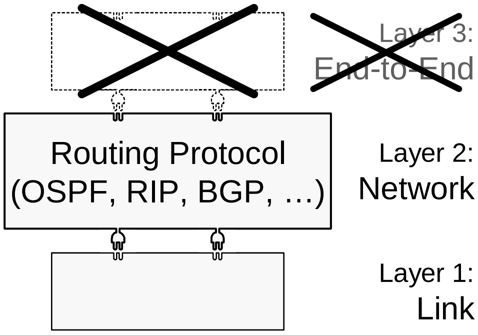


Destination	Output Interface
ece.rutgers.edu	Interface 3
cs.rutgers.edu	Interface 2

Figure 1-37: A router can be thought of as a crossroads, with connecting points corresponding to the router interfaces (also known as “network ports”). When a car (packet) enters the intersection, it is directed out by looking up the forwarding table.

- Send “boomerang” probes on round trips to the destination along the different outgoing paths. Whichever probe returns back the first is the one that carries the information about the shortest path

Figure 1-37 illustrates an analogy between a crossroads and a router. Similar to a road sign, the router maintains a forwarding table that directs the incoming packets to the appropriate exit interfaces, depending on their final destination. Of course, the road signs on different road intersections list different information depending on intersection’s location relative to the roadmap. Similarly, the routing tables in different routers list different information depending on router’s location relative to the rest of the network.



The two major problems of delivering packets in networks from an arbitrary source to an arbitrary location are:

- How to build the forwarding tables in all network nodes
- How to do forwarding (efficiently)

Usually, a requirement is that the path that a packet takes from a source to a destination should be in some sense *optimal*. There are different optimality metrics, such as quickest, cheapest, or most secure delivery. Later, in Sections 1.4.2 and 1.4.3, we will learn about some algorithms for finding shortest paths, known as *routing algorithms*. Other options for routing protocols include constraint-based routing (Section 5.4.3) and policy-based routing (Section 9.2.3).

To do their work, routers maintain two types of information: **Routing Information Base** (RIB), and **Forwarding Information Base** (FIB). RIB, or simply a **routing table**, contains current information about the network topology: the paths that lead to different destinations and costs associated with different paths. FIB, or simply a **forwarding table**, is used to make forwarding decisions for data packets (Figure 1-37). Each entry in a FIB represents a mapping from a destination address to an outgoing interface of the router. A router derives its FIB based on the information currently in its RIB and, in this sense, FIB is redundant. The reason for maintaining both information bases is to speed up the lookup of the outgoing during forwarding of data packets. Because routers forward millions of packets per second, this lookup must be very fast. The RIB information is updated based on advertisement packets received from other routers in the network. Every time there is a change in the RIB, the router also updates the associated entry in its FIB.

Pseudo code of a routing protocol module is given in Listing 1-2.

Listing 1-2: Pseudo code of a routing protocol module.

```
public class RoutingProtocol extends Thread {
    // specifies how frequently this node advertises its routing info
    public static final int ADVERTISING_PERIOD = ...; // in milliseconds

    // link layer protocol that provides services to this protocol
    private ProtocolLinkLayer linkLayerProtocol;

    // associative table of neighboring nodes
    //     (associates their addresses with this node's interface cards)
    private HashMap neighbors;

    // information received from other nodes
    private HashMap othersRoutingInfo;

    // this node's routing table
    private HashMap myRoutingTable;

    // constructor
    public RoutingProtocol(
        ProtocolLinkLayer linkLayerProtocol
    ) {
        this.linkLayerProtocol = linkLayerProtocol;

        populate myRoutingTable with costs to my neighbors;
    }

    // thread method; runs in a continuous loop and sends routing-info advertisements
    //      to all the neighbors of this node
    public void run() {
        while (true) {
            try { Thread.sleep(ADVERTISING_PERIOD); }
            catch (InterruptedException e) {
                for (all neighbors) {
                    Boolean status = linkLayerProtocol.send();
                    // If the link was down, update own routing & forwarding tables
                    //      and send report to the neighbors
                    if (!status) {
                    }
                }
            }
        }
    }

    // upcall method (called from the layer below this one, running in a bottom-layer thread)
    //      the received packet contains an advertisement/report from a neighboring node
    public void handle(byte[] data) throws Exception {

        // reconstruct the packet as in Listing 1-1 (Section 1.1.4) for a generic handle()
        //      but there is no handover to an upper-layer protocol,
        //      so this protocol handles the received report.
        synchronized (routingTable) { // critical region
            ...     // update my routing table based on the received report
        } // end of the critical region

        // update the forwarding table of the peer forwarding protocol
        //      (note that this protocol is running in a different thread!)
        call the method setReceiver() in Listing 1-1
    }
}
```

The code description is as follows: … to be described …

Routing is not an easy task as will become apparent in this and other chapters. Optimal routing requires a detailed and timely view of the network topology and link statuses. However, obtaining such information requires a great deal of periodic messaging between all nodes to notify each other about the network state in their local neighborhoods. This messaging is viewed as *overhead* because it carries control data and reduces the resources available for carrying user data. The network engineer strives to reduce overhead. In addition, the finite speed of propagating the messages and processing delays in the nodes imply that the nodes always deal with an outdated view of the network state. Moreover, autonomous systems managed by independent organizations are unwilling to disclose detailed information about their networks (Section 1.4.5). Therefore, in real-world networks routing protocols always deal with a partial and outdated view of the network state. The lack of perfect knowledge of the network state can lead to a poor behavior of the protocol or degraded performance of the applications.

Path MTU is the smallest maximum transmission unit of any link on the current path (also known as the *route*) between two hosts. The concept of MTU for individual links is defined in Section 1.1.3. Then, for path p, path MTU = min{MTU$_i$, for all links i belonging to path p}.

This section deals mainly with the control functions of routers that include building the routing tables. Later, in Section 4.1 we consider how routers forward packets from incoming to outgoing links. This process consists of several steps and each step takes time, which introduces delays in

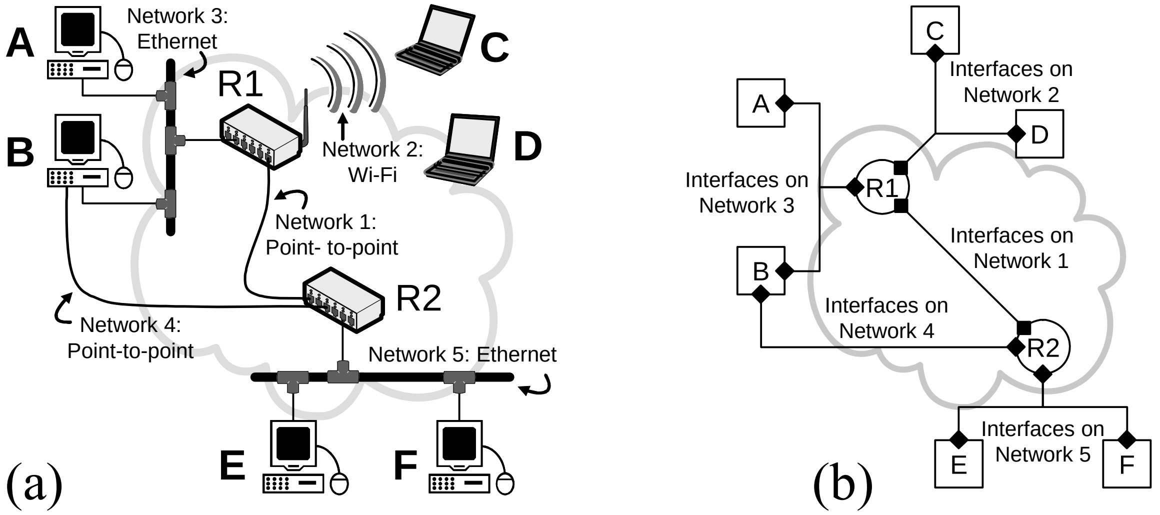


Figure 1-38: Example internetwork: (a) The physical networks include 2 Ethernets, 2 point-to-point links, and 1 Wi-Fi network. (b) Topology of the internetwork with indicated network interfaces.

packet delivery. Also, due to the limited size of router memory, some incoming packets may need to be discarded for the lack of memory space. Section 4.1 describes methods to reduce forwarding delays and packet loss due to memory shortage.

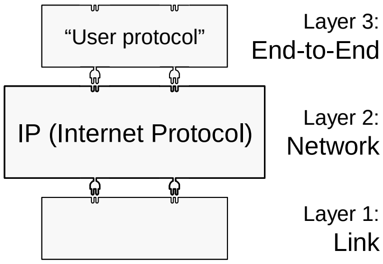
1.4.1 Networks, Internets, and the IP Protocol

A *network* is a set of computers directly connected to each other, with no intermediary nodes or networks. In other words, nodes are connected over a single link, or "hop." A network of networks is called **internetwork** or **internet**. The best known internetwork is as the *Internet* that connects computers worldwide.

Consider an example internetwork in Figure 1-38(a), which consists of five physical networks interconnected by two routers. The underlying network that a device uses to connect to other devices could be a LAN connection like Ethernet or Token Ring, a wireless LAN link such as 802.11 (known as Wi-Fi) or Bluetooth, or a dialup, DSL, or a T-1 connection. Each physical network will generally use its own packet format, and each format has a limit on how much data can be sent in a single packet (link MTU, Section 1.1.3).

Two types of network nodes are distinguished: endpoint hosts vs. routers. Each *host* usually has a single network attachment point, known as **network interface**, and therefore it cannot relay packets for other nodes. Even if a host has two or more network interfaces, such as node B in Figure 1-38(a), it is not intended to be used for transit traffic. Hosts usually do not participate in the routing process, because they do not run routing protocols. Unlike hosts, *routers* have the primary function of relaying (handing over) transit traffic between other nodes. Each router has a minimum of two, but usually many more, network attachment points (interfaces). In Figure 1-38(a), both routers R1 and R2 each have three network interfaces. Each interface on every host and router must have a network address that is globally unique.[6] A node with two or more network interfaces is said to be **multihomed**[7] (or, multiconnected). Note that multihomed hosts do *not* participate in routing or forwarding of transit traffic because they are not equipped to do so. Multihomed hosts act as any other end host, except that they may use different interfaces for different destinations, depending on the destination distance.

The idea of a network layer protocol is to implement the concept of a "virtual network" where devices can talk even though they are far away, connected using different physical network technologies. This means that the layers above the network layer do not need to worry about details, such as differences in packet formats or size limits of underlying link-layer technologies. The network layer manages these issues seamlessly and presents a uniform interface to the higher layers. The most commonly used network-layer protocol is the *Internet Protocol* (IP). The most commonly deployed version of IP is version 4 (IPv4). The next generation, IP version 6 (IPv6),[8] is designed to address the shortcomings of IPv4 and currently there is a great effort in transitioning the Internet to IPv6. IPv6 is reviewed in Section 9.1.



IP Header Format

Data transmitted over a network-of-networks (called "internet") using IP is carried in packets called IP *datagrams*. Figure 1-39 shows the format of IP version 4 datagrams. Its fields are as follows:

Version number: This field indicates the version number of the IP protocol, to allow evolution of the protocol. The value of this field for IPv4 datagrams is 4.

Header length: This field specifies the length of the IP header in 32-bit "words." Regular header length is 20 bytes, so the default value of this field equals 5 (words), which is also the minimum allowed value. In case the options field is used, the value contained in 4 bits can be up to $4^2 - 1 = 15$, which means that the options field may contain up to $(15 - 5) \times 4 = 40$ bytes.

Differentiated services: This field is used to specify the treatment of the datagram in its transmission through component networks. It was designed to carry information about the desired quality of service features, such as prioritized delivery. Originally this field was called "Type of service" (or, ToS) and it was never widely used. Its meaning has been subsequently redefined for use by a technique called Differentiated Services (Section 3.3.5), where the top 6 bits represent the service class for this packet. The bottom 2 bits are used to carry explicit congestion notification information (Section 5.3.2).

Datagram length: Total datagram length, including both the header and data, in bytes.

[6] This is not necessarily true for interfaces that are behind NATs (described in Section 8.3.3).

[7] Most notebook computers nowadays typically have two or more network interfaces, such as Ethernet, Wi-Fi, Bluetooth, etc. However, the host becomes "multihomed" only if two or more interfaces are assigned unique network-layer addresses and they are simultaneously active on their respective physical networks.

[8] IP version 5 designates the Stream Protocol (SP), a connection-oriented network-layer protocol. IPv5 was an experimental real-time stream protocol that was never widely used.

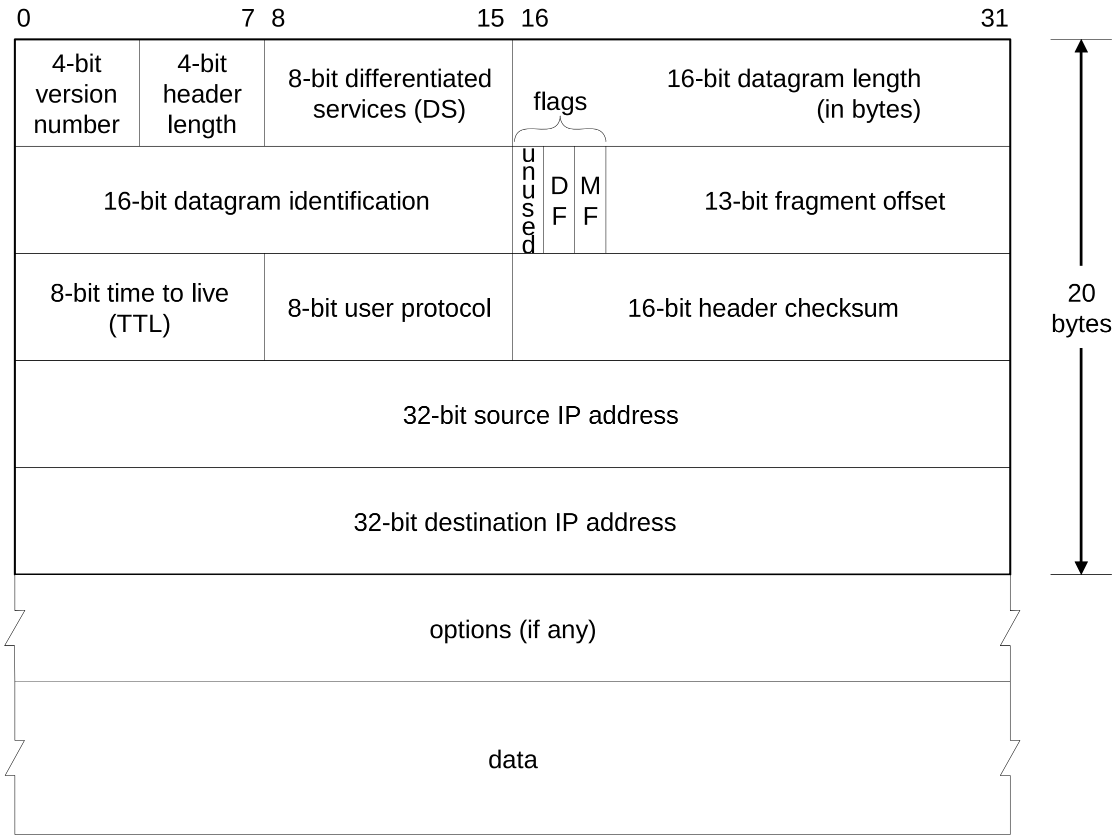


Figure 1-39: The format of IPv4 datagrams.

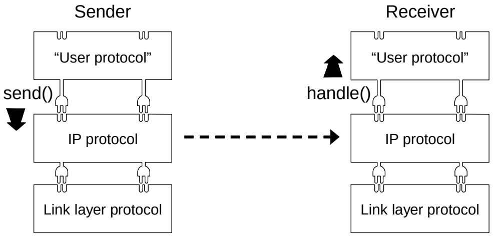
Identification: This is a unique identifier for a datagram that is used in case of datagram fragmentation, as described later in this section. This is *not* a sequence number and IP does not keep track of sequence numbers of packets. Instead, if an original datagram is fragmented into smaller datagrams, each resulting datagram carries the same identifier, so that the receiver can recognize them as pieces of the same original datagram.

Flags: There are three flag bits, of which only two are currently defined. The first bit is reserved and currently unused. The **DF** (Don't Fragment) bit when set prohibits fragmentation of this datagram. This bit may be useful if it is known that the destination does not have the capability to reassemble fragments. However, if this bit is set and the datagram exceeds the MTU size of the next link, the datagram will be discarded. The **MF** (More Fragments) bit is used to indicate the fragmentation parameters. When this bit is set, it indicates that this datagram is a fragment of an original datagram and this is not its last fragment.

Fragment offset: This field indicates the starting location of this fragment within the original datagram. The meaning of this field will become apparent later in this section, when datagram fragmentation is described. It is measured in 8-byte (64-bit) units, which implies that the length of data carried by all fragments before the last one must be a multiple of 8 bytes. The reason for specifying the offset value in units of 8-byte chunks is that only 13 bits are allocated for the offset field, which makes possible to refer to 8,192 locations. On the other hand, the datagram length field of 16 bits allows for datagrams up to 65,536 bytes long. Therefore, to be able to specify any offset value within an arbitrary-size datagram, the offset units are in 65,536 ÷ 8,192 = 8-byte units.

Time to live: The TTL field specifies how long a datagram is allowed to remain in the Internet, to catch packets that are stuck in routing loops. This field was originally set in seconds, and every router that relayed the datagram decreased the TTL value by one. In current practice, a more appropriate name for this field is *hop limit counter* and its default value is usually set to 64.

User protocol: This field identifies the higher-level protocol that uses IP for communication. At the destination node, the IP protocol consults this field to decide the upper-layer protocol to which to deliver the payload. In other words, as per Figure 1-13, this field identifies the type of the next header contained in the payload of this datagram (i.e., after the IP header). Example values are 6 for TCP, 17 for UDP, and 1 for ICMP. A complete list is maintained at http://www.iana.org/assignments/protocol-numbers.



Header checksum: This is an error-detecting code applied only to the header of this datagram. Because some header fields may change during transit (e.g., TTL, fragmentation fields), this field is reverified and recomputed at each router. The checksum is computed by adding all 16-bit words in the IP header (using 16-bit ones-complement addition) and taking the ones complement of the sum. Before this computation, the checksum field is initialized to a value of zero to exclude it from the checksum computation. Each router uses the computed checksum to compare it to the checksum found in the received datagram header. If the computed and received checksums differ, the router concludes that the datagram is corrupted and discards it.

Source IP address: This address identifies the end host that originated the datagram. The address format is described later in Section 1.4.4.

Destination IP address: This address identifies the end host that is to receive the datagram.

Options: This field encodes the options requested by the sending user. As stated earlier, the options field may contain up to 40 bytes. The possible options include: 1-bit "Copied" flag that, when set to 1, indicates that the options should be copied into all fragments of a fragmented packet; 2-bit "Option Class" indicating the options category, such as 0 for "control", and 2 for "debugging and measurement"; and variable length "Option Data" specific to a particular option.

To send messages using the IP protocol, we encapsulate the data from a higher-layer ("user") protocol into IP datagrams. These datagrams must then be sent down to the link-layer protocol, where they are further encapsulated into the frames of whatever technology is going to be used to physically convey them, either directly to their destination, or indirectly to the next intermediate step in their journey to their intended recipient. (The encapsulation process is illustrated in Figure 1-13.) The link-layer protocol puts the entire IP datagram into the data portion (the payload) of its frame format, just as IP puts end-to-end layer messages, end-to-end headers and all, into its IP Data field. Recall also that the protocol at each layer looks only at that layer's header and ignores all other headers of the packet (Figure 1-14).

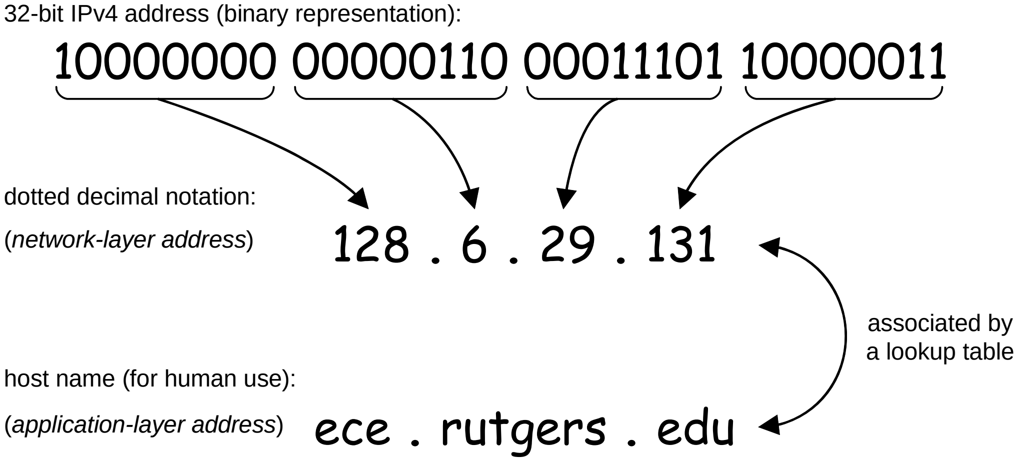

32-bit IPv4 address (binary representation):
10000000 00000110 00011101 10000011
dotted decimal notation:
(*network-layer address*)
128 . 6 . 29 . 131
associated by a lookup table
host name (for human use):
(*application-layer address*)
ece . rutgers . edu

Figure 1-40: Dotted decimal notation for IP version 4 addresses.

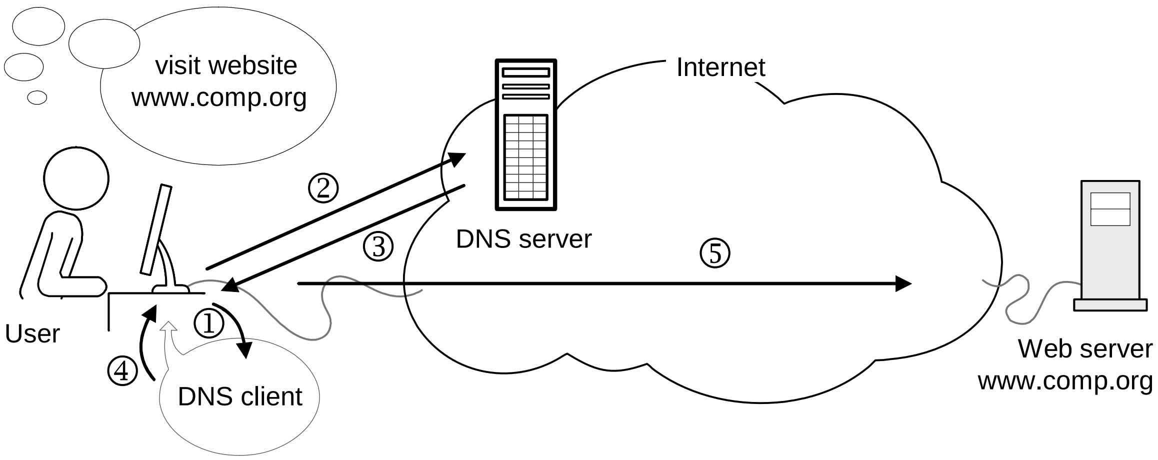
Naming and Addressing

Names and addresses play an important role in all computer systems as well as any other symbolic systems. They are labels assigned to entities such as physical objects or abstract concepts, so those entities can be referred to in a symbolic language. Because computation is specified in and communication uses symbolic language, the importance of names should be clear. It is important to emphasize the importance of naming the network nodes, because if a node is not named, it does not exist! We simply cannot target a message to an unknown entity[9]. The main issues about naming include:

- Names must be *unique* so that different entities are not confused with each other
- Names must be *bound* to and *resolved* with the entities they refer to, to determine the object of computation or communication

It is common in computing and communications to differentiate between names and addresses of objects. Technically, both are addresses (of different kind), but we distinguish them for easier usage. *Names* are usually human-understandable, therefore variable length (potentially rather long) and may not follow a strict format. *Addresses* are intended for machine use, and for efficiency reasons have fixed lengths and follow strict formatting rules. For example, you could name your computers: “My office computer for development-related work” and “My office computer for business correspondence.” The addresses of those computers could be: 128.6.236.10 and 128.6.237.188, respectively. Figure 1-40 illustrates the relationship between the binary representation of an IP address, its dotted-decimal notation, and the associated name. One could say that “names” are *application-layer addresses* and “addresses” are *network-layer addresses*. Note that in dotted-decimal notation the maximum decimal number is 255, which is the maximum number that can be represented with an 8-bit field. The mapping between the names and addresses is performed by the **Domain Name System (DNS)**, Figure 1-41. What is labeled as

[9] Many communication networks allow broadcasting messages to all or many nodes in the network. Hence, in principle the sender could send messages to nodes that it does not know of. However, this is not an efficient way to communicate and it is generally reserved for special purposes.



① Web browser passes the name “www.comp.org” to DNS client

② DNS client sends query with “www.comp.org” to DNS server

③ DNS server responds with IP address of “www.comp.org”

④ DNS client passes the IP address to Web browser

⑤ Web browser initiates connection to the Web server using its IP address

Figure 1-41: Domain Name System (DNS) supports the lookup of IP addresses based on human-readable computer names.

“DNS server” in Figure 1-41 actually is a worldwide collection of computers running a DNS protocol, as described in Section 9.4.

Distinguishing names and addresses is useful for another reason: this separation allows keeping the same name for a computer that needs to be labeled differently when it moves to a different physical place (see Mobile IP in Section 9.3.4). For example, the name of your friend may remain the same in your email address book when he or she moves to a different company and changes their email address. Of course, the name/address separation implies that there should be a mechanism for name-to-address binding and address-to-name resolution.

Two most important address types in contemporary networking are:

- Link-layer address of a device, also known as *medium access control (MAC) address*, which is a physical address for a given *network interface card* (NIC), also known as *network adaptor* or *line card*. These addresses are standardized by the IEEE group in charge of a particular physical-layer communication standard, assigned to different vendors, and hardwired into the physical devices.

- Network-layer address of a device, which is a logical address and can be changed by the end user. This address is commonly referred to as *IP address*, because IP is by far the most common network-layer protocol. Network-layer addresses are standardized by the Internet Engineering Task Force (http://www.ietf.org).

Note that a quite independent addressing scheme is used for telephone networks and it is governed by the International Telecommunications Union (http://www.itu.int).

People designed postal addresses with a structure that facilitates human memorization and post-service delivery of mail. So, a person’s address is structured hierarchically, with country name on

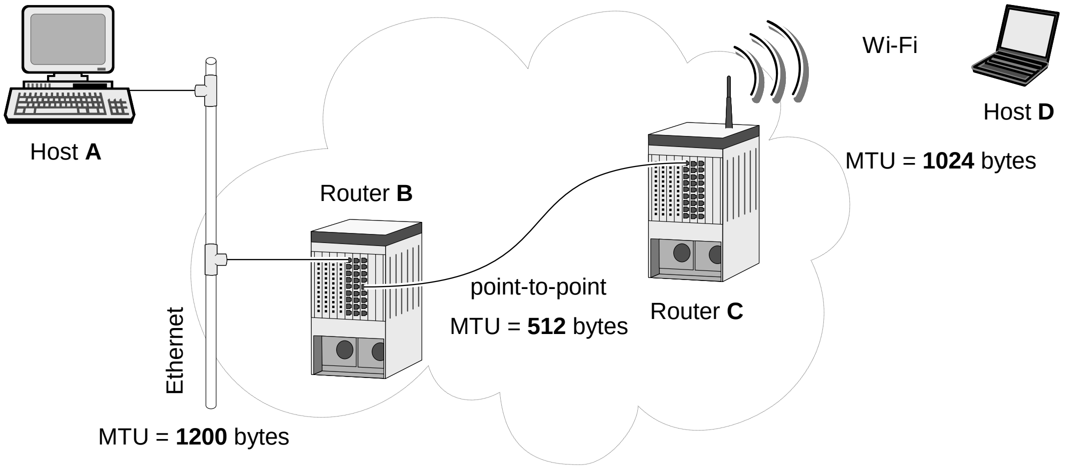


Figure 1-42: Example scenario for IP datagram fragmentation.

top of the hierarchy, followed by the city name, postal code, and the street address. One may wonder whether there is anything to be gained from adopting a similar approach for network computer naming. After all, computers deal equally well with numbers and do not need mnemonic techniques to help with memorization and recall. It turns out that in very large networks, the address structure can assist with more efficient message routing. Section 1.4.4 describes how IPv4 addresses are structured to assist routing.

Datagram Fragmentation and Reassembly

Problems related to this section: Problem 1.24

The Internet Protocol's main responsibility is to deliver data between devices on different networks, i.e., across an internetwork. For this purpose, the IP layer encapsulates data received from higher layers into IP datagrams for transmission. These datagrams are then passed down to the link layer where they are sent over physical network links.

In Section 1.1.3, we saw that underlying network technology imposes the upper limit on the frame (packet) size, known as maximum transmission unit (MTU). As the datagram is forwarded along the source-destination path, each hop may use a different physical network, with a different maximum underlying frame size. If an IP datagram is larger than the MTU of the underlying network, it may be necessary to break up the datagram into several smaller datagrams. This process is called **fragmentation**. The fragment datagrams are then sent individually and reassembled at the destination into the original datagram.

IP is designed to manage datagram size in a seamless manner. It matches the size of the IP datagram to the size of the underlying link-layer frame size, and performs fragmentation and reassembly so that the upper-layer protocols are not aware of this process. Here is an example:

Example 1.2 Illustration of IP Datagram Fragmentation

In the example scenario shown in Figure 1-42, an application on host *A*, say email client, needs to send a JPEG image to the receiver at host *D*. Assume that the sender uses the TCP protocol (described in Chapter 2), which in turn uses IP as its network-layer protocol. The first physical network is Ethernet (Section 1.5.2), which for illustration is configured to limit the size of the payload it sends to 1,200

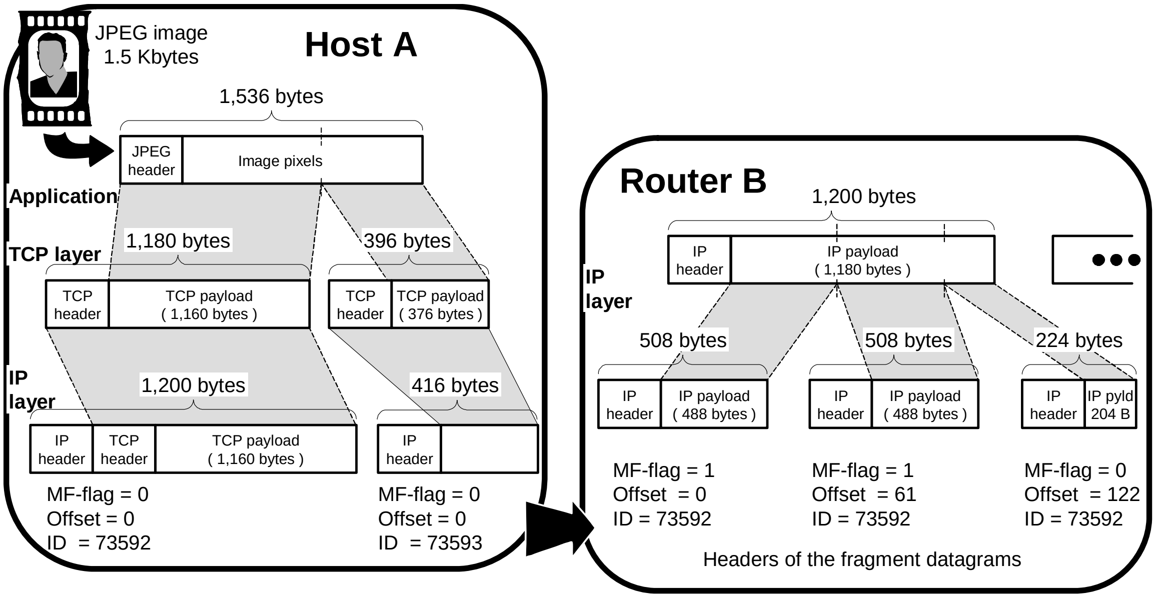


Figure 1-43: IP datagram fragmentation at Router B of the network shown in Figure 1-42. The fragments will be reassembled at the destination (Host D) in an exactly reverse process.

bytes. The second network uses a Point-to-Point protocol that limits the payload size 512 bytes and the third network is Wi-Fi (Section 1.5.3) with the payload limit equal to 1024 bytes.

Figure 1-43 illustrates the process by which IP datagrams are fragmented by the source device and possibly routers along the path to the destination. As we will learn in Chapter 2, TCP learns from IP about the MTU of the first link and prepares the TCP packets to fit this limit, so the host's IP layer does not need to perform any fragmentation. However, router *B* needs to break up the datagram into several smaller datagrams to fit the MTU of the point-to-point link. As shown in Figure 1-43, the IP layer at router *B* creates three smaller datagrams from the first datagram it receives from host *A*.

The bottom row in Figure 1-43(b) shows the contents of the fragmentation-related fields of the datagram headers (the second row of the IP header shown in Figure 1-39). Router *B* creates three fragments from the first datagram carrying 1180 bytes of payload: 488 + 488 + 204 = 1180 bytes. Recall that the length of data carried by all fragments before the last one must be a multiple of 8 bytes and the offset values are in units of 8-byte chunks. Because of this constraint, the size of the first two datagrams created by fragmentation on router *B* is 508 bytes (20 bytes for IP header + 488 bytes of IP payload). Although the MTU allows IP datagrams of 512 bytes, this would result in a payload size of 492, which is not a multiple of 8 bytes. Note also that the offset value of the second fragment is 61, which means that this fragment starts at 8 × 61 = 488 bytes in the original IP datagram from which this fragment is created.

It is important to reemphasize that lower layer protocols do not distinguish any structure in the payload passed to them by an upper-layer protocol (Figure 1-13). Therefore, although in the example of Figure 1-43 the payload of IP datagrams contains both TCP header and user data, the IP does *not* distinguish any structure within the datagram payload. When the IP layer on Router *B* receives an IP datagram with IP header (20 bytes) + 1,180 bytes of payload, it removes the IP header and does not care what is in the payload. Router *B*'s IP layer splits the 1,180 bytes into three fragments, so that when it adds its own IP header in front of each payload fragment, none of the resulting IP datagrams will exceed 512 bytes in size. Router *B* then forwards the three datagrams to the next hop.

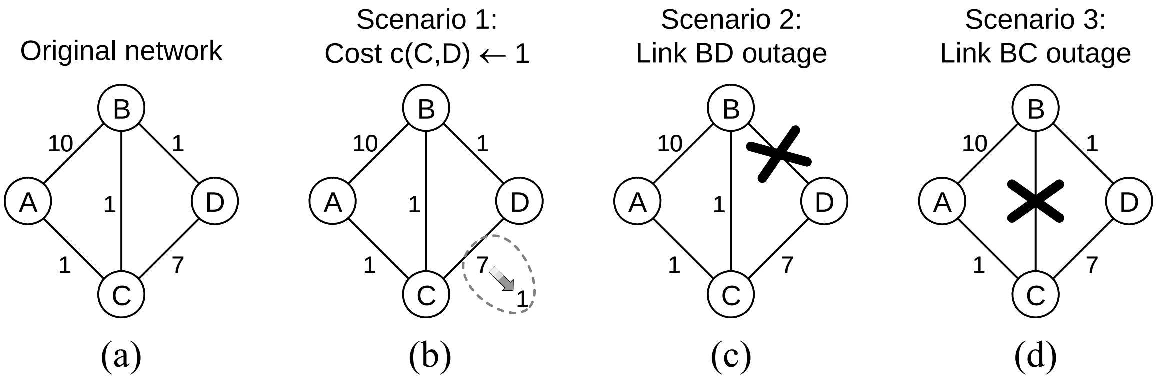


Figure 1-44: Example network used for illustrating the routing algorithms.

Datagram reassembly takes place in the destination endpoint, instead of happening somewhere in the network. In other words, the IP protocol does not take advantage of greater MTUs on packet's path. For example Router *C* in Figure 1-42 will not try to reassemble the fragments created by Router *B*, but rather will let host *D* to perform the reassembly. The reason is that some fragments would be delayed until the matching fragments arrived, which would introduce additional delays during packet transit. In addition, IP does not guarantee that all datagrams (or their fragments) will travel the same route to the destination. Routers would not know whether the matching fragments are delayed, lost, or simply took a different route.

1.4.2 Link State Routing

Problems related to this section: Problem 1.25 → Problem 1.26

Routers forward packets across the network, from source to destination. A key problem of routing algorithms is finding the *shortest path* between any two nodes, such that the sum of the costs of the links along the path is minimized. The two most popular algorithms used for this purpose are Dijkstra's algorithm, used in link state routing, and Bellman-Ford algorithm, used in distance vector routing. The link state routing is presented first, followed by the distance vector routing; Section 1.4.5 describes the path vector routing, which is similar to the distance vector routing.

When determining the minimum-cost path (i.e., distance), it is important to keep in mind that we are not interested in how people would solve this problem. Rather, we wish to know how a group of computers can solve such a problem. Computers (routers) cannot rely on what we people see by looking at the network's graphical representation; computers must work only with the information exchanged in messages.

Routers maintain the up-to-date information about the network connectivity in their routing information base (RIB), or "routing table," by exchanging control messages that advertise each router's knowledge about the network to other routers. The advertisements are not necessarily delivered using a reliable transmission protocol. However, most modern routing protocols use reliable delivery for exchange of routing information.

We use $\boldsymbol{N}$ to denote the set of all nodes in a network. For example, in Figure 1-44, $\boldsymbol{N} = \{A, B, C, D\}$. Routing algorithms usually work with types of quantities:

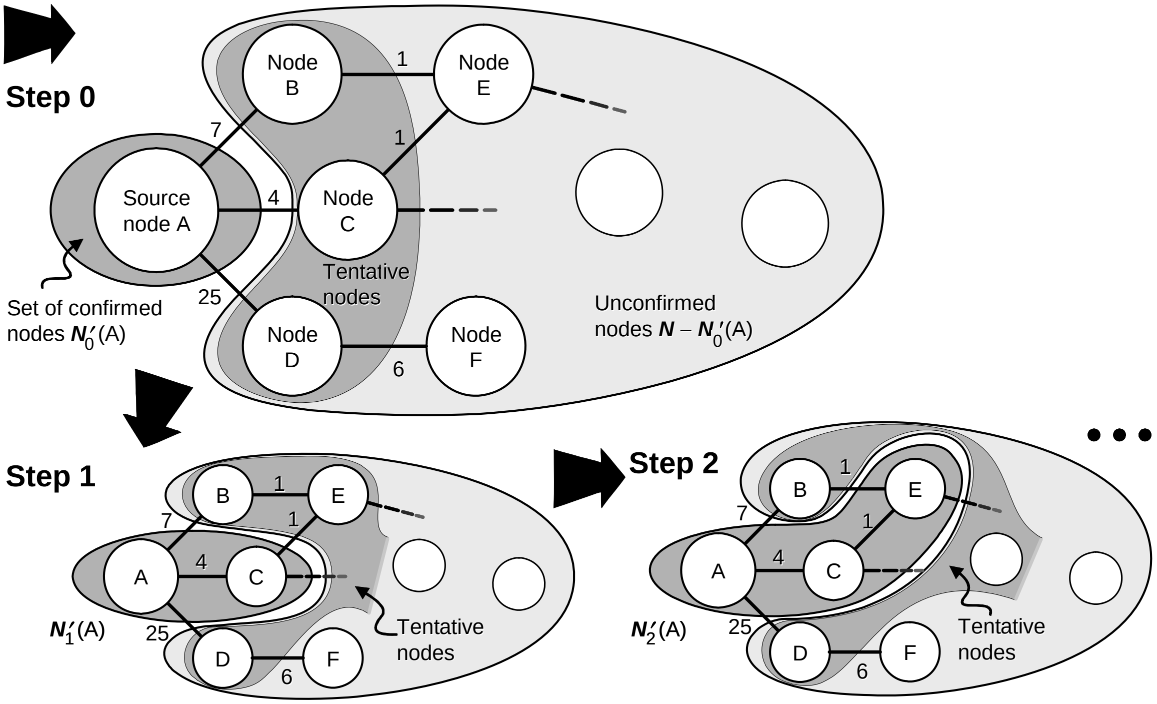


Figure 1-45: Illustration of finding the shortest path using Dijkstra's algorithm.

(i) *Link cost* is a number assigned to an individual link directly connecting a pair of nodes (routers). For example, in Figure 1-44 the cost of the link connecting the nodes *A* and *B* is labeled as "10" units, that is $c(A, B) = 10$. Link costs are given to the algorithm either by the network operator manually entering the cost values or by an independent program determines these costs.

(ii) *Node distance* for an arbitrary pair of nodes, which represents the lowest sum of link costs for all links along all the possible paths between this node pair. The distance (shortest path) from node *X* to node *Y* is denoted as $D_X(Y)$. These will be computed by the routing algorithm.

The key idea of the link state routing algorithm is to disseminate the information about local connectivity of each node to all other nodes in the network. Once all nodes gather the local information from all other nodes, each node knows the topology of the entire network and can independently compute the shortest path from itself to any other node in the network. This is done by iteratively identifying the closest node from the source node in the order of increasing path cost (Figure 1-45). At the k^{th} step we have the set $\boldsymbol{N'_k}(A)$ of *k* closest nodes to node *A*, which are called "confirmed nodes." All nodes not in $\boldsymbol{N'_k}(A)$ are called "unconfirmed nodes." We know the shortest path (distance) D_X from each node *X* in $\boldsymbol{N'_k}(A)$ to node *A*. Of all paths connecting some unconfirmed node not in $\boldsymbol{N'_k}(A)$ to node *A*, there is the shortest path that passes exclusively through nodes in $\boldsymbol{N'_k}(A)$, because $c(X, Y) \geq 0$ for any two nodes *X*, *Y*. Therefore, the $(k + 1)$st closest node should be selected among those unconfirmed nodes that are neighbors of nodes in $\boldsymbol{N'_k}(A)$. These nodes are marked as "tentative nodes" in Figure 1-45.

When a router (network node *A*) is initialized, it determines the link cost on each of its network interfaces. For example, in Figure 1-45 the cost of the link connecting node *A* to node *B* is labeled as "7" units, that is $c(A, B) = 7$. The node then advertises this set of link costs to *all* other nodes in the network (not just its neighboring nodes). Each node receives the link costs of all nodes in the network and, therefore, each node has a representation of the entire network. To advertise the link costs, the node creates a packet, known as *Link-State Advertisement* (LSA) or *Link-State Packet* (LSP), which contains the following information:

- The ID of the node that created the LSA
- A list of directly connected neighbors of this node, with the link cost to each one
- A sequence number for this packet
- A time-to-live for this packet

In the initial step, all nodes send their LSAs to all other nodes in the network using the mechanism called *broadcasting*. The shortest-path algorithm, which is described next, starts with the assumption that all nodes already exchanged their LSAs. The next step is to build a routing table (or RIB), which is an intermediate step towards building a forwarding table (or FIB).

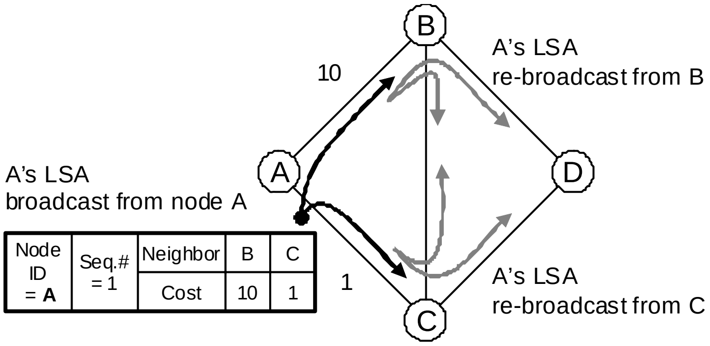
The *link state routing algorithm* works as follows. Let ***N*** denote the set of all nodes in a network. The process can be summarized as an iterative execution of the following steps

1. Check the LSAs of all nodes in the **confirmed set *N′*** (also known as the **PATH list**) to update the **tentative set** (recall that tentative nodes are unconfirmed nodes that are neighbors of confirmed nodes)
2. Move the node with the shortest path length from the tentative set to the confirmed set ***N′***.
3. Go to Step 1.

The process stops when ***N′*** = ***N***. Here is an example:

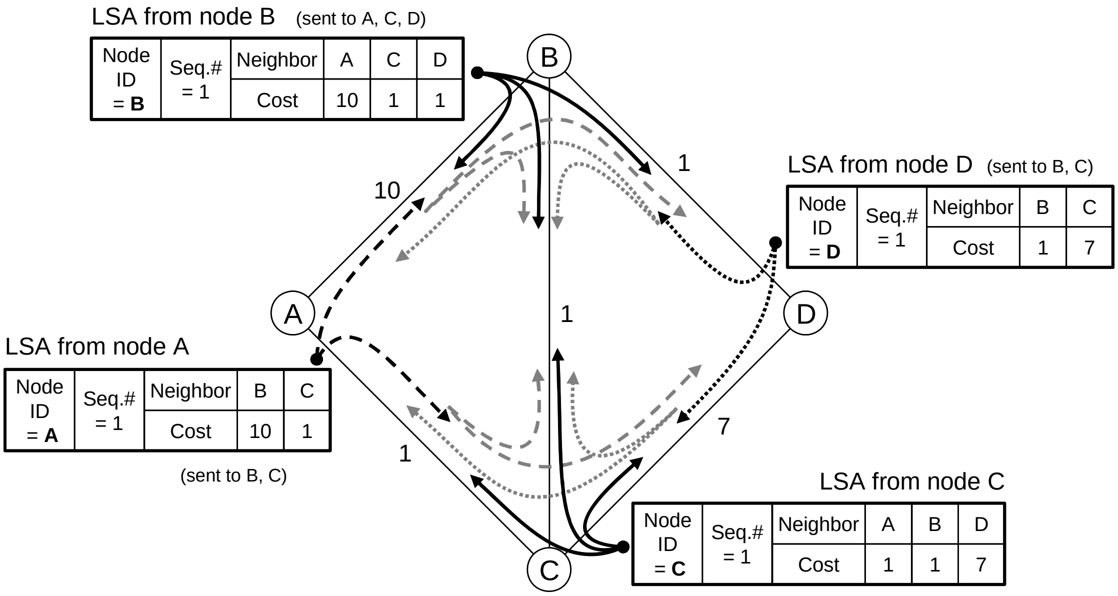
Example 1.3 Link State Routing Algorithm

Consider the network in Figure 1-44(a) and assume that it uses the link state routing algorithm. Starting from the initial state for all nodes, show how node *A* finds the shortest paths to all other nodes in the network. The figure below shows how node *A*'s link-state advertisement (LSA) is broadcast through the network.



Node ID = **A**	Seq.# = 1	Neighbor	B	C
		Cost	10	1

Assume that all nodes broadcast their LSAs and each node already received LSAs from all other nodes in the network before it starts the shortest path computation, as shown in this figure:



LSA from node B (sent to A, C, D)

Node ID = **B**	Seq.# = 1	Neighbor	A	C	D
		Cost	10	1	1

LSA from node D (sent to B, C)

Node ID = **D**	Seq.# = 1	Neighbor	B	C
		Cost	1	7

LSA from node A

Node ID = **A**	Seq.# = 1	Neighbor	B	C
		Cost	10	1

(sent to B, C)

LSA from node C

Node ID = **C**	Seq.# = 1	Neighbor	A	B	D
		Cost	1	1	7

Table 1-2 shows the process of building a routing table at node *A* of the network shown in Figure 1-44(a). Each node is represented with a triplet (*Destination node ID*, *Path length*, *Next hop*). The node *x* maintains two sets (recall Figure 1-45): Confirmed(*x*) set, denoted as ***N′***, and Tentative(*x*) set. At the end, the routing table in node *A* contains these entries: {(*A*, 0, −), (*C*, 1, *C*), (*B*, 2, *C*), (*D*, 3, *C*)}. Every other node in the network runs the same algorithm to compute its own routing table.

To account for failures of network elements, the nodes should repeat the whole procedure periodically. That is, each node periodically broadcasts its LSA to all other nodes and recomputes its routing table based on the received LSAs.

Table 1-2: Steps for building a routing table at node *A* in Figure 1-44. Each node is represented with a triplet (*Destination node ID, Path length, Next hop*).

Step	Confirmed set *N′*	Tentative set	Comments
0	(*A*, 0, −)	∅	Initially, *A* is the only member of Confirmed(*A*), so examine *A*'s LSA.
1	(*A*, 0, −)	(*B*, 10, *B*), (*C*, 1, *C*)	*A*'s LSA says that *B* and *C* are reachable at costs 10 and 1, respectively. Since these are currently the lowest known costs, put on Tentative(*A*) list.
2	(*A*, 0, −), (*C*, 1, *C*)	(*B*, 10, *B*)	Move lowest-cost member (*C*) of Tentative(*A*) into Confirmed set. Next, examine LSA of newly confirmed member *C*.
3	(*A*, 0, −), (*C*, 1, *C*)	(*B*, 2, *C*), (*D*, 8, *C*)	Cost to reach *B* through *C* is 1+1=2, so replace (*B*, 10, *B*). *C*'s LSA also says that *D* is reachable at cost 7+1=8.
4	(*A*, 0, −), (*C*, 1, *C*), (*B*, 2, *C*)	(*D*, 8, *C*)	Move lowest-cost member (*B*) of Tentative(*A*) into Confirmed, then look at *B*'s LSA.
5	(*A*, 0, −), (*C*, 1, *C*), (*B*, 2, *C*)	(*D*, 3, *C*)	Because *D* is reachable via *B* at cost 1+1+1=3, replace the Tentative(*A*) entry for *D*.
6	(*A*, 0, −), (*C*, 1, *C*), (*B*, 2, *C*), (*D*, 3, *C*)	∅	Move lowest-cost member (*D*) of Tentative(*A*) into Confirmed. END.

Limitations: Routing Loops

Link state routing needs large amount of resources to calculate routing tables. It also creates heavy traffic because of flooding the LSA packets from each node throughout the network.

On the other hand, link state routing converges much faster to correct values after link failures than distance vector routing (described in Section 1.4.3), which suffers from the so-called counting-to-infinity problem.

Before the nodes start their routing table computation (as in Table 1-2), they all must have received the same LSAs from all other nodes in the network. Link state routing specifies only the next hop, not the whole path to any destination. If not all of the nodes are working from *exactly* the same map of the network, routing loops can form. A **routing loop** is a subset of network nodes configured so that data packets may wander aimlessly in the network, making no progress towards their destination, and causing traffic congestion for all other packets. In the simplest form of a routing loop, two neighboring nodes each think the other is the best next hop to a given destination. Any packet headed to that destination arriving at either node will loop between these two nodes. Routing loops involving more than two nodes are also possible.

The reason for routing loops formation is simple: because each node computes its shortest-path tree and its routing table without interacting in any way with any other nodes, then if two nodes start with different maps, it is easy to have scenarios in which routing loops are created.

Using a reliable transmission protocol for exchanging routing information helps to prevent routing loops and other undesirable behaviors of a routing algorithm. The most popular practical implementation of link-state routing is *Open Shortest Path First* (OSPF) protocol, reviewed in Section 9.2.2.

1.4.3 Distance Vector Routing

Problems related to this section: Problem 1.27 → Problem 1.31

The key idea of the distance vector routing algorithm is that each node assumes that its neighbors already know the shortest path to each destination node. The node then selects the neighbor for which the overall distance (from the source node to its neighbor, plus from the neighbor to the destination) is minimal. The process is repeated iteratively until all nodes settle to a stable solution. This algorithm is also known by the names of its inventors as *Bellman-Ford algorithm*. Figure 1-46 illustrates the process of finding the shortest path in a network using Bellman-Ford algorithm. The straight lines indicate single links connecting the neighboring nodes. The wiggly lines indicate the shortest paths between the two end nodes (other nodes along these paths are not shown). The bold line indicates the overall shortest path from source to destination.

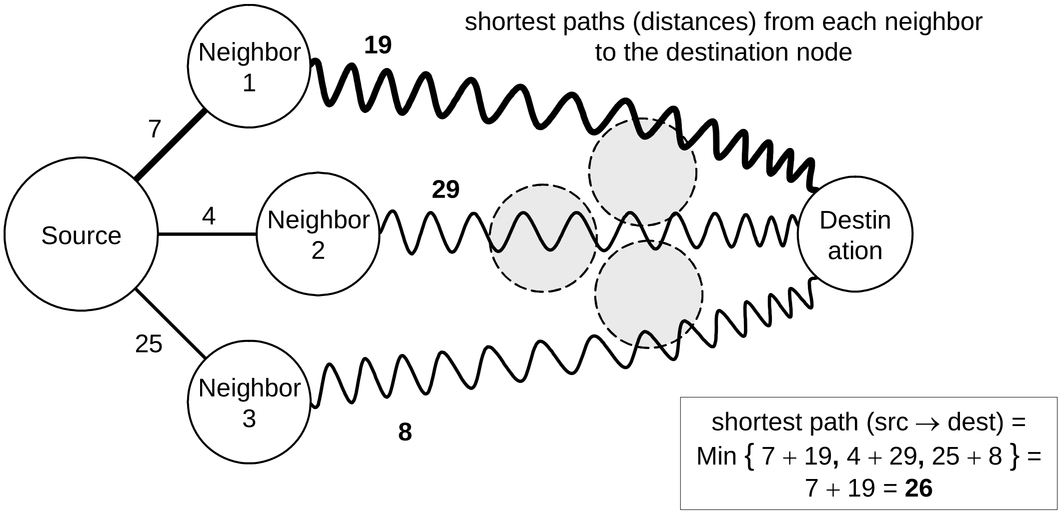

shortest paths (distances) from each neighbor
to the destination node
Neighbor 1
Neighbor 2
Neighbor 3
Source
Destin ation
7
4
25
19
29
8
shortest path (src → dest) =
Min { 7 + 19, 4 + 29, 25 + 8 } =
7 + 19 = 26

Figure 1-46: Illustration of finding the shortest path using Bellman-Ford algorithm. The thick line (crossing Neighbor 1) represents the shortest path from Source to Destination.

Next, we describe the *distance vector routing algorithm*. Let ***N*** denote the set of all nodes in a network, $c(A, B)$ denote the cost of the link connecting the nodes A and B, and $D_X(Y)$ denote the distance (shortest path) from node X to node Y. The **distance vector** of node X is the vector of distances from node X to all other nodes in the network, denoted as $DV(X) = \{D_X(Y);\ Y \in \boldsymbol{N}\}$.

Let $\eta(X)$ symbolize the set of neighboring nodes of node X. For example, in Figure 1-44 $\eta(A) = \{B, C\}$ because B and C are the only nodes directly linked to node A. The distance vector routing algorithm runs at every node X and calculates the distance to every other node $Y \in \boldsymbol{N}$, $Y \neq X$, using the following formula:

$$D_X(Y) = \min_{V \in \eta(X)} \{c(X,V) + D_V(Y)\} \tag{1.14}$$

To apply this formula, every node must receive the distance vector from all other nodes in the network. Every node maintains a *table of distance vectors*, which includes its own distance vector and distance vectors of its neighbors. Initially, the node assumes that the distance vectors of its neighbors are filled with infinite elements. Here is an example:

Example 1.4 Distributed Distance Vector Routing Algorithm

Consider the original network in Figure 1-44(a) and assume that it uses the distributed distance vector routing algorithm. Starting from the initial state for all nodes, show the first few steps until the routing algorithm reaches a stable state. For node A, the routing table initially looks as follows.

Routing table at node *A*: Initial

Distance to

From	A	B	C
A	0	10	1
B	∞	∞	∞
C	∞	∞	∞

+

Received Distance Vectors

	A	B	C	D
From B	10	0	1	1

	A	B	C	D
From C	1	1	0	7

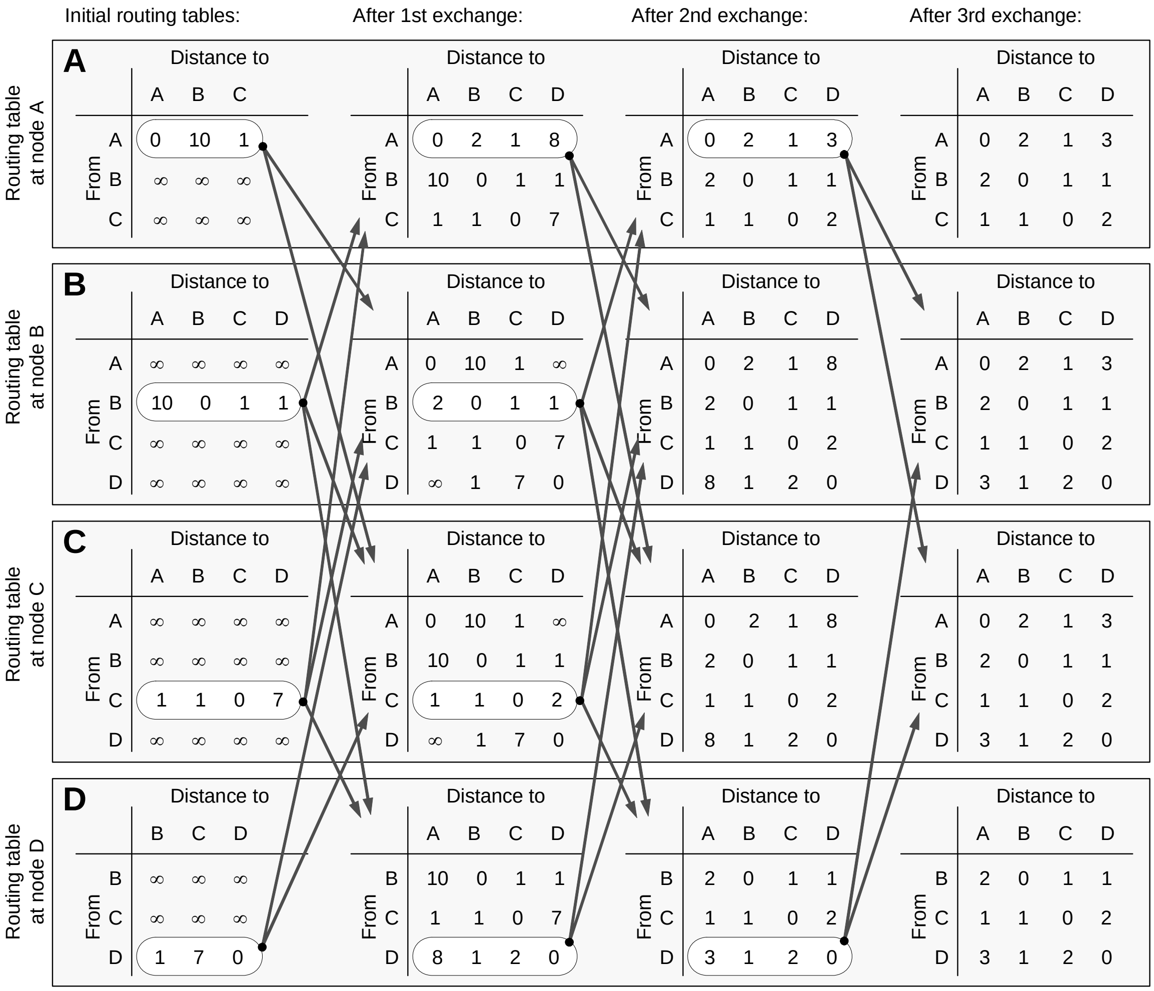
=

Routing table at node *A*: After 1st exchange

Distance to

From	A	B	C	D
A	0	2	1	8
B	10	0	1	1
C	1	1	0	7

Note that node A only keeps the distance vectors of its immediate neighbors, B and C, and not that of any other nodes, such as D. Initially, A may not even know that D exists. Next, each node sends its



Initial routing tables: After 1st exchange: After 2nd exchange: After 3rd exchange:

A Routing table at node A

Initial routing tables:

From	A	B	C
A	0	10	1
B	∞	∞	∞
C	∞	∞	∞

After 1st exchange:

From	A	B	C	D
A	0	2	1	8
B	10	0	1	1
C	1	1	0	7

After 2nd exchange:

From	A	B	C	D
A	0	2	1	3
B	2	0	1	1
C	1	1	0	2

After 3rd exchange:

From	A	B	C	D
A	0	2	1	3
B	2	0	1	1
C	1	1	0	2

B Routing table at node B

Initial routing tables:

From	A	B	C	D
A	∞	∞	∞	∞
B	10	0	1	1
C	∞	∞	∞	∞
D	∞	∞	∞	∞

After 1st exchange:

From	A	B	C	D
A	0	10	1	∞
B	2	0	1	1
C	1	1	0	7
D	∞	1	7	0

After 2nd exchange:

From	A	B	C	D
A	0	2	1	8
B	2	0	1	1
C	1	1	0	2
D	8	1	2	0

After 3rd exchange:

From	A	B	C	D
A	0	2	1	3
B	2	0	1	1
C	1	1	0	2
D	3	1	2	0

C Routing table at node C

Initial routing tables:

From	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	1	1	0	7
D	∞	∞	∞	∞

After 1st exchange:

From	A	B	C	D
A	0	10	1	∞
B	10	0	1	1
C	1	1	0	2
D	∞	1	7	0

After 2nd exchange:

From	A	B	C	D
A	0	2	1	8
B	2	0	1	1
C	1	1	0	2
D	8	1	2	0

After 3rd exchange:

From	A	B	C	D
A	0	2	1	3
B	2	0	1	1
C	1	1	0	2
D	3	1	2	0

D Routing table at node D

Initial routing tables:

From	B	C	D
B	∞	∞	∞
C	∞	∞	∞
D	1	7	0

After 1st exchange:

From	A	B	C	D
B	10	0	1	1
C	1	1	0	7
D	8	1	2	0

After 2nd exchange:

From	A	B	C	D
B	2	0	1	1
C	1	1	0	2
D	3	1	2	0

After 3rd exchange:

From	A	B	C	D
B	2	0	1	1
C	1	1	0	2
D	3	1	2	0

Figure 1-47: Distance vector (DV) algorithm for the original network in Figure 1-44.

distance vector to its immediate neighbors and, as a result, *A* receives distance vectors from *B* and *C*. For simplicity, let us assume that at every node all distance vector packets arrive simultaneously. Of course, this is not the case in reality, but asynchronous arrivals of routing packets do not affect the algorithm operation. When a node receives an updated distance vector from its neighbor, the node overwrites the neighbor's old distance vector in its routing table with the new one. As shown in the figure above, *A* overwrites the initial distance vectors for *B* and *C*. In addition, *A* re-computes its own distance vector according to Eq. (1.14), as follows:

$$D_A(B) = \min\{c(A,B) + D_B(B),\ c(A,C) + D_C(B)\} = \min\{10+0,\ 1+1\} = 2$$

$$D_A(C) = \min\{c(A,B) + D_B(C),\ c(A,C) + D_C(C)\} = \min\{10+1,\ 1+0\} = 1$$

$$D_A(D) = \min\{c(A,B) + D_B(D),\ c(A,C) + D_C(D)\} = \min\{10+1,\ 1+7\} = 8$$

The new values for *A*'s distance vector are shown in the rightmost table in the above figure.

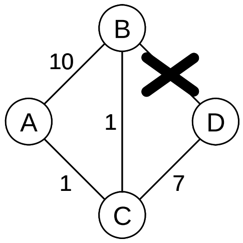
Similar computations will take place on all other nodes and the whole process is illustrated in Figure 1-47. The end result is as shown in Figure 1-47 as the second column entitled "After 1st exchange." Because for every node the newly computed distance vector is different from the previous one, Figure 1-47 shows that each node sends its new distance vector to its immediate neighbors. The cycle repeats for every node until there is no difference between the new and the previous distance vector. As shown in Figure 1-47, this happens after three exchanges.

A distance-vector routing protocol requires that each router informs its neighbors of topology changes *periodically* and, in some cases, when a change is detected in the topology of a network (*triggered updates*). Routers can detect link failures by periodically testing their links with “heartbeat” or HELLO packets (also called “keep alive”). However, if the router crashes, then it has no way of notifying neighbors of a change. Therefore, distance vector protocols must make some provision for *timing out routes* when periodic routing updates are missing for the last few update cycles. This is the task for *Invalid-Route Timer* (see Problem 1.29 for more details).

Compared to link-state protocols, which require a router to inform all the other nodes in its network about topology changes, distance-vector routing protocols have less computational complexity and message overhead (because each node informs only its own neighbors).

Limitations: Routing Loops and Counting-to-Infinity

Distance vector routing works well if nodes and links are always up, but it suffers from several problems when links fail and become restored. The problems happen because the node does not reveal the information it used to compute its distance vector when it distributes the vector to the neighbors. As a result, remote routers do not have sufficient information to determine whether their choice of the next hop will cause routing loops to form. Although reports about lowering link costs (good news) are adopted quickly, reports about increased link costs (bad news) only spread in slow increments. This problem is known as the “counting-to-infinity problem.”



Consider Scenario 2 in Figure 1-44(c) reproduced here in the figure on the right, where after the network stabilizes, the link *BD* fails. Before the failure, the distance vector of the node *B* will be as shown in the figure below. After *B* detects the link failure, it sets its own distance to *D* as ∞. (Note that *B* cannot use old distance vectors it obtained earlier from its neighbors to recompute its new distance vector, because it does not know if they are valid anymore.) If *B* sends immediately its new distance vector to *C*,[10] *C* would figure out that *D* is unreachable, because its previous best path led via *B* and now it became unavailable. However, it may happen that *C* just sent its periodic update (unchanged from before the link *BD* failure) to *B* and *B* receives it after discovering the failure of *BD* but before sending out its own update. Node *B* then recomputes its new distance to node *D* as

$$D_B(D) = \min\{c(B,A) + D_A(D),\ c(B,C) + D_C(D)\} = \min\{10+3,\ 1+2\} = 3$$

and *B* chooses *C* as the next hop to *D*. Because we humans can see the entire network topology, we know that *C* has the distance to *D* equal to 2 going via the link *BC* followed by the link *CD*. However, because *C* received from *B* only the numeric values of the distances, not the paths over which these distances are computed, *C* does *not* know that itself lays on *B*’s shortest path to *D*!

[10] Node *B* will also notify its neighbor *A*, but for the moment we ignore *A* because *A*’s path to *D* goes via *C*, and on, via *B*. Hence, *A* will not be directly affected by this situation.

Routing table at node *B* **before** *BD* outage

	Distance to			
From	A	B	C	D
A	0	2	1	3
B	2	0	1	1
C	1	1	0	2
D	3	1	2	0

1. *B* detects *BD* outage
2. *B* sets $c(B, D) = \infty$
3. *B* recomputes its distance vector
4. *B* obtains 3 as the shortest distance to *D*, via *C*

Routing table at node *B* **after** *BD* outage

	Distance to			
From	A	B	C	D
A	0	2	1	3
B	2	0	1	**3**
C	1	1	0	2
D	3	1	2	0

Given the above routing table, when *B* receives a data packet destined to *D*, it will forward the packet to *C*. However, *C* will return the packet back to *B* because for *C*, *B* is the next hop on the shortest path from *C* to *D*. The packet will bounce back and forth between these two nodes forever (or until their forwarding tables are changed). This phenomenon is called a **routing loop**, because packets may wander aimlessly in the network, making no progress towards their destination.

Because *B*'s distance vector has changed, it reports its new distance vector to its neighbors *A* and *C* (triggered update). After receiving *B*'s new distance vector, *C* will determine that its new shortest path to *D* measures 4, via *C*. Now, because *C*'s distance vector changed, it reports its new distance vector to its neighbors, including *B*. The node *B* now recomputes its new distance vector and finds that the shortest path to *D* measures 5, via *C*. *B* and *C* keep reporting the changes until they realize that the shortest path to *D* is via *C* because *C* still has a functioning link to *D* with the cost equal to 7. This process of incremental convergence towards the correct distance is very slow compared to other route updates, causing the whole network not noticing a router or link outage for a long time, and was therefore named **counting-to-infinity problem**.

A simple solution to the counting-to-infinity problem is to use **hold-down timers**. When a node detects a link failure, it reports to its neighboring nodes that an attached network has gone down. The neighbors immediately start their hold-down timer to ensure that this route will not be mistakenly reinstated by an advertisement received from another router which has not yet learned that this route is unavailable. Until the timer elapses, the router ignores *all* updates regarding this route. Router accepts and reinstates the invalid route if it receives a new update with a better metric than its own *after* the hold-down timer has expired. At that point, the network is marked as reachable again and the routing table is updated. Typically, the hold-down timer is greater than the total convergence time, providing time for accurate information to be learned, consolidated, and propagated through the network by all routers. In case of unreliable delivery of routing advertisements, the hold-down timer should be set large enough to avoid a situation where a router assumes that a neighbor is unreachable because a periodic update is lost. The reader should check Problem 1.29 for more details.

Another solution to the counting-to-infinity problem is known as **split-horizon routing**. The key idea is that it is never useful to send information about a route back in the direction from which it came. Therefore, a router never advertises the cost of a destination to its neighbor *N*, if *N* is the next hop to that destination. The split-horizon rule helps prevent two-node routing loops. In the above example, without split horizons, *C* continues to inform *B* that it can get to *D*, but it does not say that the path goes through *B* itself. Because *B* does not have sufficient intelligence, it picks up *C*'s route as an alternative to its failed direct connection, causing a routing loop. Conversely, with split horizons, *C never* advertises the cost of reaching *D* to *B*, because *B* is *C*'s next hop to *D*. Although hold-downs should prevent counting-to-infinity and routing loops, split horizon provides extra algorithm stability.

An improvement of split-horizon routing is known as **split horizon with poisoned reverse**. Here, the router advertises its full distance vector to all neighbors. However, if a neighbor is the next hop to a given destination, then the router replaces its actual distance value with an infinite cost (meaning "destination unreachable"). In a sense, a route is "poisoned" when a router marks a route as unreachable (infinite distance). Routers receiving this advertisement assume the destination network is unreachable, causing them to look for an alternative route or remove this destination from their routing tables. In the above example, *C* would *always* advertise the cost of reaching *D* to *B* as equal to ∞, because *B* is *C*'s next hop to *D*.

In a single-path internetwork (chain-of-links configuration), split horizon with poisoned reverse has no benefit beyond split horizon. However, in a multipath internetwork, split horizon with poisoned reverse greatly reduces counting-to-infinity and routing loops. The idea is that increases in routing metrics generally indicate routing loops. Poisoned reverse updates are then sent to remove the route and place it in hold-down. Counting-to-infinity can still occur in a multipath internetwork because routes to networks can be learned from multiple sources. None of the above methods works well in general cases. The core problem is that when *X* tells *Y* that it has a path to somewhere, *Y* has no way of knowing whether it itself is on the path.

The most popular practical implementation of link-state routing is *Routing Information Protocol* (RIP), reviewed in Section 9.2.1.

1.4.4 IPv4 Address Structure and CIDR

Problems related to this section: Problem 1.32 → Problem 1.34

Section 1.4.1 briefly mentions that the structure of network addresses should be designed to assist with message routing along the path to the destination. Why and how may not be obvious at first, so let us consider again the analogy between a router and a crossroads (Figure 1-37). Suppose you are driving from Philadelphia to Bloomfield, New Jersey. If the sign on a road intersection contained all small towns in all directions, you can imagine that it would be very difficult to build and use such "forwarding tables" (Figure 1-48). The intersections would be congested by cars looking-up the long table and trying to figure out which way to exit out of the intersection. In reality, the problem is solved by listing only the major city names on the signs. Note that in this case "New York" represents the *entire region* around the city, including all small towns in the region. That is, you do not need to pass through New York to reach Bloomfield, New Jersey. On your way, as you are approaching New York, at some point there will be another crossroads with a sign for Bloomfield. Therefore, *hierarchical address structure* gives a hint about the location that can be used to simplify routing.

Large computer networks, such as the Internet, encounter a similar problem with building and using forwarding tables. The solution has been to divide the network address into two parts: a fixed-length "region" portion (in the most significant bits) and an "intra-region" address (the remainder of the address). These two parts combined represent the actual network address. In this model, forwarding is simple: The router first looks at the "region" part of the destination address.

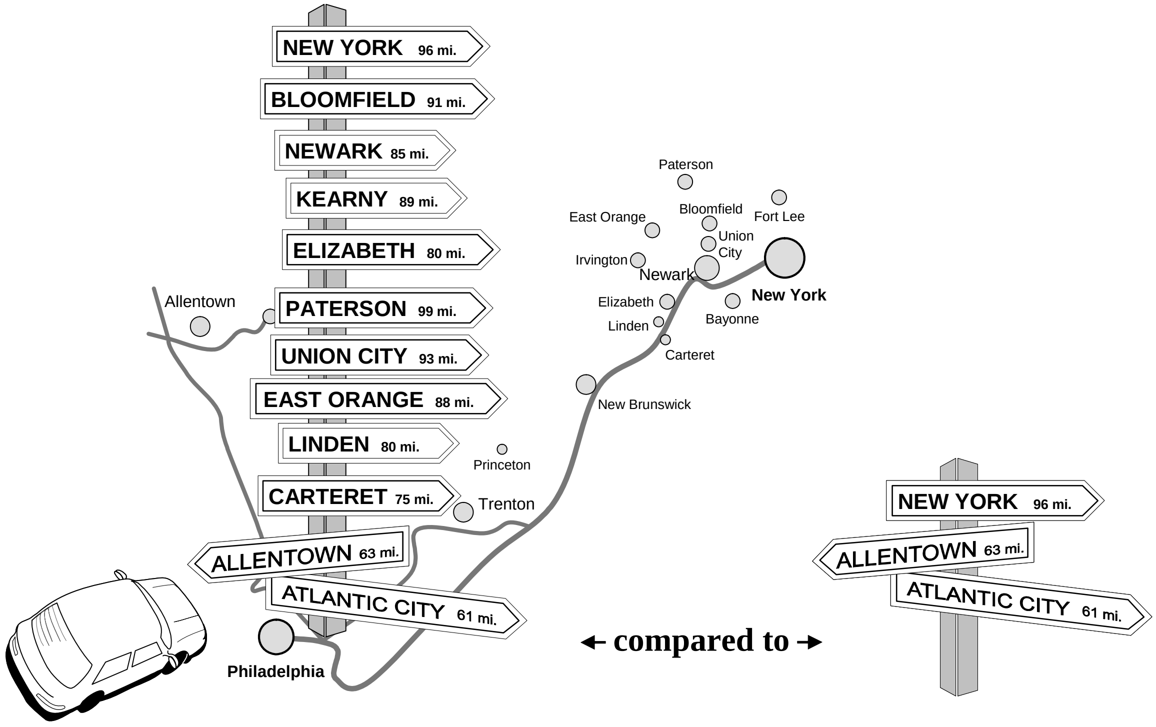


Figure 1-48: Illustration of the problem with the forwarding-table size. Real world road signs contain only a few destinations (lower right corner) to keep them manageable.

If the destination address *is* in this router's region, it looks up the forwarding table using the "intra-region" portion and forwards the packet on. Conversely, if it sees a packet with the destination address *not* in this router's region, it does a lookup on the "region" portion of the address and forwards the packet onwards. This structuring of network-layer addresses dramatically reduces the size of the forwarding tables. The amount of data in the forwarding table for routes outside the router's region is at most equal to the number of regions in the entire network, typically much smaller than the total number of possible addresses.

Note that address-structuring techniques described in this section are used only by routers to minimize the amount of routing information maintained in the memory, and to speed up packet forwarding. End hosts just use IP addresses as given to them and are not aware of any address structuring. Routers, on the other hand, now keep a single entry for a subset of addresses (see analogy in Figure 1-48), instead of having a forwarding-table entry for every individual address.

The idea of hierarchical structuring can be extended to a multi-level hierarchy, starting with individual nodes at level 0 and covering increasingly larger regions at higher levels of the addressing hierarchy. In such a network, as a packet approaches its destination, it would be forwarded more and more precisely until it reaches the destination node. The key issues in designing such hierarchical structure for network addresses include:

- Should the hierarchy be *uniform*, for example so that a region at level i+1 contains twice as many addresses as a region at level i. In other words, what is the best granularity for quantizing the address space at different levels, and should the hierarchy follow a regular or irregular pattern?

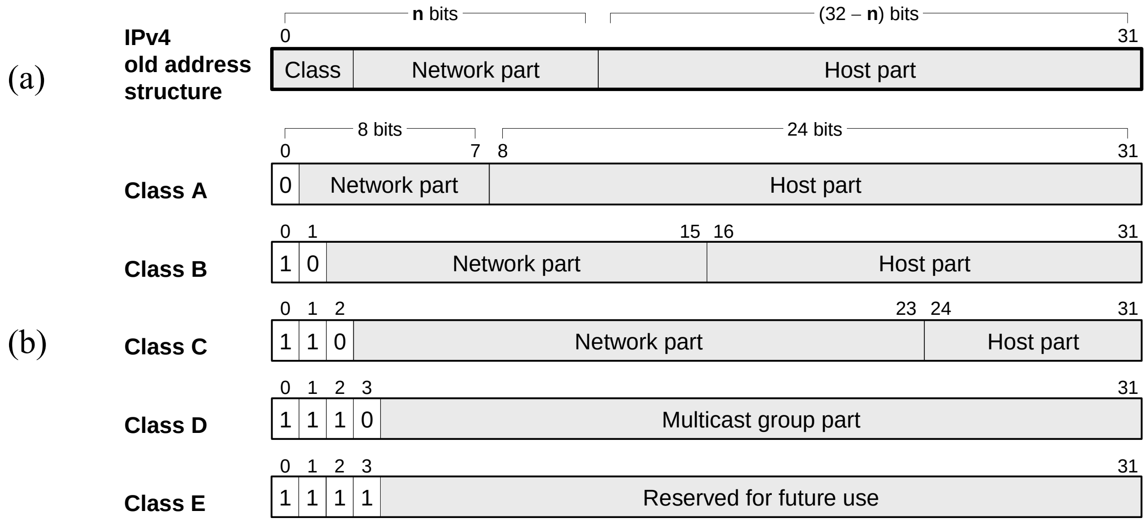


Figure 1-49: (a) Class-based structure of IPv4 addresses (deprecated). (b) The structure of the individual address classes.

- Should the hierarchy be *statically defined* or could it be *dynamically adaptive*? In other words, should every organization be placed at the same level regardless of how many network nodes it manages? If different-size organizations are assigned to different levels, what happens if an organization outgrows its original level or merges with another organization? Should organization's hierarchy (number of levels and nodes per level) remain forever fixed once it is designed?

The original solution for structuring IPv4 addresses (standardized with RFC-791 in 1981) decided to follow a uniform pattern for structuring the network addresses and opted for a statically defined hierarchy. IPv4 addresses were standardized to be 32-bits long, which gives a total of 2^{32} = 4,294,967,296 possible network addresses. At that time, the addresses were grouped into four classes, each class covering different number of addresses. In computer networks, "regions" correspond to sub-networks, or simply networks, within an internetwork (Section 1.4.1). Depending on the class, the first several bits correspond to the "network" identifier and the remaining bits to the "host identifier" (Figure 1-49). Class A addresses start with a binary "0" and have the next 7 bits for network number and the last 24 bits for host number. Class B addresses start with binary "10", use the next 14 bits for network number, and the last 16 bits for host number. For example, Rutgers University IP addresses belong to Class B because the network part starts with bits `10` (Figure 1-40), so the network part of the address is: `10000000 00000110` or `128.6.*` in dotted-decimal notation. Class C addresses start with binary "110" and have the next 21 bits for network number, and the last 8 bits for host number. A special class of addresses is Class D, which are used for IP multicast (described in Section 3.3.2). They start with binary "1110" and use the next 28 bits for the group address. Multicast routing is described later in Section 3.3.2. Addresses that start with binary "1111" are reserved for experiments.

The router-forwarding task in IPv4 is a two-step process. First, for every received packet, the router examines its destination address and determines whether it belongs to the same region as this router's addresses. If so, it looks for an exact match; otherwise, it performs a fixed-length lookup depending on the address's class.

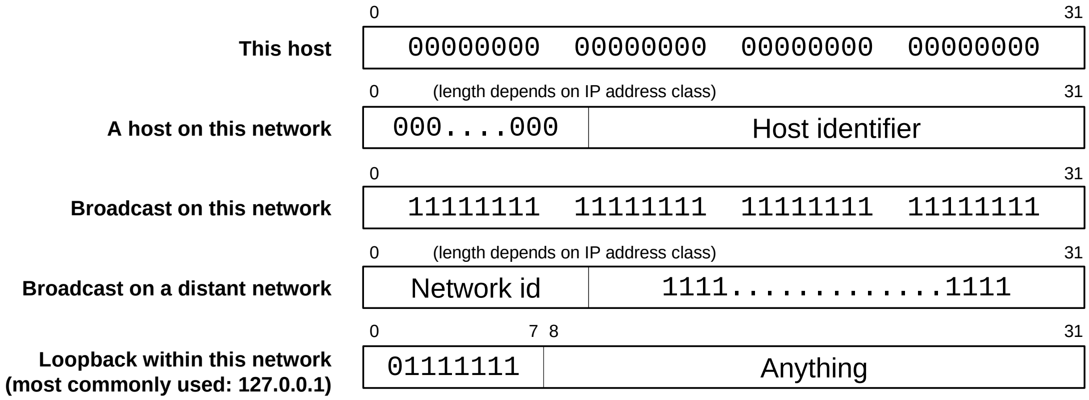


Figure 1-50: Special IP version 4 addresses.

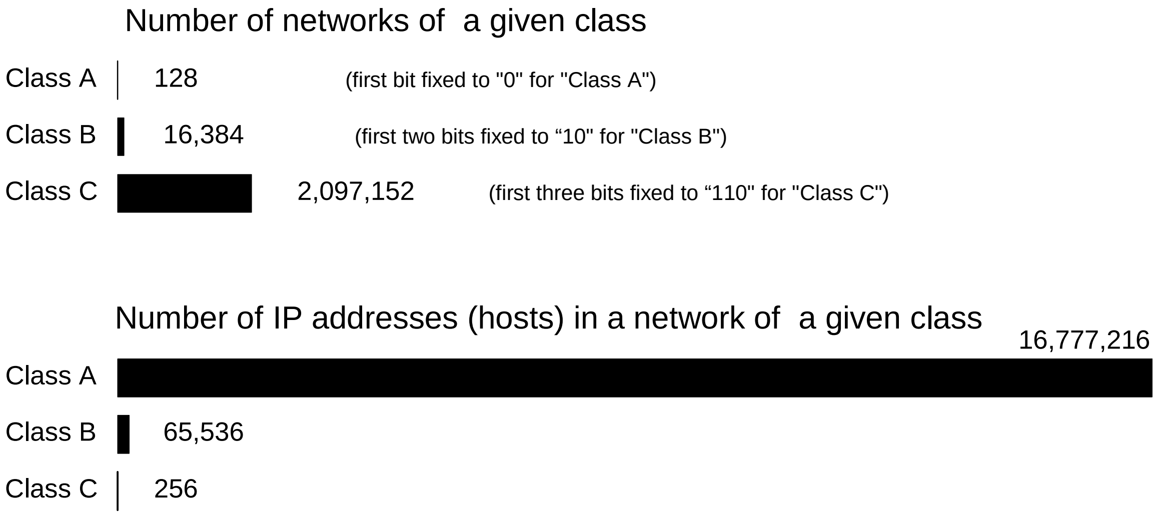


Figure 1-51: Proportions of networks in different classes (top rows) and hosts within a network of a given class (bottom rows).

Figure 1-50 lists special IPv4 addresses.

The address space has been managed by the Internet Engineering Task Force (IETF) and organizations requested and obtained a set of addresses belonging to a class. The original partitioning of IPv4 address space created regions of three sizes: Class A networks had a large number of addresses, 2^{24} = 16,777,216, Class B networks had 2^{16} = 65,536 addresses each, and Class C networks had only 2^{8} = 256 addresses each. Figure 1-51 illustrates the size proportions of networks in different classes and hosts within each network of a given class. An organization would request a block of addresses and the IETF would assign a block of addresses by specifying the first n bits of the IP address (network part), where n equaled 8, 16 or 24 bits, depending on the address class (Figure 1-49). The receiving organization would finally decide on how to assign the IP addresses to individual computers, so that all individual addresses share the same first n bits.

As the Internet grew, most organizations were assigned Class B addresses, because their networks were too large for a Class C address, but not large enough for a Class A address. Unfortunately, large part of the address space went unused. For example, if an organization had slightly more than 256 hosts and acquired a Class B address, more than 65,000 addresses may go unused and could not be assigned to another organization.

SIDEBAR 1.3: Hierarchy without Topological Aggregation

♦ There are different ways to organize addresses hierarchically. Internet addresses are aggregated *topologically*, so that addresses in the same physical subnetwork share the same address prefix (or suffix). Another option is to partition the address space by manufacturers of networking equipment. The addresses are still globally unique, but not aggregated by proximity (i.e., network topology). An example is the Ethernet link-layer address, described in Section 1.5.2. Each Ethernet attachment adaptor has assigned a globally unique address, which has two parts: a part representing the manufacturer's code, and a part for the adaptor number. The manufacturer code is assigned by a global authority, and the adaptor number is assigned by the manufacturer. Obviously, Ethernet adaptors on a given subnetwork may be from different manufacturers, and unrelated subnetworks may have adaptors from the same manufacturer. However, this type of hierarchy is not suitable for routing purposes because it does not scale to networks with tens of millions of hosts, such as the Internet. Ethernet addresses cannot be aggregated in routing tables, and large-scale networks cannot use Ethernet addresses to identify destinations. Equally important, Ethernet addresses cannot be *summarized* and exchanged by the routers participating in the routing protocols. Therefore, topological aggregation of network addresses is the fundamental reason for the scalability of the Internet's network layer. (See more discussion in Section 9.3.1.)

CIDR Scheme for Internet Protocol (IPv4) Addresses

By 1991, it became clear that the 2^{14} = 16,384 Class B addresses would soon run out and a different approach was needed. At the same time, addresses from the enormous Class C space (2^{21} blocks) were rarely allocated. The solution was proposed to assign new organizations contiguous subsets of Class C addresses instead of a single Class B address. This allowed for a refined granularity of address space assignment. In this way, the allocated set of Class C addresses could be much better matched to the organization needs than with whole Class B sets. This solution *optimizes the common case*. The common case is that most organizations require at most a few thousand addresses, and this need could not be met with individual Class C sets, while an entire Class B represented a too coarse match to the need. A middle-road solution was needed. Although initially targeting class C addresses, ultimately the entire class-based addressing was replaced with a classless approach.

Routing protocols that structure IP address so that the network part of addresses can be arbitrarily long instead of class-based are said to follow **Classless Interdomain Routing** or **CIDR** (pronounced "cider"). The CIDR-based addressing works as follows. An organization is assigned a region of the address space defined by two numbers, A and m. The assigned address region is denoted A/m. A is called the **prefix** and it is a 32-bit number (often written in dotted decimal notation) denoting the address space, while m is called the **mask** (or **subnet mask**) and it is a decimal number between 1 and 32. The subnet mask m can be written in the binary form as m ones followed by $32 - m$ zeroes. When a network is assigned A/m, it means that it includes $2^{(32-m)}$ addresses, all sharing the first m bits of A. For example, the network "192.206.0.0/21" corresponds to the $2^{(32-21)}$ = 2048 addresses in the range from 192.206.0.0 to 192.206.7.255. The subnet mask in binary form is 11111111 11111111 11111000 00000000. The organization may

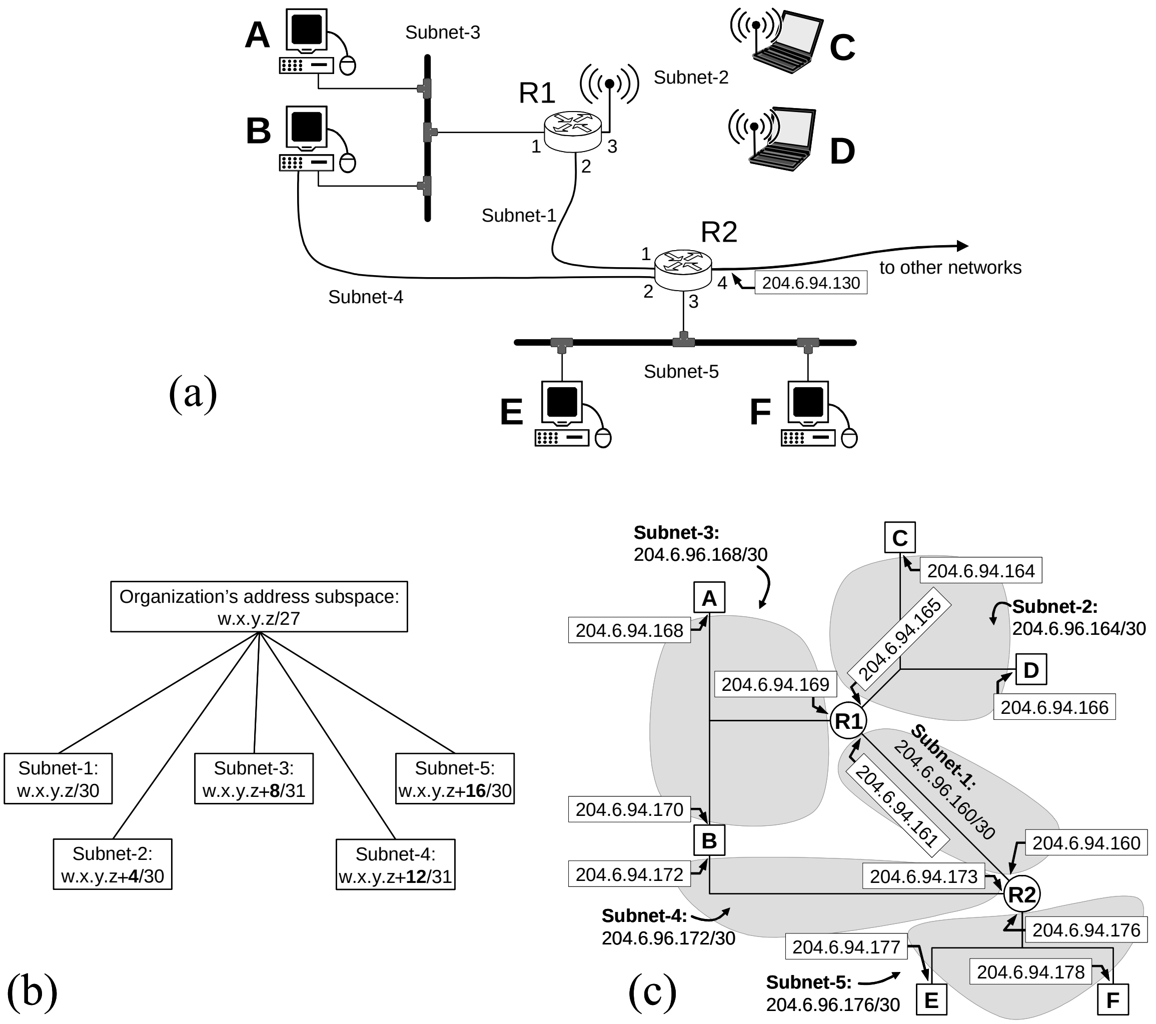


Figure 1-52: (a) Example internetwork with five physical networks reproduced from Figure 1-38. (b) Desired hierarchical address assignment under the CIDR scheme. (c) Example of an actual address assignment.

further structure its address region (known as "intranet") and subdivide it into different *A*/*n* sub-regions, where *n*>*m*.

As with any address structuring, only routers use CIDR and end hosts are not aware of CIDR or any other (e.g., class-based) address structuring. Routers use CIDR prefixes in control message advertisements to advertise destination addresses (instead of full IP addresses). Details are available in Section 9.2. On the other hand, regular data packets used for end-user communication carry only the full IP address to identify the packet destination, and never carry any CIDR prefixes.

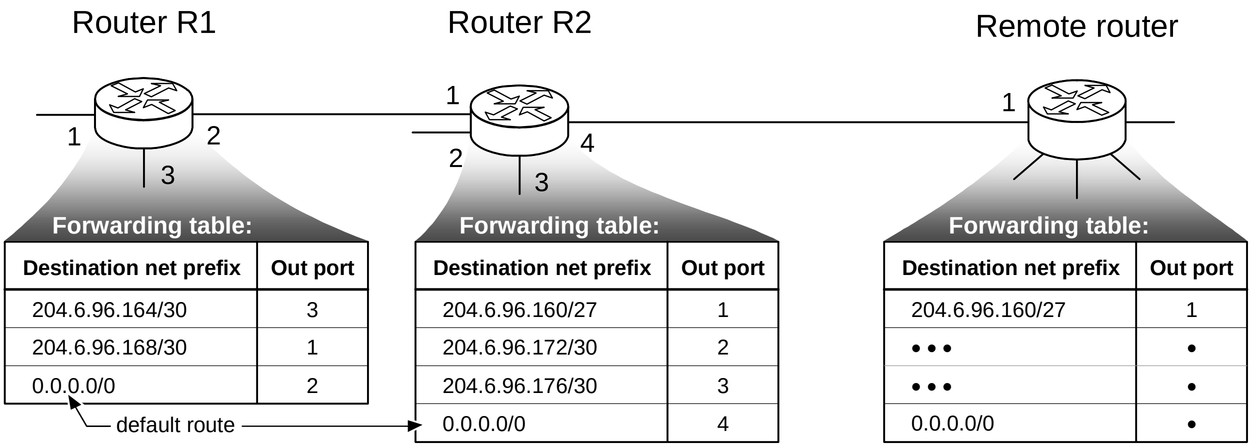
Suppose for example that you are administering your organization's network as shown in Figure 1-38, reproduced here in Figure 1-52(a). Assume that you know that this network will remain fixed in size, and your task is to acquire a set of network addresses and assign them optimally to the hosts. Assume that router R2 is connected to the rest of the world through interface 4 and this interface was previously assigned the IP address 204.6.94.130.

Table 1-3: CIDR address assignment for the internetwork in Figure 1-52.

Subnet	Network prefix	Binary representation	Interface addresses
1	204.6.94.160/30	11001100 00000110 01011110 101**000**--	R2-1: 204.6.94.160
			R1-2: 204.6.94.161
			(unused)
			b-cast: 204.6.94.163
2	204.6.94.164/30	11001100 00000110 01011110 101**001**--	C: 204.6.94.164
			R1-3: 204.6.94.165
			D: 204.6.94.166
			b-cast: 204.6.94.167
3	204.6.94.168/30	11001100 00000110 01011110 101**010**--	A: 204.6.94.168
			R1-1: 204.6.94.169
			B-1: 204.6.94.170
			b-cast: 204.6.94.171
4	204.6.94.172/30	11001100 00000110 01011110 101**011**--	R2-2: 204.6.94.172
			B-2: 204.6.94.173
			(unused)
			b-cast: 204.6.94.175
5	204.6.94.176/30	11001100 00000110 01011110 101**100**--	R2-3: 204.6.94.176
			E: 204.6.94.177
			F: 204.6.94.178
			b-cast: 204.6.94.179

Your first task is to determine how many addresses to request. As seen in Section 1.4.1, both routers R1 and R2 have 3 network interfaces each. Because your internetwork has a total of 13 interfaces (3 + 3 for routers, 2 for host B and 5 × 1 for other hosts), you need 13 unique IP addresses. In addition, you need one address for broadcast to be used by other networks (labeled "Broadcast on a distant network" in Figure 1-50). However, you would like to structure your organization's network hierarchically, so that each subnet is in its own address space, as shown in Figure 1-52(b). In Figure 1-50, subnets 1 and 4 have only two interfaces each and, with their broadcast address, they need 3 addresses each. Their assignments will have the mask $m = 30$ because $2^{(32-30)} = 4$, which is ≥ 3, and one address in each subnet will be unused. Similarly, subnets 2, 3, and 5 also need 2 bits (4 addresses) each, because they each have three interfaces and one broadcast address. Therefore, you need $5 \times 4 = 20$ addresses of which two will be unused. The next integer power of 2 is $32 = 2^5$ and your address region will be of the form *w.x.y.z*/27, which gives you $2^{(32-27)} = 2^5 = 32$ addresses of which 14 will be unused. Let us assume that the actual address subspace assignment that you acquired is 204.6.94.160/27, which corresponds to the binary prefix `11001100 00000110 01011110 101-----`. Then you could assign the individual addresses to the network interfaces as shown in Table 1-3 as well as in Figure 1-52(c). Of the last 5 bits that you control, the first three will be used to identify one of the 5 subnets, which is why the network prefixes in Table 1-3 are 30 bits long. The last 2 bits will be used to identify a host within a given subnet. Given a network prefix, the process of designating some high-order bits from the host part and grouping them to form subnets is called **subnetting**.

Each entry in the forwarding table defines a route. Figure 1-53 shows the forwarding tables for routers in the example network in Figure 1-52. The forwarding table will contain at least one entry: the default route. The **default route** entry comes after all the more specific entries, and it



Router R1 forwarding table:

Destination net prefix	Out port
204.6.96.164/30	3
204.6.96.168/30	1
0.0.0.0/0	2

Router R2 forwarding table:

Destination net prefix	Out port
204.6.96.160/27	1
204.6.96.172/30	2
204.6.96.176/30	3
0.0.0.0/0	4

Remote router forwarding table:

Destination net prefix	Out port
204.6.96.160/27	1
• • •	•
• • •	•
0.0.0.0/0	•

Figure 1-53: Forwarding tables of routers in Figure 1-52.

matches anything that failed to match a specific prefix. The default route network ID is 0.0.0.0 with the network mask of 0.0.0.0. Any destination IP address joined with 0.0.0.0 by a logical AND results in 0.0.0.0. Therefore, for any IP address, the default route produces a match. This route typically forwards the datagram to the "default gateway" for the local subnet. The default gateway is important because it is generally not feasible for all nodes to maintain knowledge of the routes to all other networks on the internetwork.

In the example in Figure 1-53, router R1 forwards all packets with destinations that do not match the prefixes 204.6.94.164/30 and 204.6.94.168/30 to its output port 2 (leading to router R2). Similarly, router R2 forwards all packets with destinations that do not match the prefix 204.6.94.160/27 out over port 4 to the remote router. We assume that router R2 will never advertise subnets, so the remote router has a single entry with the prefix 204.6.94.160/27 for all destinations on our network. Note that if router R2 receives a packet with the destination address 204.6.94.177, there will be two matching prefixes: 204.6.94.160/27 and 204.6.94.176/30. The router always chooses the *longest prefix match*, which in our case is 204.6.94.176/30.

1.4.5 Autonomous Systems and Path Vector Routing

Problems related to this section: Problem 1.36 → Problem 1.40

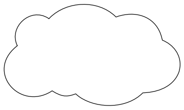
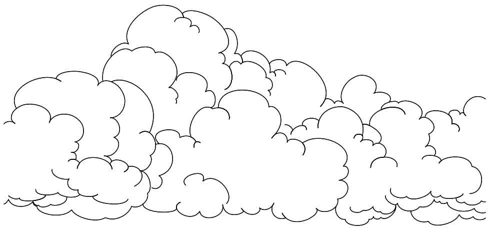


Figure 1-38 presents a naïve view of the Internet, where many hosts are mutually connected via intermediary nodes (routers or switches) that live inside the "network cloud." This would imply that the cloud is managed by a single administrative organization and all nodes cooperate to provide the best service to the customers' hosts. In reality the Internet is composed of many independent networks (or, "clouds"), managed by different organizations driven by their commercial or political interests. (The reader may also refer to Figure 1-3 for a sense of complexity of the Internet.) Each individual administrative domain is known as an **autonomous system** (AS) and different ASs may be owned and operated by different organizations or governments. Given their divergent commercial interests, these operators are more likely to compete (e.g., for profits) than to collaborate in harmony with each other.



Both distance vector and link state routing protocols have been used for *interior routing* (or, *internal routing*). That is, they have been used inside individual administrative domains or autonomous systems. However, both protocols become ineffective in large networks composed of many domains (autonomous systems). The scalability issues of both protocols were discussed earlier. In addition, they do not provide mechanisms for an administrative entity to represent its economic interests as part of the routing protocol. Economic interests can be described using *logical rules* that express the routing *policies* to reflect the economic interests. For this purpose, we need *exterior routing* (or, *external routing*) protocols for routing between different autonomous systems.

We first review the challenges posed by interacting autonomous domains and then present the path vector routing algorithm that can be used to address some of those issues.

Autonomous System Business Relationships: Peering Versus Transit

An Autonomous System (AS) can independently decide whom to exchange traffic with on the Internet, and it is not dependent upon a third party for access. Networks of Internet Service Providers (ISPs), hosting providers, telecommunications companies, multinational corporations, schools, hospitals, and even individuals can be Autonomous Systems; all one needs is a unique **Autonomous System Number (ASN)** and a block of IP addresses. A central authority (http://iana.org/) assigns ASNs and assures their uniqueness. At the time of this writing (2010), the Internet consists of over 25,000 Autonomous Systems. Most organizations and individuals do not interconnect autonomously to other networks, but connect via an ISP. One could say that an end-user is “buying transit” from their ISP.

Figure 1-54 illustrates an example of several Autonomous Systems. In order to get traffic from one end-user to another, ASs need to have an interconnection mechanism. These interconnections can be either *direct* between two networks or *indirect* via one or more intermediary networks that agree to transport the traffic. Most AS connections are indirect, because it is nearly impossible to interconnect directly with all networks on the globe. In order to make it from one end of the world to another, the traffic will often be transferred through several indirect interconnections to reach the end-user. The business agreements that allow ASs to interconnect directly and indirectly are known as “peering” or “transit,” and they are the two mechanisms that underlie the interconnection of networks that form the Internet.

A **peering** agreement (or, *swap* contract) is a voluntary interconnection of two or more autonomous systems for exchanging traffic between the customers of each AS. This is often done so that neither party pays the other for the exchanged traffic; rather, each derives revenue from its own customers. Therefore, it is also referred to as “settlement-free peering.”

In a **transit** agreement (or, *pay* contract), one autonomous system agrees to carry the traffic that flows between another autonomous system and all other ASs. Because no network connects directly to all other networks, a network that provides transit will deliver some of the traffic indirectly via one or more other transit networks. A transit provider’s routers will announce to other networks that they can carry traffic to the network that has bought transit. The transit provider receives a “transit fee” for the service. The amount of traffic carried in each direction is regulated by a service level agreement.

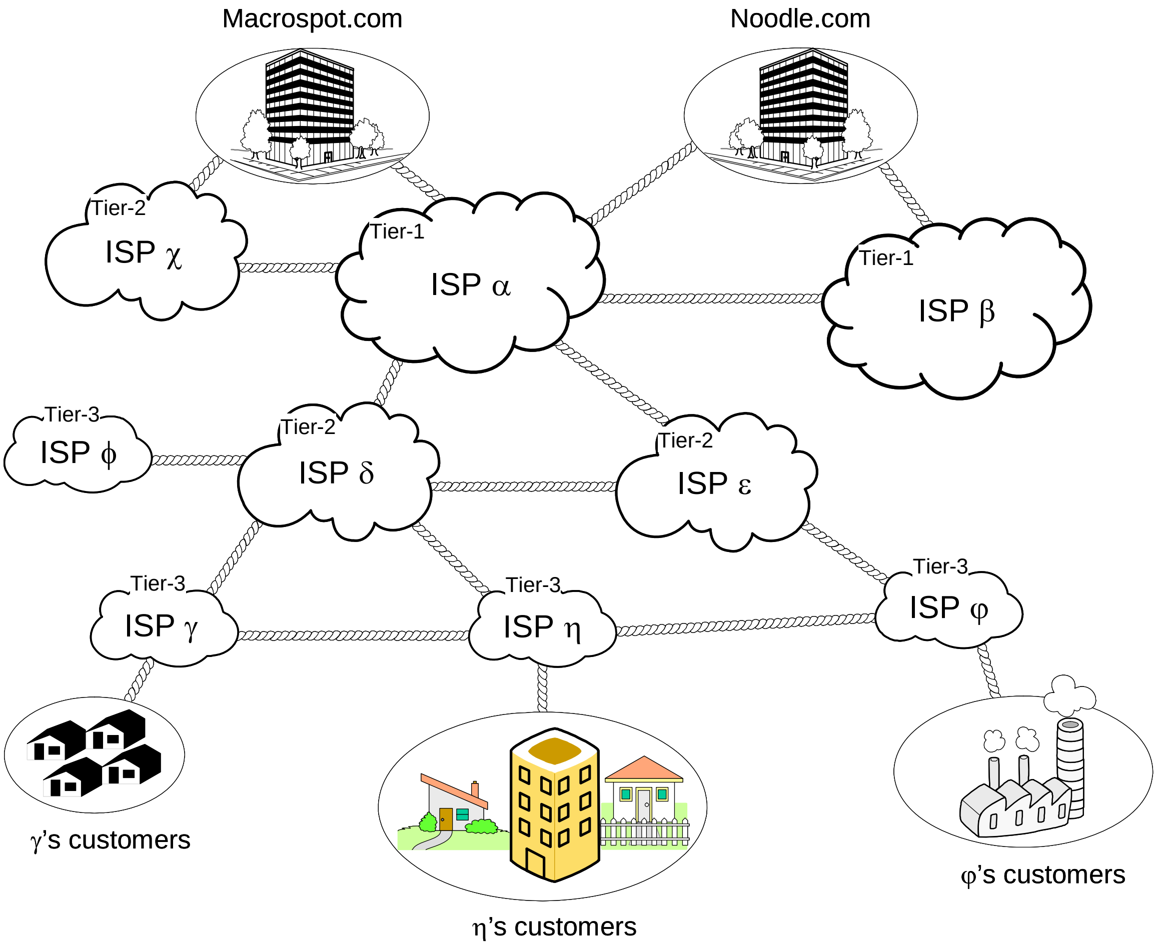


Figure 1-54: An example collection of Autonomous Systems with physical interconnections.

The transit fee is based on a reservation made up-front for a certain speed of access (in Mbps) or the amount of bandwidth used. Traffic from (upstream) and to (downstream) the network is included in the *transit fee*; when one buys 10Mbps/month from a transit provider, this includes 10 up and 10 down. The traffic can either be limited to the amount reserved, or the price can be calculated afterward (often leaving the top five percent out of the calculation to correct for aberrations). Going over a reservation may lead to a penalty.

A business agreement between ASs is implemented through (*i*) a physical interconnection of their networks, and (*ii*) an exchange of routing information through a common routing protocol. This section reviews the problems posed by autonomous administrative entities and requirements for a routing protocol between Autonomous Systems. Section 9.2.3 describes the protocol used in the current Internet, called Border Gateway Protocol (BGP), which meets these requirements.

The Internet is intended to provide *global reachability* (or, end-to-end reachability), meaning that any Internet user can reach any other Internet user as if they were on the same network. To be able to reach any other network on the Internet, Autonomous System operators work with each other in following ways (Figure 1-55):

- *Paid transit*, where one AS ("transit provider") sells transit service (or Internet access) to another AS ("transit customer"); "transit provider" in turn must also peer or pay for access.

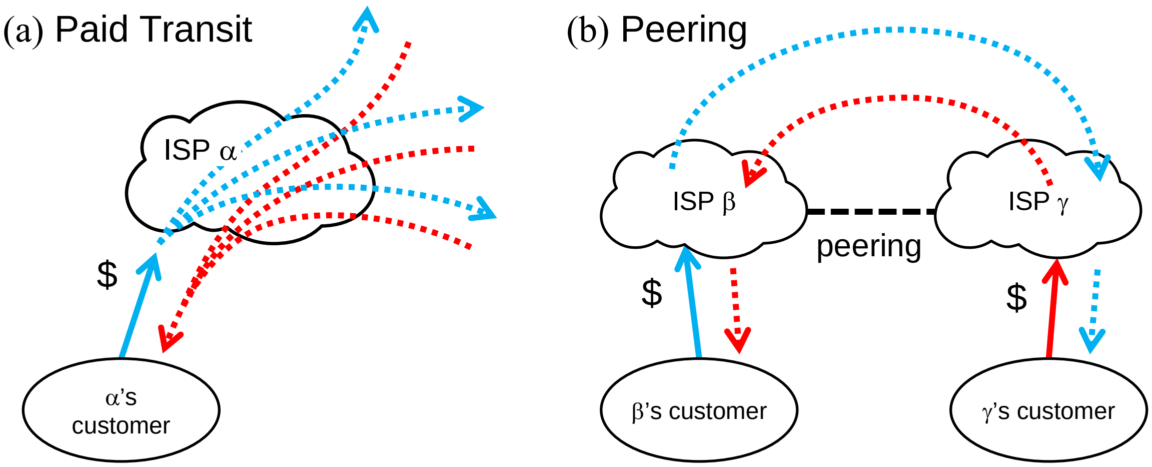


Figure 1-55: Basic types of business relationships between autonomous systems: (a) paid transit and (b) peering.

- *Peer* directly with that AS, or with an AS who sells transit service to that AS.

Therefore, any AS connected to the Internet must either pay another AS for transit, or peer with every other AS that also does not purchase transit. Figure 1-55(a) shows the paid-transit relationship. In principle, a customer pays for incoming and outgoing traffic, and expects to be able to reach all other customers or content providers on the global Internet. Permissible amounts of traffic in both directions are regulated by the *service level agreement* (SLA) between the provider and the customer. In Figure 1-55(b), ISPs β and γ agree to carry free-of-charge the traffic from each other's paying customers and not from any other peers nor from other ISPs that they may be paying for transit.

Consider the example in Figure 1-54. Tier-1 Internet Service Providers (ISPα and ISPβ) have global reachability information and can see all other networks and, because of this, their forwarding tables do not have default route entries. A default entry is usually resolved by sending the packet to a higher-level tier ISP. Tier-1 ISPs are said to be *default-free*. At present (2013) there are 14 Tier-1 ISPs in the world (http://en.wikipedia.org/wiki/Tier_1_network).

The different types of ASs (mainly by their size) lead to different business relationships between them. ISPs enter peering agreements mostly with other ISPs of the similar size (reciprocal agreements). Therefore, a Tier-1 ISP (global) would form a peering agreement with other Tier-1 ISPs, and sell transit to lower tiers ISPs. Similarly, a Tier-2 (regional or countrywide) ISP would form a peering agreement with other Tier-2 ISPs, pay for transit service to a Tier-1 ISP, and sell transit to lower Tier-3 ISPs (local). As long as the traffic ratio of the concerned ASs is not highly asymmetrical (e.g., up to 4-to-1 is a commonly accepted ratio), there is usually no financial settlement for peering.

Transit relationships are preferable because they generate revenue, whereas peering relationships usually do not. However, peering can offer reduced costs for transit services and save money for the peering parties. Other less tangible incentives ("mutual benefit") include:

- Increased redundancy (by reducing dependence on one or more transit providers) and improved performance (attempting to bypass potential bottlenecks with a "direct" path),

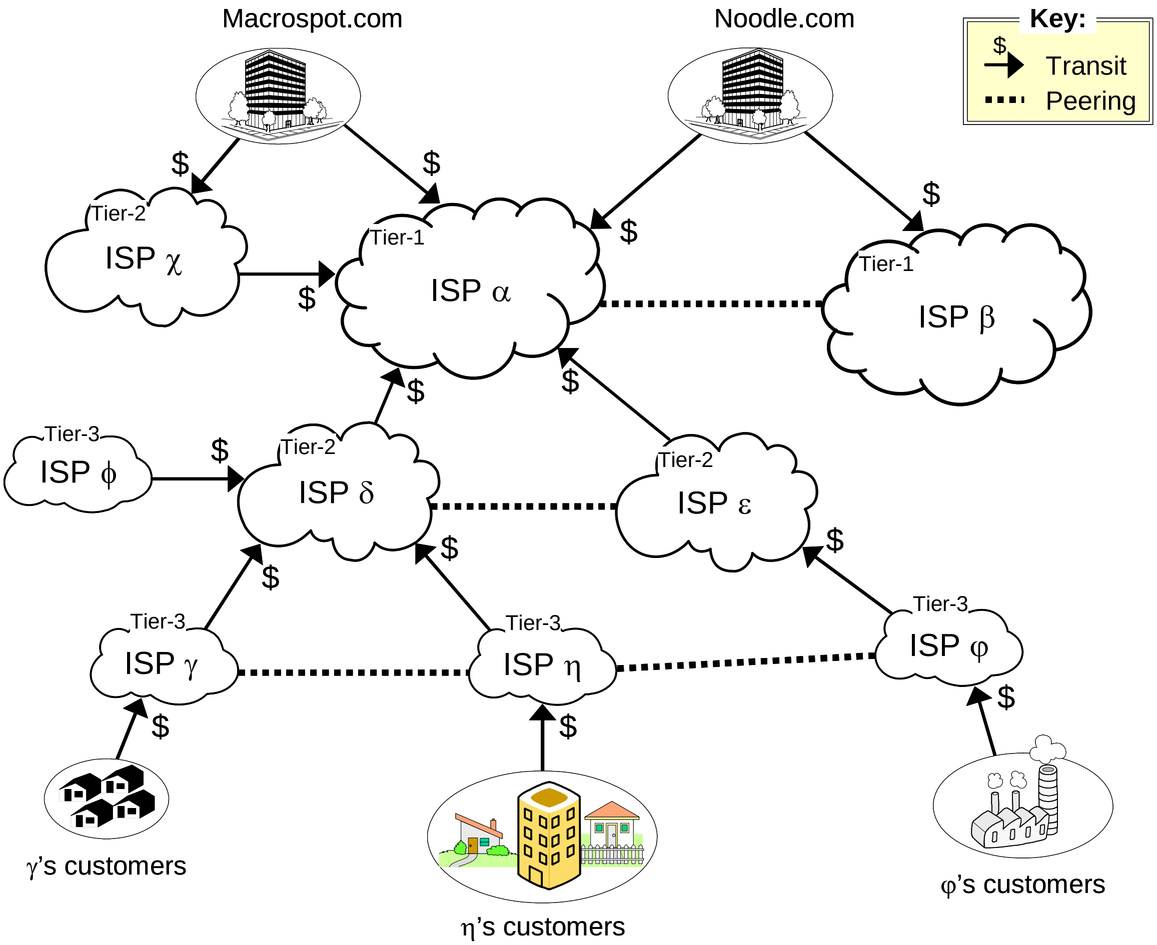


Figure 1-56: Feasible business relationships for the example ASs in Figure 1-54. Arrows only indicate the business relationships, not the flow of traffic. For example, the customers of ISP δ are ISPs ϕ, γ, and η, and ISP δ is a customer ISP α.

- Increased capacity for extremely large amounts of traffic (distributing traffic across many networks) and ease of requesting for emergency aid (from friendly peers).

Figure 1-56 shows reasonable business relationships between the ISPs in Figure 1-54. ISPϕ cannot peer with another Tier-3 ISP because it has a single physical interconnection to a Tier-2 ISPδ. An Autonomous System that has only a single connection to one other AS is called **stub AS**. The two large corporations at the top of Figure 1-56 each have connections to more than one other AS but they refuse to carry transit traffic; such an AS is called **multihomed AS**. ISPs usually have connections to more than one other AS and they are designed to carry both transit and local traffic; such an AS is called **transit AS**.

When two providers form a peering link, the traffic flowing across that link incurs a cost on the network it enters. Such a cost may be felt at the time of network provisioning: in order to meet the negotiated quantity of traffic entering through a peering link, a provider may need to increase its network capacity. A network provider may also see a cost for entering traffic on a faster timescale; when the amount of incoming traffic increases, congestion on the network increases,

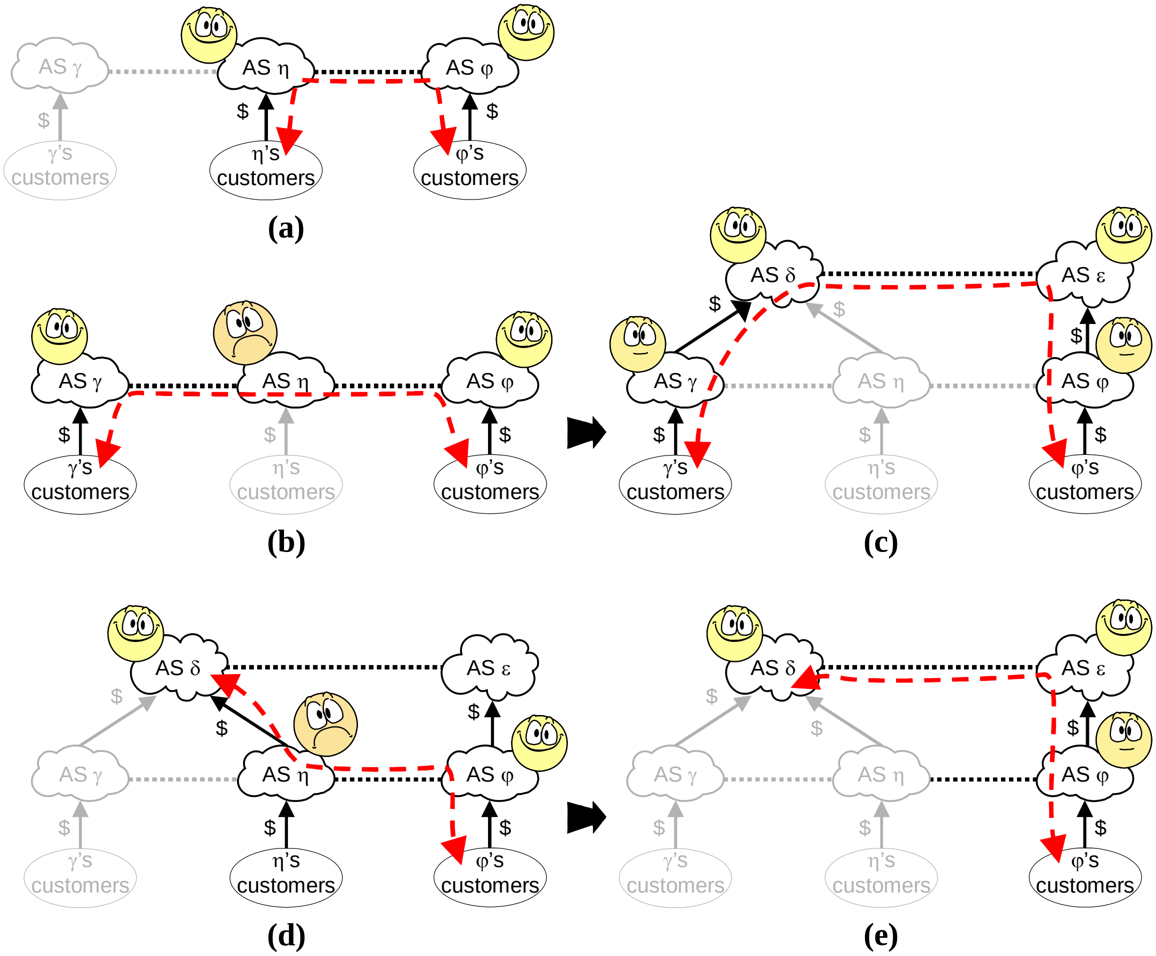


Figure 1-57: Example patterns derived from Figure 1-55 for resolving economic conflict by providing selective transit service to make or save money. Peering ASs in (a) carry traffic from each other's customers. ASγ should not use a peer ASη to reach another ASφ (b), but instead should use its transit provider ASδ (c). ASδ should not use its customer ASη to reach customers of other ASs (d), but instead should rely on peering with same-tier ASε (e).

and this leads to increased operating and network management costs. For this reason, each AS needs to decide carefully what kind of transit traffic it will support.

Each AS is in one of two types of business relationships with the ASs to which is has a direct physical interconnection: paid transit (either as provider or as customer) or peer (Figure 1-55). To its paying customers, the provider AS (i.e., ISP) wants to provide unlimited transit service. However, to its provider(s) and peers the AS probably wishes to provide a *selective transit service*. Figure 1-57 gives examples of how conflicting business interests of different parties can be resolved. To understand this figure, the reader should refer to the basic business relationships in Figure 1-55. The guiding principle is that ASs will want to avoid highly asymmetrical relationships without reciprocity. In Figure 1-57(a), both ASη and ASφ benefit from peering because it helps them to provide global reachability to their own customers. In Figure 1-57(b), ASγ and ASφ benefit from using transit service of ASη (with whom both of them are peers), but ASη may lose money in this arrangement (because of degraded service to its own customers) without gaining any benefit. Therefore, ASη should refuse to carry transit traffic between its peers. An appropriate solution is presented in Figure 1-57(c), where ASγ and ASφ use their transit providers (ASδ and ASε, respectively), to carry their mutual transit traffic. This scenario is equivalent to Figure 1-55(b). ASδ and ASε are peers and are happy to provide transit service to their transit customers (ASγ and ASφ). Figure 1-57(d) shows a scenario where higher-tier ASδ uses its transit customer ASη to gain reachability of ASφ. Again, ASη does not benefit from this arrangement, because it pays ASδ for transit and does not expect ASδ in return to use its transit service for free. The appropriate solution is shown in Figure 1-57(e), which is again equivalent to Figure 1-55(b).

To implement these economic decisions and prevent unfavorable arrangements, ASs design and enforce *routing policies*. An AS that wants to avoid providing transit between two neighboring ASs, simply does not advertise to either neighbor that the other can be reached via this AS. The neighbors will not be able to "see" each other via this AS, but via some other ASs. Routing policies for selective transit can be summarized as:

- To its transit customers, the AS should make visible (or, reachable) all destinations that it knows of. That is, all routing advertisements received by this AS should be passed on to its own transit customers;
- To its peers, the AS should make visible only its own transit customers, but not its other peers or its transit provider(s), to avoid providing unrecompensed transit;
- To its transit providers, the AS should make visible only its own transit customers, but not its peers or its other transit providers, to avoid providing unrecompensed transit.

In the example in Figure 1-56, Tier-1 ISPs (ASα and ASβ) can see all the networks because they peer with one another and all other ASs buy transit from them. ASγ can see ASη and its customers directly, but not ASφ through ASη. ASδ can see ASφ through its peer ASε, but not via its transit customer ASη. Traffic from ASφ to ASϕ will go trough ASε (and its peer ASδ), but not through ASη.

To illustrate how routers in these ASs implement the above economic policies, let us imagine example routers as in Figure 1-58. Suppose that a router in ASϕ sends an update message advertising the destination prefix `128.34.10.0/24`. The message includes the **routing path vector** describing how to reach the given destination. The path vector starts with a single AS number {ASϕ}, which is the AS that initiated this advertisement. A border router (router *K*) in ASδ receives this message and disseminates it to other routers in ASδ. As the advertisement packet travels over other ASs, their ASN will be prepended to the routing path vector. Routers in ASδ prepend their own AS number to the message path vector to obtain {ASδ, ASϕ} and redistribute the message to the adjacent ASs.

Before rebroadcasting a route advertisement to other ASs, an AS needs to consider what kind of business relationship it has with its neighbors. The AS may send different advertisement to different neighbors, based on the economic choices in Figure 1-57. For example, because ASη does not have economic incentive to advertise a path to ASϕ to its peer ASφ, it sends an update message with path vector containing only the information about ASη's customers. Therefore, ASφ will never learn that there exist physical links connecting ASη to ASδ and ASγ. On the other hand, ASε has economic incentive to advertise global reachability to its own transit customers. Therefore, routers in ASε prepend their own AS number to the routing path vector from ASδ to obtain {ASε, ASδ, ASϕ} and redistribute the update message to ASφ. Routers in ASφ update

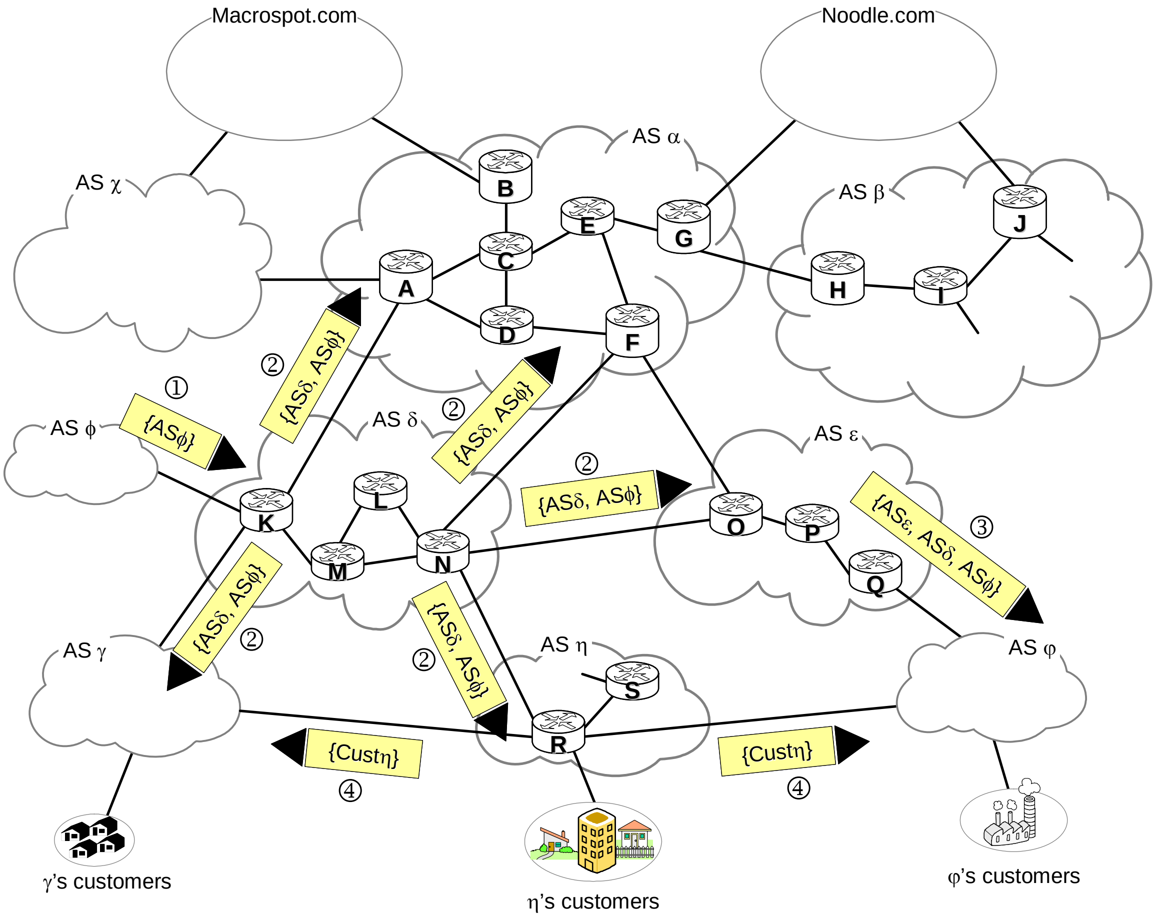


Figure 1-58: Example of routers within the ASs in Figure 1-54. Also shown is how a routing update message from ASϕ propagates to ASφ. Note that ASη does not advertise to ASφ that it has connections to δ and γ, because of the principles presented in Figure 1-57.

their routing and forwarding table entries for the prefix `128.34.10.0/24` based on the received path vector. Finally, when a router, for example in ASφ, needs to send a data packet to a destination in the subnet `128.34.10.0/24` (in ASϕ) it sends the packet first to the next hop on the path to ASϕ, which is ASε.

An important lesson here is that there may be a rich physical topology with many alternative paths connecting different autonomous systems, but those paths may never be advertised globally because of economic reasons. Most autonomous systems will have only limited knowledge about the global network connectivity.

Path Vector Routing

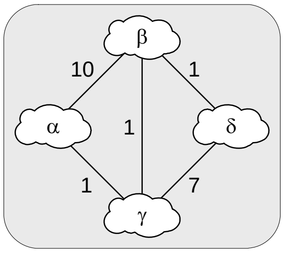
Path vector routing is used for *inter-domain* or *exterior routing* (routing between different Autonomous Systems). The path vector algorithm is somewhat similar to the distance vector algorithm (Section 1.4.3). Each border (or edge) router in a given AS advertises the destinations it can reach to its neighboring routers (in different ASs). However, instead of advertising the networks in terms of a destination address and the distance to that destination, the networks are advertised as destination addresses with path descriptions to reach those destinations. A route is defined as a pairing between a destination and the attributes of the path to that destination, thus the name, **path vector routing**. The *path vector* contains a complete path as a sequence of ASs to reach the given destination. The path vector is carried in a special path attribute that records the sequence of ASs through which the reachability message has passed. The path that contains the smallest number of ASs becomes the preferred path to reach the destination.

At predetermined times, each node advertises its own network address and a copy of its path vector down every attached link to its immediate neighbors. An example is shown in Figure 1-58, where a router in ASϕ sends a scheduled update message. After a router receives path vectors from its neighbors, it performs path selection by merging the information received from its neighbors with that already in its existing path vector. The path selection is based on some kind of path metric, similar to distance vector routing algorithm (Section 1.4.3). Again, Eq. (1.14) is applied to compute the "shortest" path. Here is an example:

Example 1.5 Path Vector Routing Algorithm

Consider the network topology in Figure 1-44(a) (reproduced below) and assume that it uses the path vector routing algorithm. Instead of router addresses, the path vector works with Autonomous System Numbers (ASNs). Starting from the initial state for all nodes, show the first few steps until the routing algorithm reaches a stable state.

The solution is similar to that for the distributed distance vector routing algorithm (Example 1.4). For AS α, the initial routing table is as in the leftmost table below. The notation ⟨*d* | χ, ξ, ζ⟩ symbolizes that the path from the AS under consideration to AS ζ is *d* units long, and χ and ξ are the ASs along the path to ζ. If the path metric simply counts the number of hops (number of ASs along the path), then the path-vector packets do not need to carry the distance *d*, because it can be determined simply by counting the ASs along the path.



Routing table at AS α: Initial

Path to

From	α	β	γ
α	$\langle 0 \mid \alpha \rangle$	$\langle 10 \mid \beta \rangle$	$\langle 1 \mid \gamma \rangle$
β	$\langle \infty \mid \rangle$	$\langle \infty \mid \rangle$	$\langle \infty \mid \rangle$
γ	$\langle \infty \mid \rangle$	$\langle \infty \mid \rangle$	$\langle \infty \mid \rangle$

+

Received Path Vectors (1st exchange)

	α	β	γ	δ
From β	$\langle 10 \mid \alpha \rangle$	$\langle 0 \mid \beta \rangle$	$\langle 1 \mid \gamma \rangle$	$\langle 1 \mid \delta \rangle$

	α	β	γ	δ
From γ	$\langle 1 \mid \gamma \rangle$	$\langle 1 \mid \beta \rangle$	$\langle 0 \mid \gamma \rangle$	$\langle 7 \mid \delta \rangle$

Routing table at node α: After 1st exchange:

=

Path to

From	α	β	γ	δ
α	$\langle 0 \mid \alpha \rangle$	$\langle 2 \mid \gamma, \beta \rangle$	$\langle 1 \mid \gamma \rangle$	$\langle 8 \mid \gamma, \delta \rangle$
β	$\langle 10 \mid \alpha \rangle$	$\langle 0 \mid \beta \rangle$	$\langle 1 \mid \gamma \rangle$	$\langle 1 \mid \delta \rangle$
γ	$\langle 1 \mid \alpha \rangle$	$\langle 1 \mid \beta \rangle$	$\langle 0 \mid \gamma \rangle$	$\langle 7 \mid \delta \rangle$

Again, AS α only keeps the path vectors of its immediate neighbors, β and γ, and not that of any other ASs, such as δ. Initially, α may not even know that δ exists. Next, each AS advertises its path vector to its immediate neighbors, and α receives their path vectors. When an AS receives an updated path vector from its neighbor, the AS overwrites the neighbor's old path vector with the new one. In addition, *A* re-computes its own path vector according to Eq. (1.14), as follows:

$$D_{\alpha}(\beta) = \min\{c(\alpha,\beta) + D_{\beta}(\beta),\ c(\alpha,\gamma) + D\gamma(\beta)\} = \min\{10+0,\ 1+1\} = 2 \qquad \Rightarrow \text{path: } \langle 2 \mid \gamma, \beta \rangle$$

$$D_{\alpha}(\gamma) = \min\{c(\alpha,\beta) + D_{\beta}(\gamma),\ c(\alpha,\gamma) + D_{\gamma}(\gamma)\} = \min\{10+1,\ 1+0\} = 1 \qquad \Rightarrow \text{path: } \langle 1 \mid \gamma \rangle$$

$$D_{\alpha}(\delta) = \min\{c(\alpha,\beta) + D_{\beta}(\delta),\ c(\alpha,\gamma) + D_{\gamma}(\delta)\} = \min\{10+1,\ 1+7\} = 8 \qquad \Rightarrow \text{path: } \langle 8 \mid \gamma, \delta\rangle$$

The new values for α’s path vector are shown in the above table at the right. Note that α’s new path to β is via γ and the corresponding table entry is ⟨2 | γ, β⟩.

Similar computations will take place on all other nodes and the whole process is illustrated in [Figure XYZ]. The end result is as shown in [Figure XYZ] as column entitled “After 1st exchange.” Because for every node the path vector computed after the first exchange is different from the previous one, each node advertises its path new vector to its immediate neighbors. The cycle repeats for every node until there is no difference between the new and the previous path vector. As shown in [Figure XYZ], this happens after three exchanges.

Note that the path vector routing messages carry only the total path length *d* and the list of autonomous systems along the path. These messages do not carry costs of individual links along the path. Each node stores the received path vectors in its routing information base (RIB).

To implement routing between autonomous systems, each autonomous system must have one or more border routers that are connected to networks in two or more ASs (its own network and a neighboring AS network). Such a node is called a **speaker node** or **gateway router**. For example, in Figure 1-58 the speaker nodes in ASα are routers *A*, *B*, *F*, and *G*; in ASβ the speaker nodes are routers *H* and *J*; and in ASδ the speakers are routers *K* and *N*. A speaker node creates a routing table and advertises it to adjoining speaker nodes in the neighboring autonomous systems. The idea is the same as with distance vector routing (Section 1.4.3), except that only speaker nodes in each autonomous system can communicate with routers in other autonomous systems (i.e., speaker nodes in those ASs). The speaker node advertises the path, not the metric of the links, in its AS or other ASs. In other words, there are no weights attached to the links in a path vector, but there is an overall cost associated with each path.

Integrating Inter-Domain and Intra-Domain Routing

Administrative entities that manage different autonomous systems have different concerns for routing messages within their own autonomous system as opposed to routing messages to other autonomous systems or providing transit service for them. Within an autonomous system, the key concern is how to route data packets from the origin to the destination in the *most efficient manner*. For this purpose, *intra-domain* or *interior routing protocols*, such as those based on distance-vector routing (Section 1.4.3) or link-state routing (Section 1.4.2). These protocols are known as **Interior Gateway Protocols (IGPs)**. Unlike this, the key concern of any given autonomous system is how to route data packets from the origin to the destination in the manner that is *most profitable for this AS*. These protocols are known as **Exterior Gateway Protocols**[11] and are based on path-vector routing, described above. This duality in routing goals and solutions means that each border router (or, speaker node) will maintain two different routing tables: one obtained by the interior routing protocol and the other by the exterior routing protocol.

A key problem is how the speaker node should integrate its dual routing tables into a meaningful forwarding table. The speaker that first receives path information about a destination in another

[11] In the literature, the acronym EGP is *not* used for a generic Exterior Gateway Protocol, because EGP refers to an actual protocol, described in RFC-904, now obsolete, that preceded BGP (Section 8.2.3).

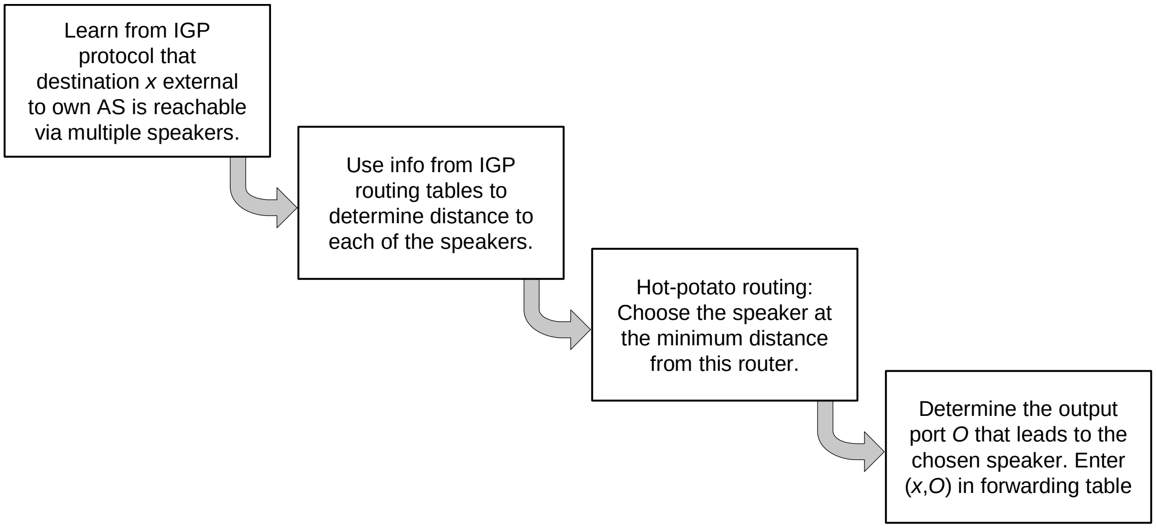

Learn from IGP protocol that destination *x* external to own AS is reachable via multiple speakers.
Use info from IGP routing tables to determine distance to each of the speakers.
Hot-potato routing: Choose the speaker at the minimum distance from this router.
Determine the output port *O* that leads to the chosen speaker. Enter (*x*,*O*) in forwarding table

Figure 1-59: Integrating an external destination to a router's forwarding table.

AS simply adds a new entry into its forwarding table. However, the problem is how to exchange the routing information with all other routers within this AS and achieve a consistent picture of the Internet viewed by all of the routers within this ASs. The goal is that, for a given data packet, each router in this AS should make the same forwarding decision (as if each had access to the routing tables of all the speaker routers within this AS). Each speaker node must exchange its routing information with all other routers within its own AS (known as *internal peering*). This includes both other speakers in the same AS (if there are any), as well as the remaining non-speaker routers. For example, in Figure 1-58 speaker *K* in ASδ needs to exchange routing information with speaker *N* (and vice versa), as well as with non-speaker routers *L* and *M*. Note again that only speaker routers run both IGP and Exterior Gateway Protocol (and each maintains two routing tables); non-speaker routers run only IGP and maintain a single routing table.

The forwarding table contains pairs ⟨*destination*, *output-port*⟩ for all possible destinations. The output port corresponds to the IP address of the next hop router to which the packet will be forwarded. Recall that each router performs the longest CIDR prefix match on each packet's destination IP address (Section 1.4.4). All forwarding tables must have a default entry for addresses that cannot be matched, and only routers in Tier-1 ISPs are default-free because they know prefixes to all networks in the global Internet.

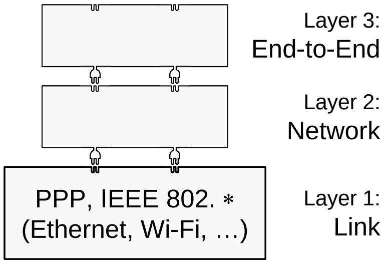
Consider again the scenario shown in Figure 1-58 where ASφ advertises the destination prefix `128.34.10.0/24`. If the AS has a single speaker node leading outside the AS, then it is easy to form the forwarding table. For example, in Figure 1-58 ASη has a single speaker router *R* that connects it to other ASs. If non-speaker router *S* in ASη receives a packet destined to ASφ, the packet will be forwarded along the shortest path (determined by the IGP protocol running in ASη) to the speaker *S*, which will then forward it to *N* in ASδ. Consider now a different situation where router *B* in ASα receives a packet destined to ASφ. *B* should clearly forward the packet to another speaker node, but which one? As seen in Figure 1-58, both *A* or *F* will learn about ASφ but via different routes. To solve the problem, when a speaker node learns about a destination outside its own AS, it must disseminate this information to all routers within its own AS. This dissemination is handled by the AS's interior gateway protocol (IGP).

When a router learns from an IGP advertisement about a destination outside its own AS, it needs to add the new destination into its forwarding table. This applies to both non-speaker routers and speaker routers that received this IGP advertisement from the fellow speaker router (within the same AS), which first received the advertisement via exterior gateway protocol from a different AS. One approach that is often employed in practice is known as **hot-potato routing**. In hot-potato routing, the Autonomous System gets rid of the packet (the "hot potato") as quickly as possible (more precisely, as inexpensively as possible). This is achieved by having the router send the packet to the speaker node that has the minimum distance to this router among all the speakers with a path to the destination. Figure 1-59 summarizes the steps taken at a router for adding the new entry to its forwarding table. In Figure 1-58, when *B* receives a packet for ASφ it will send it to *A* or *F* based on the lowest cost *within* ASα only, rather than overall lowest cost to the destination.

The most popular practical implementation of path vector routing is *Border Gateway Protocol* (BGP), currently in version 4 (BGP4). Section 9.2.3 describes how BGP4 meets the above requirements.

1.5 Link-Layer Protocols and Technologies

In packet-switched networks, blocks of data bits (generally called packets) are exchanged between the communicating nodes. That is, the nodes send packets rather than continuous bit-streams. At the link layer, packets are called *frames*. The key function of the link layer is transferring frames from one node to an adjacent node over a communication link. This task is complex because there are a great variety of communication link types. The key characteristics of a link include *data rate*, *duplexity* (half or full duplex), and *multiplicity* of the medium (i.e., point-to-point or shared broadcast). The link-layer services include:



• *Framing* is encapsulating a network-layer datagram into a link-layer frame by adding the header and the trailer. It is particularly challenging for a receiving node to recognize where an arriving frame begins and ends. For this purpose, special control bit-patterns are used to identify the start and end of a frame. On both endpoints of the link, receivers are continuously hunting for the start-of-frame bit-pattern to synchronize on the start of the frame. Having special control codes, in turn, creates the problem of *data transparency* (the need to avoid confusion between control codes and data) and requires *data stuffing* (described earlier in Figure 1-15).

• *Medium access control* (MAC) allows sharing a broadcast medium (Section 1.3.3) MAC addresses are used in frame headers to identify the sender and the receiver of the frame. MAC addresses are different from IP addresses and require a special mechanism for translation between different address types (Section 9.3.1). Point-to-point protocols do not need MAC.

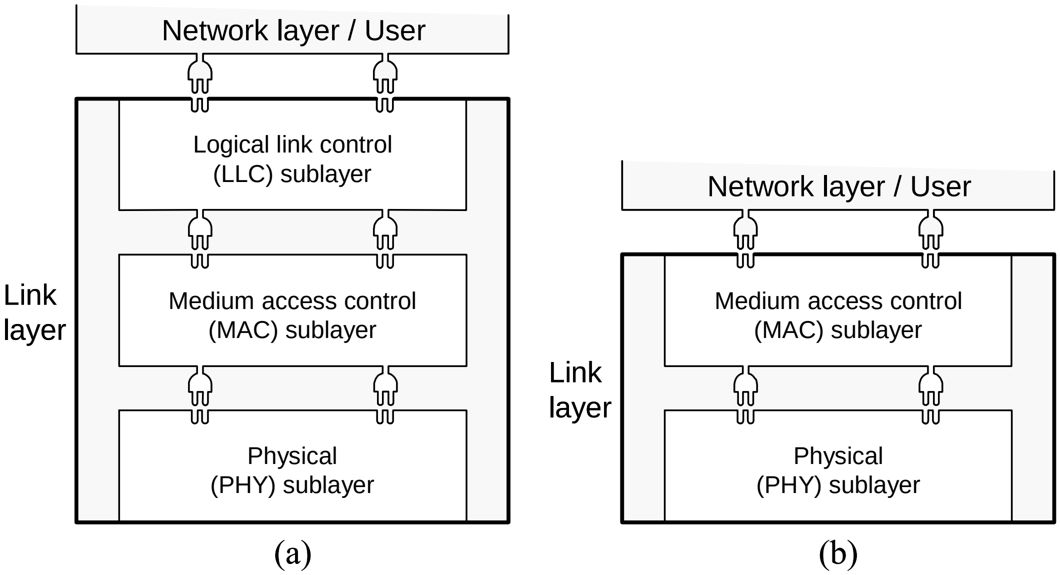


Figure 1-60: Sublayers of the link layer for broadcast communication links.

• *Reliable delivery* between adjacent nodes includes error detection and error recovery. The techniques for error recovery include forward error correction code (Section 1.2) and retransmission by ARQ protocols (Section 1.3).

• *Connection liveness* is the ability to detect a link outage that makes impossible to transfer data over the link. For example, a wire could be cut, or a metal barrier could disrupt the wireless link. The link-layer protocol should signal this error condition to the network layer.

• *Flow control* is pacing between adjacent sending and receiving nodes to avoid overflowing the receiving node with messages at a rate it cannot process. A link-layer receiver is expected to be able to receive frames at the full datarate of the underlying physical layer. However, a higher-layer receiver may not be able receive packets at this full datarate. It is usually left up to the higher-layer receiver to throttle the higher-layer sender. (An example for the TCP protocol is given in Section 2.1.4.) Sometimes the link layer may also participate in flow control. A simple way of exerting backpressure on the upper-layer protocol is shown in Listing 1-1 (Section 1.1.4) at the start of the method `send()`, where an exception is thrown if the buffer for storing the unacknowledged packets is full.

There are two types of communication links: (1) *point-to-point link* with one sender and one receiver on the link, and no medium access control (MAC) or explicit MAC addressing; and, (2) *broadcast link* over a shared wire or air medium. Point-to-point link is easier to work with than a broadcast link because broadcast requires coordination of many stations for accessing the medium. The basics of medium access control are already described in Section Section 1.3.3 and more will be covered later in this section.

Because broadcast links are so complex, it is common to subdivide the link layer of the protocol stack into three sublayers (Figure 1-60): logical-link-control (LLC) sublayer, medium access control (MAC) sublayer, and physical (PHY) sublayer. In the OSI reference model (Section 1.1.4), Layer 2 is subdivided into two sublayers: LLC and MAC sublayer. The network layer may directly use the services of a MAC sublayer (Figure 1-60(b)), or it may interact with a logical-

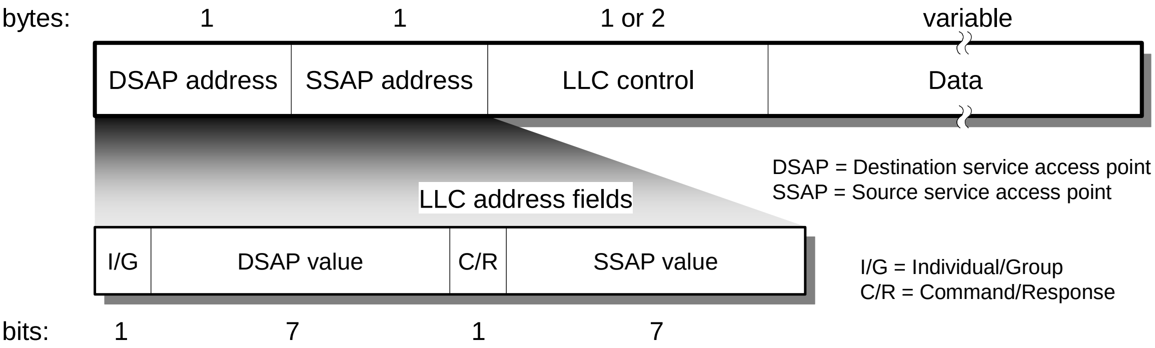


Figure 1-61: Packet format for Logical Link Control (LLC) protocol.

link-control (LLC) sublayer (Figure 1-60(a)). We will see examples of both approaches later in this section. The IP protocol (Section 1.4.1) usually directly interacts with a MAC sublayer.

IEEE specified the 802.2 standard for LLC, which is the common standard for all broadcast links specified by the IEEE Working Group 802, such as Ethernet (Section 1.5.2) and Wi-Fi (Section 1.5.3) broadcast links. 802.2 LLC hides the differences between various kinds of IEEE 802 links by providing a single frame format and service interface to the network layer. 802.2 LLC also provides options for reliable delivery and flow control. Figure 1-61 shows the LLC packet format. The two address fields specify the destination and source users of LLC, where the "user" is usually an upper-layer protocol, such as IP (Figure 1-60). The LLC user addresses are referred to as "service access points" (SAPs), which is the OSI terminology for the user of a protocol layer. The DSAP address field identifies one or more destination users for which the LLC packet data is intended. This field corresponds to the `receivingProtocol` field in Listing 1-1. The SSAP address field identifies the upper-layer protocol that sent the data.

Section 1.5.1 reviews a link-layer protocol for point-to-point links. Sections 1.5.2 and 1.5.3 review link-layer protocol for broadcast links: Ethernet for wire broadcast links and Wi-Fi for wireless broadcast links. Within a single building, broadcast local-area networks such as Ethernet or Wi-Fi are commonly used for interconnection. However, most of the wide-area (long distance) network infrastructure is built up from point-to-point leased lines.

1.5.1 Point-to-Point Protocol (PPP)

Problems related to this section: Problem 1.41

Figure 1-62 illustrates two typical scenarios where point-to-point links are used. The first is for telephone dialup access, where a customer's PC calls up an Internet service provider's (ISP) router and then acts as an Internet host. When connected at a distance, each endpoint needs to be fitted with a *modem* to convert analog communications signals into a digital data stream. Figure 1-62 shows modems as external to emphasize their role, but nowadays computers have built-in modems. Another frequent scenario for point-to-point links is connecting two distant routers that belong to the same or different ISPs (right-hand side of Figure 1-62). In this case, *dedicated modems* may be used. These modems do not dial, and instead automatically connect when they are powered on. They also reconnect, without any operator intervention, if a disturbance on the line is severe enough to break the connection.

Visit http://en.wikipedia.org/wiki/HDLC for more details on High-Level Data Link Control (HDLC)

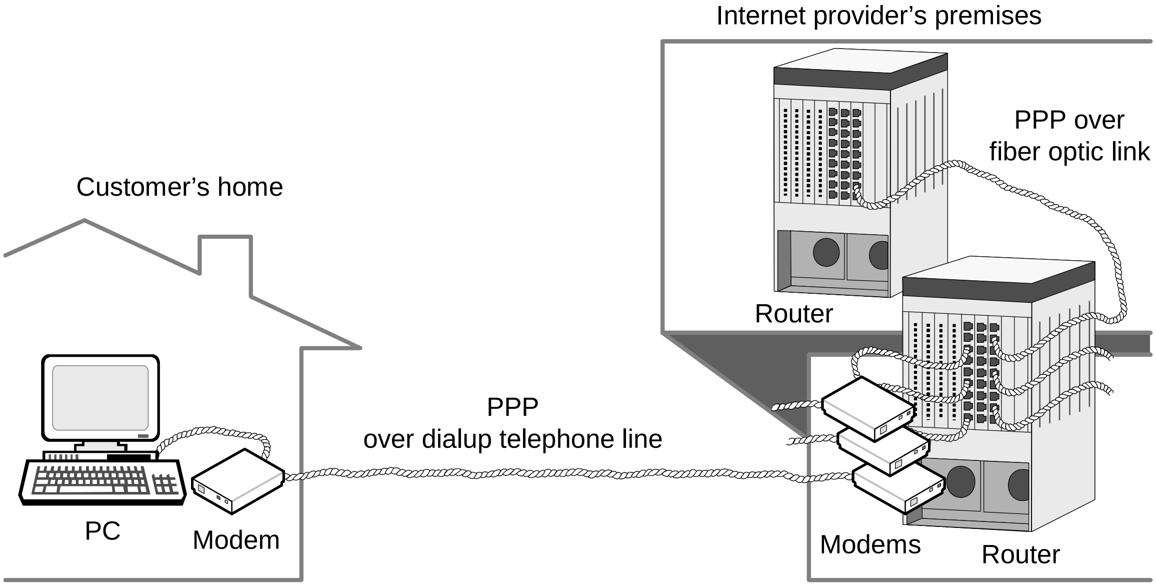
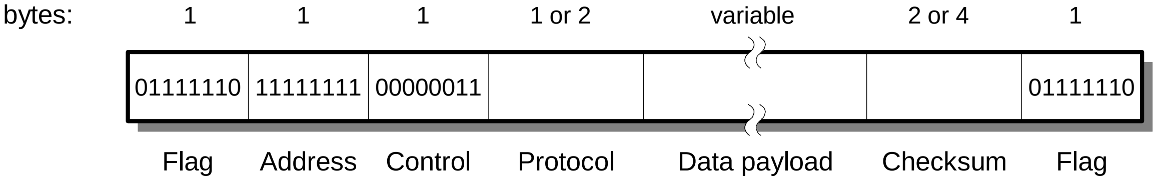


Figure 1-62: Point-to-point protocol (PPP) provides link-layer connectivity between a pair of network nodes over many types of physical networks.



bytes:	Flag	Address	Control	Protocol	Data payload	Checksum	Flag
	1	1	1	1 or 2	variable	2 or 4	1
	01111110	11111111	00000011				01111110

Figure 1-63: Point-to-point protocol (PPP) frame format.

Two most popular point-to-point link-layer protocols are **PPP** (point-to-point protocol), which is *byte-oriented*, viewing each frame as a collection of bytes; and **HDLC** (high-level data link control), which is *bit-oriented*. PPP, although derived from HDLC, is simpler and includes only a subset of HDLC functionality. (This book does not cover HDLC and the reader should check the bibliography in Section 1.7 for relevant references.)

The format of a PPP frame is shown in Figure 1-63. The PPP frame always begins and ends with a special character (called "flag"). The Flag makes it possible for the receiver to recognize the boundaries of an arriving frame. Note that the PPP frame header does not include any information about the frame length, so the receiver recognizes the end of the frame when it encounters the trailing Flag field. The second field (Address) normally contains all ones (the broadcast address of HDLC), which indicates that all stations should accept this frame. Because there are only two hosts attached to a PPP link, PPP uses the broadcast address to avoid having to assign link-layer addresses. The third field (Control) is set to a default value 00000011. This value indicates that PPP is run in connectionless mode, meaning that frame sequence numbers are *not* used and out-of-order delivery is acceptable.

Because the Address and Control fields are always constant in the default configuration, the nodes can negotiate an option to omit these fields and reduce the overhead by 2 bytes per frame.

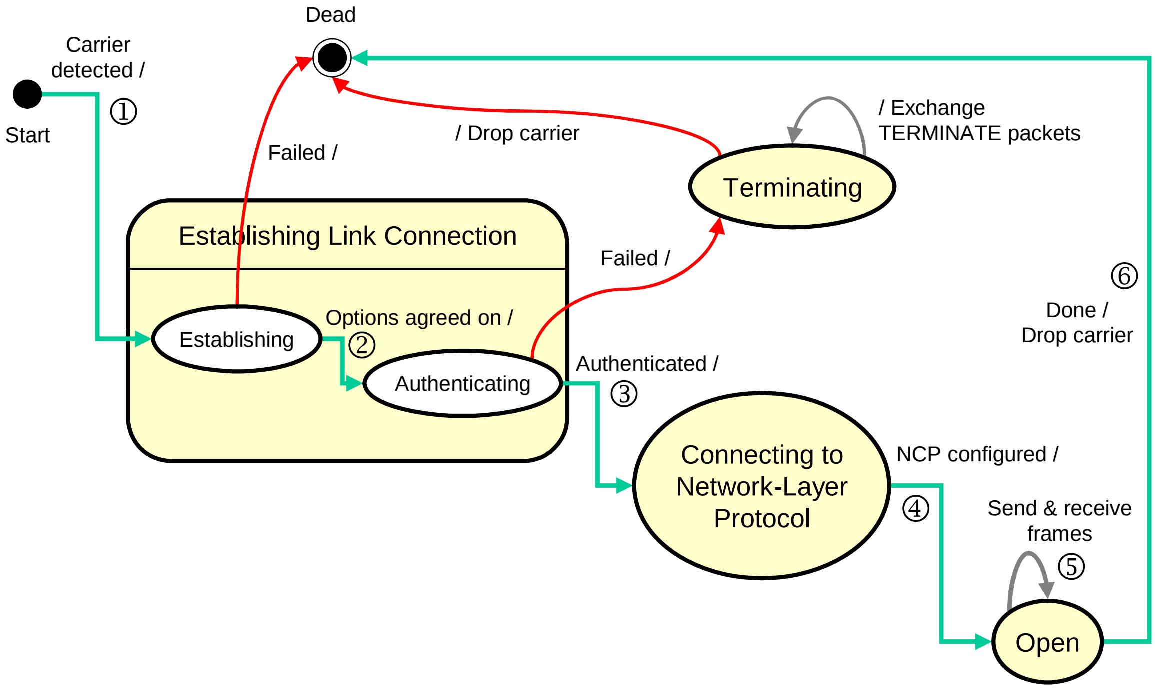


Figure 1-64: State diagram for the point-to-point protocol (PPP).

The Protocol field is used for demultiplexing at the receiver: it identifies the upper-layer protocol (e.g., the IP, Section 1.4.1) that should receive the payload of this frame. The code for the IP protocol is hexadecimal 21_{16}. The reader may wish to check Listing 1-1 (Section 1.1.4) as to how the method `handle()` calls `upperProtocol.handle()` to handle the received payload.

The Payload field is of variable length, up to some negotiated maximum; if not negotiated, the default length of 1500 bytes is used. After Payload comes the Checksum field, which is by default 2 bytes, but can be negotiated to a 4-byte checksum. PPP checksum only detects errors, and has no capability for error correction/recovery—it does not implement any ARQ protocol. A PPP receiver silently drops a packet with the checksum indicating an error in transmission.

Figure 1-64 summarizes the state diagram for PPP; the actual finite state machine of the PPP protocol is more complex and the interested reader should consult RFC-1661 [Simpson, 1994]. There are two key steps before the endpoints can start exchanging network-layer data packets:

1. **Establishing link connection:** during this phase, the link-layer connection is set up. The link-layer peers must configure the PPP link. This includes negotiating the parameters such as the maximum frame length, authentication, whether to omit the Address and Control fields, and the number of bytes for the protocol and checksum fields (Figure 1-63). PPP's **Link Control Protocol (LCP)** is used for this purpose.

2. **Connecting to network-layer protocol:** after the link has been established and options negotiated by the LCP, PPP must choose and configure one or more network-layer protocols that will operate over the link. PPP's **Network Control Protocol (NCP)** is used for this purpose. Once the chosen network-layer protocol has been configured, datagrams can be sent over the link.

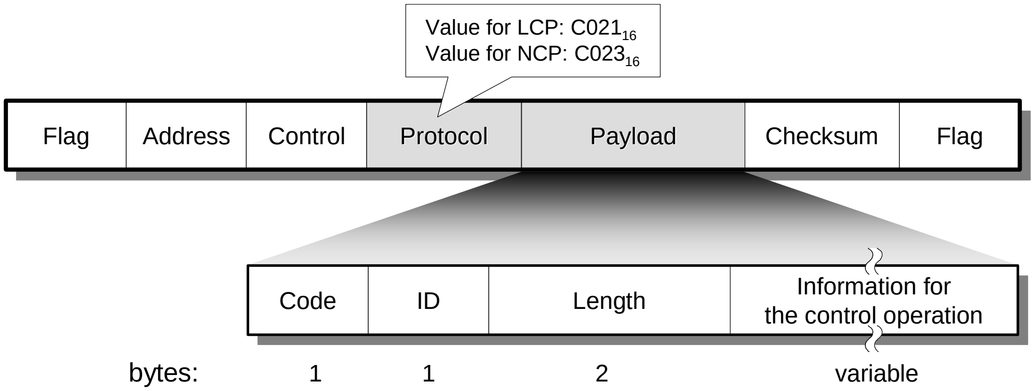

Value for LCP: $C021_{16}$
Value for NCP: $C023_{16}$
Flag
Address
Control
Protocol
Payload
Checksum
Flag
Code
ID
Length
Information for the control operation
bytes:
1
1
2
variable

Figure 1-65: LCP or NCP packet encapsulated in a PPP frame (see Figure 1-63).

LCP and NCP are based on PPP and run on top of it. If the transition through these two states is successful, the connection goes to the *Open* state, where data transfer between the endpoints takes place.

The *Authenticating* state (sub-state of *Establishing Link Connection*) is optional. The two endpoints may decide, during the *Establishing* sub-state, not to go through authentication. If they decide to proceed with authentication, they will run Challenge Handshake Authentication Protocol (CHAP, defined in RFC-1994) to verify each other's identity using a three-way handshake.

Listing 1-1 in Section 1.1.4 shows how application calls the protocols down the protocol stack when sending a packet. However, before `send()` can be called, `lowerLayerProtocol` must be initialized. Link-layer protocol is usually built-in in the firmware of the network interface card, and the initialization happens when the hardware is powered up or user runs a special application. Therefore, NCP in step 2 above establishes the connection between the link-layer PPP protocol and the higher-layer (e.g., IP) protocol that will use its services to transmit packets.

LCP and NCP protocols send control messages encapsulated as the payload field in PPP frames (Figure 1-65). The receiving PPP endpoint delivers the messages to the receiving LCP or NCP module, which in turn configures the parameters of the PPP connection.

Although PPP frames do not use link-layer addresses, PPP provides the capability for network-layer address negotiation: endpoint can learn and/or configure each other's network-layer address.

In summary, PPP has no error correction/recovery (no ARQ, only error detection), no flow control, and out-of-order delivery is acceptable. No specific protocol is defined for the physical layer in PPP.

1.5.2 Ethernet (IEEE 802.3)

Problems related to this section: Problem 1.42 → Problem 1.45

Ethernet is a network protocol for local area networks (LANs). The MAC protocol for Ethernet is based on the CSMA/CD protocol shown in Figure 1-33. The frame format for Ethernet is shown in Figure 1-66. Ethernet was first standardized by DEC, Intel and Xerox, known as the DIX

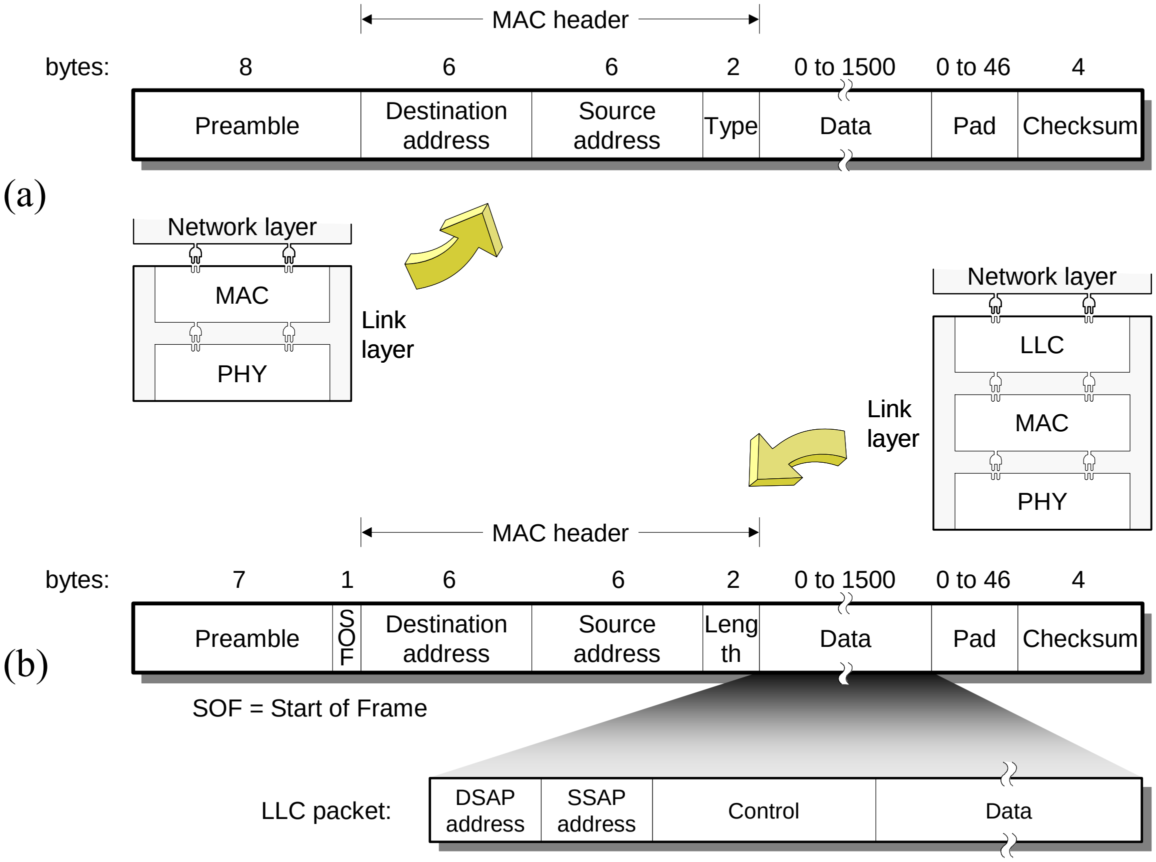


Figure 1-66: Link-layer frame format for DIX standard Ethernet Version 2.0 (a) and for IEEE standard 802.3 (b). The LLC packet format at the bottom is shown in Figure 1-61.

standard. (See Section 1.7 for an overview of Ethernet history.) When IEEE released the 802.3 standard, it adopted a slightly different frame format, as shown in Figure 1-66(b). The Type field in a DIX frame represents the upper-layer protocol that is using Ethernet as its link layer. On the other hand, an 802.3 frame carries instead the frame Length. In 802.3 frame, the upper-layer protocol is specified in the LLC frame as DSAP address (also see Figure 1-61). Because the DIX standard was widely used by the time IEEE 802.3 was released, a compromise was reached as follows. If the Type/Length field contains a number ≤1500 than it represents the frame Length and the receiver should look for the upper-layer protocol in the contained LLC packet. If the Type/Length field contains a number >1500 than it identifies the upper-layer protocol, and the data field does not contain an LLC-formatted packet, but rather a network-layer packet (e.g., an IP datagram). All versions of Ethernet up to date use this frame format.

The **Ethernet link-layer address** or **MAC-48 address** is a globally unique 6-byte (48-bit) string that comes wired into the electronics of the Ethernet attachment. An Ethernet address has two parts: a 3-byte manufacturer code, and a 3-byte adaptor number. IEEE acts as a global authority and assigns a unique manufacturer's registered identification number, while each manufacturer gives an adaptor a unique number (also see Sidebar 1.3 in Section 1.4.4). Although intended to be a permanent and globally unique identification, it is possible to change the MAC address on most of today's hardware, an action often referred to as *MAC spoofing*.

When a frame arrives at an Ethernet attachment, the electronics compares the destination address with its own and discards the frame if the addresses differ, unless the address is a special "broadcast address" which signals that the frame is meant for all the nodes on this network. The CSMA/CD protocol (Section 1.3.3) does not implement ARQ for reliable data transmission. The receiver silently drops corrupted frames (detected using the checksum, Figure 1-66). If the dropped frames were important, the sending application would need to detect their loss and retransmit them.

We know that in CSMA/CD the transmission time of the smallest valid frame must be larger than one maximum round-trip propagation time. This requirement limits the distance between two computers on an Ethernet LAN. The smallest valid frame size is 64 bytes (512 bits). This 64-byte value is derived from the original 2500-m maximum distance between Ethernet interfaces plus the transit time across up to four repeaters plus the time the electronics takes to detect the collision. The 64 bytes correspond to 51.2 μs over a 10 Mbps link, which is larger than the round-trip time across 2500 m (about 18 μs) plus the delays across repeaters and the electronics to detect the collision. The Ethernet standards impose strict limitations on the size of encapsulated packets, requiring small packets to be padded up to a minimum size. The *Pad* field (Figure 1-66) allows up to 46 bytes of padding, which with the MAC header and checksum produces a 64-byte frame for frames with zero bytes of data.

Sensing the medium idle takes time, so there will necessarily be an idle period between transmissions of Ethernet frames. This period is known as the **interframe space** (IFS), *interframe gap*, or *interpacket gap*. The IFS is a self-imposed quiet time appended to the end of every frame. It is the spacing between two non-colliding frames, from start of idle after the last bit of the FCS field of the first frame to the first bit of the Preamble of the subsequent frame. This idle time gives the network media a chance to stabilize, and receiving nodes to process the received frame and prepare themselves for the next transmission. If an Ethernet network adapter senses that there is no signal energy entering the adapter from the channel for IFS-bit times, it declares the channel idle and starts to transmit the frame. The minimum interframe space for Ethernet is 96-bit times (the time it takes to transmit 96 bits of raw data on the medium), which is 9.6 μs for 10 Mbps Ethernet, 960 ns for 100 Mbps (fast) Ethernet, 96 ns for 1 Gbps (gigabit) Ethernet, and 9.6 ns for 10 Gbps (10 gigabit) Ethernet. (Note that the IFS period is not shown in Figure 1-33 to keep it simple, but it occurs on the middle path in the diagram, when sensing the carrier idle before starting the transmission.)

Table 1-4: Parameter values for the Ethernet MAC protocol (CSMA/CD).

Parameter	**Data rate**		
	Up to and including 100 Mbps	1 Gbps	10 Gbps
Backoff slot time	512 bit times	4096 bit times	not applicable
Interpacket gap / IFS	96 bits	96 bits	96 bits
Attempts limit	16	16	not applicable
Backoff limit	10	10	not applicable
Jam size	32 bits	32 bits	not applicable
Maximum frame size	1518 bytes	1518 bytes	1518 bytes
Minimum frame size	512 bits (64 bytes)	512 bits (64 bytes)	512 bits (64 bytes)

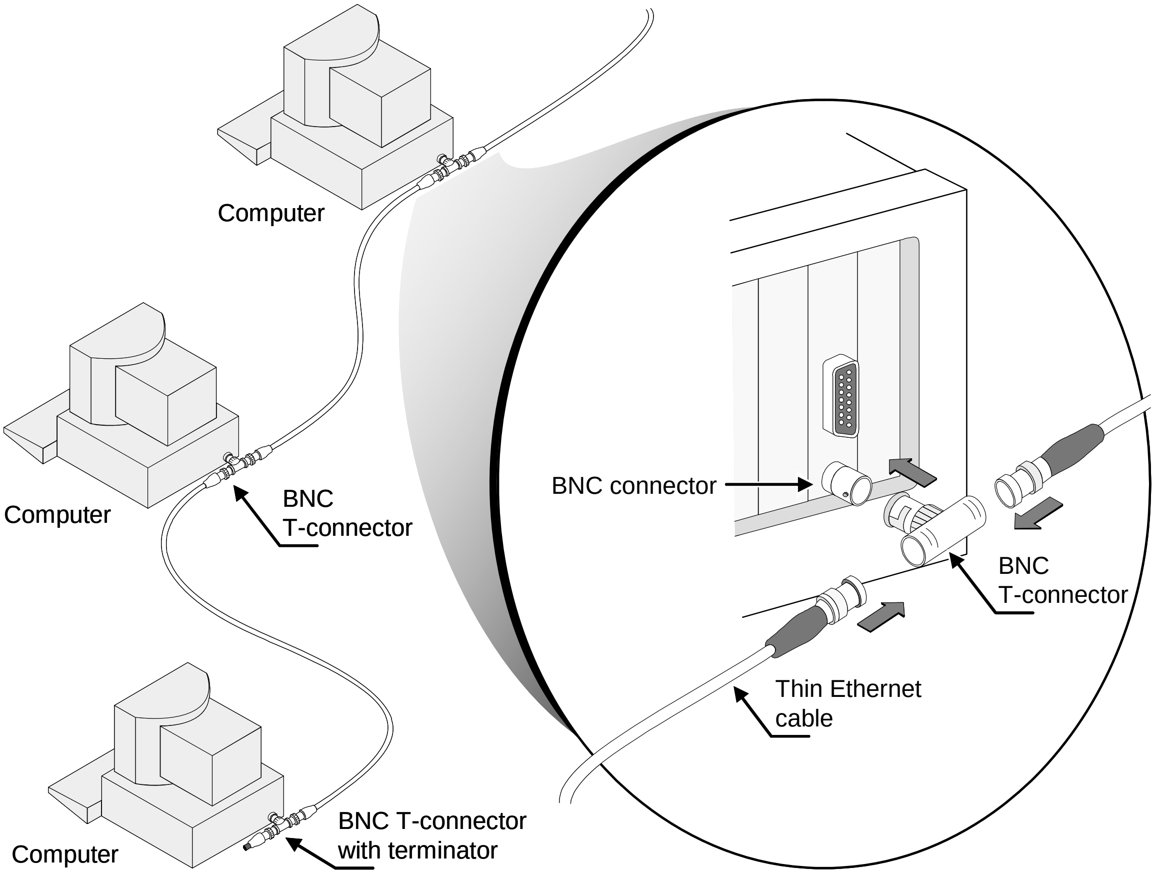


Figure 1-67: Thin coaxial cable Ethernet represents a bus-based design.

The Ethernet specification for a bus-based design allows no more than 1,024 hosts and it can span only a geographic area of 2,500 m. Table 1-4 lists some important parameters of the Ethernet MAC protocol for different data rates of the physical sublayer.

Evolution of Ethernet

Ethernet has evolved over the past 35 years since it was invented. This evolution was shaped by physical characteristics of communication links, such as *data rate*, *duplexity* (half or full duplex), and *multiplicity* of the medium (i.e., point-to-point or shared broadcast). Ethernet operation is specified for data rates from 1 Mbps to 10 Gbps using a common MAC protocol (CSMA/CD). In 1997, IEEE Std 802.3x specified *full duplex* operation. The CSMA/CD MAC protocol specifies shared medium (half duplex) operation where frame collisions can occur, as well as full duplex operation that operates without collisions. Ethernet Physical Sublayer (PHY) is standardized for operation over coaxial, twisted-pair or fiber-optic cables.

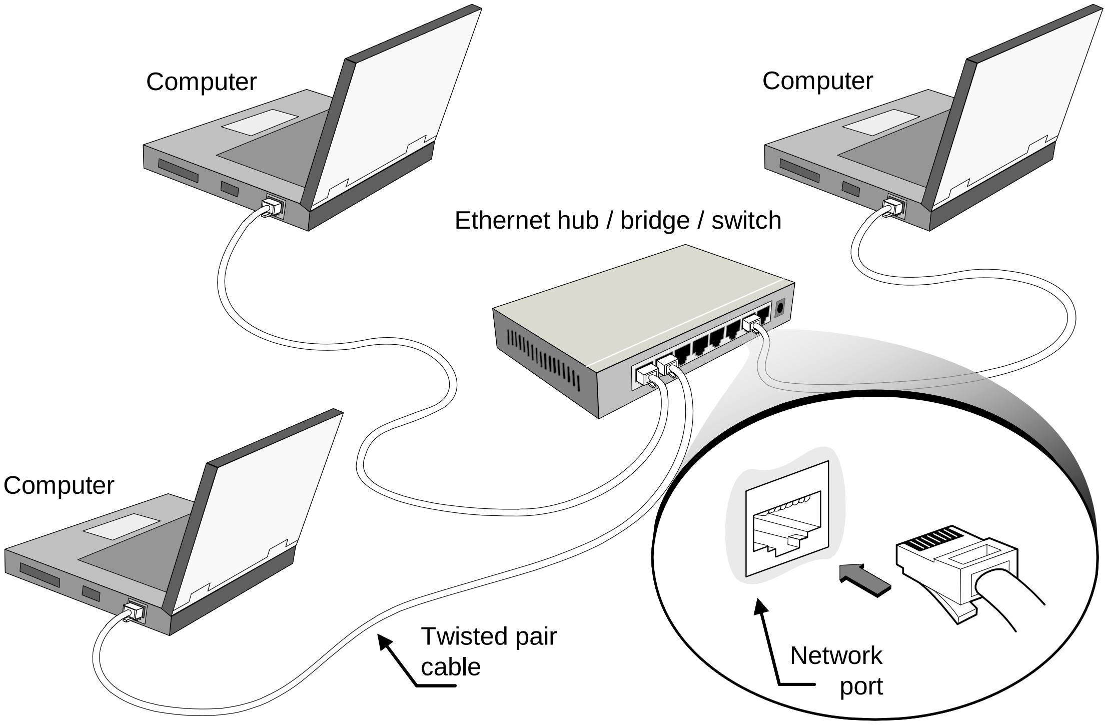
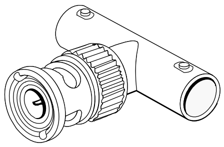


Figure 1-68: Bridged or switched Ethernet represents a star-based (hub-and-spokes) design.



Ethernet was first standardized for operation over coaxial cables (Figure 1-67). A cable (ether) with multiple devices attached to it in parallel is called a *multidrop cable*. This is also known as **bus**-based design for Ethernet. The multidrop cable with all stations attached to it together are called a **collision domain**. If two or more stations in a collision domain transmit simultaneouly, their frames will collide and will not be successfully received. The first Ethernet design was the so-called *Thick Ethernet* (or, 10Base5) that appeared in the early 1970s. It used a thick coaxial cable with markings to show where transcievers can be screwed onto the cable (2.5 meters apart). The subsequent cable type was *Thin Ethernet* (or, 10Base2), which used standard BNC connectors to form T-junctions on the carrier cable (Figure 1-67). Multidrop-cable Ethernets were followed by a star-patterned wiring, where all computers in the LAN have a cable running to a central **hub** and the incident spokes where endpoint hosts were connected (Figure 1-68). Historically, the first instance of this design is 10Base-T. The Ethernet version notation consists of three parts, as follows:

Data rate (e.g., 10 Mbps, 10 Gbps)	Baseband/Broadband transmission	Wiring type (e.g., coaxial, twisted pair or fiber optic)

For example, 10Base-T means 10 Mbps baseband transmission over unshielded twisted-pair cable (Category 5 UTP); 10GBase-X means 10 Gbps baseband over two pairs of twisted-pair cable.

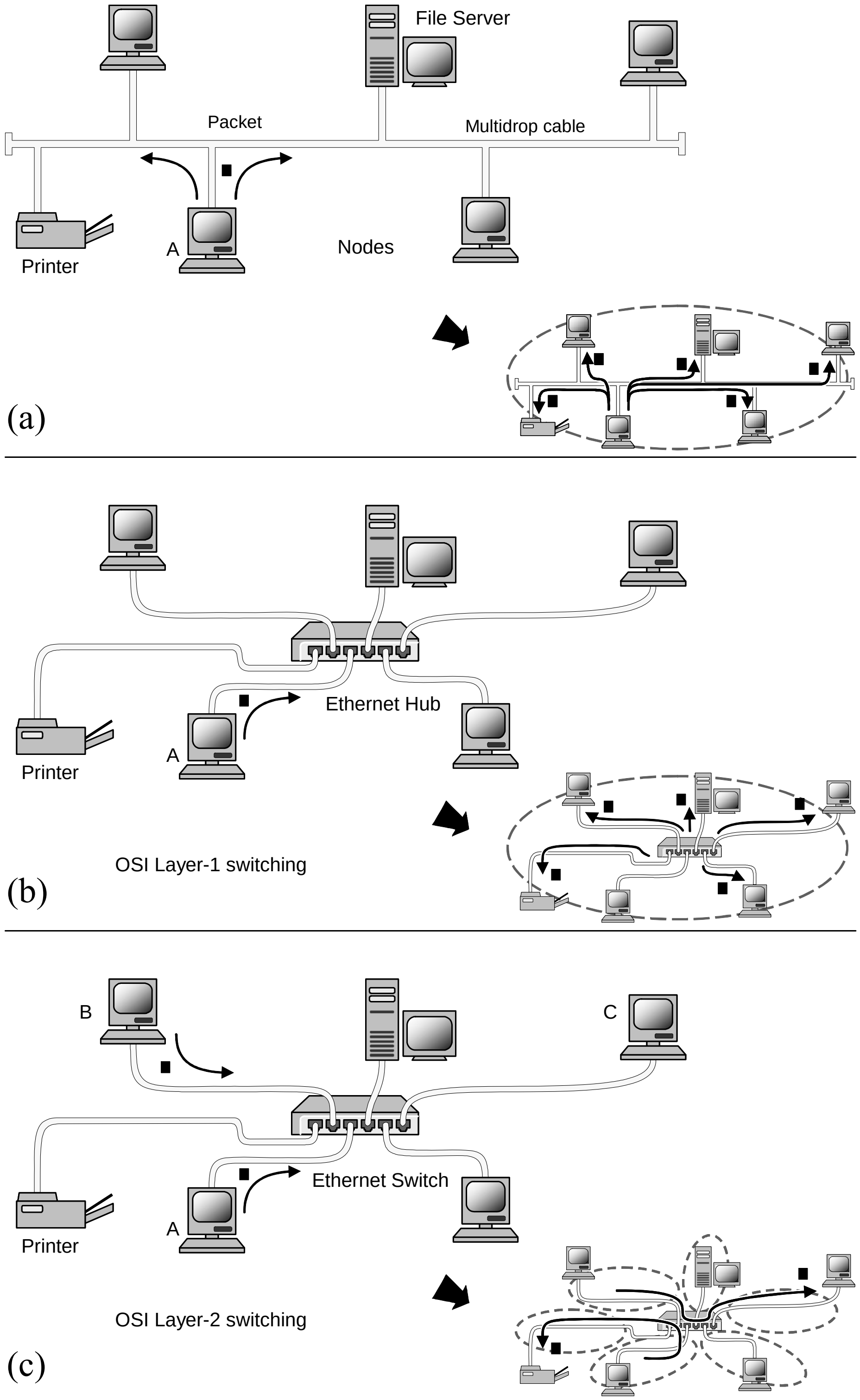


Figure 1-69: Comparing bus-based Ethernet (a), hub-based Ethernet (b) and switch-based Ethernet (c). Dashed-outline ovals indicate independent collision domains.

The star topology in Figure 1-68 has many variations, depending on whether the central device operates at the physical layer (OSI Layer 1) or at the link layer (OSI Layer 2) and whether the

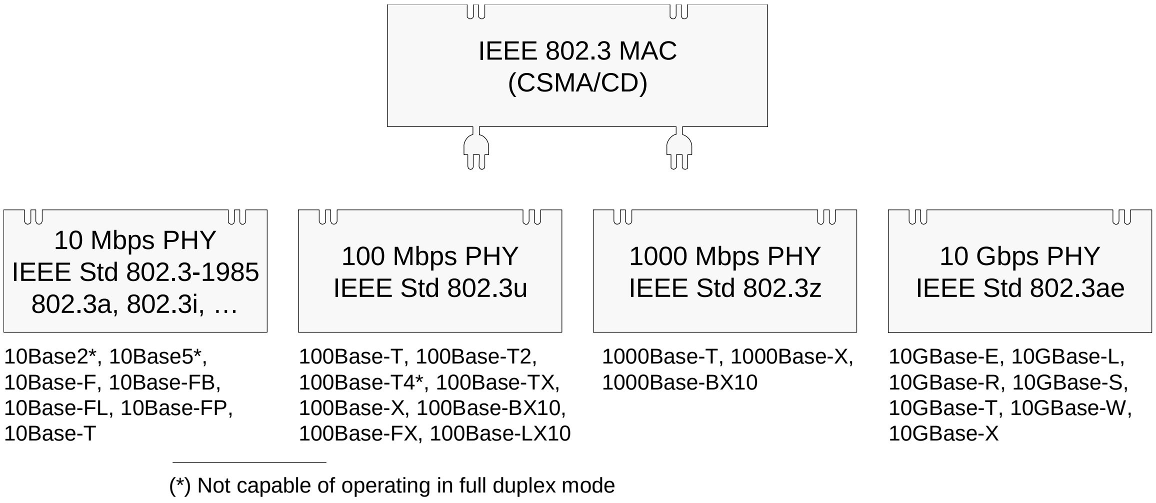

IEEE 802.3 MAC
(CSMA/CD)
10 Mbps PHY
IEEE Std 802.3-1985
802.3a, 802.3i, …
100 Mbps PHY
IEEE Std 802.3u
1000 Mbps PHY
IEEE Std 802.3z
10 Gbps PHY
IEEE Std 802.3ae
10Base2*, 10Base5*, 10Base-F, 10Base-FB, 10Base-FL, 10Base-FP, 10Base-T
100Base-T, 100Base-T2, 100Base-T4*, 100Base-TX, 100Base-X, 100Base-BX10, 100Base-FX, 100Base-LX10
1000Base-T, 1000Base-X, 1000Base-BX10
10GBase-E, 10GBase-L, 10GBase-R, 10GBase-S, 10GBase-T, 10GBase-W, 10GBase-X
(*) Not capable of operating in full duplex mode

Figure 1-70: Ethernet standards family for IEEE Std 802.3-2008 (current).

links are half duplex or full duplex. The central device was historically first called **bridge**. In the simplest version, the bridge is called **hub** or **repeater** and it operates at the physical layer in a half-duplex mode. A *hub* does not understand anything beyond individual bits, i.e., does not recognize frames nor knows about device addresses. It simply switches bits that come in on one network interface (or, port) to *all* other interfaces. The whole network forms a *single collision domain*, so conceptually this design is equivalent to a bus-based design.

A bridge with greater capabilities is known as a **switch**, but the term “bridge” is also often used synonymously. An Ethernet switch moves frames from input to output ports based on their Layer-2 destination addresses (described above as MAC-48 addresses). In other words, unlike a hub which switches bits to all interfaces, a switch switches a frame exclusively to a port determined by the frame’s destination address.

Figure 1-69 illustrates the difference between hubs and switches. Ethernet hubs (Figure 1-69(b)) are conceptually equivalent to the bus-based Ethernet (Figure 1-69(a)) because both designs form a single collision domain. Conversely, each network port of an Ethernet switch forms an *independent collision domain* (Figure 1-69(c)). With switch-based design, each cable has only two stations attached: on one end is a switch’s port and on the other end is a computer host. This is essentially a point-to-point link, but collisions are still possible between the endpoints and CSMA/CD must be employed. More detail on Ethernet switches is provided later in this section.

Figure 1-70 summarizes the current family of Ethernet protocols. The figure also indicates whether a physical sublayer has the ability to perform full-duplex link transmission and reception, which is described next.

Full-duplex Mode and Collision-free Switched Ethernet

Traditionally, Ethernet MAC sublayer implements the CSMA/CD algorithm, which creates a half-duplex link. In half-duplex mode, media access method is the means by which two or more stations share a common transmission medium (broadcast). To transmit, a station defers the transmission and waits for a quiet period on the medium, when no other station is transmitting. Then, it sends the intended message in bit-serial form. If, after initiating a transmission, the message collides with that of another station, then each transmitting station intentionally transmits the jam signal to ensure propagation of the collision throughout the system. The station remains silent for a random amount of time (backoff) before attempting to transmit again.

The current Ethernet 802.3 standard (IEEE Std 802.3-2008) provides for two modes of operation of the MAC sublayer:

(a) In **half-duplex mode**, stations contend for the use of the physical medium, using the traditional CSMA/CD contention-based operation. Bidirectional communication is accomplished by sequential exchange of frames, rather than simultaneous transmission in both directions. Half-duplex operation is possible on all supported media; it is required on those media that are incapable of supporting simultaneous transmission and reception without interference, such as 10Base2 and 100Base-T4 (Figure 1-70).

(b) The **full-duplex mode** of operation allows simultaneous communication between a pair of stations using point-to-point media (dedicated channel). Full-duplex operation does not require that transmitters defer, nor do they monitor or react to receive activity. There is no "collision detection" in this mode because there is no contention for a shared medium. Full-duplex operation can be used when *all* of the following are true:

1) The physical medium is capable of supporting simultaneous transmission and reception without interference (Figure 1-70).

2) There are *exactly two* stations connected with a full duplex point-to-point link. Because there is no contention for use of a shared medium, the multiple access (i.e., CSMA/CD) algorithms are unnecessary.

3) Both stations on the LAN are capable of, and have been configured to use, full duplex operation.

The most common configuration envisioned for full-duplex operation consists of a central switch (or, bridge) with a dedicated cable connecting each switch port to a single station. Ethernet hubs or repeaters are incapable of full duplex operation. By definition, an IEEE 802.3 LAN operating in full-duplex mode comprises exactly two stations, so full-duplex mode creates an **Ethernet point-to-point link**.

An Ethernet device operates in either half or full duplex mode at any one time. A device is configured for one specific mode of operation (e.g. 1000Base-X Full Duplex). Auto-Negotiation is performed as part of the initial set-up of the link, and allows the physical layers (PHYs) at each end to advertise their capabilities (speed, PHY type, half or full duplex) and to automatically select the operating mode for communication on the link. The term "CSMA/CD MAC" is used synonymously with "802.3 MAC," and may represent an instance of either a half duplex or full duplex mode device, although full-duplex devices do not implement the traditional CSMA/CD algorithm. In full-duplex mode, stations do not implement the CSMA/CD algorithms traditionally used to arbitrate access to shared-media LANs. Full-duplex operation constitutes a proper subset of the MAC functionality required for half-duplex operation.

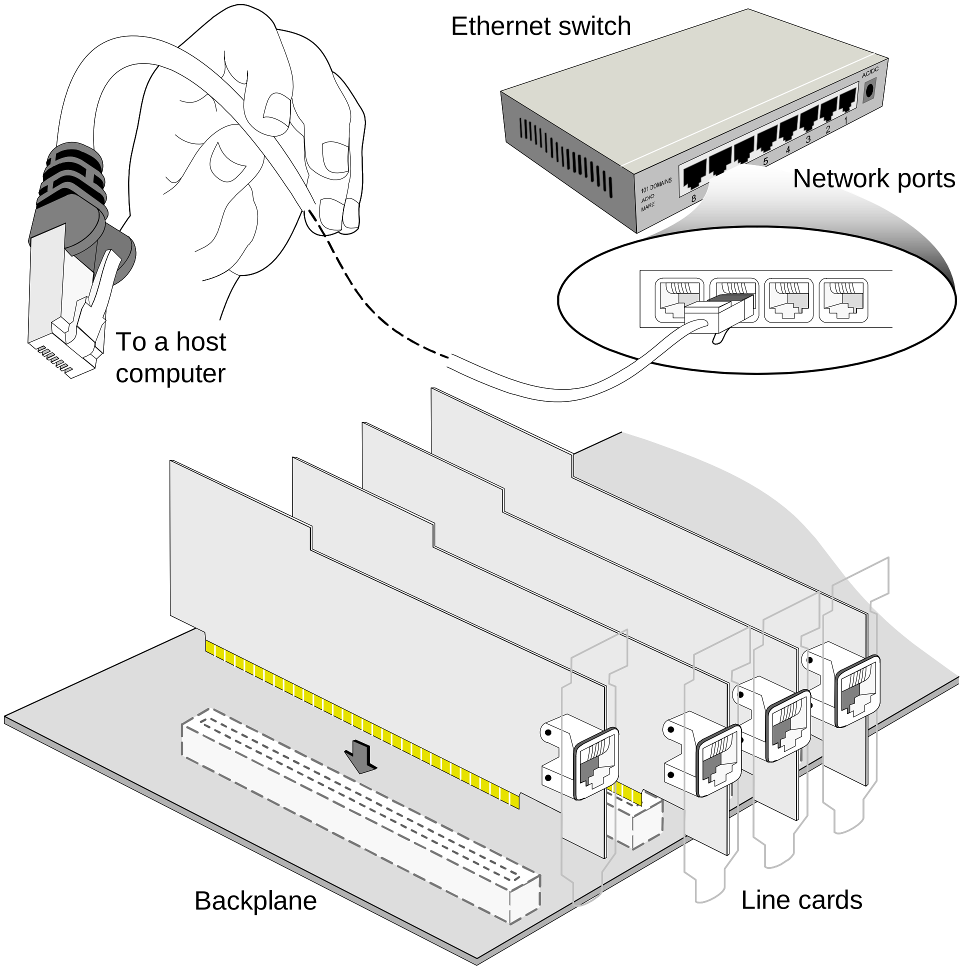


Figure 1-71: Ethernet switch architecture. Line cards are also known as Network Interface Cards (NICs).

Ethernet Switches and the Spanning Tree Protocol

An Ethernet switch consists of a high-speed backplane and a number of plug-in line cards, typically 4 to 32 (Figure 1-71). Each line card contains one or more (e.g., eight) network ports or connectors. A twisted pair cable leads from each connector to a host computer. When a computer sends a frame, the frame first reaches an associated line card, which checks whether the frame is destined to a station connected to the same card. If so, the frame is copied to the given port/connector on this line card. If not, the frame is sent over the backplane to the destination computer's line card. The backplane typically runs at data rates of many Gbps, using a proprietary protocol. More about switch design is available in Section 4.1.

A hub or repeater transmits a frame on an output port while it is being received on an input port. This is known as *cut-through switching*. Unlike a hub/repeater, a switch or bridge first receives the entire frame then stores it, waiting for the network attached to the frame's outgoing port to become idle. This is known as *store-and-forward switching*. With store-and-forward switching, it

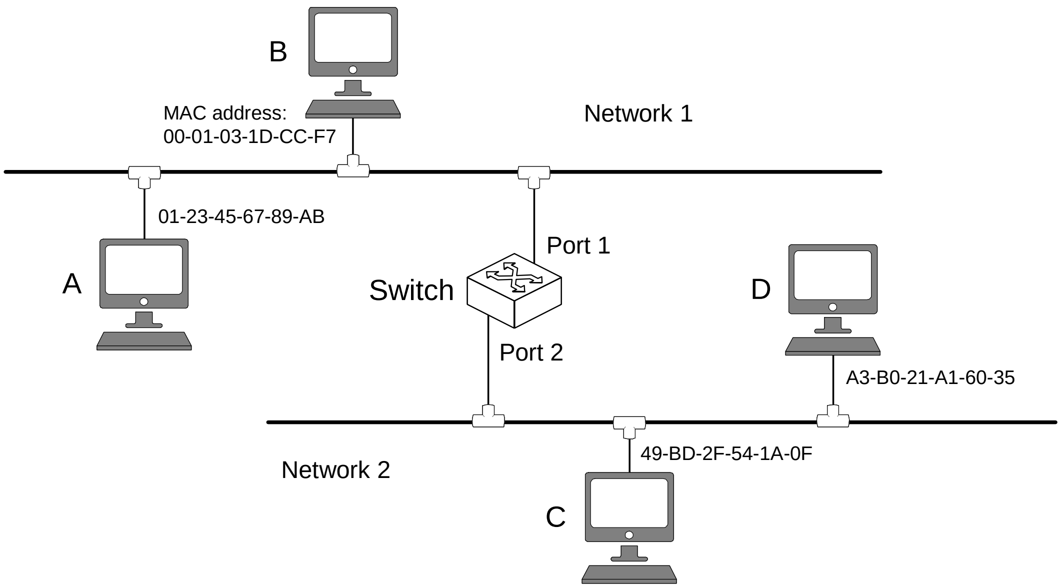


Figure 1-72: Example Ethernet networks connected by an Ethernet switch.

is possible for two stations on different ports of the switch to transmit simultaneously without a collision. We say that switch ports form independent collision domains (Figure 1-69).

As already pointed out, switches switch packets based on their link-layer (or, MAC) addresses, i.e., switches operate at OSI Layer-2. They are also known as *LAN switches*. In Section 1.4 we also learned about another kind of switches: routers. Routers switch packets based their network-layer addresses, i.e., switches operate at OSI Layer-3. Routers are more complex because they need to run routing protocols to discover the topology of the entire internetwork consisting of many networks. It is also said that LAN switches are *transparent* to the computers in the network. Unlike routers, where nodes know of the next-hop routers, LAN nodes are unaware of intermediate switches and their forwarding role. When a computer sends a frame, the frame is addressed to another computer, rather than addressing the frame to a switch. The frame will pass through a switch when going from one LAN segment to another without the switch identifying itself as the device that transmitted the frame to the next segment. Therefore, switches are transparent to each other, as well. Routers are described in Section 4.1.

LAN switches perform two basic functions: frame forwarding and frame filtering. **Frame forwarding** helps move a frame toward its ultimate destination. A switch moves a frame from an input port to an output port based on frame's MAC address by looking up the **switching table**. The switching table is similar to a router's forwarding table. Consider a network in Figure 1-72. The switching table of the switch is shown in Table 1-5. The MAC addresses of stations *A*, *B*, and *D* are listed in the table. For example, if a frame arrives on Port-2 destined for MAC address 00-01-03-1D-CC-F7 (station *A*), the switch outputs the frame on Port-1. If a frame arrives on Port-1

Table 1-5: The switching table for the example LAN in Figure 1-72.

MAC address	Network port	Time last frame received
00-01-03-1D-CC-F7	1	10:39
01-23-45-67-89-AB	1	10:52
A3-B0-21-A1-60-35	2	10:17

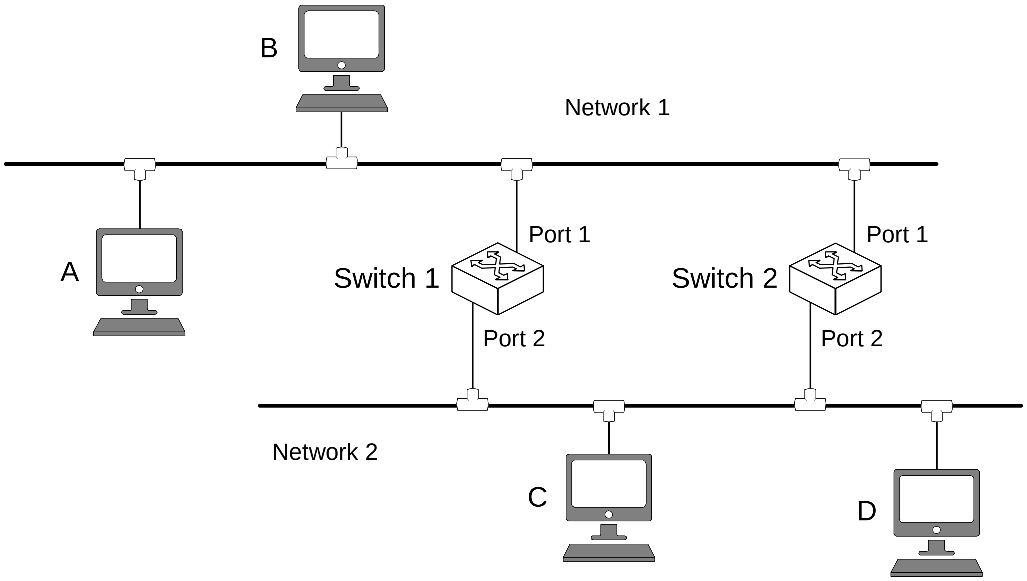


Figure 1-73: Example switched network with a loop formed by two switches.

destined for 49-BD-2F-54-1A-0F (station *C*), which is currently not listed in the switching table, the switch will output the frame to all other ports. In this example, Port-2 is the only other port.

Frame filtering relates to discarding frames that are headed in a direction where they do not need to go. For example, if in Figure 1-72 a frame arrives on Port-2 with the destination address 00-01-03-1D-CC-F7 (station *A*), then according to Table 1-5 this frame should be output on Port-1. On the other hand, assume that a frame arrives on Port-2 with the destination address A3-B0-21-A1-60-35. The switch realizes that it is a station on a network segment attached on Port-2 sending a frame to another station on the same segment. In our case, it is station *C* sending a frame to station *D*. There is no need to forward this frame because all other stations on the same segment already received this frame, so the switch filters this frame and discards it.

The switching table can be filled up manually, but this is a tedious and error-prone task for large number of stations. Instead, the switch performs **backward learning** of the switching table. Initially, the table is empty. When the switch receives a frame from a station for which it has no address in the table, the switch automatically creates a new entry. The entry records the MAC address in the frame's *Source address* field (Figure 1-66), the network port on which the frame arrived, and the time of the arrival. If every station on all attached networks sends a frame, then every station will eventually be recorded in the table. Note that this learning algorithm does not use control packets—the learning process piggybacks on upper-layer data packets and avoids introducing the overhead of control packets.

The parameter called **aging time** determines how long the table entries are valid. If the switch does not receive a frame with a given address as its source address, the entry will be deleted from the table. For example, in Table 1-5 a frame with source address A3-B0-21-A1-60-35 (station *D* Figure 1-66) arrived last time at 10:17 on Port-2. Suppose that the aging time for this switch is 50 minutes. If no frame arrives with source address A3-B0-21-A1-60-35 arrives until 11:07, the switch will remove the entry for station *D* from the table. In this way, if a computer is unplugged or moved around the building and plugged in again somewhere else, the network can operate without manual intervention.

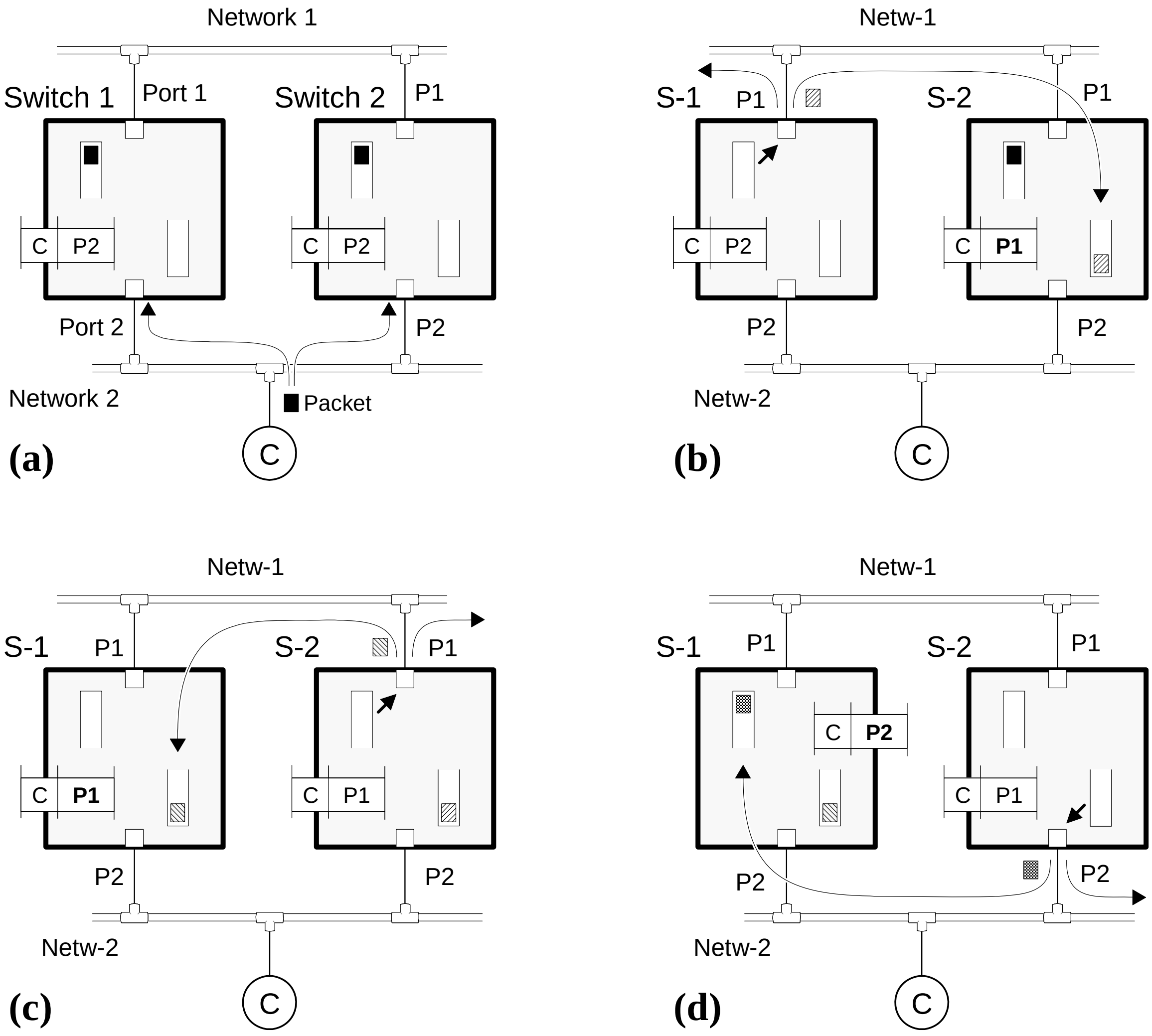
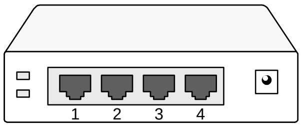


Figure 1-74: Packet proliferation in the network with a loop, from Figure 1-73.

Consider now the network in Figure 1-73. The two switches connect the two networks via two alternative paths, thus forming a *loop* (or, cycles) in the topology. This may happen by accidence, if the network administrator is not careful when upgrading the network, or with the purpose to provide for alternate paths in case of switch failures (fault tolerance by redundancy). Let us assume that the switching tables of both switches are as in Table 1-5 and that station *C* sends a frame to another station (the destination to which *C* sends is irrelevant for this example). The frame will arrive on both switches on Port-2, and each switch will record in its switching table that a packet from *C* arrived on Port-2, implying that *C* resides on Network 2. The frame will be enqueued for forwarding on Port-1 of each switch (Figure 1-74(a)). Suppose that Switch 1 is the first to seize the access to the medium and succeed in relaying the frame to Network 1. Because switches are transparent to each other and do not add any information into data frames, the frame will appear on Port-1 of Switch 2 exactly as if transmitted by station *C*. Switch 2 will record in its table that *C* arrived on Port-1 (as if *C* now resides on Network 1!) and enqueue the frame for forwarding on its Port-2 (Figure 1-74(b)). Next, suppose that Switch 2 succeeds in transmitting its first received frame onto Network 1 (Figure 1-74(c)). Now Switch 1 will record that *C* moved to Port-1 and enqueue the frame on its Port-2. Figure 1-74(d) shows one more iteration, where

Switch 2 transmits its second received frame onto Network 2, and this process continues forever. Note also that during this process any frames from other stations heading to station *C* may be misdirected and eventually discarded.

The solution to the problem of loops/cycles in a general graph is to remove them and produce a tree topology. A **spanning tree** of a graph is a subgraph of this graph that connects (spans) all the nodes, but contains no cycles. That is, a spanning tree keeps all the nodes of the original graph, but removes some links. In Ethernet networks, each LAN segment corresponds to a graph node, and each switch corresponds to a link in the graph. A spanning tree of a network can be derived automatically using the *spanning tree protocol* (STP), specified in the IEEE 802.1D standard.



Each switch in the network sends a configuration message on all of its attached networks, which includes the MAC address of the switch. The configuration message carries a unique switch identifier and a port identifier from which the configuration message was sent. The *switch identifier* consists of the switch's MAC address (48-bit binary number) and a configurable priority number (16 bits). The *port identifier* consists of the port number (12 bits) and a priority number (4 bits). The switches use the Spanning Tree Protocol to compute the spanning tree, which has these five steps:

1. Elect a root switch. Each switch chooses the switch with the smallest (lowest) identifier currently known to it as the *root switch* of the spanning tree. To compare two IDs, the priority is compared first. If two switches have equal priority, then their MAC addresses are compared and the switch with the smaller address is chosen as the root switch. The root switch always forwards frames out over all of its ports.

2. Compute the shortest path to the root. Each switch determines the cost of each possible path from itself to the root. (This cost may be incorrect initially during the convergence process.) From these paths, it selects one with the smallest cost ("shortest path"). The cost of traversing a path is the sum of the costs of the LAN segments on the path. Different technologies have different default costs for LAN segments. A common approach is to assign to each segment the cost of 1 (i.e., one hop). The port connecting this switch to the shortest path becomes the *root port* of this switch. All shortest paths in the given Ethernet network form a spanning tree.

3. Determine any designated ports. All switches on a LAN segment collectively decide which one among them has the shortest path to the root. The elected switch becomes the *designated switch* that will be responsible for forwarding frames from this LAN segment toward the root switch. The port connecting the designated switch to the given LAN segment becomes a *designated port* of this switch. A switch may have no designated ports or may have more than one designated port (because each switch is connected to several LAN segments).

4. Disable all other ports. Every switch *blocks* all of its active ports that do not qualify as a root port or a designated port. In case there are ties, go to the next step.

5. Resolve the ties. It may happen that two or more ports on a single switch are attached to shortest paths to the root or two or more switches on the same LAN segment have equal least-cost paths to the root. Such ties are broken as follows:

5.a) Breaking ties for root ports. When there are several shortest paths from a switch, choose the neighbor switch with the lowest identifier. The root port is thus the one connecting to this neighbor switch.

5.b) Breaking ties for designated ports. When more than one switch on a segment has a shortest path to the root, the switch with the smallest identifier is chosen as designated to forward messages to the root. The port attaching the designated switch to the LAN segment is its designated port. A loser switch sets the port to the given LAN segment as being blocked.

5.c) The final tiebreaker. In some cases, there may still be a tie, as when two switches are connected by multiple cables. In this case, multiple ports on a switch are candidates for root port. The switch chooses the path that passes through the lowest-priority port on the neighbor switch.

The switches run the STP protocol iteratively and exchange *configuration messages* using special data frames called Bridge Protocol Data Units (BPDUs). A BPDU frame carries this information:

BPDU format:

Protocol ID	Protocol version ID	BPDU type	Flags	Root ID	Root path cost	Switch ID	Port ID	Message age	Max age	Hello time	Forward delay
2 bytes	1 byte	1 byte	1 byte	8 bytes	4 bytes	8 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes

Particularly important for STP are:

(1) *Switch ID*—the identifier for the switch sending the message (includes the MAC address and a configurable priority number);

(2) *Rot ID*—the identifier for what the sending switch believes to be the root switch;

(3) *Root path cost*—the distance from the sending switch to the root switch, usually measured in hops; the IEEE 802.1D standard also specifies the value ranges depending on the link data rate, with the full range of 1–200,000,000, from fastest links (value 1 for 10 Tbps) to slowest links (value 200,000,000 for ≤ 100 Kbps).

Before configuration messages from all switches are received, some switches may have made incorrect choice of the root switch due to insufficient information. Initially, each switch thinks itself is the root, and so it sends a configuration message out on each of its ports identifying itself to others as the root, with the root path cost set to 0. When a switch receives a message on a port, the switch checks if this message is better than the best configuration message previously recorded for this port. As detailed in the STP algorithm, the message is considered "better" if:

- It identifies a root with a smaller identifier, or
- It identifies the same root but with a shorter distance (lower cost path), or
- It identifies the same root and distance, but the sending switch has a smaller identifier.

When a switch decides that it is not the root switch, it stops sending own configuration messages and only forwards messages from other switches. Similarly, when a switch decides that it is not the designated switch for a given LAN segment, it stops sending configuration messages over the port attached to this segment. When the system stabilizes, only the root switch will be generating configuration messages and all the other switches will be forwarding these messages only over the ports for which they are the designated switch.

When a link directly connected to a switch fails, the failure could be detected in two ways: by sensing signal loss at the physical layer, or by missing BPDU frames for a specified amount of time. The switch will react to link failure only if the port connected to this link is "root" port—the

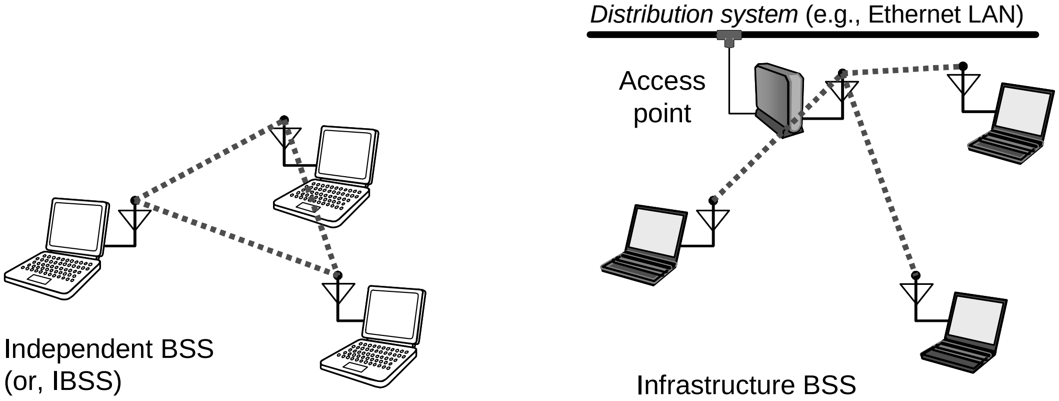

Distribution system (e.g., Ethernet LAN)
Access point
Independent BSS (or, IBSS)
Infrastructure BSS

Figure 1-75: IEEE 802.11 (Wi-Fi) independent and infrastructure basic service sets (BSSs).

switch will ignore a link failure event if the associated port is “blocking” or “designated.” For a link failure on a “root” port, the switch will first try to identify a new “root” port based on stored information. If such port cannot be found, the switch declares itself as “root switch” and starts announcing this in its BPDUs. Downstream switches will ignore this message until their old information about the “root switch” becomes outdated, at which point they will join the reconfiguration process. Similarly, when a new switch is connected to the network, the new switch will start sending BPDUs declaring itself as “root switch” and triggering the reconfiguration process.

Spanning tree protocol (STP) assumes a single network administrator and no adversary spoofing of switch priorities. STP and switching-table learning algorithm work completely independently of each other. First STP configures the network topology to remove loops. On top of this topology, but completely unaware and independent of it, the learning algorithm learns the switching table that associates destination addresses with outgoing ports on a switch. This independent operation of the two mechanisms is in contrast to routers, where the forwarding table (FIB) is derived from the routing table (RIB).

1.5.3 Wi-Fi (IEEE 802.11)

Problems related to this section: Problem 1.46 → Problem 1.48

Wi-Fi is any *wireless local area network* (WLAN) that is based on the evolving family of IEEE 802.11 standards. This section mainly covers 802.11b, which is the most basic and considered “legacy.” More recent variants are described in Section 6.3.1. A device that uses Wi-Fi (such as a personal computer, video-game console, smartphone, digital camera, tablet computer, or digital audio player) can connect to the Internet via a wireless network access point. Such an access point (or “hotspot”) has a range of about 20 meters indoors and a greater range outdoors. Wi-Fi also allows communications directly from one computer to another without an access point intermediary, forming a so-called “ad hoc network.”

Architecture and Basics

The basic building block of an IEEE 802.11 network is the **basic service set** (BSS), which is simply a set of stations that communicate with one another. A BSS does not generally refer to a

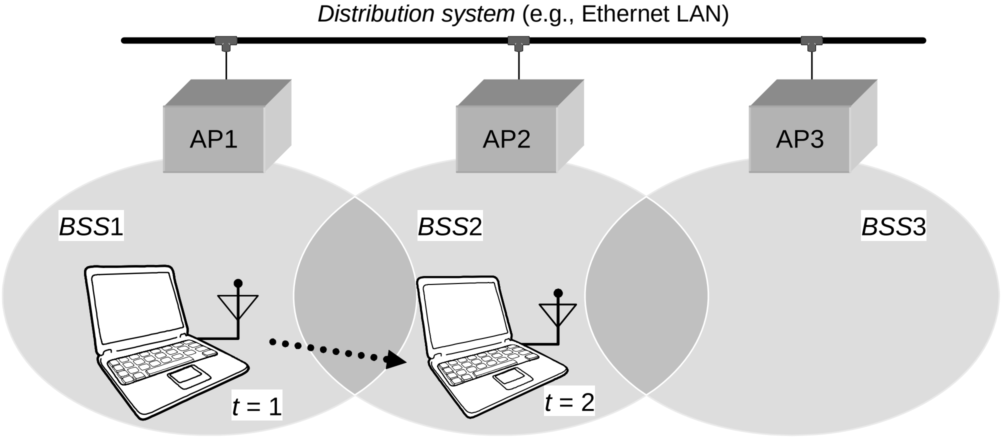


Figure 1-76: IEEE 802.11 (Wi-Fi) extended service set (ESS) allows connecting multiple access points to support long-range roaming.

particular area, because of the uncertainties of electromagnetic propagation. There are two types of BSS, as shown in Figure 1-75. When all of the stations in the BSS are wireless and there is no connection to a wired network, the BSS is called an **independent BSS** (or, IBSS). The IBSS is the entire network and only those stations communicating with each other in the IBSS are part of the LAN. This type of network is called an **ad hoc network** (see Chapter 6).

When all of the mobile stations in the BSS communicate with an **access point** (AP), the BSS is called an **infrastructure BSS** (never called an IBSS!). This configuration is also known as *wireless local area network* or W-LAN. The access point provides both the connection to the wired LAN (wireless-to-wired bridging), if any, and the local relay function for all stations in its BSS. Therefore, if one mobile station in the BSS must communicate with another mobile station, the packet is sent first to the AP and then relayed from the AP to the destination station. This causes communications to consume more transmission capacity than in the case where the communications are directly between the source and the destination (as in the IBSS). However, in many cases the benefits provided by the AP outweigh the drawbacks. One of the benefits provided by the AP is that the AP can assist the mobile stations in saving battery power. The mobile stations can operate at a lower power, just to reach the AP, and not worry about how far away is the destination host. Also, the AP can buffer (temporarily store) the packets for a mobile station, if the station is currently in a power saving mode and not listening to arriving packets.

Extended service set (ESS) extends the range of mobility from a single infrastructure BSS Figure 1-75 to an arbitrary range by interconnecting a set of infrastructure BSSs (Figure 1-76). In ESS, multiple APs communicate among themselves to forward traffic from one BSS to another and to facilitate the roaming of mobile stations between the BSSs. This is conceptually similar to the cellular telephony network and can be implemented in a campus network with overlapping access points. The APs perform this communication via the *distribution system*, such as an Ethernet-based wireline network. The stations in an ESS see the wireless medium as a single link-layer connection. ESS is the highest-level abstraction supported by 802.11 networks. Roaming between geographically distributed ESSs is not supported by IEEE 802.11 and must be supported by a higher-level protocol, e.g., Mobile IP (Section 9.3.4).

Wi-Fi physical layer (PHY) defines several transmission rates; the lowest rate is 1 Mbps, and the other rates are different for different variants of the 802.11 standard. Wi-Fi supports dynamic

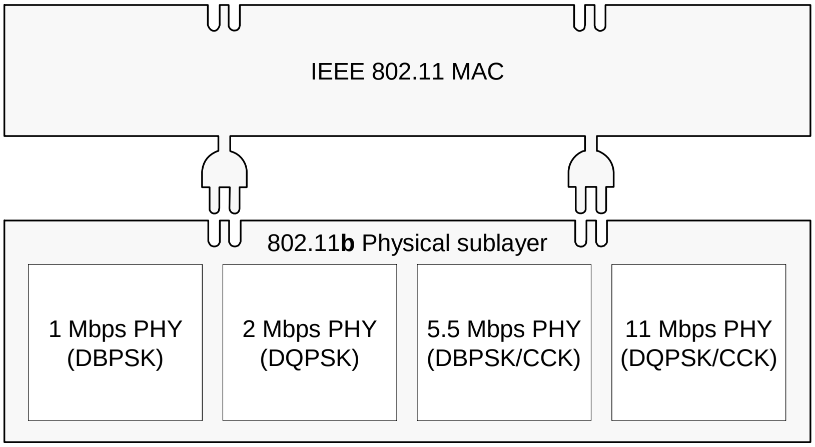
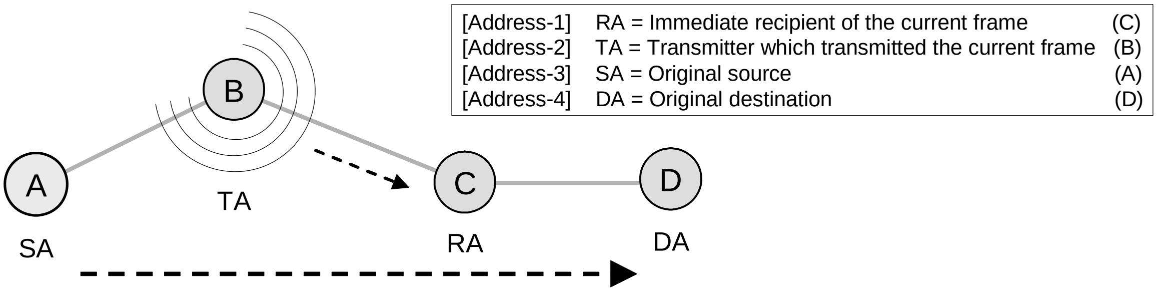


Figure 1-77: Wi-Fi supports dynamic data-rate adaptation at the physical sublayer. This figure shows the available rates for 802.11b.

data-rate adaptation to the current conditions of the wireless channel (Figure 1-77). Unlike wire media, where channel conditions are stable and error rate is low, wireless channel state changes dynamically. The goal is to select the data rate that minimizes the errors due to the channel noise. This behavior is not implemented in Ethernet, which operates over wire media. Ethernet's physical data rate is pre-configured and does not change at runtime (compare to Figure 1-70).

Figure 1-78 shows the 802.11 frame format. The general MAC-layer format (Figure 1-78(a)) is used for all data and control frames, but not all fields are used in all types of frames. 802.11 uses the same 48-bit MAC address format as Ethernet (Section 1.5.2). There can be up to four address fields in an 802.11 frame. One of the fields could be the BSS identifier, used in the probe request and response frames, when mobile stations scan an area searching for an access point.

In the most general case, when all four address fields are present, their meaning is as follows. Consider an ad hoc network where station *A* needs to send a frame to station *D*, via intermediary stations *B* and *C*. We need four addresses to successfully deliver the frame:



[Address-1]	RA = Immediate recipient of the current frame	(C)
[Address-2]	TA = Transmitter which transmitted the current frame	(B)
[Address-3]	SA = Original source	(A)
[Address-4]	DA = Original destination	(D)

The *Duration*/*Connection-ID* field indicates the time (in microseconds) the channel will be reserved for successful transmission of a data frame. The stations that receive this frame, but are not intended receivers, use this information to defer their future transmissions until the current frame transmission is completed. The deferral period is called **network allocation vector** (NAV), and we will see later in Figure 1-84 how it is used. In some control frames, this field contains a network association, or connection identifier, in case of multiple access points to know which AP the node is currently associated to.

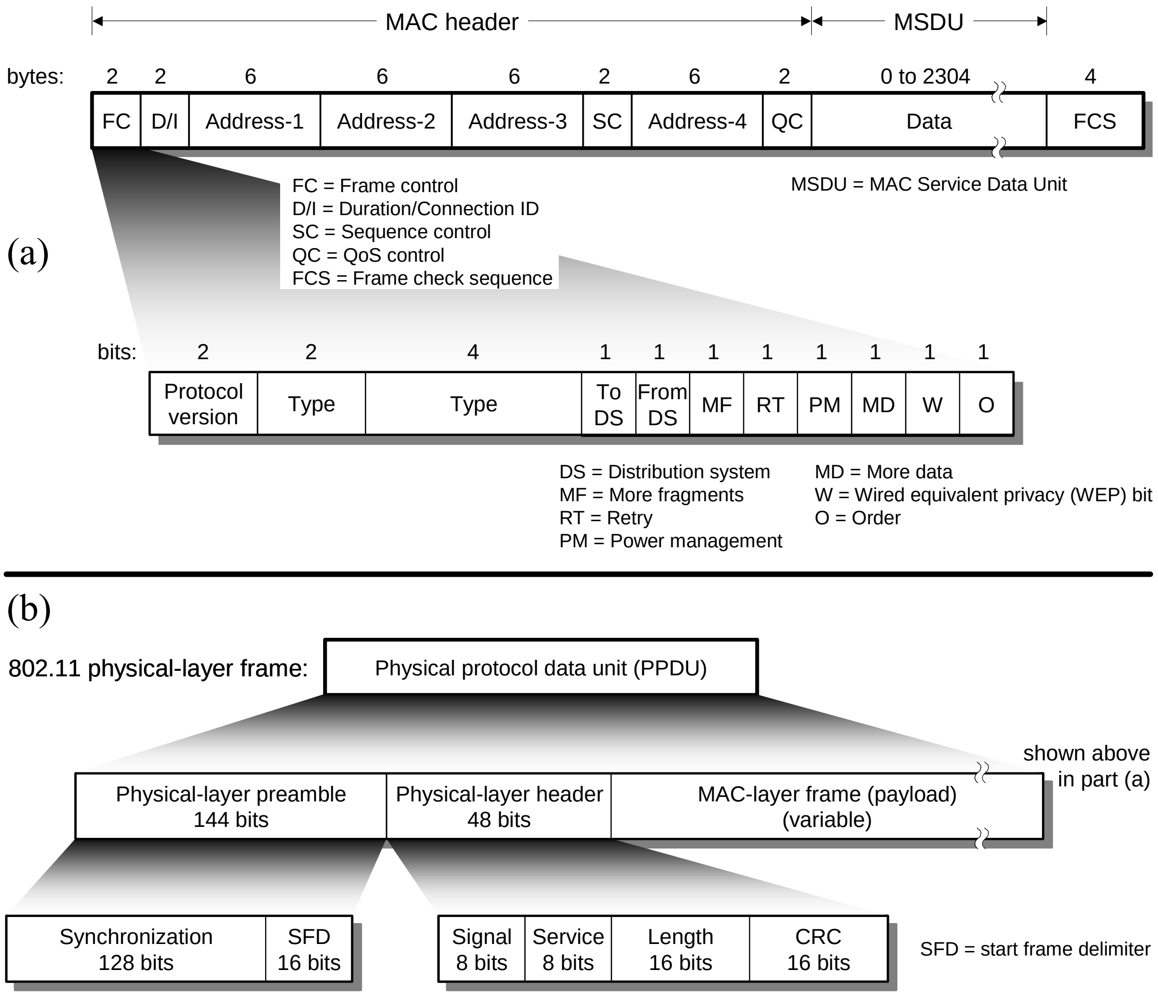


Figure 1-78: IEEE 802.11 (Wi-Fi) frame formats. (a) Link-layer (or, MAC-layer) frame format. (b) Physical-layer frame format (also known as Long PPDU format).

In the infrastructure BSS case (Figure 1-75), only three addresses are used along with the bits "To DS" and "From DS" (Figure 1-78(a)). If the frame has the "To DS" bit set, then Address 3 is the destination address; if "From DS" set to 1, then Address-3 is the original source address.

The 802.11 physical-layer frame (Figure 1-78(b)) is known as PLCP protocol data unit (PPDU), where PLCP stands for "physical (PHY) layer convergence procedure." The version shown in Figure 1-78(b) is known as *Long PPDU format*. A **preamble** is a bit sequence that allows a receiver to distinguish silence from transmission periods and detect the beginning of a new packet. Receivers watch for a preamble to lock onto the rest of the frame transmission. (We encountered preambles in Section 1.1.4, as well as in Ethernet, Figure 1-66.) There are two different preamble and header formats defined for 802.11 physical-layer frames. The mandatory supported long preamble and header, shown in Figure 1-78(b), is interoperable with the basic 1 Mbps and 2 Mbps data transmission rates. There is also an optional short preamble and header (not illustrated here), known as *Short PPDU format*. This format is used at higher transmission rates to reduce the control overhead and improve the network performance. (More discussion is provided in Chapter 6.)

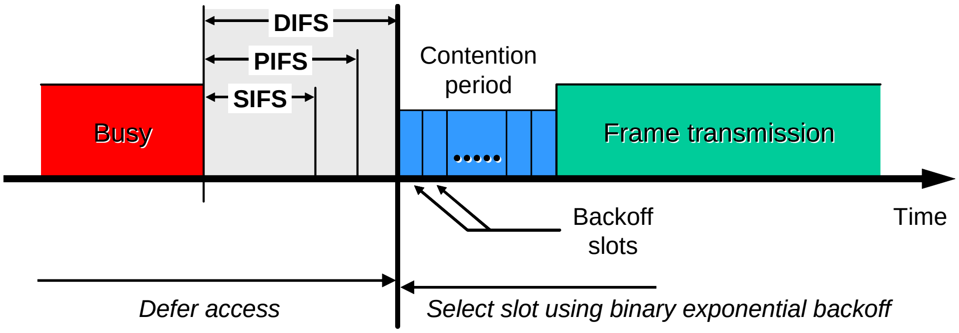


Figure 1-79: The timing diagram for IEEE 802.11 frame transmissions. The *collision-avoidance* (or "contention") period follows the *interframe space* (xIFS) and contains a random number of backoff slots. The IFS length is used to control the priority of a station.

Medium Access Control (MAC) Protocol

The medium access control (MAC) protocol for IEEE 802.11 is a CSMA/CA protocol (Section 1.3.3). A CSMA/CA sender tries to avoid collision by introducing a variable amount of delay before starting with transmission. This is known as the *access deferral state*. Similar to an Ethernet adapter (Section 1.5.2), a Wi-Fi adapter needs time to decide that the channel is idle. Again, this period is known as *interframe space* (IFS). However, unlike Ethernet, the IFS delay is not fixed for all stations to 96-bit times. Wi-Fi has an additional use of the IFS delay, so that it can differentiate stations of different *priority*. Each station must delay its transmission according to the IFS period assigned to the station's priority class. A station with a higher priority is assigned a shorter interframe space and, conversely, lower priority stations are assigned longer IFSs. The idea behind different IFSs is to create different priority levels for different types of traffic. Then, high-priority traffic waits less time after the medium has become idle. If there is any high-priority traffic, it grabs the medium before a lower-priority station has a chance to try.

When a station wants to transmit data, it first senses if the medium is busy. Three rules apply:

1. If the station found the medium idle, and it stayed idle for the duration of IFS corresponding to this station's priority level, the transmission can begin immediately.
2. If the medium is busy, the station continuously senses the medium, waiting for it to become idle. When the medium becomes idle, the station enters the *collision-avoidance state* (or, *access deferral state*). The station can transmit the packet if the medium is idle after the collision-avoidance period expires.
3. If during the collision-avoidance state the channel is sensed busy (because another station transmitted), this station will remain silent for the duration of that transmission.

The event sequence and IFS relationships are illustrated in Figure 1-79. First, the station waits for its assigned IFS duration, which may be different for different stations, but once assigned it is fixed. After IFS, the station enters collision-avoidance by setting a **contention timer** (or, "backoff timer") to a time interval randomly selected in the range [0, CW–1], and counting down to zero while sensing the carrier. If the carrier is idle when the backoff countdown reaches zero, the station transmits. If the carrier becomes busy before the backoff reaches zero, the station freezes the counter and resumes the countdown when the medium again becomes idle.

To assist with interoperability between different data rates (Figure 1-77), the interframe space is a fixed length of time, independent of the physical layer bit rate (even for the latest 802.11 standard, see Chapter 6). There are two basic intervals determined by the physical layer (PHY): the *short interframe space* (SIFS), which is equal to the parameter β (Section 1.3.3), and the *slot time*, which is equal to $2\times\beta$. To be precise, the 802.11 slot time is the sum of the physical-layer Rx-Tx turnaround time[12], the *clear channel assessment* (CCA) interval, the propagation delay on the air medium, and the link-layer processing delay. The station must sense the channel state (idle or busy) for a full slot duration.

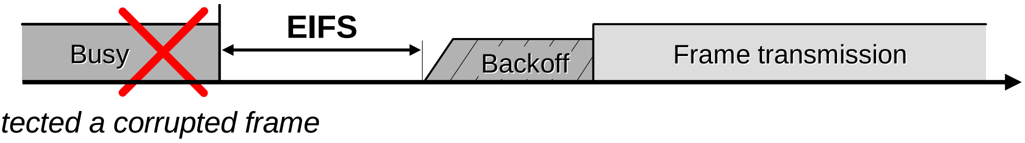
The four different types of IFSs defined in 802.11 (three of which are shown Figure 1-79) are:

SIFS: Short interframe space is used for the highest priority transmissions, such as control frames, or to separate transmissions belonging to a single dialog (including Data-and-ACK frames, Figure 1-80). This value is a fixed value per PHY and is calculated in such a way that the transmitting station will be able to switch back to the receive mode and be capable of decoding the incoming packet.

PIFS: PCF (or priority) interframe space is used by the *point coordination function* (PCF) during contention-free operation. The coordinator station (usually the Access Point) uses PIFS when issuing polls and the polled station may transmit after the SIFS has elapsed and preempt any contention-based traffic. PIFS is equal to SIFS plus one slot time.

DIFS: DCF (or distributed) interframe space is the minimum time that the medium should be idle for asynchronous frames contending for access. Stations may have immediate access to the medium if it has been idle for a period longer than the DIFS. DIFS is equal to SIFS plus two slot times.

EIFS: Extended interframe space (not illustrated in Figure 1-79) is much longer than any of the other interframe spaces. If a station has received a frame containing errors while waiting for transmission, then a station has to defer EIFS duration instead of DIFS before transmitting. This station could not decode the duration information of the received frame (D/I field in Figure 1-78) and correctly set its deferral period (defined later as "network allocation vector"). EIFS ensures that this station is prevented from colliding with a future acknowledgment frame belonging to the current dialog.



More IFS types are defined in recent variants of the 802.11 standard, see Chapter 6 for details. The values of some important 802.11b system parameters are shown in Table 1-6. The values shown are for the 1Mbps channel bit rate and some of them are different for other bit rates.

[12] *Rx-Tx turnaround time* is the maximum time (in μs) that the physical layer requires to change from receiving to transmitting the start of the first symbol. More information about the Rx-Tx turnaround time is available in: "IEEE 802.11 Wireless Access Method and Physical Specification," September 1993; doc: IEEE P802.11-93/147: http://www.ieee802.org/11/Documents/DocumentArchives/1993_docs/1193147.doc

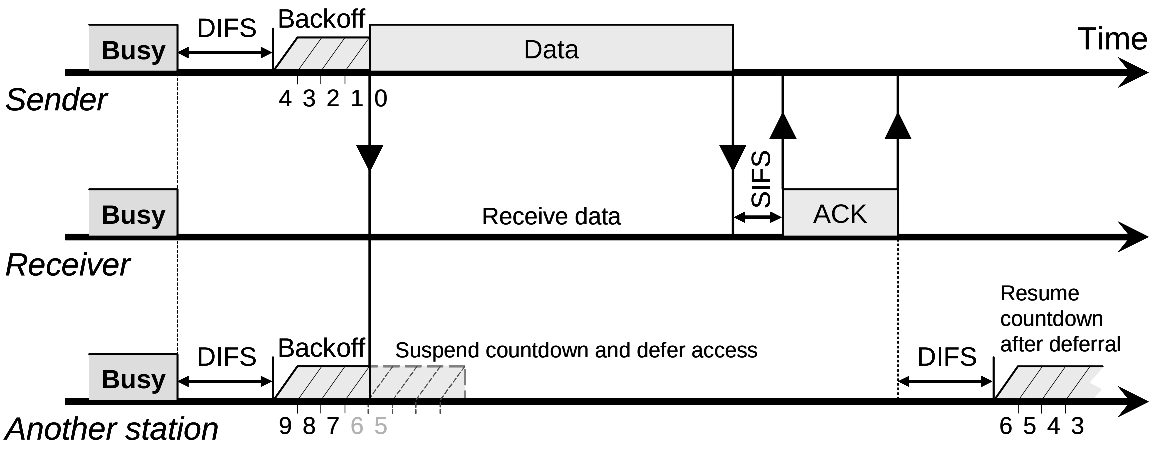


Figure 1-80: Timing diagram for the IEEE 802.11 basic transmission mode, which is based on the stop-and-wait ARQ. Note the backoff slot countdown during the contention period.

Table 1-6: Example IEEE 802.11b system parameters.

Parameter	Value for 1 Mbps channel bit rate
Slot time	20 μsec
SIFS	10 μsec
DIFS	50 μsec (DIFS = SIFS + 2 × Slot time)
EIFS	SIFS + PHY_preamble + PHY_header + ACK + DIFS = 364 μsec
CW_{min}	32 (minimum contention window size)
CW_{max}	1024 (maximum contention window size)
PHY_preamble	144 bits (144 μsec)
PHY_header	48 bits (48 μsec)
MAC data header	28 bytes = 224 bits
ACK	14 bytes + PHY_preamble + PHY_header = 304 bits (304 μsec)
RTS	20 bytes + PHY_preamble + PHY_header = 352 bits (352 μsec)
CTS	14 bytes + PHY_preamble + PHY_header = 304 bits (304 μsec)
CTS-timeout	SIFS + CTS + Slot time = 10 + 304 + 20 = 334 bits (334 μsec)
MTU*	Adjustable, up to 2304 bytes for frame body before encryption

(*) The Maximum Transmission Unit (MTU) size specifies the maximum size of a payload (in bytes) that a frame can carry, not counting the frame's header and trailer (see Section 1.1.3). MAC-layer MTU is also known as MSDU, for *M*AC *S*ervice *D*ata *U*nit (see Figure 1-78(a)). The reader may also encounter the numbers 2312, 2324, or 2320, which are 802.11 MTU sizes *after encryption*, depending on the encryption method. The maximum MAC frame size is 2346 bytes, known as MPDU, for *M*AC *P*rotocol *D*ata *U*nit which including the MAC header and FCS (Figure 1-78(a)). In practice, MSDU size seldom exceeds 1500 bytes because of the need to bridge with Ethernet.

Unlike Ethernet (Section 1.5.2) or PPP (Section 1.5.1), 802.11 implements a reliable transmission ARQ with acknowledgments and retransmissions. An example of an 802.11 frame transmission in the so-called **basic mode** is shown in Figure 1-80. Note that even the units of the atomic transmission (data and acknowledgment frames) are separated by SIFS, which is intended to give the transmitting station a short break so it will be able to switch back to receive mode and be capable of decoding the incoming (in this case ACK) frame.

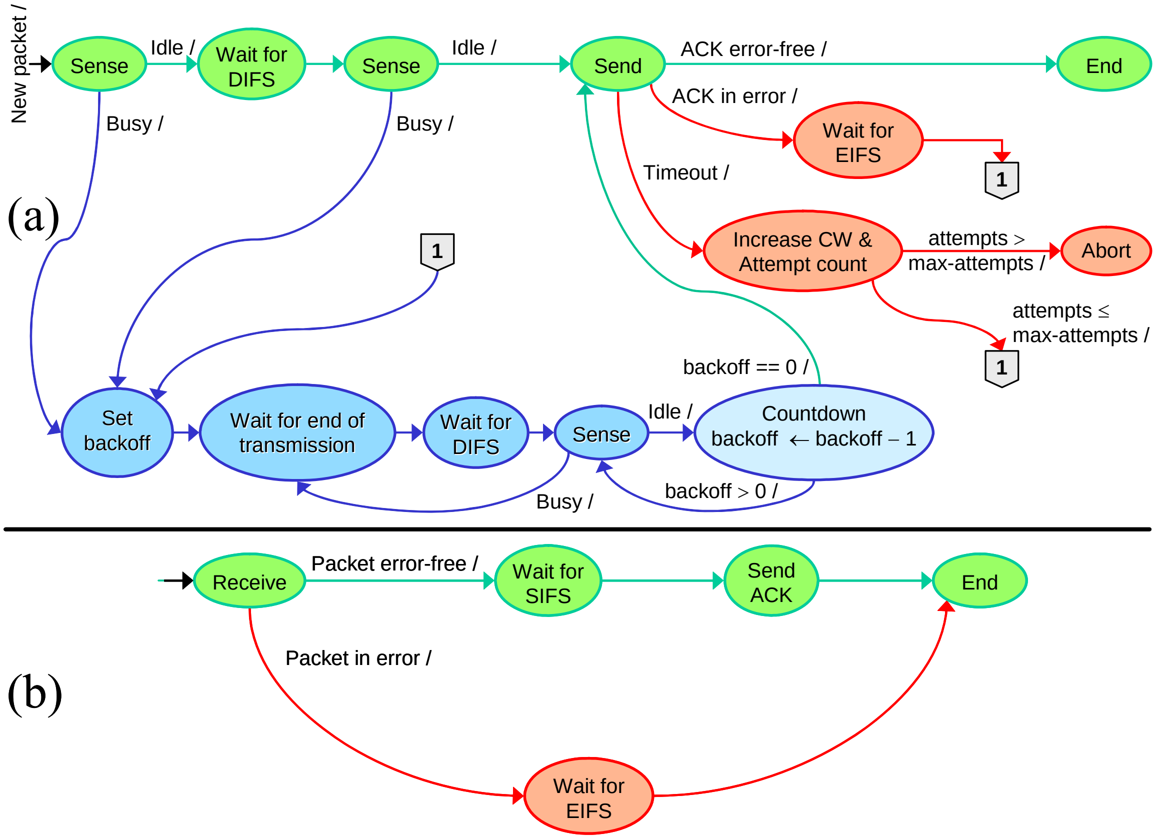


Figure 1-81: (a) Sender's state diagram for 802.11 MAC protocol basic mode. Compare to CSMA/CA in Figure 1-36. In "Set backoff," the backoff counter is set randomly to a number ∈ {0, …, *CW*−1}. (b) Receiver's state diagram for 802.11 MAC protocol.

The state diagrams for 802.11 senders and receivers are shown in Figure 1-81. Note that sender's state diagram is based on the CSMA/CA protocol shown in Figure 1-36, slightly modified.

Here is an example:

Example 1.6 Illustration of Timing Diagrams for IEEE 802.11 (Basic Mode)

Consider a local area network (infrastructure BSS) using the IEEE 802.11 protocol shown in Figure 1-81. Show the timing diagrams for the following scenarios:

(a) A single station has two frames ready for transmission on an idle channel.

(b) A single station has one frame ready for transmission on a busy channel. The acknowledgment for the frame is corrupted during the first transmission.

(c) A single station has one frame ready for transmission on a busy channel. The data frame is corrupted during the first transmission.

The solutions are shown in Figure 1-82. Sender's actions are shown above the time axis and receiver's actions are shown below the time axis. A crossed block represents a loss or erroneous reception of the corresponding frame.

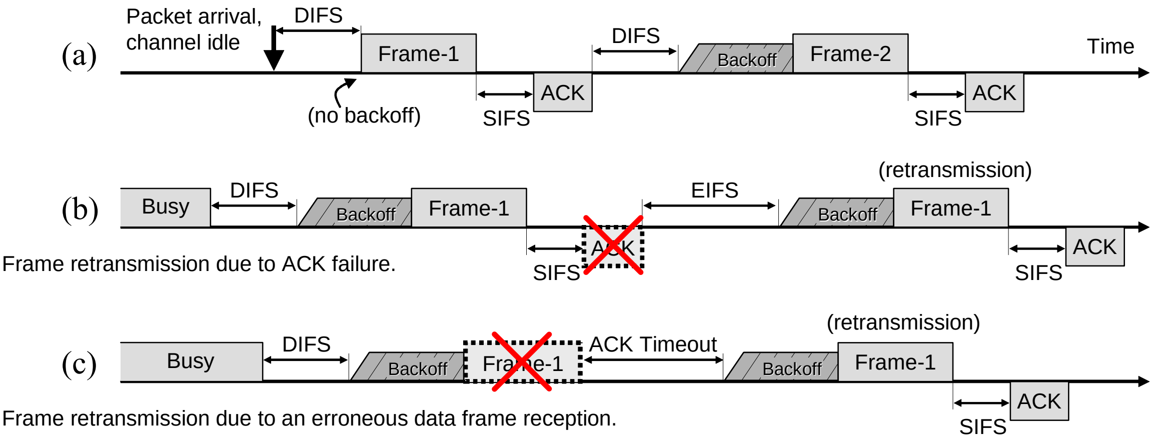


Figure 1-82: Timing diagrams for the 802.11 basic mode, showing the sender's activity above each timeline and receiver's activity below the timeline. (a) Timing of successful frame transmissions under the DCF. (b) Frame retransmission due to ACK failure. (c) Frame retransmission due to an erroneous data frame reception.

The timing of successful frame transmissions is shown in Figure 1-82(a). If the channel is idle upon the packet arrival, the station transmits immediately (after a mandatory DIFS), without backoff. The rationale is that because there was no previous transmission to wait to end, there is no reason that another station would synchronize and collide with this one. However, the station has to backoff for its own second transmission.

Figure 1-82(b) shows the case where an ACK frame is received in error, i.e., received with an incorrect frame check sequence (FCS). The transmitter re-contends for the medium to retransmit the frame after an EIFS interval. This is also indicated in the state diagram in Figure 1-81. The reason for EIFS is that our station has not received a (error-free) ACK; therefore, it need to (re-)transmit a data frame. However, it just detected an erroneous frame (corrupted ACK). As a result, before contending for the medium for retransmission, the station has to wait EIFS instead of DIFS.

On the other hand, if no ACK frame is received within a timeout interval (even a corrupted ACK), due possibly to an erroneous reception at the receiver of the preceding data frame, as shown in Figure 1-82(c), the transmitter contends again for the medium to retransmit the frame after an ACK timeout. (Note that the ACK timeout is much shorter than the EIFS interval; in fact, ACK_timeout = $t_{\mathrm{SIFS}} + t_{\mathrm{ACK}} + t_{\mathrm{slot}}$. Check Table 1-6 for the values.)

Hidden Stations Problem

The hidden and exposed station problems are described earlier in Section 1.3.3. A common solution is to induce the receiver to transmit a brief "warning signal" so that other potential transmitters in its neighborhood are forced to defer their transmissions. IEEE 802.11 extends the basic access method (Figure 1-80) with two control frames: *request-to-send* (RTS) and *clear-to-send* (CTS) frames, which are very short frames exchanged before the data and ACK frames. (Frame lengths, including RTS and CTS durations, are shown in Table 1-6.) The process is shown in Figure 1-83, where node *A* needs to send data to node *B* and node *C* is hidden to *A* and may potentially interfere with its transmission. Sender *A* first sends the RTS frame to receiver *B* (Figure 1-83(a)). If the transmission succeeds, receiver *B* responds by a short frame (CTS). The

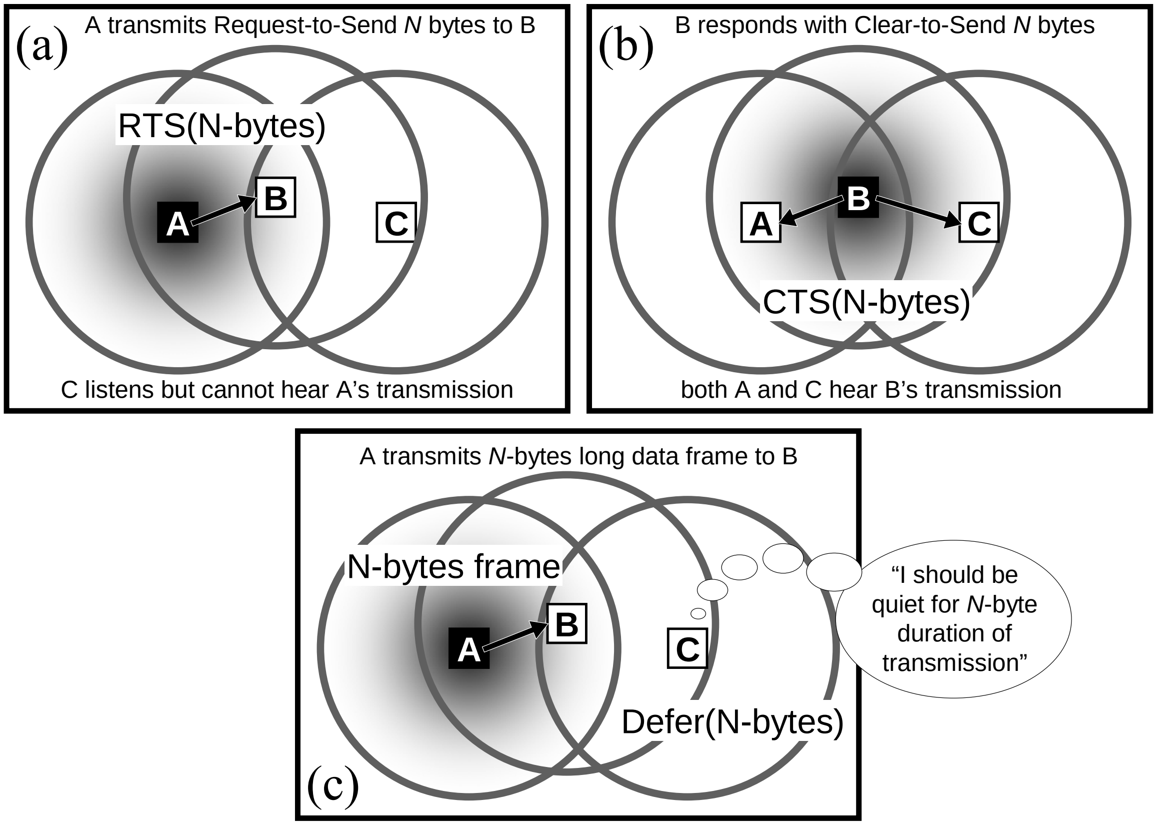

(a)
A transmits Request-to-Send *N* bytes to B
RTS(N-bytes)
A
B
C
C listens but cannot hear A's transmission
(b)
B responds with Clear-to-Send *N* bytes
B
A
C
CTS(N-bytes)
both A and C hear B's transmission
A transmits *N*-bytes long data frame to B
N-bytes frame
A
B
C
"I should be quiet for *N*-byte duration of transmission"
Defer(N-bytes)
(c)

Figure 1-83: IEEE 802.11 protocol is augmented by RTS/CTS frames to reserve the channel and prevent interference by hidden stations. The "N-bytes" information informs the overhearing stations about the duration of the forthcoming transmission.

CTS frame is intended not only for sender *A*, but also for all other stations in the receiver's range (Figure 1-83(b)). All stations that receive a CTS frame know that this frame signals a transmission in progress and must avoid transmitting for the duration of the upcoming data frame. Through this indirection, the sender performs "floor acquisition" so it can speak unobstructed because all other stations will remain silent for the duration of transmission (Figure 1-83(c)).

The 4-way handshake of the RTS/CTS/DATA/ACK exchange of the 802.11 protocol (Figure 1-84) requires that the roles of sender and receiver be interchanged several times between a pair of communicating nodes, so neighbors of both these nodes must remain silent *during the entire exchange*. Because some stations may be hidden from others (e.g., in Figure 1-83 stations *A* and *C* and hidden to each other), the stations cannot rely on the regular carrier sense mechanism. For this purpose, 802.11 introduced the so called **virtual carrier sense** mechanism. Each frame carries in its D/I field of the MAC header (Figure 1-78(a)) the remaining duration (in microseconds) needed to receive the next frame transmission. The neighboring nodes that overhear the RTS or CTS frames will set their *network allocation vector* (NAV) values from the D/I value to ensure that atomic operations are not interrupted. Figure 1-84 indicates how the NAV duration is set for RTS, CTS, data and ACK frames.

The additional RTS/CTS exchange shortens the vulnerable period from the entire data frame in the basic method (Figure 1-80) down to the duration of the RTS/CTS exchange in the RTS/CTS method (Figure 1-84). If a "covered station" transmits simultaneously with the sender, they will

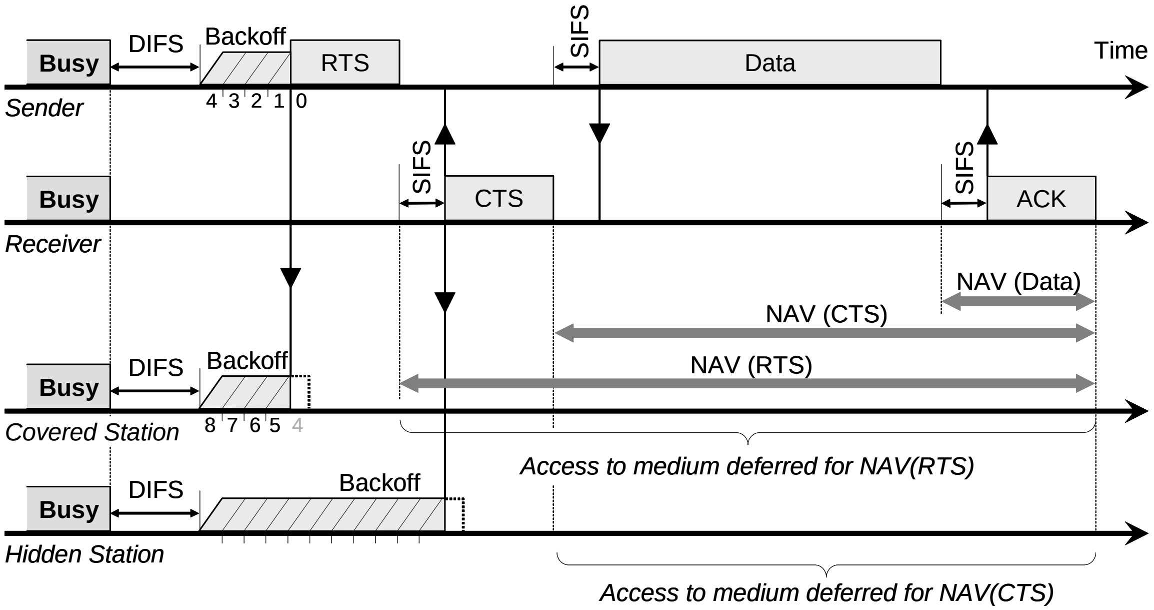


Figure 1-84: The 802.11 protocol atomic unit exchange in RTS/CTS transmission mode consists of four frames: RTS, CTS, Data, and ACK. (Compare to Figure 1-80.)

collide within the RTS frame. If a hidden station overhears a CTS frame, it knows that another station requested a transmission reservation and it will *not* interfere with the subsequent data frame transmission. In either case, the sender will detect collision by the lack of the CTS frame. If collision happens, it will last only a short period because RTS and CTS frames are very short relatively to data frames. (For example, according to Table 1-6, RTS equals 352 bits or 44 bytes, which is 18.1 slot times.)

Note, however, that within an access point area ("infrastructure BSS") some stations may be communicating in the RTS/CTS access mode and others in the basic access mode (Figure 1-80). The NAV vector is set only in the RTS/CTS mode and *not* in the basic mode, because RTS/CTS frames perform channel reservation for the subsequent data frame. To a station that operates in the basic mode, an RTS/CTS/DATA/ACK exchange would appear as two successive transmissions. The vulnerable period lengths would be different, as well, depending on the access mode in which a given station operates.

This RTS/CTS exchange partially solves the hidden station problem but the exposed station problem remains unaddressed. The hidden station problem is solved only partially, because if a hidden station starts its transmission *simultaneously* with the CTS frame from the receiver, the hidden station will not hear the CTS frame, the sender will receive the CTS frame correctly and start with the data frame transmission, and this will result in a collision at the receiver. (Of course, the probability of this event is very low.)

802.11 RTS/CTS protocol does not solve the exposed station problem (Figure 1-32(b)). Exposed stations could maintain their NAV vectors to keep track of ongoing transmissions. However, if an exposed station gets a packet to transmit while another transmission is in progress, it is allowed to transmit for the remainder of the NAV, before the sender needs to receive the ACK. Tailoring the frame to fit this interval and accompanied coordination is difficult and is not implemented as part of 802.11.

Recent extensions in the evolution of the 802.11 standard are described in Chapter 6.

1.6 Quality of Service Overview

The focus of this chapter has been on *what* networks do and *how* they do it. We also need to consider ***how well*** they do their work, particularly for multimedia applications. Multimedia applications deal with analog signals of sound, light, touch, etc. Unlike symbolic data which can be experienced in abstract chunks without strict dependence on time, analog signals require faithful reproduction in time for best experience.

The Internet Protocol (IP, Section 1.4.1) is ubiquitous and very powerful because it is simple. IP just tries to deliver packets to their destination; it does not make any promises about *how well* the packets will be delivered or, in other words, it does not try to do anything complicated. Such approach (known as “best effort”) works well for wired networks and traditional data applications. All a network needs to know is where we want our packets delivered. With the advent of wireless networks and multimedia applications, people became aware of the “how well” issue (or, “quality of service”). Because the old telephone network dealt with a multimedia application (spoken conversations) from the outset, the telephone networkers have been well aware of quality issues and effective ways of dealing with it. On the other hand, telephone network never had to deal with such complexity of scenarios and diversity of applications as the Internet does. This raises the question of whether the Internet should provide better-quality service to some packets. The answer is not straightforward and to understand the issues we will need to understand the mechanisms for providing differentiated quality of service.

Users exchange information through computing applications. A distributed application at one end accepts information, uses the network for transport, and presents the information at the other end. Therefore, it is common to talk about the characteristics of information transfer of different applications. These characteristics describe the traffic that the applications generate as well as the acceptable delays and information losses by the intermediaries (network) in delivering that traffic.

Figure 1-85 illustrates multimedia information delivery over the network. The recurring themes for Quality-of-Service (QoS) include ***packet loss, delay and their statistical properties***. We have seen that the network capacity is subject to *physical or economic constraints* and for this reason many customers share the same network resources (“statistical multiplexing,” Section 1.1.3). Because customer usage patterns are random and changing over time, statistical multiplexing inevitably results in random packet delays and loss. Users working with applications based on symbolic data (known as *elastic applications*, such as email or Web browsing) can tolerate significant delays and lost packets can be retransmitted, if important. One the other hand, users of multimedia-based applications (such as telephone calls or video streaming) are much more sensitive to delays and lost packets.

A complement of delay is *capacity*, also referred to as *bandwidth*. Constraints on the network capacity are imposed by physical and economic limitations. Constraints on the delay (also referred to as *latency*), on the other hand, are imposed subjectively by the task of the information

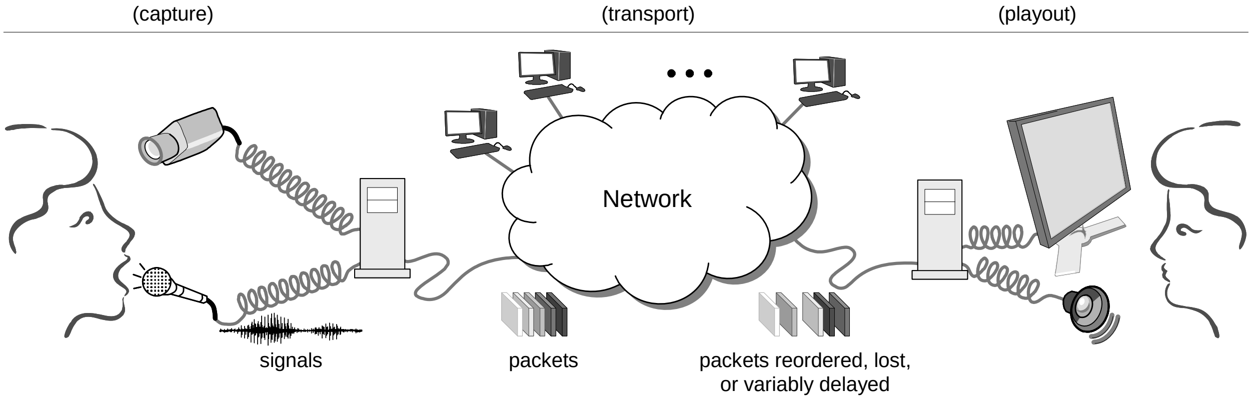


Figure 1-85: Conceptual model of multimedia information delivery over the network.

recipient—information often loses value for the recipient if not obtained in a timely manner. In addition to actual packet loss, packets that do not arrive within the delay bounds are considered lost. A key issue is how to limit the delay so it meets the application requirements. Therefore, three relevant forces shape the design of a QoS solution:

(1) Desired application performance

(2) Bandwidth needed to provide that performance

(3) Complexity or cost of the required mechanisms

The information that applications generate can take many forms: text, audio, voice, graphics, pictures, animations, and videos. In addition, the information transfer may be one-way, two-way, broadcast, or multipoint. Although timeliness is always important, there are different degrees of time constraints, such as soft, hard, as well as their statistical properties. Often compromises must be made and a sub-optimal service agreed to as a better alternative to unacceptable delays or no service at all. Generally, elastic applications can take advantage of however much or little bandwidth is available. If all information must be received without loss, then any loss can be compensated by retransmission. By contrast, inelastic applications require a minimum level of bandwidth and a maximum latency to function. Any retransmission may introduce unacceptable latencies. Various parameters that need to be considered are shown in Figure 1-86. IP networks are subject to much impairment, including:

* Packet loss due to network congestion or corruption of the data

* Variation in the amount of delay of packet delivery, which can result in poor voice quality

* Packets arriving out of sequence, which can result in discarded packets and cause more delay and disruption

* Hardware and software failures

1.6.1 Performance Bounds and Quality of Service

There is more than one way to characterize quality-of-service (QoS). Generally, QoS is the ability of a network element (e.g., an application, a host or a router) to provide some level of assurance

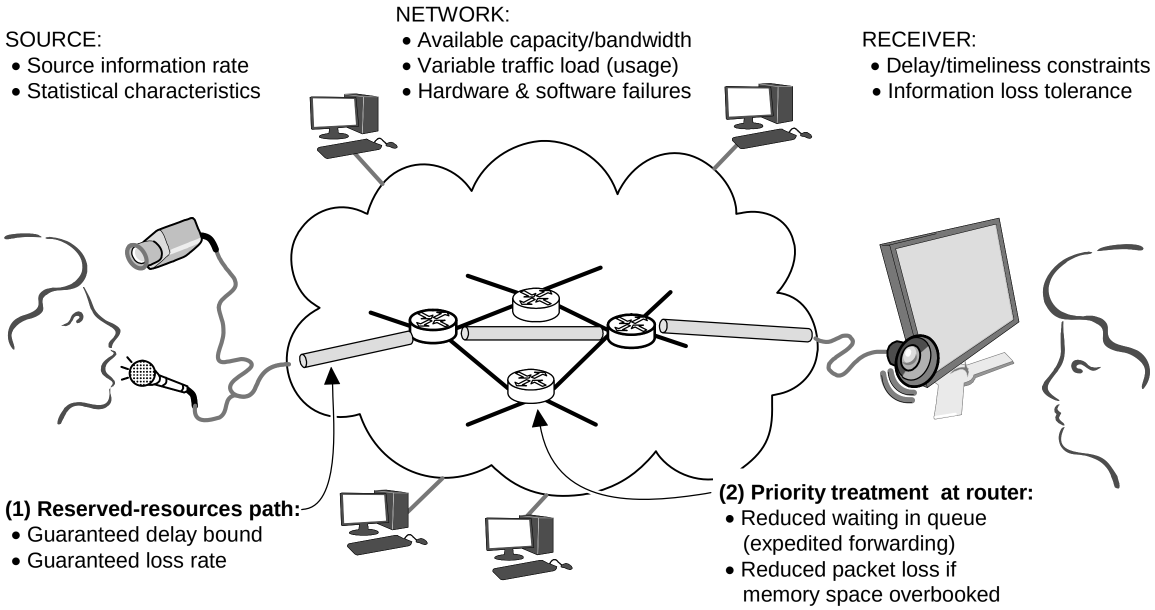

SOURCE:
• Source information rate
• Statistical characteristics
NETWORK:
• Available capacity/bandwidth
• Variable traffic load (usage)
• Hardware & software failures
RECEIVER:
• Delay/timeliness constraints
• Information loss tolerance
(1) Reserved-resources path:
• Guaranteed delay bound
• Guaranteed loss rate
(2) Priority treatment at router:
• Reduced waiting in queue (expedited forwarding)
• Reduced packet loss if memory space overbooked

Figure 1-86: Key requirements and approaches to quality of service assurance.

for consistent network data delivery. Some applications are more stringent about their QoS requirements than other applications.

Network performance bounds can be expressed either *deterministically* or *statistically*. A deterministic bound holds for every packet sent on a connection. One the other hand, a statistic bound characterizes the overall network performance. For example, a deterministic delay bound of 200 ms means that every packet sent on a connection experiences an end-to-end (from sender to receiver) delay smaller than 200 ms. On the other hand, a statistical bound of 200 ms with a parameter of 0.99 means that the probability that a packet will be delayed by more than 200 ms is smaller than 0.01.

Network operator can guarantee performance bounds for a connection only by provisioning sufficient network resources, either in advance by building extra capacity, or dynamically by reservation during the connection-establishment phase before use.

QoS guarantees: hard or soft

QoS can be provided in absolute or relative terms. Absolute QoS provides certain guarantees to a set of participants, regardless of other participants. Relative QoS offers the favored packets lower delay and lower loss rate *relative* to less favored packets.

Two approaches are considered for addressing the needs of inelastic applications:

- Add more capacity to the network
- Provide preferential service within the network

Adding capacity is a simple solution, but involves economic costs. The provision of preferential service options may look like a better solution. For example, a heavily loaded router can reduce the delay for some packets by dropping some of the other packets that are waiting for transmission. However, the decision making as to which packets should be dropped and which ones should be expedited is very complex to implement and difficult to evaluate for effectiveness. The following chapters will describe the mechanisms involved in providing tiered services.

Two basic approaches have been devised for network-based quality of service (Figure 1-86):

- **Prioritization** (known as "differentiated services"): network traffic is classified and apportioned network resources according to bandwidth management policy criteria. To enable QoS, network elements give preferential treatment to classifications identified as having more demanding requirements.
- **Resource reservation** (known as "integrated services"): network resources are apportioned according to an application's QoS request, and subject to bandwidth management policy.

Consider this analogy. If you traveled by car to another city, you have no guarantees about the arrival time. Online map services may predict the travel time based on past traffic statistics, but they cannot predict what will actually happen. Suppose now that we introduce a new class of vehicles, called "emergency vehicles," such as ambulance cars, fire trucks, or police cars. Travelers belonging to this class may receive priority treatment at intersections (or, "routers"). Even for them, there are no guarantees on their origin-to-destination (i.e., end-to-end) delays. In this sense, their preferential treatment ("differentiated service") is defined only in terms of "per-hop behaviors" and not "end-to-end." On the other hand, if you traveled by train, the transportation resources are reserved ahead of time ("integrated services"). Your origin-to-destination (i.e., end-to-end) travel time is predictable—train timetables are fixed and widely publicized.

Our primary concerns here are *delay* and *loss* requirements that applications impose on the network. We should keep in mind that other requirements, such as reliability/availability and security may be important. When one or more links or intermediary nodes fail, the network may be unable to provide a connection between source and destination until those failures are repaired. *Reliability* refers to the frequency and duration of such failures. Some applications (e.g., control of electric power plants, hospital life support systems, critical banking operations) demand extremely reliable network operation. Typically, we want to be able to provide higher reliability between a few designated source-destination pairs. Higher reliability is achieved by providing multiple disjoint paths between the designated node pairs.

Application performance clearly depends on delay, loss, and their statistical characteristics. However, this dependency is not a simple function that can be clearly expressed and monitored. In addition, application performance depends on other things, such as user's perception of information quality. It also depends on adaptive application behavior, such as adjusting the sending rate, adjusting the coding (to mask errors), and adjusting the time point when the information is played back to the receiver, as described in Chapter 3.

1.6.2 Network Neutrality and QoS Outlook

Network neutrality (or, net neutrality or Internet neutrality) is the principle that Internet service providers (ISPs) should not be allowed to block or degrade Internet traffic from their competitors in order to speed up their own. There is a great political debate going on at present related to this topic. On one hand, consumers' rights groups and large Internet companies, such as Google and eBay, have lobbied the US Congress to pass laws restricting ISPs from blocking or slowing Internet traffic. On the other hand, net neutrality opponents, such as major telecommunications companies Verizon and AT&T, argue that in order to keep maintaining and improving network performance, ISPs need to have the power to use tiered networks to discriminate in how quickly they deliver Internet traffic. The fear is that net neutrality rules would relegate them to the status of "dumb pipes" that are unable to effectively make money on value-added services.

Today many ISPs enforce a usage cap to prevent "bandwidth hogs" from monopolizing Internet access. (Bandwidth hogs are usually heavy video users or users sharing files using peer-to-peer applications.) Service providers also enforce congestion management techniques to assure fair access during peak usage periods. The issue is particularly pertinent in wireless networks. Consumers currently do not have influence on these policies, and can only "vote" by exploiting a competitive market and switching to another ISP that has a larger usage cap or no usage cap at all. Consider, on the other hand, a business user who is willing to pay a higher rate that guarantees a high-definition and steady video stream that does not pause for buffering. Unfortunately, currently this is not possible, because regardless of the connection speeds available, Internet access is still a best effort service.

To discriminate against heavy video and file-sharing users, providers use what is known as deep packet inspection. *Deep packet inspection* is the act of an intermediary network node of examining the payload content of IP datagrams for some purpose. Normally, an intermediary node (router or switch) examines only link or network-layer packet headers but not the network-layer payload.

The Internet with its "best effort" service is not neutral in terms of its impact on applications that have different requirements. It is more beneficial for elastic applications that are latency-insensitive than for real-time applications that require low latency and low jitter, such as voice and real-time video. Some proposed regulations on Internet access networks define net neutrality as equal treatment among similar applications, rather than neutral transmissions regardless of applications.

There has been a great amount of research on QoS in wireline networks, but very little of it ended up being employed in actual products. Although the need for tiered services in wired networks has been debated, there is a widespread recognition that service discrimination is necessary in wireless networks. QoS in wireless networks is becoming particularly important with the growth of real-time and multimedia applications. Unlike traditional network applications, such as email or Web browsing, where data transmission process is much more transparent, in real-time communications, such as phone calls, delay and loss problems are much more apparent to the users. A consensus is that QoS techniques will be actually employed in wireless networks. Here are some of the arguments:

Wireline Network	**vs.**	**Wireless Network**
• Deals with *thousands* of traffic flows, thus not feasible to control		• Deals with *tens* of traffic flows (max about 50), thus it is feasible to control
• Am I a *bottleneck*?		• I am the *bottleneck*!
• Easy to add capacity		• Hard to add capacity

• Scheduling interval ~ 1 μs	• Scheduling interval ~ 1 ms, so a larger period is available to make decision

As will be seen in Chapter 3, service quality is always defined relative to human users of the network. As strange as it may sound, it is interesting to point out that the content providers may not always want to have the best network performance. In other words, the two types of end users (content providers vs. customers) may have conflicting objectives on network performance. For example, recent research of consumer behavior [Liu, 2008] has shown that task interruptions "clear the mind," changing the way buyers make decisions. The buyers who were temporarily interrupted were more likely to shift their focus away from "bottom-up" details such as price. Instead, they concentrated anew on their overall goal of getting satisfaction from the purchased product, even if that meant paying more. The uninterrupted buyers remained more price-conscious. This seems to imply that those annoying pop-up advertisements on Web pages—or even a Web page that loads slowly—might enhance a sale!

See further discussion about net neutrality in Section 10.4 (Chapter 10 is available separately on the book website).

1.7 Summary and Bibliographical Notes

This chapter covers some basic aspects of computer networks and wireless communications. Some topics are covered only briefly and many other important topics are left out. Practical implementations of the Internet protocols will be described in Chapter 9. To learn more about the basics of computer networking, the reader may also consult other networking books. Perhaps two of the most regarded introductory networking books currently are [Peterson & Davie 2007] and [Kurose & Ross 2010].

Section 1.1: Introduction

Full duplex links are commonly implemented over wired media. It is generally not possible simultaneously to receive and transmit on the same wireless link (in the same frequency band) the mutual interference would garble both signals. Thus, bidirectional systems must separate the uplink and downlink channels into orthogonal signaling dimensions, typically using time or frequency dimensions [Haykin, 2009]. Choi et al. [2010] have successfully designed a single channel full-duplex wireless transceiver. The design uses a combination of radio-0frequency and baseband techniques to achieve full-duplexing with minimal effect on link reliability.

The OSI reference architecture for communication protocols is described in [Day & Zimmermann, 1983].

The end-to-end principle was formulated by Saltzer *et al*. [1984], who argued that reliable systems tend to require end-to-end processing to operate correctly, in addition to any processing

Search the Web for net neutrality

in the intermediate system. They pointed out that most features in the lowest level of a communications system present costs for all higher-layer clients, even if those clients do not need the features, and are redundant if the clients have to reimplement the features on an end-to-end basis.

Keshav [1997: Chapter 5] argued from first principles that there are good reasons to require at least five layers and that no more are necessary. The layers that he identified are: physical, link, network, transport, and application layers. He argued that the functions provided by the session and presentation layers can be provided in the application with little extra work.

Early works on protocol implementation include [Clark, 1985; Hutchinson & Peterson, 1991; Thekkath, *et al.*, 1993]. Clark [1985] described the upcall architecture. Hutchinson and Peterson [1991] described a threads-based approach to protocol implementation. [Thekkath, *et al.*, 1993] is a pionerring work on user-level protocol implementation.

RFC-2679 [Almes, *et al.*, 1999(a)] defines a metric for one-way delay of packets across Internet paths. RFC-2681 [Almes, *et al.*, 1999(b)] defines a metric for round-trip delay of packets across Internet paths and follows closely the corresponding metric for one-way delay described in RFC-2679.

Section 1.2: Reliable Transmission via Redundancy

Section 1.3: Reliable Transmission by Retransmission

Automatic Repeat reQuest (ARQ) was invented by H.C.A. van Duuren during World Ward II to provide reliable transmission of characters over radio [van Duuren, 2001]. A classic early paper on ARQ and framing is [Gray, 1972]. RFC-3366 [Fairhurst & Wood, 2002] provides advice to the designers for employing link-layer ARQ techniques. This document also describes issues with supporting IP traffic over physical-layer channels where performance varies, and where link ARQ is likely to be used.

Broadcast media require medium access coordination. The earliest medium access control (MAC) protocol is called ALOHA. ALOHA was invented in the late 1960s by Norman Abramson and his colleagues at the University of Hawaii. Their goal was to use low-cost commercial radio equipment to connect computer users on Hawaiian Islands with a central time-sharing computer on the main campus. Because of its simplicity, ALOHA is used in some modern systems. For example, radio-frequency identification (RFID) systems are using the ALOHA protocol (see Section 6.3 of this book).

Another MAC protocol is called Carrier Sense Multiple Access (CSMA). This means that before the sender sends a packet, it senses the medium to see if it is idle. If it is, the sender transmits the packet. A variant of CSMA called CSMA/CD (for: Collision Detection) continues to sense the carrier during the transmission to detect whether a collision will happen because some other sender connected on the same medium is transmitting at the same time. If a collision is detected, then the sender will wait for a random period of time before transmitting again. The CSMA protocol was first proposed by Kleinrock and Tobagi in [1975].

CSMA/CD is normally implemented in wired networks. There are two main barriers to implement CSMA/CD in wireless networks. First, a wireless transmitter cannot simultaneously transmit and listen for a collision. Second, any channel activity around the transmitter may not be an indicator of collision at the receiver. A scheme called CSMA/CN (collision notification) has been developed to approximate CSMA/CD in wireless networks [Sen, et al., 2010].

Proper functioning of the backoff mechanism depends on a good **collision model**, which describes how stations determine whether collision took place. A collision model makes assumptions on the sequence of events that produce packet collision in a given system. Modeling can also include deriving the probability of packet collision under different scenarios. Wireless random-access protocols usually assume that packet errors or packet loss occurred due to collision. However, errors or loss can occur also due to channel noise or jamming by an adversary. The subsequent strategy for dealing with the loss may be inappropriate if the loss is not due collisions. For example, the delay introduced by the backoff mechanism would be wasteful if packet were not lost to a collision. Distinguishing collisions from other communication problems is often difficult or impossible.

Section 1.4: Routing and Addressing

IP version 4 along with the IPv4 datagram format was defined in RFC-791. Currently there is a great effort by network administrators to move to the next generation IP version 6 (IPv6), reviewed in Section 9.1.

Path MTU Discovery for IP version 4 is described in RFC-1191, and for IPv6 in RFC-1981.

Internet routing protocols are designed to rapidly detect failures of network elements (nodes or links) and route data traffic around them. In addition to routing around failures, sophisticated routing protocols take into account current traffic load and dynamically shed load away from congested paths to less-loaded paths.

The original ARPAnet distance vector algorithm used queue length as metric of the link cost. That worked well as long everyone had about the same line speed (56 Kbps was considered fast at the time). With the emergence of orders-of-magnitude higher bandwidths, queues were either empty or full (congestion). As a result, wild oscillations occurred and the metric was not functioning anymore. This and other reasons caused led to the adoption of Link State Routing as the new dynamic routing algorithm of the ARPAnet in 1979.

The first link-state routing concept was invented in 1978 by John M. McQuillan [McQuillan *et al*., 1980] as a mechanism that would calculate routes more quickly when network conditions changed, and thus lead to more stable routing.

When IPv4 was first created, the Internet was rather small, and the model for allocating address blocks was based on a central coordinator: the *Internet Assigned Numbers Authority* (http://iana.org/). Everyone who wanted address blocks would go straight the central authority. As the Internet grew, this model became impractical. Today, IPv4's classless addressing scheme (CIDR) allows variable-length network IDs and hierarchical assignment of address blocks. Big Internet Service Providers (ISPs) get large blocks from the central authority, then subdivide them, and allocate them to their customers. In turn, each organization has the ability to subdivide further their address allocation to suit their internal requirements. CIDR not only solved the problem of address shortages, but also by aggregating routes based on arbitrary-length IP prefixes in a classless manner, instead of aggregation limited by classes, it further reduced the forwarding table sizes.

In Section 1.4.4, it was commented that CIDR optimizes the common case. *Optimizing the common case* is a widely adopted technique for system design. Check, for example, [Keshav, 1997, Section 6.3.5] for more details and examples. Keshav [1997] also describes several other widely adopted techniques for system design.

The IANA (http://iana.org/) is responsible for the global coordination of the DNS Root (Section 9.4), IP addressing, Autonomous System numbering (RFC-1930, RFC-4893), and other Internet protocol resources. It is operated by the *Internet Corporation for Assigned Names and Numbers*, better known as ICANN.

In Section 1.4.5, we saw how the global Internet with many independent administrative domains creates the need to reconcile economic forces with engineering solutions. A routing protocol within a single administrative domain (Autonomous System) just needs to move packets as efficiently as possible from the source to the destination. Unlike this, a routing protocol that spans multiple administrative domains must allow them to implement their economic preferences. In 1980s, NSFNet provided the backbone network (funded by the US National Science Foundation), and the whole Internet was organized in a single tree structure with the backbone at the root. The backbone routers exchanged routing advertisements over this tree topology using a routing protocol called Exterior Gateway Protocol (EGP), described in RFC-904. In the early 1990s, the Internet networking infrastructure opened up to competition in the US, and a number of ISPs of different sizes emerged. The evolution of the Internet from a singly-administered backbone to its current commercial structure made EGP obsolete, and it is now replaced by BGP (Section 9.2.3). Bordering routers (called speaker nodes) implement both intra-domain and inter-domain routing protocols. Inter-domain routing protocols are based on path vector routing. Path vector routing is discussed in RFC-1322 (http://tools.ietf.org/html/rfc1322)

[Berg, 2008] introduces the issues about peering and transit in the global Internet. [Johari & Tsitsiklis, 2004]. [He & Walrand, 2006] present a generic pricing model for Internet services jointly offered by a group of providers and propose a fair revenue-sharing policy based on the weighted proportional fairness criterion. [Shrimali & Kumar, 2006] develop game-theoretic models for Internet Service Provider peering when ISPs charge each other for carrying traffic and study the incentives under which rational ISPs will participate in this game. [Srinivas & Srikant, 2006] study economics of network pricing with multiple ISPs.

Section 1.5: Link-Layer Protocols and Technologies

IEEE 802.2 is the IEEE 802 standard defining Logical Link Control (LLC). The standard is available online here: http://standards.ieee.org/getieee802/802.2.html. Both Ethernet (802.3) and Wi-Fi (802.11) have different physical sublayers and MAC sublayers but converge on the same LLC sublayer (i.e., 802.2), so they have the same interface to the network layer.

The IETF has defined a serial line protocol, the **Point-to-Point Protocol (PPP)**, in RFC-1661 [Simpson, 1994] (see also RFC-1662 and RFC-1663). RFC-1700 and RFC-3232 define the 16-bit protocol codes used by PPP in the Protocol field (Figure 1-63). PPP uses HDLC (bit-synchronous or byte synchronous) framing. High-Level Data Link Control (HDLC) is a link-layer protocol developed by the International Organization for Standardization (http://www.iso.org/). The current standard for HDLC is ISO-13239, which replaces previous standards. The specification of PPP adaptation for IPv4 is RFC-1332, and the IPv6 adaptation specification is RFC-5072. Current use of the PPP protocol is in traditional serial lines, authentication exchanges in DSL networks using PPP over Ethernet (PPPoE) [RFC-2516], and in the Digital Signaling Hierarchy (generally referred to as Packet-on-SONET/SDH) using PPP over SONET/SDH [RFC-2615].

Ethernet was originally developed at Xerox's Palo Alto Research Center (PARC) in 1973 to 1975 by Robert Metcalfe and David Boggs. The name "Ethernet" refers to the cable (the ether) and it originates from the *luminiferous ether*, through which electromagnetic radiation was once thought to propagate. Network hosts on an Ethernet network use a MAC protocol based on CSMA/CD to coordinate their access to the broadcast medium. As a historic curiosity, in March 1974, Robert Z. Bachrach wrote a memo to Metcalfe and Boggs and their management, stating that "technically and conceptually there is nothing new in your proposal" and that "analysis would show that your system would be a failure." However, this simple technology pretty much blew away any sophisticated technologies that competed with it over more than thirty years. (Check also Mr. Bachrach's response here:
http://www.reddit.com/comments/1xz13/in_1974_xerox_parc_engineers_invented_ethernet).

Digital Equipment Corporation (DEC), Intel, and Xerox published the Ethernet Version 1.0 standard in 1978 for a 10-Mbps version of Ethernet, called the **DIX standard**. In September 1980, the IEEE 802.3 working group released a draft standard **802.3** of the 10-Mbps version of Ethernet, with some minor changes from the DIX standard. In 1982, DEC, Intel, and Xerox published **Ethernet Version 2.0** or **Ethernet II**. Meanwhile, the IEEE draft standard was approved in 1983 and was subsequently published as an official standard in 1985 (ANSI/IEEE Std 802.3-1985). Although there are some minor differences between the two technologies, the terms **Ethernet** and **802.3** are generally used synonymously. Since then, a number of supplements to the standard have been defined to take advantage of improvements in the technologies and to support additional communication media and higher data rate capabilities, plus several new optional medium access control features. The latest version of the 802.3 standard is available online at: http://standards.ieee.org/getieee802/802.3.html.

Ethernet's collision detection does severely limit practical throughput on loaded, unswitched networks. As a result, almost no one uses unswitched networks anymore (known as "classic Ethernet"), and full duplex Gigabit Ethernet does not support unswitched operation. Basically, today's modern, high speed Ethernet connections are synchronized connections that can use the full bandwidth of the wire without worrying about collisions.

The significance of full-duplex, collision-free Ethernet is that it represents a MAC-layer improvement. Traditionally, Ethernet has progressed by physical layer protocol improvements, and the MAC layer has essentially remained unchanged. Collisions reduce the available bandwidth. Ethernet switches help, but they could have bus-based network segments on each port, with more than one station in each collision domain. In addition, half-duplex means 1/2 bandwidth. Therefore, full-duplex, collision-free Ethernet represents a major improvement.

The first spanning tree protocol (STP) was invented in 1985 at the Digital Equipment Corporation by Radia Perlman. Based on this algorithm, the IEEE published in 1990 the first standard for the protocol as the IEEE 802.1D standard. When a new switch is connected to the network, the newly switch will send the reconfiguration BPDU, and the other connected device will comply. All traffic is stopped for roughly 30 to 50 seconds while a spanning tree calculation takes place to respond to a topology change. In 2001, the IEEE introduced *Rapid Spanning Tree Protocol (RSTP)* as the IEEE 802.1w standard. RSTP provides significantly faster spanning tree convergence after a topology change. RSTP is typically able to respond to changes within a few milliseconds of a physical link failure. Standard IEEE 802.1D-2004 incorporates RSTP and obsoletes the original STP standard, but RSTP is backwards compatible with the regular STP. See [Cisco Systems Inc., 2006] for additional information.

The **IEEE Std 802.11** is a wireless local area network specification. [Crow, *et al*., 1997] discusses IEEE 802.11 wireless local area networks. In fact, 802.11 is a family of evolving wireless LAN standards. The 802.11n specification is described in this book in Section 6.3.1. The latest version of the 802.11 standard is available online at:
http://standards.ieee.org/getieee802/802.11.html.

The moniker CSMA/CA originally designated a method used by the Apple LocalTalk wire network in the 1980s. The RTS/CTS access mode in 802.11 is based on the MACA protocol (Multiple Access with Collision Avoidance) [Karn, 1990], which is an adaptation of the handshake protocols used in RS-232-C between computers and peripheral equipment, such as printers. An improvement to this basic protocol, called Multiple Access with Collision Avoidance for Wireless (MACAW), was introduced by [Bharghavan, *et al*., 1994; Fullmer & Garcia-Luna-Aceves 1997]. An additional feature of MACAW is the use of an acknowledgement from the receiver after the successful reception of a packet. MACAW also introduces some other features that are not included in the IEEE 802.11 standard.

Section 1.6: Quality of Service Overview

This section introduces the theme of how well the network delivers the packets—the network quality or service. The rest of the book explores the issues and solutions in depth.

X.-P. Xiao [2008] provides a high-level overview of issues and solutions for quality of service.

Raj Jain, “Books on Quality of Service over IP,”
Online at: http://www.cse.ohio-state.edu/~jain/refs/ipq_book.htm

Useful information about QoS can be found here:

Leonardo Balliache, “Practical QoS,” Online at: http://www.opalsoft.net/qos/index.html

Problems

Note: *Find problem solutions on the back of this book, starting on page* 459.

Problem 1.1

Suppose you wish to transmit a long message from a source to a destination over a network path that crosses two routers. Assume that all communications are error free and no acknowledgments are used. The propagation delay and the bit rate of the communication lines are the same. Ignore all delays other than transmission and propagation delays.

(a) Given that the message consists of *N* packets, each *L* bits long, how long will it take to transmit the entire message?

(b) If, instead, the message is sent as 2×*N* packets, each *L*/2 bits long, how long will it take to transmit the entire message?

(c) Are the durations in (a) and (b) different? Will anything change if we use smaller or larger packets? Explain why yes or why no.

Problem 1.2

Problem 1.3

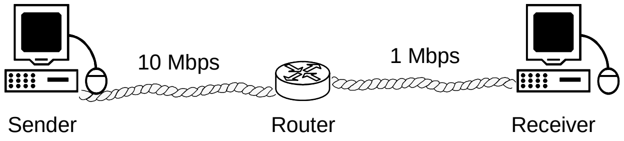
Suppose two hosts, sender *A* and receiver *B*, communicate using Stop-and-Wait ARQ method. Subsequent packets from *A* are alternately numbered with 0 or 1, which is known as the *alternating-bit protocol*.

(a) Show that the receiver *B* will never be confused about the packet duplicates *if and only if* there is a single path between the sender and the receiver.

(b) In case there are multiple, alternative paths between the sender and the receiver and subsequent packets are numbered with 0 or 1, show step-by-step an example scenario where receiver *B* is unable to distinguish between an original packet and its duplicates.

Problem 1.4

Assume that the network configuration shown in the figure below runs the Stop-and-Wait protocol. The signal propagation speed for both links is 2×10^8 m/s. The length of the link from the sender to the router is 100 m and from the router to the receiver is 10 km. Determine the sender utilization.



Problem 1.5

Suppose two hosts are using Go-back-2 ARQ. Draw the time-sequence diagram for the transmission of seven packets if packet 4 was received in error.

Problem 1.6

Consider a system using the Go-back-*N* protocol over a fiber link with the following parameters: 10 km length, 1 Gbps transmission rate, and 512 bytes packet length. (Propagation speed for fiber $\approx 2 \times 10^8$ m/s and assume error-free and full-duplex communication, i.e., link can transmit simultaneously in both directions. Also, assume that the acknowledgment packet size is negligible.) What value of *N* yields the maximum utilization of the sender?

Problem 1.7

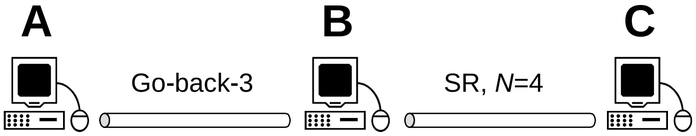
Consider a system of two hosts using a sliding-window protocol sending data simultaneously in both directions. Assume that the maximum frame sizes (MTUs) for both directions are equal and the acknowledgments may be piggybacked on data packets, i.e., an acknowledgment is carried in the header of a data frame instead of sending a separate frame for acknowledgment only.

(a) In case of a full-duplex link, what is the minimum value for the retransmission timer that should be selected for this system?

(b) Can a different values of retransmission timer be selected if the link is half-duplex?

Problem 1.8

Suppose three hosts are connected as shown in the figure. Host *A* sends packets to host *C* and host *B* serves merely as a relay. However, as indicated in the figure, they use different ARQ's for reliable communication (Go-back-*N* vs. Selective Repeat). Note that *B* is *not* a router; it is a regular host running both receiver (to receive packets from *A*) and sender (to forward *A*'s packets to *C*) applications. *B*'s receiver immediately relays in-order packets to *B*'s sender.

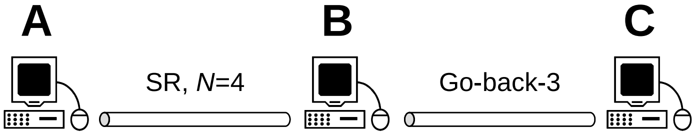


Draw side-by-side the timing diagrams for A→B and B→C transmissions up to the time where the first seven packets from *A* show up on *C*. Assume that the 2nd and 5th packets arrive in error to host *B* on their first transmission, and the 5th packet arrives in error to host *C* on its first transmission.

Discuss the merits of sending ACKs end-to-end, from destination C to source A, as opposed to sending ACKs independently for each individual link.

Problem 1.9

Consider the network configuration as in Problem 1.8. However, this time around assume that the protocols on the links are reverted, as indicated in the figure, so the first pair uses Selective Repeat and the second uses Go-back-*N*, respectively.



Draw again side-by-side the timing diagrams for A→B and B→C transmissions assuming the same error pattern. That is, the 2nd and 5th packets arrive in error to host *B* on their first transmission, and the 5th packet arrives in error to host *C* on its first transmission.

Problem 1.10

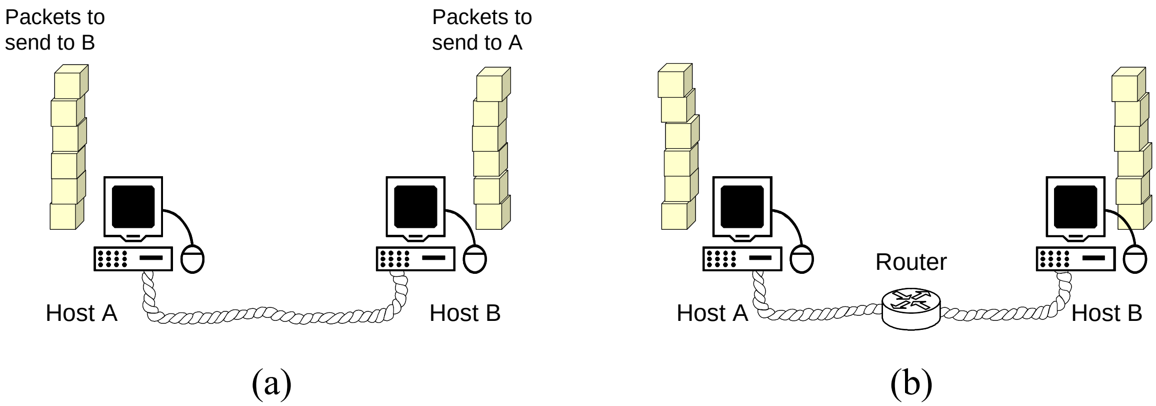
Assume the following system characteristics (see the figure below):

The link transmission speed is 1 Mbps; the physical distance between the hosts is 300 m; the link is a copper wire with signal propagation speed of 2×10^8 m/s. The data packets to be sent by the hosts are 2 Kbytes each, and the acknowledgment packets (to be sent separately from data packets) are 10 bytes long. Each host has 100 packets to send to the other one. Assume that the transmitters are somehow synchronized, so they *never* attempt to transmit simultaneously from both endpoints of the link.

Each sender has a *window* of 5 packets. If at any time a sender reaches the limit of 5 packets outstanding, it stops sending and waits for an acknowledgment. Because there is no packet loss (as stated below), the timeout timer value is irrelevant. This is similar to a Go-back-*N* protocol, with the following difference.

The hosts do *not* send the acknowledgments immediately upon a successful packet reception. Rather, the acknowledgments are sent periodically, as follows. At the end of an 82 ms period, the host examines whether any packets were successfully received during that period. If one or more packets were received, a single (cumulative) acknowledgment packet is sent to acknowledge all the packets received in this period. Otherwise, no acknowledgment is sent.

Consider the two scenarios depicted in the figures (a) and (b) below. The router in (b) is 150 m away from either host, i.e., it is located in the middle. If the hosts in each configuration start sending packets at the same time, which configuration will complete the exchange sooner? Show the process.



Assume no loss or errors on the communication links. The router buffer size is unlimited for all practical purposes and the processing delay at the router approximately equals zero. Note that the router can simultaneously send and receive packets on different links.

Problem 1.11

Consider two hosts directly connected and communicating using Go-back-*N* ARQ in the presence of channel errors. Assume that data packets are of the same size, the transmission delay t_x per packet, one-way propagation delay t_p, and the probability of error for data packets equals p_e. Assume that ACK packets are effectively zero bytes and always transmitted error free.

(a) Find the expected delay per packet transmission. Assume that the duration of the timeout t_{out} is large enough so that the source receives ACK before the timer times out, when both a packet and its ACK are transmitted error free.

(b) Assume that the sender operates at the maximum utilization and determine the expected delay per packet transmission.

Note: This problem considers only the expected delay from the start of the first attempt at a packet's transmission until its successful transmission. It does not consider the *waiting delay*, which is the time the packet arrives at the sender until the first attempt at the packet's transmission. The waiting delay will be considered later in Section 4.4.

Problem 1.12

Given a 64Kbps link with 1KB packets and RTT of 0.872 seconds:

(a) What is the maximum possible throughput, in packets per second (pps), on this link if a Stop-and-Wait ARQ scheme is employed and all transmissions are error-free?

(b) Again assuming that S&W ARQ is used, what is the expected throughput (pps) if the probability of error-free transmission is p=0.95?

(c) If instead a Go-back-*N* (GBN) sliding window ARQ protocol is deployed, what is the average throughput (pps) assuming error-free transmission and fully utilized sender?

(d) For the GBN ARQ case, derive a lower bound estimate of the expected throughput (pps) given the probability of error-free transmission p=0.95.

Problem 1.13

Assume a slotted ALOHA system with 10 stations, a channel with transmission rate of 1500 bps, and the slot size of 83.33 ms. What is the maximum throughput achievable per station if packets are arriving according to a Poisson process?

Problem 1.14

Consider a slotted ALOHA system with *m* stations and unitary slot length. Derive the following probabilities:

(a) A new packet succeeds on the first transmission attempt

(b) A new packet suffers exactly *K* collisions and then a success

Problem 1.15

Consider a slotted ALOHA network with *m* mobile stations and packet arrivals modeled as a Poisson process with rate λ. Solve the following:

(a) Assuming that this system operates with maximum efficiency, what are the fractions of slots that, on average, go unused (idle), slots that are used for successful transmission, and slots that experience packet collisions?

(b) Describe under what scenarios the system would operate with a less-than-maximum efficiency. Under such scenarios, what are the fractions of idle, successful, and collision slots?

(c) Given a steady arrival rate λ, would each non-maximum-efficiency operating point remain stable? Explain why yes or why no.

Hint: Carefully examine Figure 1-31 and other figures related to ALOHA.

Problem 1.16

Problem 1.17

Suppose two stations are using nonpersistent CSMA with a modified version of the binary exponential backoff algorithm, as follows. In the modified algorithm, each station will always wait 0 or 1 time slots with equal probability, regardless of how many collisions have occurred.

(a) What is the probability that contention ends (i.e., one of the stations successfully transmits) on the first round of retransmissions?

(b) What is the probability that contention ends on the second round of retransmissions (i.e., success occurs after one retransmission ends in collision)?

(c) What is the probability that contention ends on the third round of retransmissions?

(d) In general, how does this algorithm compare against the nonpersistent CSMA with the normal binary exponential backoff algorithm in terms of performance under different types of load?

Problem 1.18

A network using random access protocols has three stations on a bus with source-to-destination propagation delay τ. Station A is located at one end of the bus, and stations B and C are together at the other end of the bus. Frames arrive at the three stations are ready to be transmitted at stations A, B, and C at the respective times $t_A = 0$, $t_B = \tau/2$, and $t_C = 3\tau/2$. Frames require transmission times of 4τ. In appropriate timing diagrams with time as the horizontal axis, show the transmission activity of each of the three stations for (a) Pure ALOHA; (b) Non-persistent CSMA; (c) CSMA/CD.

Note: In case of collisions, show only the first transmission attempt, *not* retransmissions.

Problem 1.19

Problem 1.20

Consider a local area network using the CSMA/CA protocol shown in Figure 1-36. Assume that three stations have frames ready for transmission and they are waiting for the end of a previous transmission. The stations choose the following backoff values for their first frame: STA1 = 5, STA2 = 9, and STA3=2. For their second frame backoff values, they choose: STA1 = 7, STA2 = 1, and STA3=4. For the third frame, the backoff values are STA1 = 3, STA2 = 8, and STA3=1. Show the timing diagram for the first 5 frames. Assume that all frames are of the same length.

Problem 1.21

Consider a CSMA/CA protocol that has a backoff window size equal 2 slots. If a station transmits successfully, it remains in state 1. If the transmission results in collision, the station randomly chooses its backoff state from the set {0, 1}. If it chooses 1, it counts down to 0 and transmits. (See Figure 1-36 for details of the algorithm.) What is the probability that the station will be in a particular backoff state?

Hint: Figure 1-87 shows an analogy with a playground slide. A kid is climbing the stairs and upon reaching Platform-1 decides whether to enter the slide or to proceed climbing to Platform-2 by tossing a fair coin. If the kid enters the slide on Platform-1, he slides down directly through Tube-1. If the kid enters the slide on Platform-2, he slides through Tube-2 first, and then continues down through Tube-1. Think of the above problem as the problem of determining the probabilities that the kid will be found in Tube-1 or in Tube-2. For simplicity, we distort the reality and assume that climbing the stairs takes no time, so the kid is always found in one of the tubes.

Problem 1.22

Consider a CSMA/CA protocol with two backoff stages. If a station previously was idle, or it just completed a successful transmission, or it experienced two collisions in a row (i.e., it exceeded the retransmission limit, equal 2), then it is in the first backoff stage. In the first stage, when a new packet arrives, the station randomly chooses its backoff state from the set {0, 1}, counts down to 0, and then transmits. If the transmission is successful, the station remains in the first backoff stage and waits for another packet to send. If the transmission results in collision, the

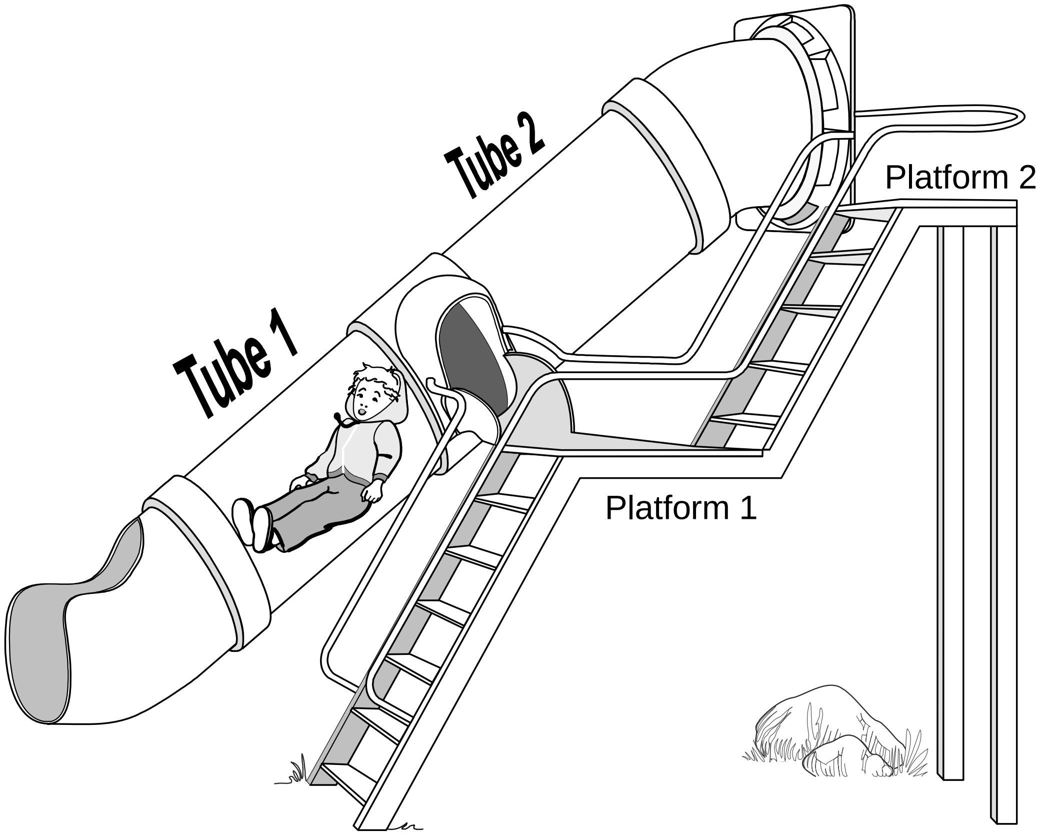


Figure 1-87: A playground-slide analogy to help solve Problem 1.21.

station jumps to the second backoff stage. In the second stage, the station randomly chooses its backoff state from the set {0, 1, 2, 3}, counts down to 0, and then retransmits the previously collided packet. If the transmission is successful, the station goes to the first backoff stage and waits for another packet to send. If the transmission results in collision, the station discards the packet (because it reached the retransmission limit, equal 2), and then jumps to the first backoff stage and waits for another packet to send.

Continuing with the playground-slide analogy of Problem 1.21, we now imagine an amusement park with two slides, as shown in Figure 1-88. The kid starts in the circled marked “START.” It first climbs Slide-1 and chooses whether to enter it at Platform-11 or Platform-12 with equal probability, i.e., 0.5. Upon sliding down and exiting from Slide-1, the kid comes to two gates. Gate-1 leads back to the starting point. Gate-2 leads to Slide-2, which consists of four tubes. The kid decides with equal probability to enter the tube on one of the four platforms. That is, on Platform-21, the kid enters the tube with probability 0.25 or continues climbing with probability 0.75. On Platform-22, the kid enters the tube with probability 1/3 or continues climbing with probability 2/3. On Platform-23, the kid enters the tube with probability 0.5 or continues climbing with probability 0.5. Upon sliding down and exiting from Slide-1, the kid always goes back to the starting point.

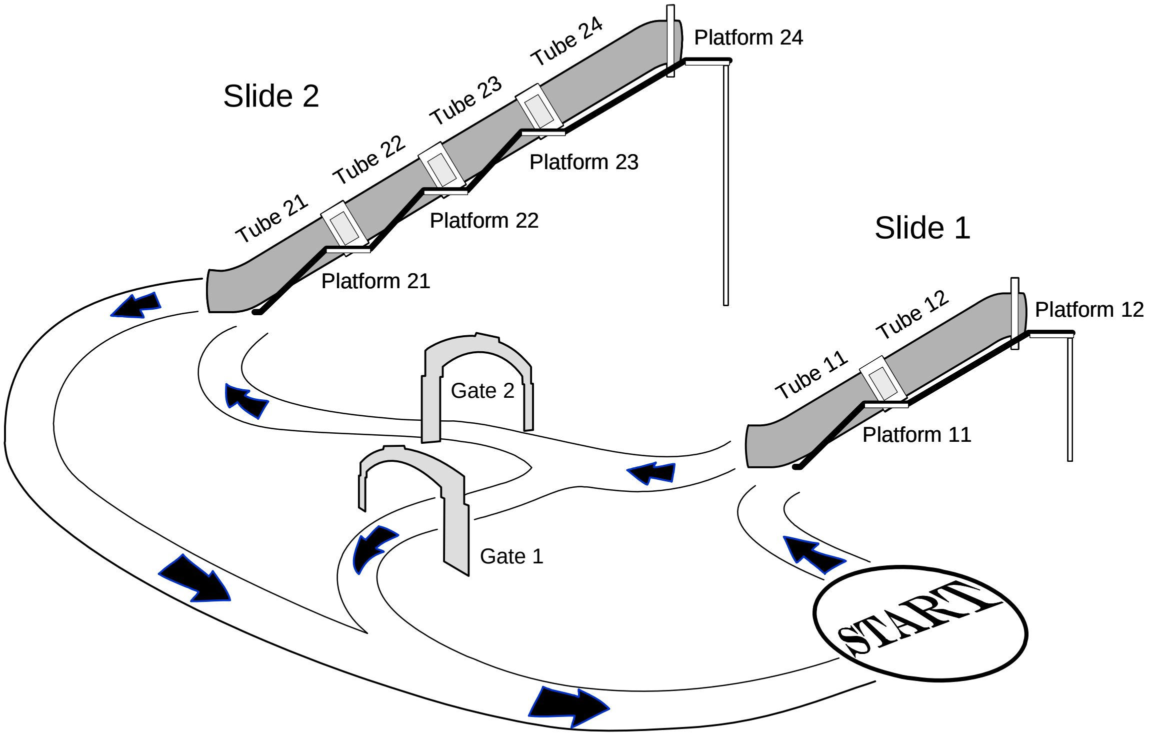
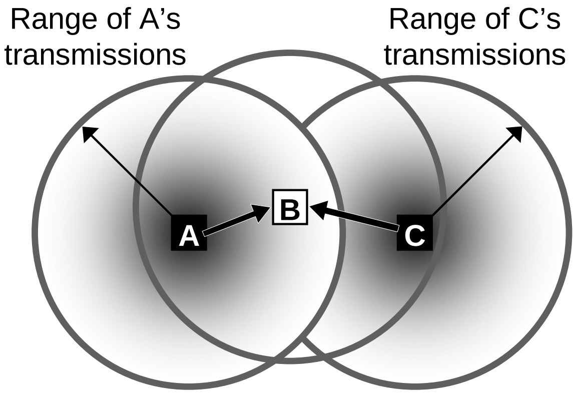


Figure 1-88: An analogy of a playground with two slides to help solve Problem 1.22.

Problem 1.23

Consider three wireless stations using the CSMA/CA protocol at the channel bit rate of 1 Mbps. The stations are positioned as shown in the figure. Stations *A* and *C* are hidden from each other and both have data to transmit to station *B*. Each station uses a timeout time for acknowledgments equal to 334 μsec. The initial backoff window range is 32 slots and the backoff slot duration equals 20 μsec. Assume that both *A* and *C* each have a packet of 44 bytes to send to *B*.



Suppose that stations *A* and *C* just heard station *B* send an acknowledgment for a preceding transmission. Let us denote the time when the acknowledgment transmission finished as $t = 0$. Do the following:

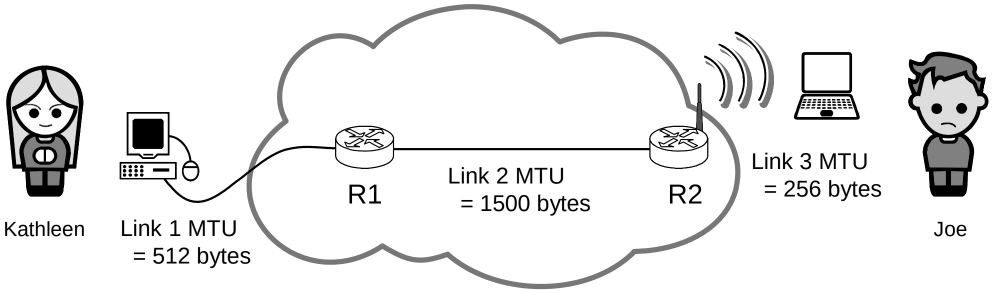
(a) Assuming that station *A* selects the backoff countdown $b_{A1} = 12$ slots, determine the vulnerable period for reception of the packet for *A* at receiver *B*.

(b) Assuming that simultaneously station C selects the backoff countdown b_{C1} = 5 slots, show the exact timing diagram for any packet transmissions from all three stations, until either a successful transmission is acknowledged or a collision is detected.

(c) After a completion of the previous transmission (ended in step (b) either with success or collision), assume that stations A and C again select their backoff timers (b_{A2} and b_{C2}, respectively) and try to transmit a 44-bytes packet each. Assume that A will start its second transmission at t_{A2}. Write the inequality for the range of values of the starting time for C's second transmission (t_{C2}) in terms of the packet transmission delay (t_x), acknowledgment timeout time (t_{ACK}) and the backoff periods selected by A and C for their first transmission (b_{A1} and b_{C1}, respectively).

Problem 1.24

Kathleen is emailing a long letter to Joe. The letter size is 16 Kbytes. Assume that TCP is used and the connection crosses 3 links as shown in the figure below. Assume link layer header is 40 bytes for all links, IP header is 20 bytes, and TCP header is 20 bytes.



(a) How many packets/datagrams are generated in Kathleen's computer on the IP level? Show the derivation of your result.

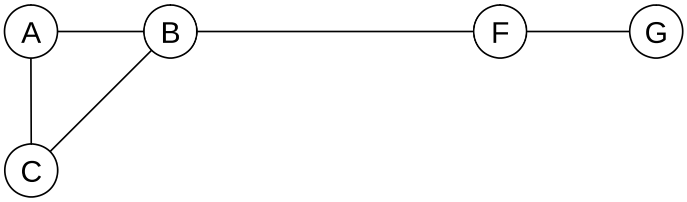
(b) How many fragments Joe receives on the IP level? Show the derivation.

(c) Show the first 4 and the last 5 IP fragments Joe receives and specify the values of all relevant parameters (data length in bytes, ID, offset, flag) in each fragment header. Fragment's ordinal number should be specified. Assume initial ID = 672.

(d) What will happen if the very last fragment is lost on Link 3? How many IP datagrams will be retransmitted by Kathleen's computer? How many retransmitted fragments will Joe receive? Specify the values of all relevant parameters (data length, ID, offset, flag) in each fragment header.

Problem 1.25

Consider the network in the figure below, using link-state routing (the cost of all links is 1):



Suppose the following happens in sequence:

(a) The *BF* link fails

(b) New node *H* is connected to *G*

(c) New node *D* is connected to *C*

(d) New node *E* is connected to *B*

(e) A new link *DA* is added

(f) The failed *BF* link is restored

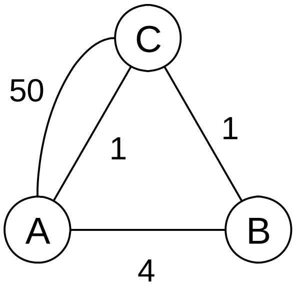
Show what link-state advertisements (LSAs) will flood back and forth for each step above. Assume that (*i*) the initial LSA sequence number at all nodes is 1, (*ii*) no packets time out, (*iii*) each node increments the sequence number in their LSA by 1 for each step, and (*iv*) both ends of a link use the same sequence number in their LSA for that link, greater than any sequence number either used before.

[You may simplify your answer for steps (b)-(f) by showing only the LSAs which change (not only the sequence number) from the previous step.]

Problem 1.26

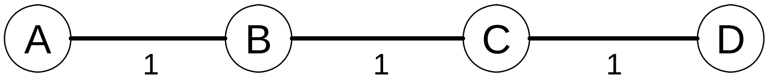
Problem 1.27

Consider the network in the figure below and assume that the distance vector algorithm is used for routing. Show the distance vectors *after* the routing tables on all nodes are stabilized. Now assume that the link $\overline{AC}$ with weight equal to 1 is broken. Show the distance vectors on all nodes for up to five subsequent exchanges of distance vectors or until the routing tables become stabilized, whichever comes first.



Problem 1.28

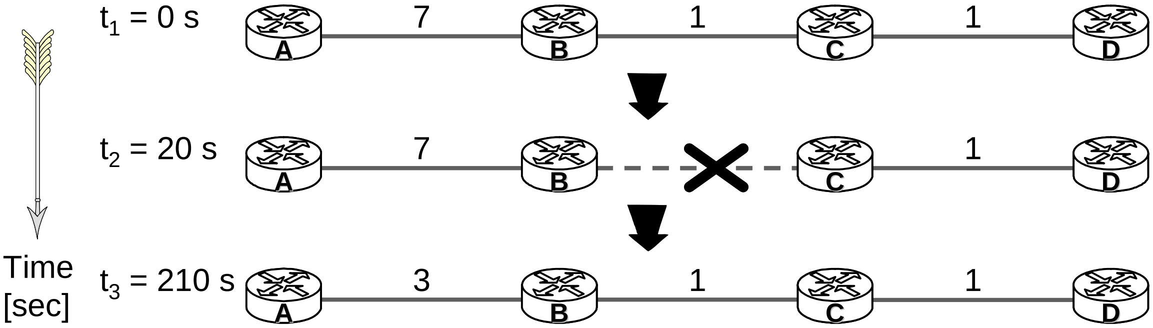
Consider the following network, using distance-vector routing:



Suppose that, after the network stabilizes, link *C–D* goes down. Show the routing tables on the nodes *A*, *B*, and *C*, for the subsequent five exchanges of the distance vectors. How do you expect the tables to evolve for the future steps? State explicitly all the possible cases and explain your answer.

Problem 1.29

Consider a network shown in the figure below using the distance vector routing protocol with *hold-down timers* (but not split-horizon routing). Assume that the network has already converged and all routers have built their routing tables. Assume also that the routers will exchange periodic update messages at regular intervals, starting at time t_1=0. In addition, the routers will send *triggered updates* when a cost of a link changes, a link goes down, or comes up. At time t_2, the link *BC* is downed and then reestablished at t_3. Note also that at t_3 the cost of the link *AB* is reduced from 7 down to 3.



Assume that every router maintains four timers (all in seconds) for every route in its routing table:

1. *Update Timer*: how often (in seconds) to send periodic update messages, also known as "keep-alive" messages
2. *Invalid-Route Timer*: how many seconds since seeing a valid update, to mark the route "invalid" and initiate its hold-down and flush timers. The route is marked as "invalid" but is retained in the routing table until the flush timer expires
3. *Hold-Down Timer*: how long (in seconds) to *ignore* any updates for the downed route
4. *Flush Timer*: how many seconds until the "invalid" route is removed from the routing table

Note that hold-down and flush timers for a downed route are started simultaneously, upon expiration of the invalid timer. Once the hold-down timer expires, the router will now accept any new updates for that route, and the route with the best metric will be placed in the routing table. However, the flush timer will still continue to keep counting, and if no new updates come in for the downed route, then that route will be flushed from the routing table.

Assume that the timer values are set as follows: Update = 30 seconds; Invalid = 180 s; Hold-Down = 180 s; and Flush = 240 s.

Solve the following:

(a) Show the routing tables on all routers at time t_1.

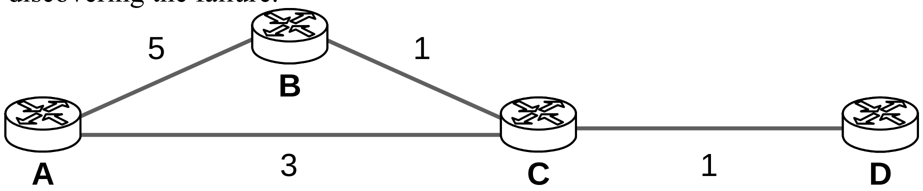
(b) List all the messages that will be exchanged between all routers from time 0 until 450 seconds (included).

(c) Show the routing tables just on routers *B* and *C* at times *t* = 60 sec, 180 s, 360 s, 390 s, and 420 s.

If several events happen simultaneously on the same router, then assume the following order: 1) a timer expires; 2) messages are sent and received instantaneously, with zero delay; 3) routing tables are updated, based on the received messages; 4) a timer is started.

Problem 1.30

Consider the network shown in the figure, using the distance vector routing algorithm. Assume that all routers exchange their distance vectors periodically every 60 seconds regardless of any changes. If a router discovers a link failure, it broadcasts its updated distance vector within 1 second of discovering the failure.

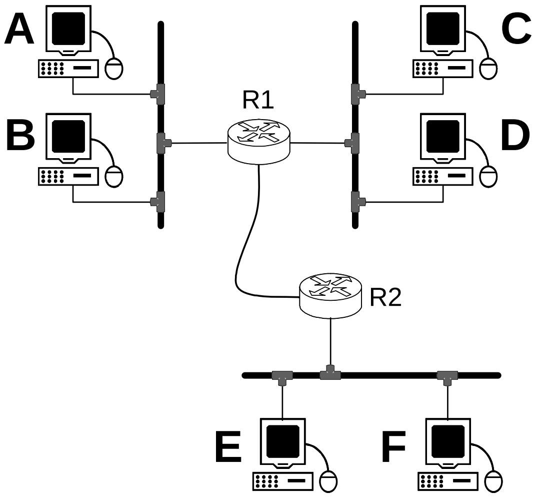


(a) Start with the initial state where the nodes know only the costs to their neighbors, and show how the routing tables at all nodes will reach the stable state.
(b) Use the results from part (a) and show the forwarding table at node *A*. [Note: use the notation *AC* to denote the output port in node *A* on the link to node *C*.]
(c) Suppose the link *CD* fails. Give a sequence of routing table updates that leads to a routing loop between *A*, *B*, and *C*.
(d) Would a routing loop form in (c) if all nodes use the split-horizon routing technique? Would it make a difference if they use split-horizon with poisoned reverse? Explain your answer.

Problem 1.31

Problem 1.32

You are hired as a network administrator for the network of sub-networks shown in the figure. Assume that the network will use the CIDR addressing scheme.



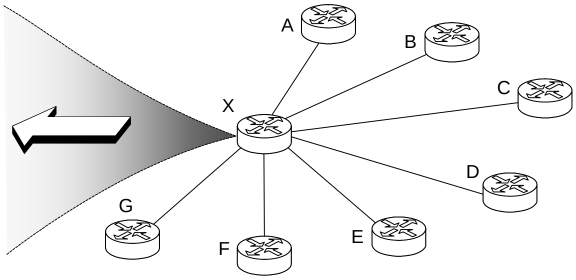
(a) Assign meaningfully the IP addresses to all hosts on the network. Allocate the minimum possible block of addresses for your network, assuming that no new hosts will be added to the current configuration.

(b) Show how routing/forwarding tables at the routers should look *after* the network stabilizes (do not show the process).

Problem 1.33

The following is a forwarding table of a router *X* using CIDR. Note that the last three entries cover every address and thus serve in lieu of a default route.

Subnet Mask	Next Hop
223.92.32.0 / 20	A
223.81.196.0 / 12	B
223.112.0.0 / 12	C
223.120.0.0 / 14	D
128.0.0.0 / 1	E
64.0.0.0 / 2	F
32.0.0.0 / 3	G



State to what next hop the packets with the following destination IP addresses will be delivered:

(a) 195.145.34.2

(b) 223.95.19.135

(c) 223.95.34.9

(d) 63.67.145.18

(e) 223.123.59.47

(f) 223.125.49.47

(Keep in mind that the default matches should be reported only if no longer match is found.)

Problem 1.34

Suppose a router receives a set of packets and forwards them as follows:

(a) Packet with destination IP address **128.6.4.2**, forwarded to the next hop **A**

(b) Packet with destination IP address **128.6.236.16**, forwarded to the next hop **B**

(c) Packet with destination IP address **128.6.29.131**, forwarded to the next hop **C**

(d) Packet with destination IP address **128.6.228.43**, forwarded to the next hop **D**

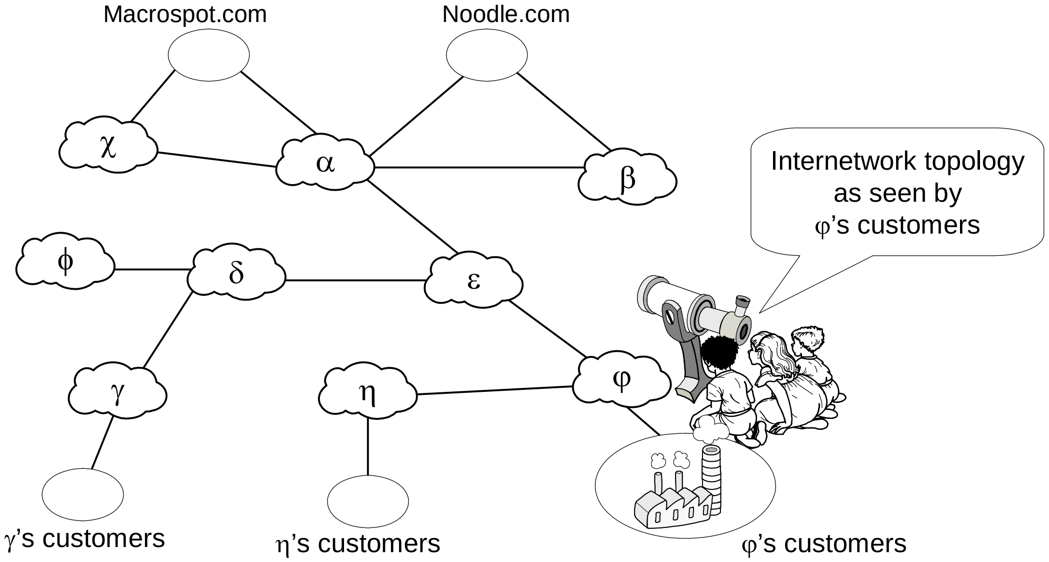
Reconstruct only the part of the router's forwarding table that you suspect is used for the above packet forwarding. Use the CIDR notation and select the ***shortest network prefixes*** that will produce unambiguous forwarding:

Network Prefix	Subnet Mask	Next Hop
……………………………	………………..	…
……………………………	………………..	…
……………………………	………………..	…
……………………………	………………..	…

Problem 1.35

Problem 1.36

Consider the internetwork of autonomous systems shown in Figure 1-56. Assume that all stub ASs already advertised the prefixes of their networks so by now the internetwork topology is known to all other ASs. (We assume that non-stub ASs do not advertise any destinations within themselves—they advertise their presence only if they lay on a path to a stub-AS destination.) Given the business interests of different ASs, and as illustrated in Figure 1-57 and Figure 1-58, customers of different ISPs will not learn about all links between all ASs in the internetwork. For example, customers of ISP φ will see the internetwork topology as shown in the figure below:

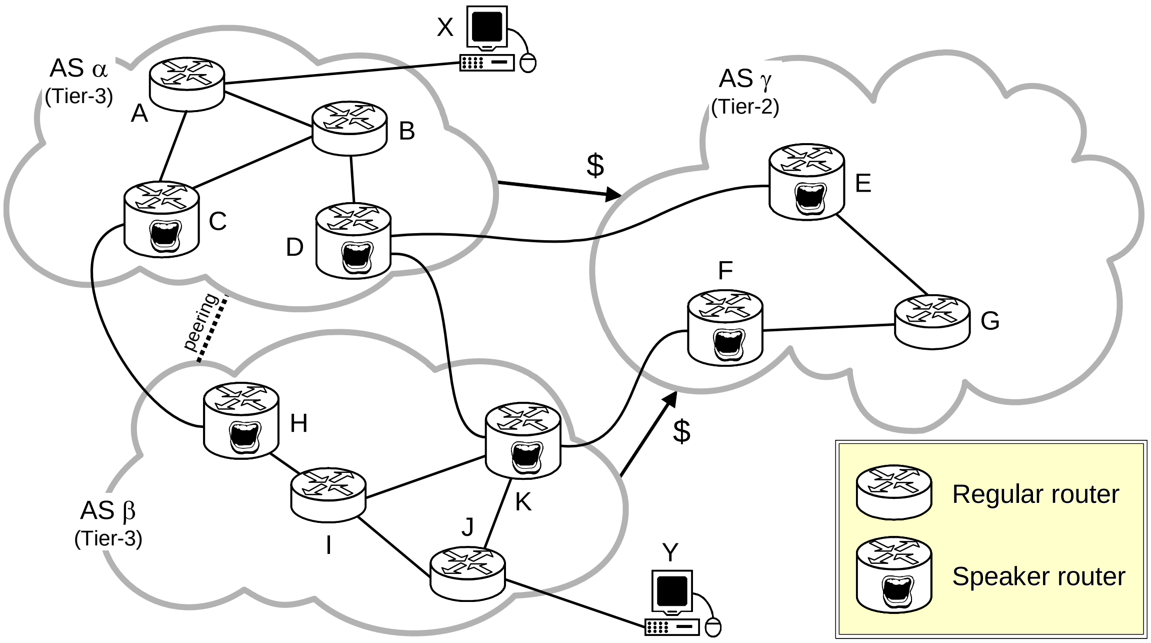


As explained in the description of Figure 1-57, ASη has no interest in carrying ASφ's transit traffic, so it will not advertise that it has a path to ASδ. Therefore, ASφ and its customers will not learn about this path. For the same reason the connection between ASδ and ASα will not be visible. Note that ASφ will learn about alternative paths from/to Macrospot.com or Noodle.com through ASs χ and β, respectively, because multihomed ASs will advertise their network prefixes to all directly connected ISPs.

Your task is to show the respective views of the internetwork topology for the customers of ASs γ and η, and for the corporations Macrospot.com or Noodle.com. For each of these ASs, draw the internetwork topology as they will see it and explain why some connections or ASs will not be visible, if there are such.

Problem 1.37

Consider the internetwork of autonomous systems shown in the figure below. Autonomous systems α and β are peers and they both buy transit from AS γ. Assume that all links cost 1 (one hop) and every AS uses hot-potato routing to forward packets (destined for other ASs) towards speaker nodes.



(d) How many paths (in terms of autonomous systems, not individual routers) to customers of AS β are available for use to reach customers of AS α?

(e) What path will traffic from host X to host Y normally take? List the autonomous systems and within each autonomous system list the individual routers (hop-by-hop). How about traffic from host Y to host X?

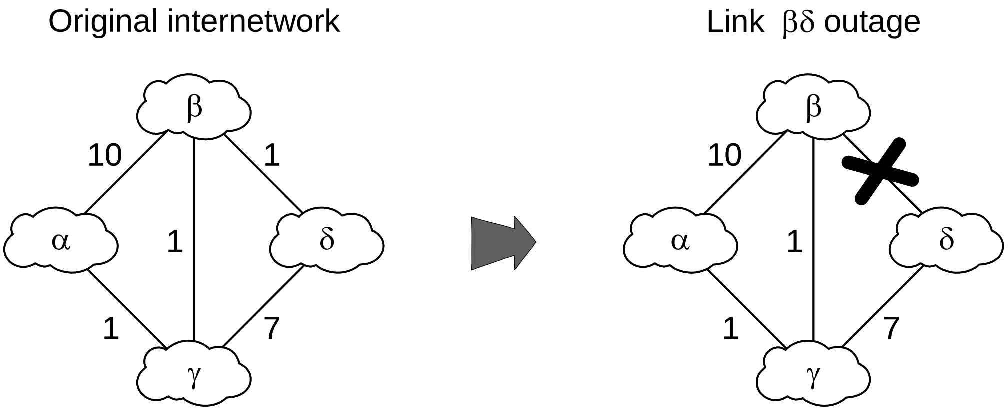
(f) Is it possible for *all* traffic from host X to host Y and vice versa to take the same path? Explain why yes or why no, and if yes, how this can be achieved.

(g) Is it possible for traffic from host X to host Y and vice versa to take the same that includes AS γ? Explain why yes or why no, and if yes, how this can be achieved.

Problem 1.38

Problem 1.39

Consider the internetwork of autonomous systems (ASs) shown in Example 1.5, reproduced here:



Assume that the internetwork has converged and all ASs have built their path table for all destination ASs. Next, consider the failure of the link between ASβ and ASδ, as illustrated in the figure on the right.

ASβ will send an "unreachable" message to its neighbors, stating that path 〈β,δ〉 is not available. Every AS that receives this message will check whether any of the paths in its path table are using this path and mark them as unavailable. The AS will also propagate an "unreachable" message to its other neighbors, in case they are also using this AS as the next hop for destination ASδ. If an AS receives an "unreachable" message and realizes that it can reach ASδ without using path 〈β,δ〉, it will inform the neighbor from whom it received "unreachable" about an alternative path to ASδ. On learning about this new path vector, that neighbor will, in turn, propagate a follow-up path vector message to its other neighbors.

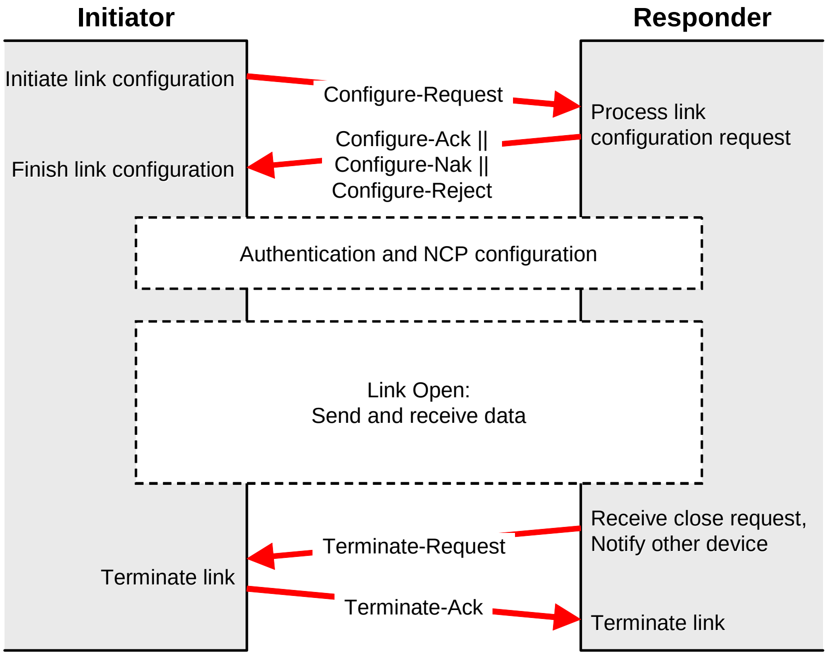
Solve the following:

(a) Show the path table entry at each autonomous system just for the single destination ASδ before the link βδ outage.
(b) List the ASs for which the routing will be affected by the link βδ outage.
(c) List ***all*** the messages (include the exact path advertised in a message) that will be sent in the internetwork until it has converged to a new equilibrium state.
(d) Show the path table entry at each autonomous system just for the single destination ASδ in the new equilibrium state.

Problem 1.40

Problem 1.41

PPP uses the Link Control Protocol (LCP) to establish, maintain, and terminate the physical connection. During the link establishment phase, the configuration parameters are negotiated by an exchange of LCP frames. Before information can be sent on a link, each of the two computers that make up the connection must test the link and agree on a set of parameters under which the link will operate. If the negotiation converges, the link is established and either the authentication is performed or the network layer protocol can start sending data. If the endpoints fail to negotiate a common configuration for the link, it is closed immediately. LCP is also responsible for maintaining and terminating the connection.



The *Configure-Request* message is sent to request a link establishment and it contains the various options requested. This request is responded with a *Configure-Ack* ("acknowledge") if every requested option is acceptable. A *Configure-Nak* ("negative acknowledge") is sent if all the requested options are recognizable but some of their requested values are not acceptable. This message includes a copy of each configuration option that the Responder found unacceptable and it suggests an acceptable negotiation.

A *Configure-Reject* is sent if any of the requested options were either unrecognizable or represent unacceptable ways of using the link or are not subject to negotiation. This message includes the objectionable options. Configure-Request frames are transmitted periodically until either a Configure-Ack is received, or the number of frames sent exceeds the maximum allowed value.

A simplified format of an LCP frame is as follows (Figure 1-65):

Code	Identifier	Data

The *Code* field identifies the type of LCP frame, such as *Configure-Request*, *Configure-Ack*, *Terminate-Request*, etc. The *Identifier* field carries an identifier that is used to match associated requests and replies. When a frame is received with an invalid Identifier field, the frame is silently discarded without affecting the protocol execution.

- For a Configure-Request frame, the Identifier field *must* be changed whenever the contents of the Data field changes (Data carries the link configuration options), and whenever a valid reply has been received for a previous request. For retransmissions, the Identifier *may* remain unchanged.
- For a configuration response frame (Configure-Ack, Configure-Nak, or Configure-Reject), the Identifier field is an exact copy of the Identifier field of the Configure-Request that caused this response frame.
- For a Terminate-Request frame, the Identifier field *must* be changed whenever the content of the Data field changes, and whenever a valid reply has been received for a previous request. For retransmissions, the Identifier *may* remain unchanged.

- For a Terminate-Ack frame, on reception, the Identifier field of the Terminate-Request is copied into the Identifier field of the Terminate-Ack frame.

During the link establishment phase, only LCP frames should be transmitted in the PPP frames. Any non-LCP frames received during this phase must be silently discarded.

The LCP link termination frames are: *Terminate-Request*, which represents the start of the link termination phase; and, *Terminate-Ack*, which acknowledges the receipt of a recognizable Terminate-Request frame, and accepts the termination request. Under ideal conditions, the link termination phase is signaled end-to-end using LCP link termination frames. However, the link termination phase also can be caused by a loss of carrier or an immediate shutdown by the system administrator.

Explain the following:

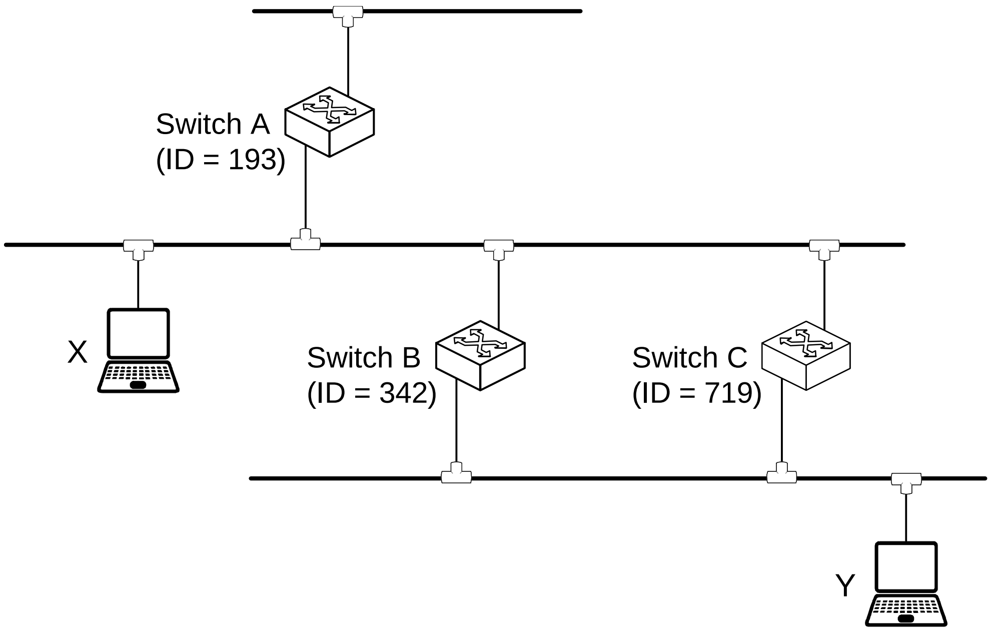
(a) Why are unique identifiers needed in LCP frames? Why a configuration response frame, such as Configure-Ack, without the Identifier field is insufficient? Illustrate your argument by drawing a time-diagram for a scenario where LCP link configuration would fail if the Identifier field did not exist.

(b) Why is the "two-way handshake" using Terminate-Ack frame needed in the link termination phase? Illustrate your argument by drawing a time-diagram of a scenario where LCP link termination would fail if Terminate-Ack were not sent.

Problem 1.42

Problem 1.43

Consider the following Ethernet network where all the switches employ the spanning tree protocol (STP) to remove the loops in the network topology. The numbers in parentheses represent the switches' identifiers.



Start when the network is powered up and stop when the network stabilizes (i.e., only the root switch remains generating configuration frames). Assume that all switches send configuration messages in synchrony with each other (although in reality generally this is not the case). Do the following:

(a) List all the configuration messages sent by all switches until the network stabilizes. Recall that a configuration message carries ⟨source-ID, root-ID, root-distance⟩.

(b) For each switch, indicate which ports will be selected as "root," "designated," or "blocked" by the spanning tree protocol.

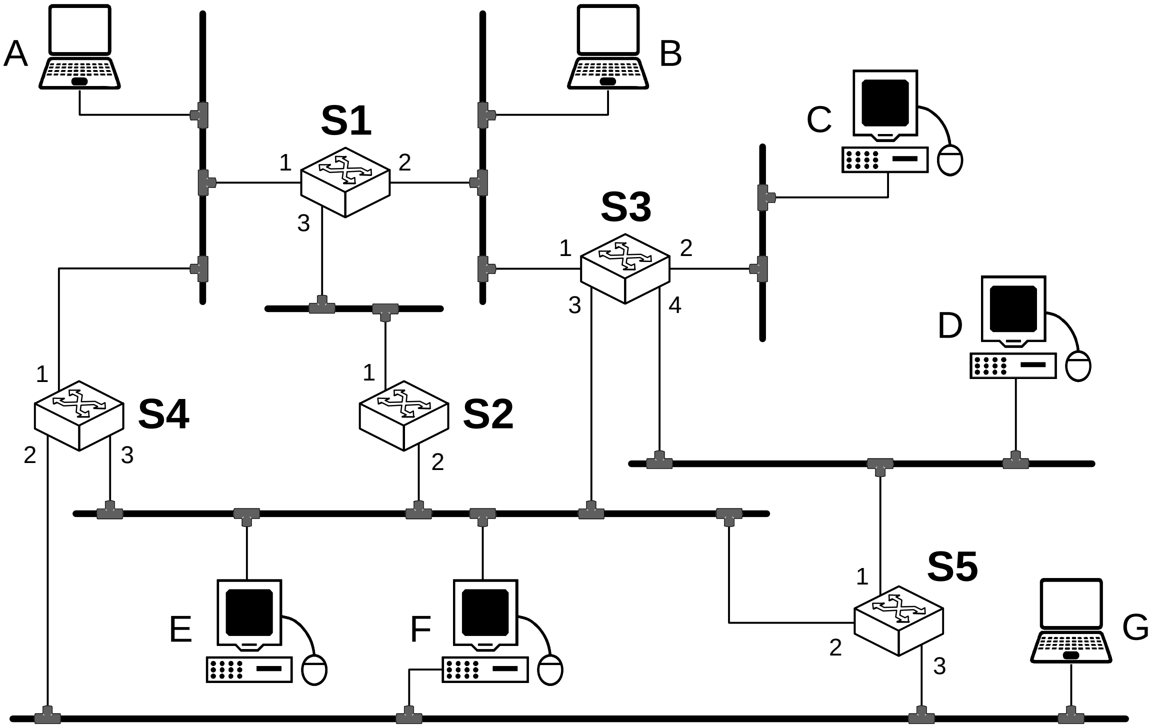
(c) How many iterations will take for the network to stabilize?

(d) After the network stabilizes, draw the path that a frame sent by station *X* will traverse to reach station *Y*.

Problem 1.44

Problem 1.45

Consider the following extended Ethernet network with switches employing the spanning tree protocol. The switches' identifiers are ordered by their names (S1, S2, S3, etc.).



(a) After the network stabilizes, for each switch specify how the ports will be labeled by the spanning tree protocol ("root," "designated," or "blocked").

(b) Assume now that switch *S*3 suffers a breakdown. For the remaining switches, list the ports that will be selected as "designated" after a new tree has been formed.

Problem 1.46

Consider a local area network using the CSMA/CA protocol shown in Figure 1-36. Assume that three stations have frames ready for transmission and they are waiting for the end of a previous transmission. The stations choose the following backoff values for their first frame: STA1 = 5, STA2 = 9, and STA3=2. For their second frame backoff values, they choose: STA1 = 7, STA2 = 1, and STA3=4. For the third frame, the backoff values are STA1 = 3, STA2 = 8, and STA3=3. Show the timing diagram for the first 5 frames. Assume that all frames are of the same length.

Note: Compare the solution with that of Problem 1.20.

Problem 1.47

Consider an independent BSS (IBSS) with two mobile STAs, *A* and *B*, where each station has a single packet to send to the other one. Draw the precise time diagram for each station from the start to the completion of the packet transmission. For each station, select different packet arrival time and a reasonable number of backoff slots to count down before the station commences its transmission so that no collisions occur during the entire session. (Check Table 1-6 for contention window ranges.) Assume that both stations are using the basic transmission mode and only the first data frame transmitted is received in error (due to channel noise, not collision).

Problem 1.48

Chapter 2

Transmission Control Protocol (TCP)

Contents

Transmission Control Protocol (TCP) is usually not associated with quality of service; but one could argue that TCP offers QoS in terms of assured delivery and efficient use of bandwidth, although it provides no delay guarantees. TCP is, after all, mainly about efficiency and adaptation: how to deliver data utilizing the maximum available (but fair) share of a dynamically changing network capacity so to reduce the delay. That is why our focus here is one aspect of TCP—congestion avoidance and control. We start quality-of-service review with TCP because it does not assume any knowledge of or any cooperation from the network. The network is essentially seen as a black box.

In Chapter 1 we have seen that pipelined ARQ protocols, such as Go-back-*N*, increase the utilization of network resources by allowing multiple packets to be simultaneously in transit (or, in flight) from sender to receiver. The number of unacknowledged packets (known as "flight size") is controlled by a parameter called window size which must be set according to the available network resources. The network is responsible for data from the moment it accepts them at sender's end until they are delivered at receiver's end. In a sense, the network is storing the data for the "flight duration," either in a router memory or travelling over a link. For this purpose, the network must reserve resources to avoid the possibility of becoming overbooked. In case of two end hosts connected by a single link, the optimal window size is easy to determine and remains constant for the duration of session. However, this task is much more complex in a general multi-hop network and that is a key purpose of TCP.

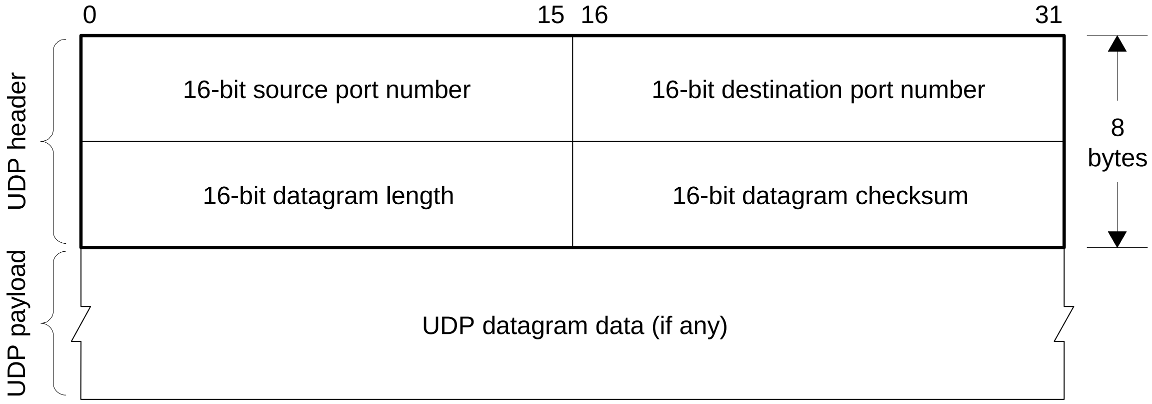

0
15
16
31
UDP header
UDP payload
16-bit source port number
16-bit destination port number
16-bit datagram length
16-bit datagram checksum
8 bytes
UDP datagram data (if any)

Figure 2-1: UDP datagram format.

2.1 Introduction to UDP and TCP

The End-to-End layer of the three-layer protocol stack (Figure 1-11) or the Transport layer of the OSI protocol stack (Figure 1-16) provides convenient services such as connection-oriented data stream support, reliability, flow control, and multiplexing. We start by introducing a simple end-to-end protocol called User Datagram Protocol (UDP).

2.1.1 User Datagram Protocol (UDP)

The *User Datagram Protocol (UDP)* is a simple end-to-end protocol for transmission of data between endpoint applications. UDP does not provide reliable packet transmission, i.e., it is not an Automatic Repeat Request (ARQ) protocol (Section 1.3), which means that it exposes any unreliability of the underlying network protocol to the application. For this reason, UDP is sometimes referred to as *Unreliable* Datagram Protocol. UDP provides checksums to allow the receiver to check for data integrity. However, there is no guarantee of message delivery, correct ordering of message, or duplicate protection.

UDP introduces transport-layer *ports* that make possible multiplexing several data streams at the sender and demultiplexing for delivery to the correct receiver application. A **port** is a software structure represented by the **port number** that identifies the sending or the receiving process at an endpoint device. The port number is a 16-bit integer value, allowing for port numbers between 0 and 65535. Port 0 is reserved, but is a permissible source port value if the sending process does not expect messages in response.

UDP packets are called *datagrams*, same as IP datagrams (Section 1.4.1), because they are essentially not much different. Figure 2-1 shows the format of a UDP datagram. The fields are:

Source Port Number: This field identifies the sending application when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be set to zero. If the source host is a client, the port number is likely to be temporary. If the source host is a server, the port number is likely to be a well-known port number permanently assigned to the server application.

Destination Port Number: This field identifies the receiver's port and is required. Similar to the source port number, the port number may be temporary or permanently assigned and well-known.

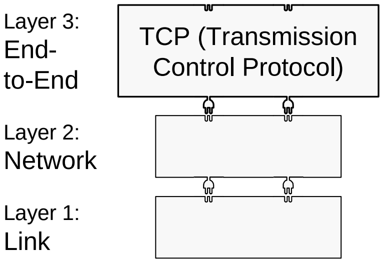
Datagram Length: This field specifies the length (in bytes) of the entire datagram, including both header and data. The minimum length is 8 bytes because that is the length of the UDP header. Given 16 bits, UDP datagram length is limited to 65,535 bytes (8-byte header plus 65,527 bytes of data). However, we must also account for the underlying IP protocol header, which in case of IPv4 leaves 65,507 bytes (65,535 − 8-byte UDP header − 20-byte IP header). IPv6 defines the so-called jumbograms where it is possible to have UDP packets of size greater than 65,535 bytes.

Datagram Checksum: The checksum field is used for error-checking of the header *and* data. The checksum calculation also includes part of the IP header and the reader should check other sources for details. If the sender generates no checksum, the field should be set to zero.

In essence, the only services that UDP provides above IP are multiplexing (via port numbers) and data integrity verification (via an optional checksum). UDP is suitable for uses where reliable transmission is either not necessary or performed by the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets. Various attributes make UDP particularly useful for certain application types:

- It is *transaction-oriented*, suitable for simple query-response applications.
- It uses datagrams, which is suitable for modeling other protocols such as in IP tunneling.
- It is simple, suitable for bootstrapping or other purposes without a full protocol stack.
- It does not maintain any connection state (such as information about unacknowledged packets or packets delivered out-of-order), which makes it suitable for large numbers of clients, such as in streaming media applications.
- It does not provide retransmission, which avoids retransmission-related delays and makes it suitable for real-time applications where timeliness is more important than reliable delivery.
- Works well in unidirectional communication, suitable for broadcasting of information such as in many kinds of service discovery or routing protocol advertisements.

2.1.2 TCP and Reliable Byte Stream Service



TCP provides a *byte stream service*, which means that a stream of 8-bit bytes (also known as "octets") is exchanged across a TCP connection between two endpoint applications. Once a connection is established, all data written by the application is considered a single continuous stream of bytes. TCP does not automatically insert any delimiters of the data records. An application using TCP might "write" to it several times only to have the data compacted into a single segment and delivered as such to its peer. For example, if the application on one end writes 20 bytes, followed by a write of 30 bytes, followed by a write of 10 bytes, the application at the other end of the connection cannot tell what size the individual writes were. The other end may read the 60 bytes in two reads

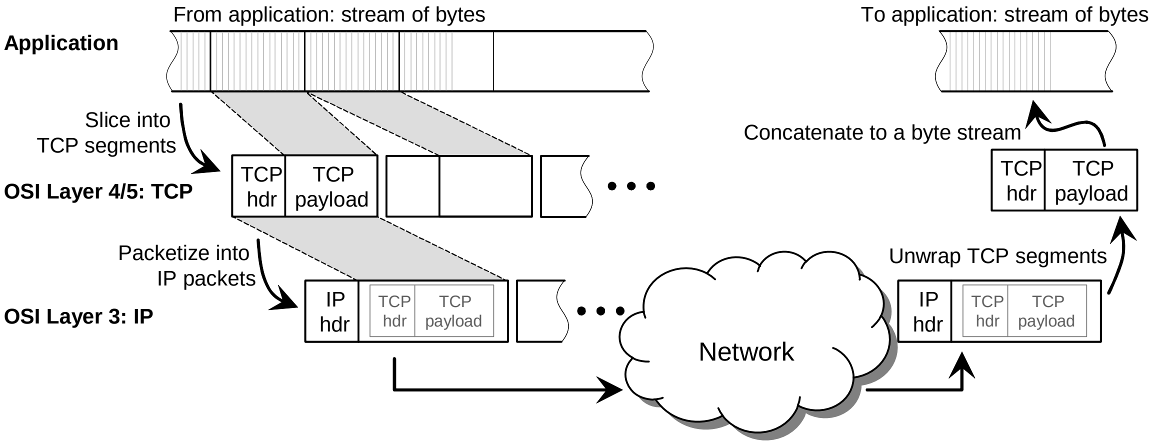

From application: stream of bytes
To application: stream of bytes
Application
Slice into TCP segments
OSI Layer 4/5: TCP
TCP hdr
TCP payload
Concatenate to a byte stream
Packetize into IP packets
OSI Layer 3: IP
IP hdr
Unwrap TCP segments
Network

Figure 2-2: TCP accepts a stream of bytes as input from the application, slices it into segments, and passes to the IP layer as IP packets. This process is reversed at the receiver.

of 30 bytes at a time. One end puts a stream of bytes into TCP and the same, identical stream of bytes appears at the other end.

A TCP session is continuously open and applications can at random times pass data to their TCP sender for transmission. A session is closed only upon an explicit request by the application, or if the communication link is accidentally broken. If an application decides to send more data, it must request opening a new session to do so.

It is common to use the term "segment" for TCP packets, because they represent segments of the input bytestream. The TCP segments are encapsulated into IP packets and sent to the destination (Figure 2-2). Compare this figure to Figure 1-13.

Like any other data packet, a TCP segment consists of the header and the data payload (Figure 2-3). The header consists of a 20-byte mandatory part, plus a variable-size options field. Most of regular TCP segments found on the Internet will have a fixed 20-byte header and the options field is rarely used. The description of the header fields is as follows.

Source port number and **destination port number:** These numbers identify the sending and receiving applications on their respective hosts. A network application is rarely the sole "inhabitant" of a host computer; usually, the host runs multiple applications (processes), such as a web browser, email client, multimedia player, etc. Similar to an apartment building, where to identify the recipient uniquely an apartment number is needed in addition to the street address, the applications communicating over TCP are uniquely identified by their hosts' IP addresses and the applications' port numbers.

Sequence number: The 32-bit sequence number field identifies the position of the *first* data byte of this segment in the sender's byte stream during data transfer (when SYN flag is not set). Because TCP provides a byte-stream service, each byte of data has a sequence number (instead of packet-based sequence numbers, as in Section 1.3).

Acknowledgment number: The 32-bit acknowledgment number field identifies the sequence number of the next data byte that the receiver expects to receive. This field is valid only if the ACK bit is set; otherwise, it should be ignored by the recipient of this segment.

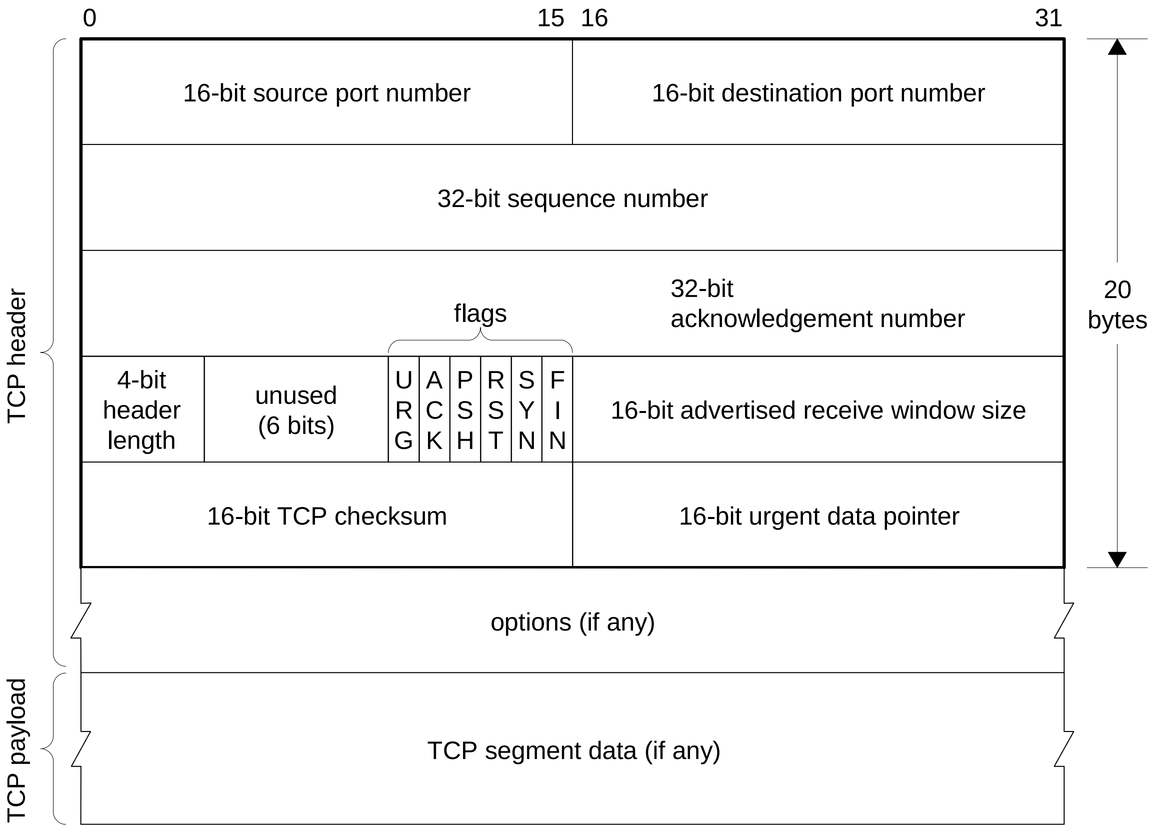

0
15
16
31
16-bit source port number
16-bit destination port number
32-bit sequence number
flags
32-bit acknowledgement number
4-bit header length
unused (6 bits)
URG
ACK
PSH
RST
SYN
FIN
16-bit advertised receive window size
16-bit TCP checksum
16-bit urgent data pointer
options (if any)
TCP segment data (if any)
20 bytes
TCP header
TCP payload

Figure 2-3: TCP segment format.

Header length: This field specifies the length of the TCP header in 32-bit words. This field is also known as the **Offset** field, because it informs the segment receiver where the data begins relative to the start of the segment. Regular header length is 20 bytes, so the default (and minimum allowed) value of this field equals 5. In case the options field is used, the value can be up to $4^2 - 1 = 15$, which means that the options field may contain up to $(15 - 5) \times 4 = 40$ bytes.

Unused: This field is reserved for future use and must be set to 0 (unless used for experimental purposes). Two bits of the Unused filed have recently been reserved for use in Explicit Congestion Notification (ECN), see Section 5.3.2 for details.

Flags: There are six bits allocated for the flags field, as follows:

> **URG:** If this bit is set, the urgent data pointer field of the header is valid (described later).
>
> **ACK:** When this bit is set, the acknowledgment number field of the header is valid and the recipient of this segment should pay attention to the acknowledgment number.
>
> **PSH:** If the Push bit is set, it requires the TCP receiver to pass the received data to the receiving application immediately. Normally, this bit is not set and the TCP receiver may choose to buffer the received segment until it accumulates more data in the receive buffer.
>
> **RST:** When set, the Reset bit requires the receiver of this segment to abort the connection because of some abnormal condition. For example, the sender of this segment may have received a segment it did not expect to receive and wants to abort the connection (e.g., a

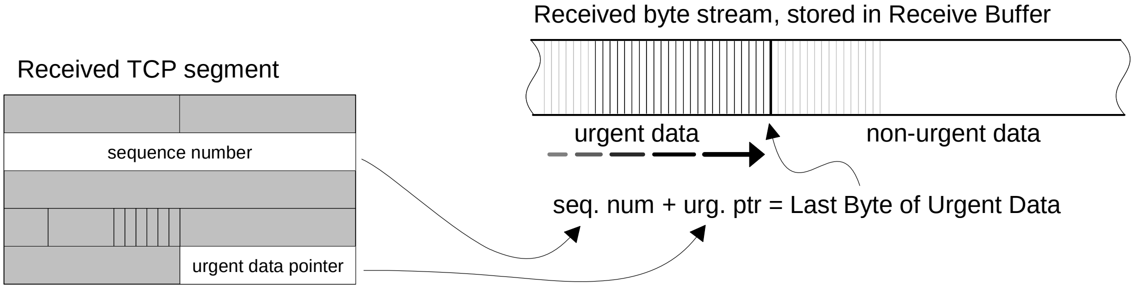

Received byte stream, stored in Receive Buffer
Received TCP segment
sequence number
urgent data pointer
urgent data
non-urgent data
seq. num + urg. ptr = Last Byte of Urgent Data

Figure 2-4: Urgent data pointer points to the last byte of the urgent data. First byte of urgent data never explicitly defined and any data in receive buffer up to last byte may be considered urgent.

potential attacker performing a port scan or simply a misconfigured host). (See Sidebar 2.1 in Section 2.2.)

SYN: This bit requests establishing a connection (discussed later).

FIN: When set, this bit informs the TCP receiver that the sender does not have any more data to send. The sender can still receive data from the receiver until it receives a segment with the FIN bit set from the receiver's direction.

Receive window size: This field specifies the number of bytes the sender of this segment is currently willing to accept (when acting as the receiver for TCP segments). This field can be used to control the flow of data and congestion, as described later in Sections 2.1.4 and 2.2, respectively.

Checksum: This field helps in detecting errors in the received segments.

Urgent data pointer: When the URG flag is set, the value of this field should be added to the value of the sequence number field to obtain the location of the last byte of the "urgent data" (Figure 2-4). The first byte of the urgent data is never explicitly defined. Because the TCP receiver passes data to the application in sequence, any data in the receive buffer up to the byte pointed by the urgent-data pointer may be considered urgent.

Options: The options field may be used to provide functions other than those covered by the regular TCP header. This field may be up to 40 bytes long and if its length is not a multiple of 32 bits, extra padding bits should be added. The options field is used by the TCP sender and receiver at the connection establishment time, to exchange some special parameters, as described later. Other uses include carrying timestamps for computing the retransmission timer (Section 2.1.3).

The reader may have noted that TCP header does not carry network address information or segment length information. This is because TCP relies on the underlying IP protocol and avoids duplication of such data. The TCP protocol software module on the receiving end obtains the length information from the IP module when the payload data is passed up the protocol stack.

The pseudo code in Listing 2-1 summarizes the TCP sender side protocol. In reality, both TCP sender and TCP receiver are implemented within the same TCP protocol module, because most applications need to be able to both send and receive data. Note also that the method `send()` is part of sender's code, whereas the method `handle()` is part of receiver's code. However, to keep the discussion manageable, I decided to focus only on sender's side. See also Figure 2-5 for the explanation of the buffering parameters and Figure 2-11 for TCP sender's state diagram. The reader can find a simple TCP protocol simulator written in the Java programming language on the book website.

Listing 2-1: Pseudo code for RTO retransmission timer management in a TCP sender.

```
public class TCPSender {

    // window size that controls the maximum number of outstanding segments
    //     equation (2.3a) explains how EffectiveWindow is calculated
    private long effectiveWindow;

    // maximum segment size (MSS)
    private long MSS;

    // sequence number of the last byte sent thus far (Figure 2-5), initialized randomly
    private long lastByteSent;

    // sequence number of the last byte for which the acknowledgment
    //     from the receiver arrived thus far, initialized randomly
    private long lastByteAcked;

    // list of unacknowledged segments that may need to be retransmitted
    private ArrayList unacknowledgedSegments = new ArrayList();

    // network layer protocol that provides services to TCP protocol (normally, IP protocol)
    private ProtocolNetworkLayer networkLayerProtocol;

    // class constructor
    public TCPSender(ProtocolNetworkLayer networkLayerProtocol) {
        this.networkLayerProtocol = networkLayerProtocol;

        lastByteSent = initial sequence number;  // randomly generated
        lastByteAcked = initial sequence number; // randomly generated
    }

    // reliable byte stream service offered to an upper layer (or application)
    //     takes as input a long message ('data' input parameter) and transmits in segments
    public void send(
        byte[] data, String destinationAddr,
        ProtocolLayer_iUP upperProtocol
    ) throws Exception {
        // slice the application into segments of size MSS and send one-by-one
        for (i = 0; i < (data.length % MSS); i++) {

          // if the sender already used up the limit of outstanding packets, then wait
          if (effectiveWindow - unacknowledgedSegments.size() > 0){
           suspend this thread;
           wait until some ACKs are received;
        }

        // create a new TCP segment with sequence number equal to LastByteSent;
21a        // if (data.length < (i+1)*MSS), i.e., the last data slice is smaller than one MSS
21b        //     then use padding or Nagle's algorithm (described later)
22         current_data_pointer = data + i*MSS;
22         TCPSegment outgoingSegment =
23             new TCPSegment(
23a              current_data_pointer, destinationAddr, upperProtocol
23b            );

24         if (RTO timer not already running) {  start the timer;  }

25         unacknowledgedSegments.add(outgoingSegment);
26         lastByteSent += outgoingSegment.getLength();

27         // hand the packet down the stack to IP for transmission
27a        // Note: outgoingSegment must be serialized to a byte-array as in Listing 1-1
28         networkLayerProtocol.send(              // (omitted for clarity)
28a             outgoingSegment, destinationAddr, this
28b        );
29     }

30     // upcall method (called from the IP layer), when an acknowledgment is received
31     public void handle(byte[] acknowledgment) {

32         // acknowledgment carries the sequence number of
32a        //     the next byte expected by the TCP receiver
33         if (acknowledgment.nextByteExpected > lastByteAcked) {
34             discard the acknowledged segment from
34a                the unacknowledgedSegments array;
35             lastByteAcked = acknowledgment.nextByteExpected;
36             if (lastByteAcked < lastByteSent) {
37                 re-start RTO timer; ;  // restart the retransmission timer
38             } // i.e., there are segments not yet acknowledged
39         }
40     }

41     // this method is called when the RTO retransmission timer expires (timeout)
41a    // this event signifies segment loss, and the oldest unacknowledged segment is retransmitted
42     public void RTOtimeout() {
43         retrieve the segment with sequence_number == LastByteAcked
43a            from unacknowledgedSegments array and retransmit it;

43         double the TimeoutInterval; // exponential backoff (see Sidebar 1.2)
44         start RTO timer countdown; // the timer for the retransmitted segment
45     }
46 }
```

The code description is as follows: … to be described … The reader should compare Listing 2-1 to Listing 1-1 in Section 1.1.4 for a generic protocol module.

The method `handle()`, which starts on Line 31, normally handles bidirectional traffic and processes TCP segments that are received from the other end of the TCP connection. Recall that TCP acknowledgments are piggybacked on TCP segments as part of the header (Figure 2-3).

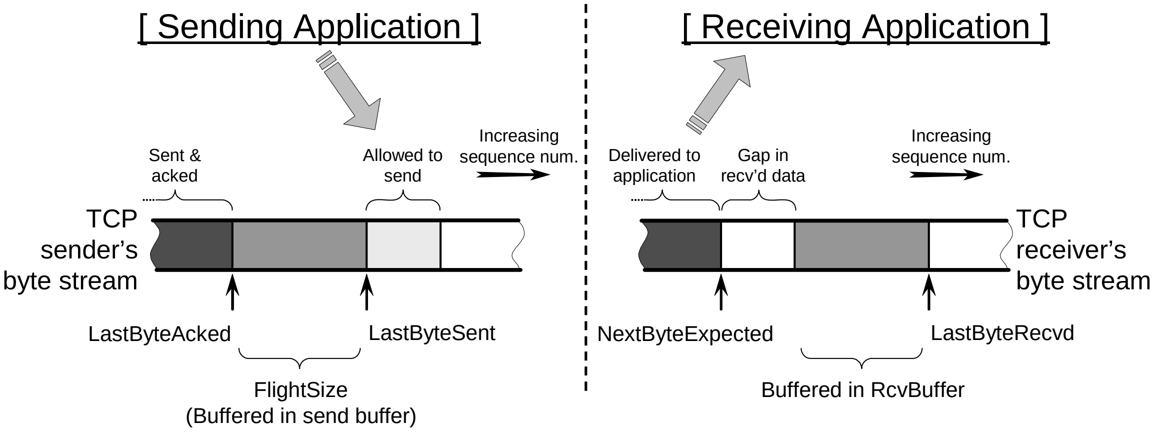


Figure 2-5: Parameters for TCP send and receive buffers pointing at different locations of the data bytestream.

Listing 2-1 provides only a basic skeleton of a TCP sender. The details will be completed in the following sections as we learn more about different issues.

The first action that TCP sender and receiver do is to establish a session. This task is accomplished by the first three segments that they exchange, known as the **three-way handshake** procedure (Figure 2-6). Here we briefly summarize the three-way handshake. The interested reader should consult another source for more details, e.g., [Stevens 1994; Peterson & Davie 2007; Kurose & Ross 2010].

The roles of "client" versus "server" are defined based on which endpoint listens for connection requests ("server") versus which endpoint requests a connection establishment ("client"). The first three TCP segments are special in that they do not contain data payload (i.e., they have only a header), and the SYN flag in the header is set (Figure 2-3). During the connection-establishment phase, the client and the server will transition through different states, such as LISTEN, SYN_SENT, and ESTABLISHED (marked in Figure 2-6 on the right-hand side). In this example, the client offers `RcvWindow` = 2048 bytes, and the server offers `RcvWindow` = 4096 bytes. Note that both client and server can simultaneously act as senders and receivers. They also exchange the maximum segment size of the future segments, `MSS` (to be described later, Table 2-1), and settle on the smaller one of 1024 and 512 bytes, i.e., 512 bytes.

As stated earlier, both sender and receiver instantiate their sequence numbers randomly. In Figure 2-6, the sender selects 122750000, while the receiver selects 2363371521. Hence, the sender sends a zero-data segment with the following information in the header:

```
    122750000:122750000(0)
```

The notation x:y(z) represents a segment of the data bytestream starting with bytes x through y, making a total of $z = y - x + 1$ bytes. The receiver acknowledges the sender's segment and sends its own initial sequence number by sending a header:

```
    2363371521:2363371521(0); ack 122750000
```

i.e., it sends zero data bytes and acknowledges zero data bytes received from the sender (the ACK flag is set). Given that these are zero-payload segments, the reader may wonder where the information about `MSS` and other parameters not present in the TCP header is carried. Such

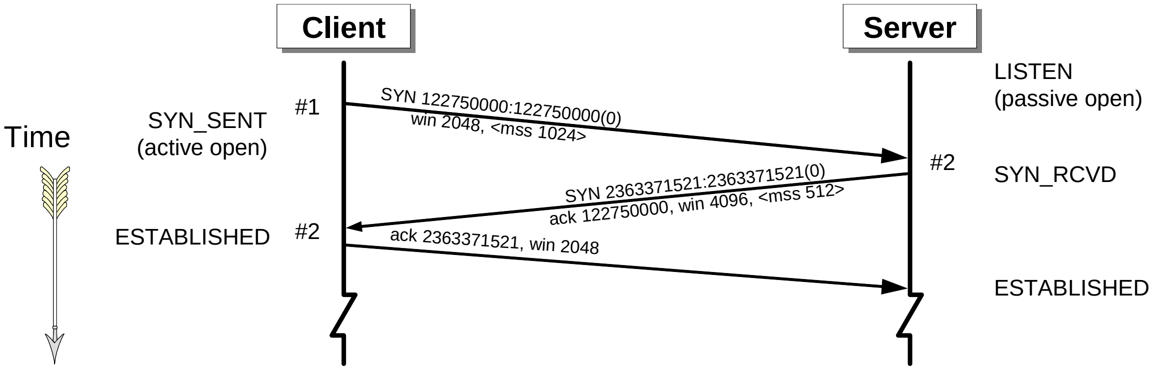


Figure 2-6: TCP connection establishment. See example session continuation in Figure 2-9.

information is carried in the Options field (Figure 2-3). After establishing the connection, the sender starts sending segments containing the application's data.

The framework for programming network application based on TCP connections for inter-process communication is called *network sockets*.

2.1.3 TCP Retransmission Timer

Problems related to this section: Problem 2.1 → ??, Problem 2.13, and Problem 2.15

An important parameter for reliable transmission is the retransmission timeout (RTO). This timer triggers the retransmission of packets that are presumed lost. We have encountered RTO in Section 1.3, but not much attention was paid to setting the value of this parameter. It is very important to set the right value for the RTO timer. For, if the timeout time is too short and the acknowledgment arrives soon after RTO expired, the packets will be unnecessarily retransmitted thus wasting the network bandwidth. And, if the timeout time is too long, the sender will unnecessarily wait when it should have already retransmitted thus underutilizing and perhaps wasting the network bandwidth.

It is relatively easy to set the timeout timer for single-hop networks because the propagation time remains effectively constant. However, in multihop networks queuing delays at intermediate routers and variable propagation delays over alternate paths introduce significant uncertainties.

TCP has a special algorithm for dynamically updating the retransmission timeout value. The details are available in RFC-2988 [Paxson & Allman, 2000], and here is a summary. The RTO timer value, denoted as `TimeoutInterval`, is initially set as 3 seconds. If a segment's acknowledgment is received before the retransmission timer expires, the TCP sender measures the round-trip time (RTT) for this segment, denoted as `SampleRTT`. The parameter `SampleRTT` is measured by including the timestamp of each segment in the *Options* field of TCP header (Figure 2-3), having it bounced back by the receiver, and then computing the difference of the current time minus the sending timestamp. TCP only measures `SampleRTT` for segments that have been transmitted *once* and *not* for segments that have been retransmitted.

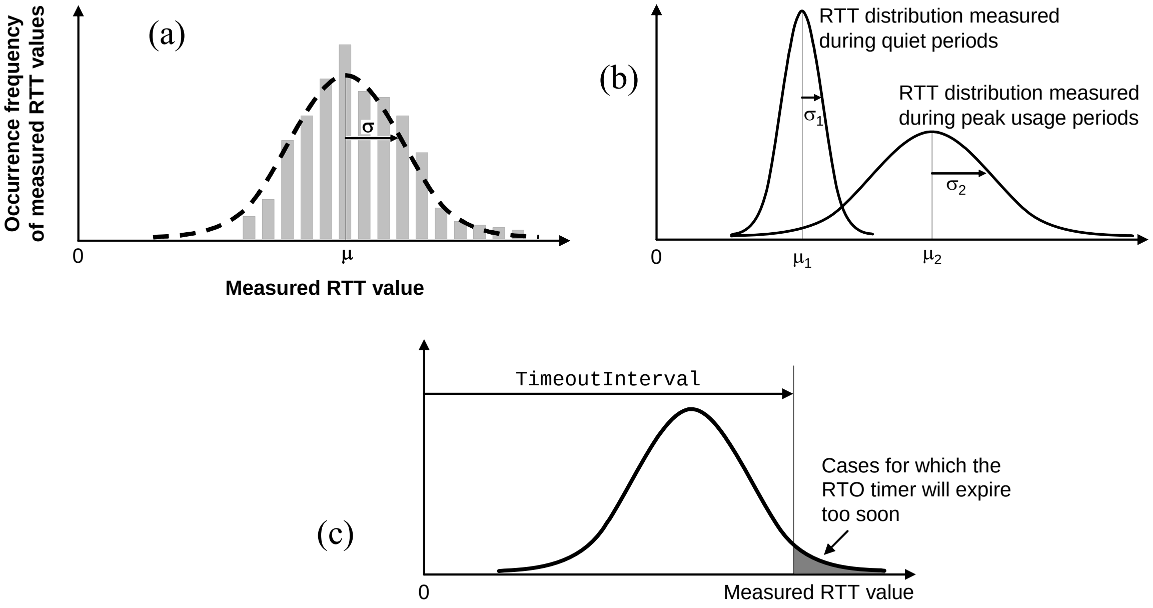


Figure 2-7: Distribution of measured round-trip times for a given sender-receiver pair.

Suppose you want to determine the statistical characteristics of round-trip time values for a given sender-receiver pair. As illustrated in Figure 2-7(a), the histogram obtained by such measurement can be approximated by a normal distribution[13] $N(\mu, \sigma)$ with the mean value μ and the standard deviation σ. If the measurement were performed during different times of the day, the obtained distributions may look quite different, Figure 2-7(b). This implies that the timeout interval should be *dynamically adapted* to the observed network condition. The remaining decision is about setting the value of `TimeoutInterval`. As illustrated in Figure 2-7(c), there will always be some cases for which any finite `TimeoutInterval` is too short and the acknowledgment will arrive after the timeout already expired. Setting `TimeoutInterval` = $\mu + 4\sigma$ will cover nearly 100 % of all the cases (see the Appendix on statistics). Therefore, for the subsequent data segments, the `TimeoutInterval` is set according to the following equation:

$$\texttt{TimeoutInterval}(t) = \texttt{EstimatedRTT}(t) + \max\{G, 4 \cdot \texttt{DevRTT}(t)\} \qquad (2.1)$$

where `EstimatedRTT`(t) is the currently estimated mean value of the round-trip time, `DevRTT`(t) is the round-trip time variation, and G is the system clock granularity (in seconds).[14] When the retransmission timer expires (because of a presumably lost packet), the earliest unacknowledged data segment is retransmitted and the next timeout interval is set to twice the previous value (the initial value being `backoff` = 1):

$$\texttt{backoff}(t) = 2 \times \texttt{backoff}(t\text{–}1)$$

[13] In reality, multimodal RTT distributions (i.e., with several peaks) are observed. The interested reader can find relevant links at this book's website—follow the link "Related Online Resources," then look under the topic of "Network Measurement."

[14] Using the clock granularity G is not required and it is usually omitted in most books. RFC-6298 recommends it because if the delay variability `DevRTT`(t) equals zero (perhaps because of coarse clock granularity), the sender may behave oddly and retransmit segments to early. See discussion related to Example 2.1 in Section 2.2.

Search the Web for RTT distribution measurement

$$\text{RTO timer} \leftarrow \texttt{backoff}(t) \times \texttt{TimeoutInterval}(t) \tag{2.2}$$

This property of doubling the RTO on each timeout is known as **exponential backoff**.[15]

The round-trip time is continually estimated as:

$$\texttt{EstimatedRTT}(t) = (1-\alpha) \cdot \texttt{EstimatedRTT}(t-1) + \alpha \cdot \texttt{SampleRTT}(t)$$

The initial value is set as $\texttt{EstimatedRTT}(0) = \texttt{SampleRTT}(0)$ for the first RTT measurement. This approach to computing the running average of a variable is called Exponential Weighted Moving Average (EWMA). Similarly, the current standard deviation of RTT, $\texttt{DevRTT}(t)$, is estimated as:

$$\texttt{DevRTT}(t) = (1-\beta) \cdot \texttt{DevRTT}(t-1) + \beta \cdot |\texttt{SampleRTT}(t) - \texttt{EstimatedRTT}(t-1)|$$

The initial value is set as $\texttt{DevRTT}(0) = \dfrac{1}{2}\texttt{SampleRTT}(0)$ for the first RTT measurement. The recommended values of the control parameters α and β are $\alpha = 0.125$ and $\beta = 0.25$. These values were determined empirically. To prevent spurious retransmissions of segments that are only delayed and not lost, RFC-2988 conservatively recommends the minimum `TimeoutInterval` as 1 second.

In theory, it is simplest to maintain an individual retransmission timer for each outstanding packet. In practice, timer management involves considerable complexity, so most protocol implementations maintain single timer per sender. RFC-2988 recommends maintaining *single* retransmission timer per TCP sender, even if there are multiple transmitted-but-not-yet-acknowledged segments. Of course, individual implementers may decide otherwise, but this book follows the single-timer recommendation for TCP.

TCP sends segments in *bursts* (or, groups of segments), every burst containing the number of segments limited by the current window size. Recall from Section 1.3.2 that in all sliding window protocols, the sender is allowed to have only up to the window-size outstanding amount of data (yet to be acknowledged). The same holds for the TCP sender. Once the window-size worth of segments is sent, the sender stops and waits for acknowledgments to arrive. For every arriving ACK, the sender is allowed to send certain number of additional segments, as governed by the rules described later. The retransmission timer management is included in the pseudo code in Listing 2-1 (Section 2.1.2). The following summary extracts and details the key points of the retransmission timer management from Listing 2-1:

```
In method TCPSender.send(), Listing 2-1 // called by the application from above
 in Line 24:
    if (RTO timer not already running) {
        start the RTO timer using Eq. (2.2);
    }
```

[15] We encountered exponential backoff in Section 1.3.3 (Sidebar 1.2). In the TCP case, the sender assumes that concurrent TCP senders are contending for the network resources (router buffers), thereby causing congestion and packet loss. To reduce the congestion, the sender doubles the retransmission delay by doubling its RTO. Of course, this assumption may be incorrect because other senders may be using UDP or another protocol that does not apply congestion control or care about fairness.

```
In method TCPSender.handle() // called by IP layer from below when ACK arrives
    in Lines 36 - 38:        // for a segment not previously acknowledged ("new ACK")
        if ((lastByteAcked < lastByteSent) {
            calculate the new value of TimeoutInterval; // Eq. (2.1)
            reset the RTO backoff to 1;
            re-start the RTO timer using Eq. (2.2);
        }

In method TCPSender.RTOtimeout(), Listing 2-1 // triggered by RTO timer timeout
    in Line 43:
           double the RTO backoff;
           re-start the RTO timer using Eq. (2.2);
```

An important peculiarity to note about TCP running a single retransmission timer per connection is as follows. When a window-size worth of segments is sent, the timer is set for the first one, assuming that the timer is not already running (Line 24 in Listing 2-1). For every acknowledged segment of the burst, the timer is restarted for its subsequent segment in Line 37 in Listing 2-1. Thus, the actual timeout time for the segments towards the end of an earlier burst can run quite longer than for those near the beginning of the burst. An example will be seen later in Section 2.2 in the solution of Example 2.1.

2.1.4 TCP Flow Control

TCP receiver accepts out-of-order segments, but they are buffered and not delivered to the application above the TCP layer before the gaps in sequence numbers are filled. For this buffering, the receiver allocates memory space of the size `RcvBuffer`, which is typically set to 4096 bytes, although older versions of TCP set it to 2048 bytes. The rationale for this relatively small buffer size is that many gaps in the segment sequence likely indicate a bad connection and the sender needs to slow down instead of overwhelming the network and receiver with out-of-order segments. Lost segments will need to be retransmitted and will introduce various delays associated with retransmission, including the timeout timer delay. It may be better to slow down and succeed transmitting at a lower rate. We will discuss the topic of loss detection and sending-strategy adaptation in Section 2.2. The *receive buffer* is used to store in-order segments as well, because the application may be busy with other tasks and does not fetch the incoming data immediately as they arrive. For simplicity, in the following discussion we will assume that in-order segments are immediately fetched by the application, unless stated otherwise.

To avoid having its receive buffer overrun, the receiver continuously advertises the remaining buffer space to the sender using a field in the TCP header; we call this parameter `RcvWindow`. Its value is dynamically changing to reflect the current occupancy state of the receiver's buffer. The sender should never have more than the current `RcvWindow` amount of data outstanding (unacknowledged). This process is called *flow control*. Figure 2-8 illustrates the difference between flow control and congestion control (congestion control is described later in Section 2.2).

Figure 2-9 shows how an actual TCP session might look like, continuing from Figure 2-6 after the connection is established. In our example, the client happens to be the "sender," but server or both client and server can simultaneously be senders and receivers. We use the same notation so that 0:511(512) represents a segment of the data bytestream starting with bytes 0 through 511,

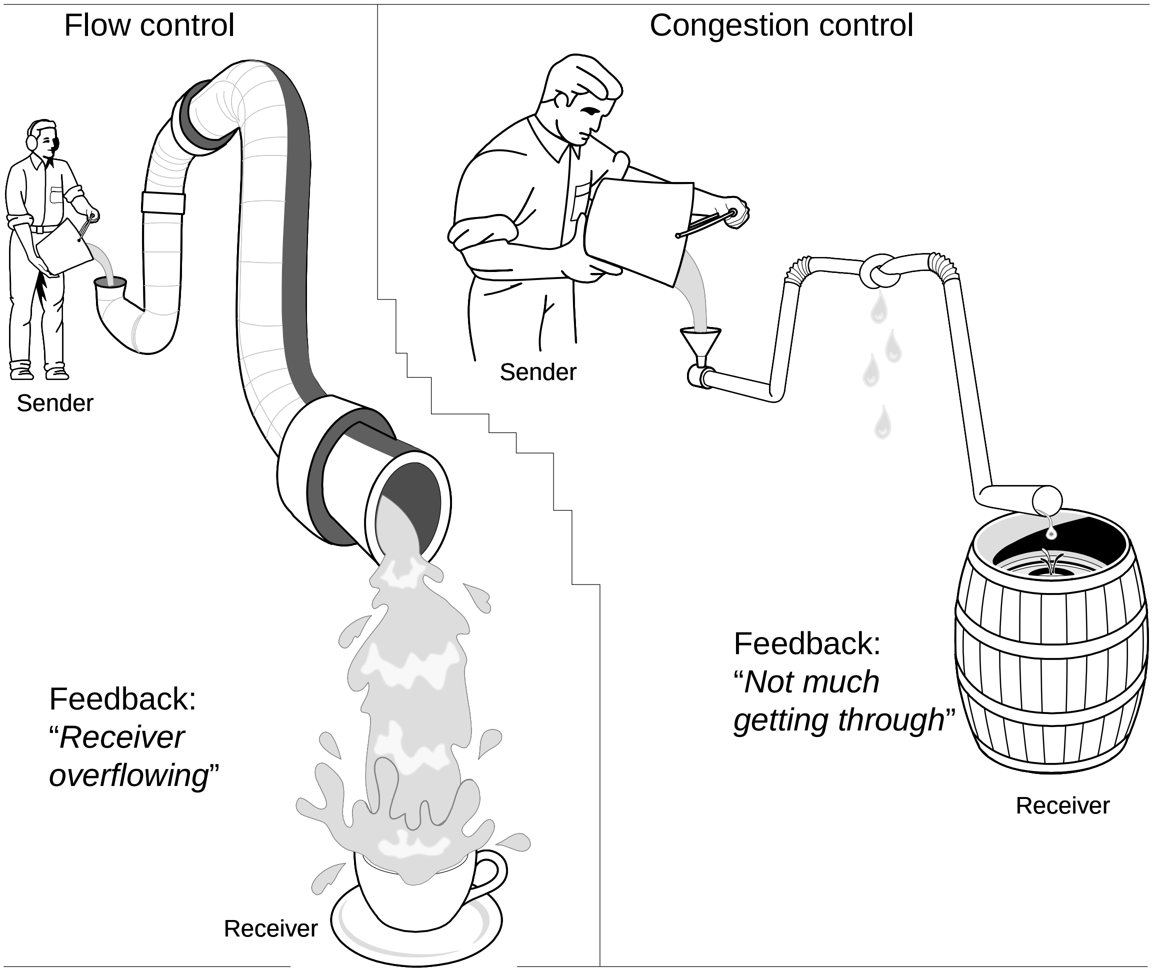


Figure 2-8: Flow control compared to congestion control.

which is a total of 512 bytes (the number in the parentheses). To avoid further cluttering the diagram, we simply assume that the sender started with the sequence number equal to zero instead of realistic numbers shown in Figure 2-6. In Figure 2-9, the server sends no data to the client, so the sender keeps acknowledging the first segment from the receiver, and the acknowledgments from the sender to receiver carry the value 0 for the sequence number (recall that in Figure 2-6 the actual value of the server's sequence number is 2363371521).

After establishing the connection, the sender starts sending packets. Figure 2-9 illustrates how TCP incrementally increases the number of outstanding segments. This procedure is called **slow start** and will be described later. TCP assigns *byte-based* sequence numbers, but for simplicity we usually show imagined packet-based sequence numbers. Note that the receiver is *not* obliged to acknowledge *individually* every single in-order segment—it can use *cumulative ACKs* to acknowledge several in-order segments up to the most recent one. However, the receiver *must* immediately generate (duplicate) ACK—*dupACK*—for every out-of-order segment, because dupACKs help the sender detect segment loss (as described in Section 2.2).

Note that the receiver might send dupACKs even for successfully transmitted segments because of random re-ordering of segments in the network. This is the case with segment #7 (detail at the bottom of Figure 2-9), which arrives after segment #8. Therefore, if a segment is delayed further

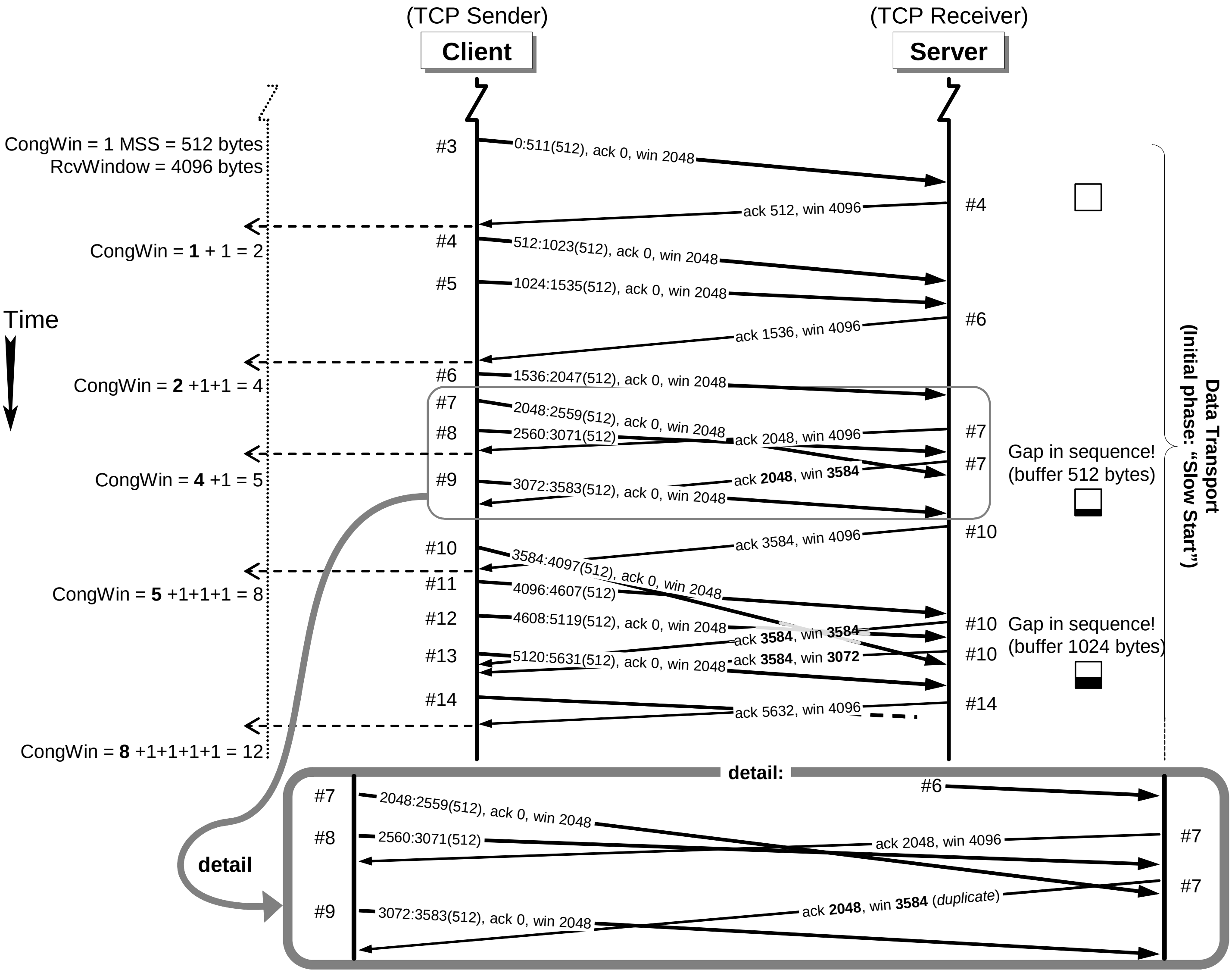


Figure 2-9: The time line of the initial part of a TCP session, continued from Figure 2-6. See text for details. (The CongWin parameter on the left represents the sender's window size, described in Section 2.2.)

than three or more of its successors, the duplicate ACKs will trigger the sender to re-transmit the delayed segment. The receiver may eventually receive a duplicate of such a segment in which case the receiver would simply discard the duplicate.

The receiver may sometimes advertise a window of `RcvWindow` = 0, for example because the receiving application is busy with something else and has not been fetching the received data. Zero-size window effectively stops the sender from transmitting data, until the window becomes nonzero. However, the receiver will not proactively notify the sender if the receive buffer empties. If there were still unacknowledged packets, some of them may arrive after the receive window became nonzero. The subsequent ACK packet will carry a new, nonzero `RcvWindow` advertisement and the sender would learn that the receive buffer has emptied. However, an ACK may also be lost. Acknowledgments are not reliably transmitted—TCP does not acknowledge acknowledgments, it only ACKs segments containing data. In any case, to prevent a deadlock from occurring the sender must initiate a **persist timer** (also called *the zero-window-probe timer*) when it receives a zero-window advertisement. The persist timer causes the sender to query the receiver periodically, to find out if the receive window has grown. These tiny, 1-byte segments from the sender are called *window probes* and are treated by the receiver the same as any other segment. The sender keeps sending these tiny segments until the effective window becomes non-zero or a segment loss is detected. The first value of the persist timer may be set to `TimeoutInterval` and the normal TCP exponential backoff will be used for calculating the subsequent values.

2.2 Congestion Control

TCP is a window-based ARQ protocol (Section 1.3.2) and as such, the number of outstanding packets is limited by the window size. However, unlike protocols considered in Section 1.3.2 that work over a single link, TCP communicates over arbitrary network paths with many intermediate nodes. Therefore, its window size depends not only on the storage space available on the receiver for storing out-of-order packets, but also on the storage space available in the intermediate nodes (routers). This duality of window size constraints is illustrated in Figure 2-8. A key problem addressed by TCP is to determine the optimal window size, which must be estimated *dynamically* because available network resources on different routers vary dynamically and frequently.

The problem is illustrated in Figure 2-10, where the whole network is abstracted as a single *bottleneck router*. (For simplicity, we assume that all packets belonging to a TCP connection travel over the same path, which in reality may not be true.) Both the network and the receiver have limited memory capacity to hold packets from our session, denoted respectively as `CongWindow`(t) and `RcvWindow`(t). Both capacities vary with time. It is easy for the receiver to know about its own available buffer space and advertise the current window size `RcvWindow`(t) to the sender in the header of an ACK segment. The problem is with the intermediate router(s), which serve data flows between many sources and receivers. As Figure 2-8(right) suggests, the network is congested somewhere and leaks (drops) packets. Bookkeeping and policing of fair use of router's resources is a difficult task, because the router must forward packets as quickly as possible, and it is practically impossible to dynamically determine the "right window size" `CongWindow`(t) of the router's memory allocated for each flow and advertise it back to its sender. Even if it could be done, by the time this information reached the sender, it would be long outdated.

TCP maneuvers to avoid network congestion in the first place, and controls the damage if congestion occurs. The key characteristic of TCP is that all the intelligence for congestion avoidance and control is in the end hosts—no help is expected from the intermediary network nodes.[16]

[16] This statement is not true anymore, because recently active queue management techniques have been deployed in the network to help TCP to better manage congestion; see Section 5.3.

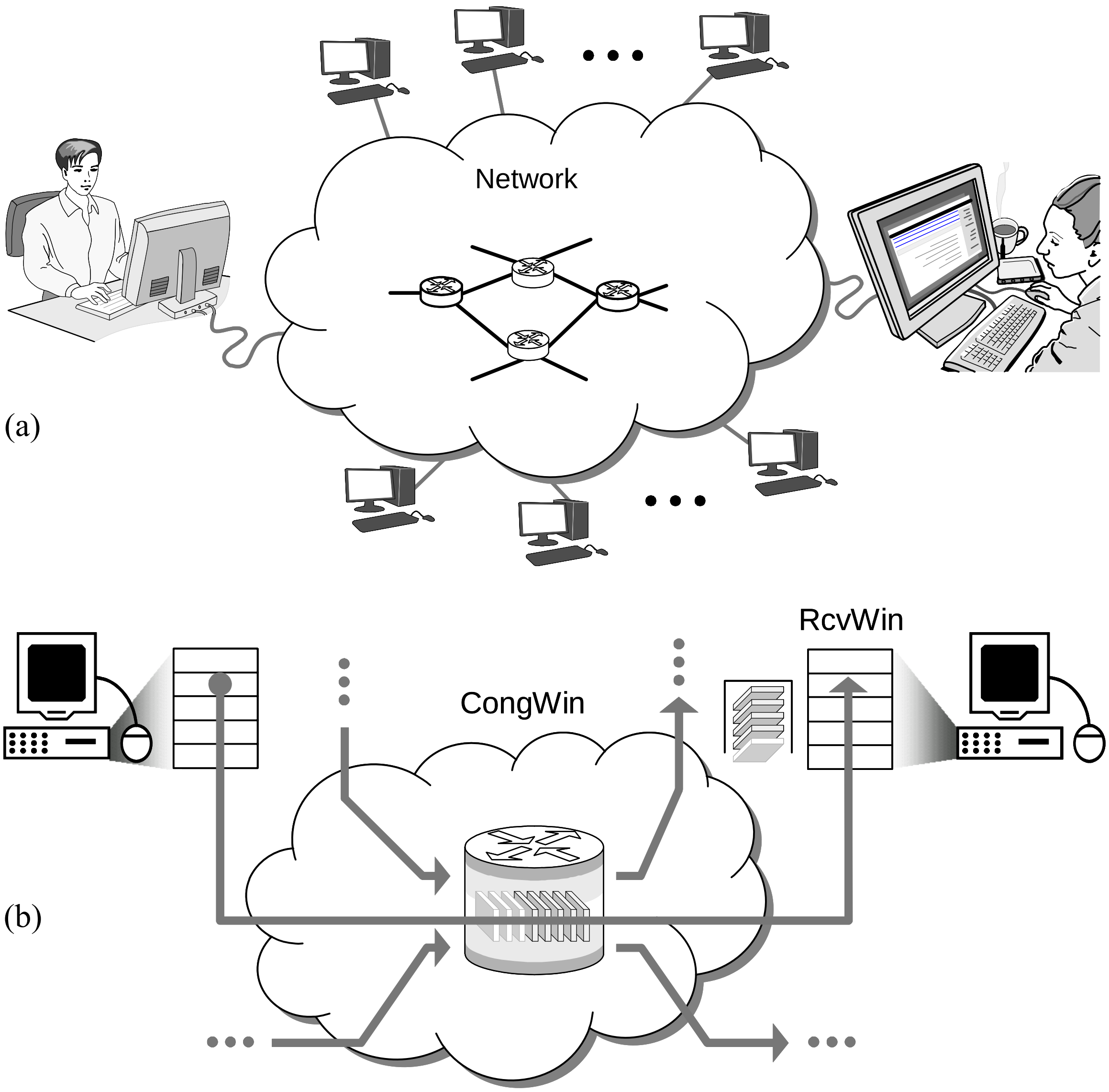


Figure 2-10: Simple congestion-control scenario. The entire network (a) is abstracted as a single router (b) that has `CongWin` memory slots available for packets from our session.

TCP evolved over the last three decades, so it is hard to talk about one TCP algorithm for congestion control. Early versions of TCP would start a connection with the sender injecting multiple data segments into the network, up to the window size advertised by the receiver. The problems would arise due to intermediate router(s), which must queue the packets before forwarding them. If that router ran out of memory space, large number of packets would be lost and had to be retransmitted. Jacobson [1988] showed how this naïve approach could reduce the throughput of a TCP connection drastically.

An improved TCP approached this problem by having the sender dynamically probe the network and adjust the amount of data in flight to match the bottleneck resource. The algorithm used by TCP sender can be summarized as follows:

1. Sender starts with a small size window (one segment)

2. Sends a burst (size of the current sender window) of packets into the network
3. Waits for a feedback about the success rate (acknowledgments from the receiver end)
4. When obtains the feedback:
    a. If the transmissions are successful (i.e., acknowledged), *increases* the sender window size and goes to Step 2
    b. If a loss is detected, performs the loss recovery procedure, *decreases* the sender window size and goes to Step 2

This simplified procedure will be elaborated later. It is important to note that TCP *controls* congestion in the sense that it first needs to cause congestion, next to observe it through some sort of feedback, and then to react by reducing the input. This cycle is repeated in a never-ending loop (as long as there is application data to send). Section 2.4 describes other variants of TCP that try to avoid congestion, instead of causing it (and then controlling it).

There are several design questions that TCP designers needed to answer to make the above summarized algorithm work:

Q1: How to determine how much data can be effectively sent (the effective window size)?

Q2: How to detect packet loss that occurred during transport in the network?

Q3: How to react when a packet loss is detected?

There are many different versions of TCP that answer these questions differently. We begin by summarizing what is roughly common for different approaches and then providing more details on different TCP variants. Table 2-1 shows the most important parameters used in TCP congestion control (all the parameters are maintained in integer units of bytes). Buffering parameters for TCP sender and receiver are shown in Figure 2-5. Figure 2-11 and Figure 2-12 summarize the algorithms run by the sender and receiver. These are digested from RFC-5681 and RFC-2581 and the reader should check the details on TCP congestion control in [Allman *et al*. 1999; Stevens 1997]. [Stevens 1994] provides a detailed overview with the traces of actual runs.

Table 2-1. TCP congestion control parameters (measured in integer number of bytes). See RFC-2581 for details and see Figure 2-5 for an illustration of the parameters.

Variable	Definition
`MSS`	The maximum segment size that the sender can transmit. MSS does *not* include the TCP/IP headers and options—only the data payload. This value can be based on the maximum transmission unit (MTU) of the first link, the path MTU discovery algorithm, or other factors. By setting MSS to the path MTU, the sender may avoid packet fragmentation (Section 1.4.1), although this is difficult to achieve because routers change routes dynamically. [Note that RFC-2581 distinguishes the sender maximum segment size (`SMSS`) and the receiver maximum segment size (`RMSS`).]
`RcvWindow`	The size of the most recently advertised receiver window.
`CongWindow`	Sender's current estimate of the available buffer space in the bottleneck router.

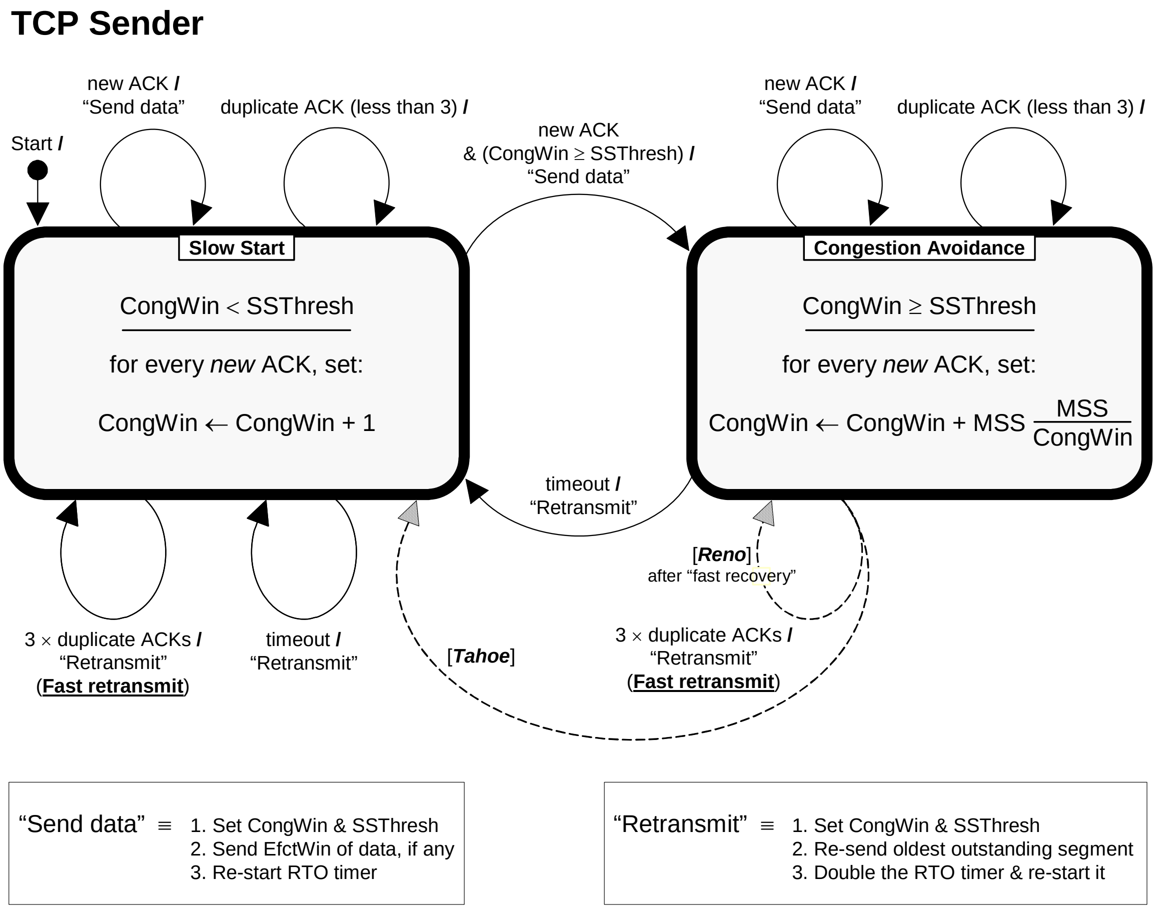

TCP Sender
Start /
new ACK / "Send data"
duplicate ACK (less than 3) /
Slow Start
CongWin < SSThresh
for every new ACK, set:
CongWin ← CongWin + 1
new ACK & (CongWin ≥ SSThresh) / "Send data"
Congestion Avoidance
CongWin ≥ SSThresh
for every new ACK, set:
CongWin ← CongWin + MSS MSS/CongWin
timeout / "Retransmit"
[Reno] after "fast recovery"
3 × duplicate ACKs / "Retransmit" (Fast retransmit)
timeout / "Retransmit"
[Tahoe]
3 × duplicate ACKs / "Retransmit" (Fast retransmit)
"Send data" ≡ 1. Set CongWin & SSThresh 2. Send EfctWin of data, if any 3. Re-start RTO timer
"Retransmit" ≡ 1. Set CongWin & SSThresh 2. Re-send oldest outstanding segment 3. Double the RTO timer & re-start it

Figure 2-11: Simplified TCP sender state diagram. CongWin and SSThresh are computed differently for different TCP senders (Table 2-2). This figure will be detailed in Figure 2-15 for TCP Tahoe and in Figure 2-19 for TCP Reno.

`LastByteAcked`	The highest sequence number currently acknowledged.
`LastByteSent`	The sequence number of the last byte the sender sent.
`FlightSize`	The amount of data that the sender has sent, but not yet had acknowledged.
`EffectiveWindow`	The maximum amount of data that the sender is currently allowed to send. At any given time, the sender must not send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of `CongWindow` and `RcvWindow`.
`SSThresh`	The slow start threshold used by the sender to decide whether to employ the slow-start or congestion-avoidance algorithm to control data transmission. The *slow start* algorithm is used when `CongWindow` < `SSThresh`, while the *congestion avoidance* algorithm is used when `CongWindow` > `SSThresh`. When `CongWindow` and `SSThresh` are equal the sender may use either slow start or congestion avoidance.

As a window-based protocol, TCP sender must assure at all times the invariant that the amount of unacknowledged data is smaller than the window size:

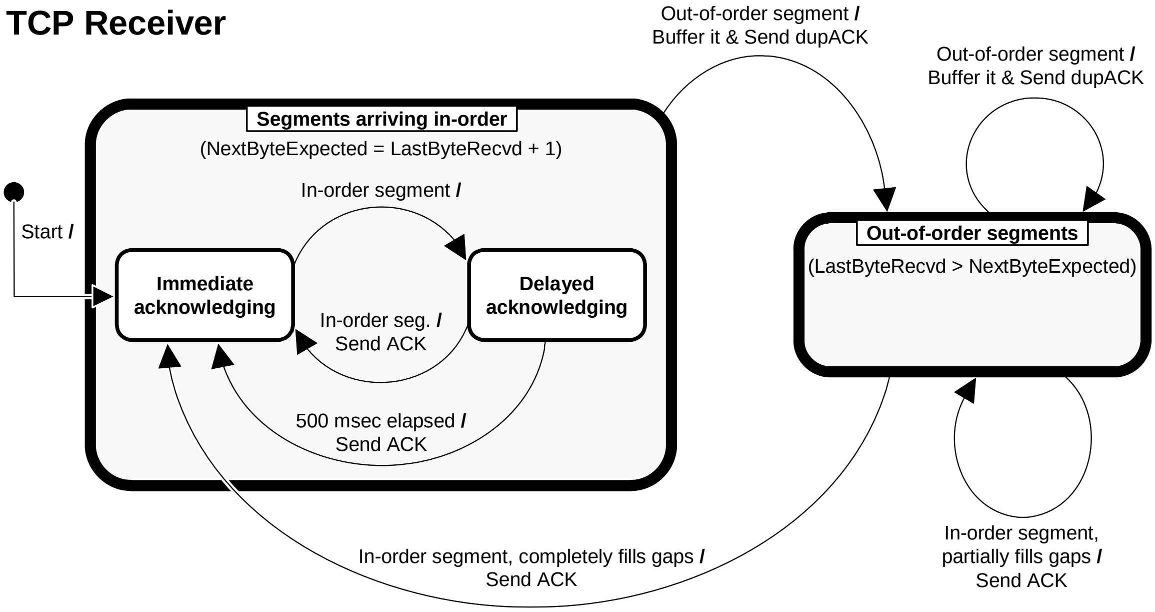

TCP Receiver
Segments arriving in-order
(NextByteExpected = LastByteRecvd + 1)
Start /
Immediate acknowledging
Delayed acknowledging
In-order segment /
In-order seg. / Send ACK
500 msec elapsed / Send ACK
Out-of-order segment / Buffer it & Send dupACK
Out-of-order segment / Buffer it & Send dupACK
Out-of-order segments
(LastByteRecvd > NextByteExpected)
In-order segment, completely fills gaps / Send ACK
In-order segment, partially fills gaps / Send ACK

Figure 2-12: TCP receiver state diagram.

$$\texttt{LastByteSent} \le \texttt{LastByteAcked} + \min\{\texttt{CongWindow}, \texttt{RcvWindow}\}$$

Therefore, the amount of unacknowledged data (denoted as `FlightSize`) should not exceed this value at any time:

$$\texttt{FlightSize} = \texttt{LastByteSent} - \texttt{LastByteAcked} \le \min\{\texttt{CongWindow}, \texttt{RcvWindow}\}$$

At any moment during a TCP session, the maximum amount of data the TCP sender is allowed to send (marked as "*allowed to send*" in Figure 2-5 and also known as *usable window*) is:

$$\texttt{EffectiveWindow} = \min\{\texttt{CongWindow}, \texttt{RcvWindow}\} - \texttt{FlightSize} \tag{2.3a}$$

If the calculated `EffectiveWindow` equals zero, nothing is sent (i.e., the sender does not send a zero-data segment but remains silent). We assume that the sender can only send `MSS`-size segments; the sender defers transmission until it collects at least an `MSS` worth of data. This is not always true, and the application can request a speedy transmission, thus generating small packets, known as *tinygrams*. The application does this using the `TCP_NODELAY` socket option, which sets PSH flag (Figure 2-3). This is particularly the case for interactive applications, such as telnet or secure shell. Nagle's algorithm [Nagle 1984] constrains the sender to have unacknowledged at most one segment smaller than one `MSS`. In other words, the sender cannot send another small segment before all previous small segments are acknowledged. For simplicity, we assume that the effective window is always rounded down to the integer number of `MSS`-size segments using the floor operation $\lfloor \cdot \rfloor$:

$$\texttt{EffectiveWindow} = \lfloor \min\{\texttt{CongWindow}, \texttt{RcvWindow}\} - \texttt{FlightSize} \rfloor \tag{2.3b}$$

Figure 2-9 illustrates the TCP slow start phase, with the congestion window sizes shown on the left side of the figure. In **slow start**, `CongWindow` starts at one segment and gets incremented by one segment every time an ACK is received. As seen, this opens the congestion window *exponentially*: send one segment, then two, four, eight and so on.

The only "feedback" TCP receives from the network is by indirectly detecting packets lost in transport. TCP sender detects packet loss based on two types of events (whichever occurs first):

1. RTO retransmission timer expiration (Section 2.1.3)
2. Reception of three[17] duplicate ACKs (*four* identical ACKs without the arrival of any other intervening packets from the receiver)

It is important to keep in mind that the sender declares packet loss based on *indirect observation*—the sender cannot observe packet loss as it occurs somewhere in the network, on a router. Rather, the sender observes signs of loss: acknowledgment did not arrive before RTO timer expired, or three duplicate acknowledgments arrived consecutively. These signs do not tell *when a packet loss happened* (somewhere in the network); they just say that it is feasible to ***assume** that a loss might have happened* (based on these signs observed at the TCP sender). It may also be that the packet (or its acknowledgment) has not been lost but rather just delayed and will arrive later. Packet loss happens in the network and the network is not expected to notify the TCP endpoints about the loss. Packet loss is of little concern to TCP receiver, except that it buffers out-of-order segments and waits for the gap in sequence to be filled. TCP sender is the one mostly concerned about the loss and the one that takes actions in response to a detected loss.

The next design question is how to react to a presumed packet loss. TCP designers decided to focus on the main cause of loss (at that time, 1980s and 1990s), which was memory overflow on a bottleneck router (Figure 2-10). Of course, packets may be lost to channel noise or even to a broken link, but these other types of loss were uncommon. A design is good as long as its assumptions hold and TCP has worked fine over wired networks. However, recent emergence of wireless data networks has challenged this underlying assumption about packet loss and caused great problems (see Section 2.5).

There are two aspects of reacting to presumed loss: (a) loss recovery, and (b) adaptation of the sending strategy. As an ARQ protocol, TCP performs loss recovery by retransmission. As for the adaptation, given the assumption that loss is caused by router memory overflow, TCP sender dampens its transmission rate to allow the router memory drain. There are many versions of TCP, each having different reaction to loss. The two most popular versions are **TCP Tahoe** (details described in Section 2.2.1) and **TCP Reno** (Section 2.2.2), of which TCP Reno is more recent and currently prevalent in the Internet (updated as TCP NewReno, see Section 2.2.3). Table 2-2 summarizes how they detect and handle segment loss.

Using duplicate acknowledgments as an indication of packet loss is an interesting approach. Recall from Section 1.3.2 that the Go-Back-N protocol reacts on duplicate acknowledgments to retransmit all of the currently unacknowledged packets (i.e., the full window of *N*). However, TCP designers realized that duplicate ACKs are telling the sender that some packets got through (although out of order) and there is no need to retransmit them. Because the regular TCP does not

[17] The reason for *three* dupACKs is as follows. Because TCP does not know whether a lost segment or just a reordering of segments causes a dupACK, it waits for a small number of dupACKs to be received. It is assumed that if there is just a reordering of the segments, there will be only one or two dupACKs before the reordered segment is processed, which will then generate a fresh ACK. Such is the case with segments #7 and #10 in Figure 2-9. If three or more dupACKs are received in a row, it is a strong indication that a segment has been lost. TCP implementations usually allow changing the threshold of dupACKs and using values other than three.

Table 2-2: How different TCP senders detect and deal with segment loss.

Event	TCP Version	TCP Sender's Action
Timeout	Tahoe	Set `CongWindow` = 1×`MSS` and `SSThresh` = max {⌊½ `CongWindow`⌋, 2×`MSS`}
	Reno	Set `CongWindow` = 1×`MSS` and `SSThresh` = max {⌊½ `FlightSize`⌋, 2×`MSS`}
≥ 3×dup ACKs	Tahoe	Set `CongWindow` = 1×`MSS` and `SSThresh` = max {⌊½ `CongWindow`⌋, 2×`MSS`}
	Reno	Set `CongWindow` = max {½ `FlightSize`, 2×`MSS`} + 3×`MSS` and `SSThresh` = max {⌊½ `FlightSize`⌋, 2×`MSS`}

send specific acknowledgments like the Selective-Repeat protocol (there is a TCP version that does), the sender cannot know which packets got through and which were lost. Therefore, the sender considers it safe to assume only that the oldest outstanding packet is lost and retransmits only that one. After three duplicate ACKs, TCP sender also performs **fast retransmit** of what appears to be the lost segment, without waiting for the RTO timer to expire. The name "fast retransmit" comes from the fact that when a segment is lost, three duplicate ACKs will arrive much sooner than the RTO timer will expire. We will see later how this approach can be refined in different versions of TCP.

Upon detecting the loss, TCP sender adapts its sending strategy to *avoid* further loss by reducing its transmission rate. As seen in Table 2-2, different versions react differently to three dupACKs: the more recent version of TCP (TCP Reno) reduces the congestion window size to a lesser degree than the older version (TCP Tahoe). The reason is that researchers realized that three dupACKs signalize lower degree of congestion than RTO timeout. If the RTO timer expires, this may signal a "severe congestion" where nothing is getting through the network. Conversely, three dupACKs imply that three packets got through, although out of order, so this signals a "mild congestion."

The initial value of the slow start threshold `SSThresh` is commonly set to 65535 bytes = 64 KB. When a TCP sender detects segment loss, the new value of `SSThresh` is calculated as shown in Table 2-2. This calculation is different for different TCP versions, as explained later.

SIDEBAR 2.1: Optional TCP Keep-alive Messages

♦ When two hosts have established a TCP session, the connection may remain silent for long periods. Maintaining TCP connections requires resources, so an endpoint may wish to know if the other endpoint is still connected and release resources if not. Also, protocols such as NAT (Section 9.3.3) may terminate inactive TCP connections. RFC-1122 specifies (page 101) that "keep-alive" segments may be used to determine if the connection is still valid, and maintain it alive if needed. An endpoint can periodically send an empty probe TCP segment with the ACK flag set to the other endpoint, soliciting a response. If the connection is valid, the remote host (which does not need to support the keepalive feature, just a regular TCP) will reply with no data and the ACK set. If a certain number of keepalives are sent and no response (ACK) is received within a timeout time then the sending host will terminate the connection from its end.

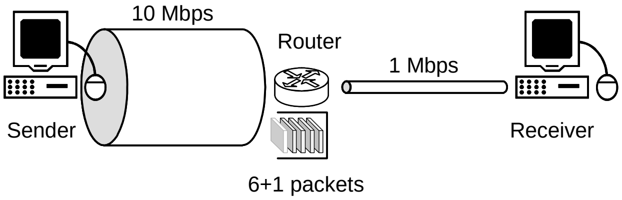
If a connection has been terminated due to a TCP keep-alive timeout and the other host eventually sends a packet for the old connection, the host that terminated the connection will send a packet with the RST flag set to signal the other host that the old connection is no longer valid. This will force the other host to terminate its end of the connection and a new connection could be established.

Typically, TCP keep-alive probes are sent every 45 or 60 seconds on an idle TCP connection, although some applications use longer intervals, e.g., 2 hours. The connection is dropped after three ACKs in sequence are missed. Although TCP keepalives are not universally accepted, most implementations include this optional feature and have it activated. RFC-2525 describes typical implementation problems with TCP keepalive. Note that the keepalive feature is not included in Figure 2-11.

Congestion can occur when packets arrive on a big pipe (a fast LAN) and are sent out a smaller pipe (a slower WAN). Congestion can also occur when multiple input streams arrive at a router whose output capacity (transmission speed) is less than the sum of the input capacities. Here is an example:

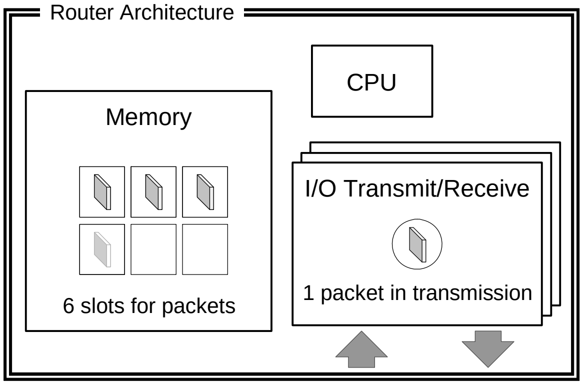
Example 2.1 Congestion Due to Mismatched Pipes with Limited Router Resources

Consider an FTP application that transmits a large file (e.g., 20 MBytes) from host A to B over the two-hop path shown in the figure. The link between the router and the receiver is called the "bottleneck" link because it is much slower than any other link on the sender-receiver path. This scenario is common: TCP sender connected by a broadband link from office or home over a high-speed LAN to a lower-speed long-distance link. (The situation would be reversed for wireless mobile devices.)



Assume that the router can always allocate the buffer size of only six packets for our session and in addition have one of our packets currently being transmitted. We assume the router architecture as in the second figure on the right. (See Section 4.1 for more details about router architectures.) Packets are only dropped when the buffer fills up. We will assume that there is no congestion or queuing on the path taken by the ACKs.

Assume `MSS` = 1KB, a large `RcvWindow` (e.g., 64 KB) and error-free transmission on all the links. The clock granularity *G* equals one RTT. Finally, to simplify the graphs, assume that all ACK arrivals occur exactly clocked at unit increments of RTT and that the associated `CongWindow` update occurs exactly at that time, too.



Draw the graphs for the values of `CongWindow` (in KBytes) over time (in RTTs) for the first 20 RTTs if the sender's TCP congestion control uses the following:

(a) TCP Tahoe: Additive increase / multiplicative decrease and slow start and fast retransmit.

(b) TCP Reno: All the mechanisms in (a) plus fast recovery.

The solutions for (a) and (b) are shown in Figure 2-13 through Figure 2-21. The discussion of the solutions is in the following text. Note that, unlike Figure 2-9, the transmission rounds are "clocked" and neatly aligned to the units of RTT. This idealization is only for the sake of illustration and the real world would look more like Figure 2-9. [Note that this idealization would stand in a scenario with propagation delays much longer than transmission delays.]

We use this example to highlight the differences between two well-known variants of TCP: TCP Tahoe and TCP Reno. Both types of senders start in the slow start state, as shown in Figure 2-11. As the sender's burst size grows, at some point it will exceed the bottleneck router's buffering capacity and one or more packets will be dropped at the router. The sender will detect loss either by three or more duplicate acknowledgments or by the retransmission timer expiration (Table 2-2). Different sender types manage segment loss differently, and we detail their approaches in the sections that follow.

We first consider how packet loss occurs at the router, in the scenario of Example 2.1, as illustrated in Figure 2-14. Recall the illustration in Figure 1-18 showing that packets are first completely received at the link layer before they are passed up the protocol stack (to IP and on to TCP). In Example 2.1, the link speeds are mismatched by a factor of 10 : 1, so the router will transmit only a single packet on the second link while the sender already transmitted ten packets on the first link. Normally, this would only delay the packets at the router, but with limited router resources, packets may also be dropped. This case occurs in Figure 2-13, where at time 4×RTT the sender transmits 14 segments, one after another. Figure 2-14 shows details of what happens as there packets keep arriving at the router. First, packet #16 arrives on an idle router and goes immediately into transmission on the second link. While packet #16 is being transmitted, ten more packets will arrive (because the link speeds are mismatched by a factor of 10 : 1). The router will buffer the first six (packets #17 through #22) and the four packets in excess of the router buffer capacity will be discarded (numbered #23 through #26). Thereafter, until the queue slowly drains, the router will have one buffer slot become available for every ten new packets that arrive. As a result, more packets will be dropped. In our case, packet #27 will take place vacated by packet #17 which moves into transmission, and packets #28 and #29 will be dropped. (More details about how routers forward packets are available in Section 4.1.)

Note the delay between the time packet loss *occurs* in the router and the time it is *detected* by the sender (Figure 2-13). The first loss occurred at time 3×RTT when the router dropped packet #15, but it was detected at time 5×RTT when the sender received three (or more) duplicate ACKs.

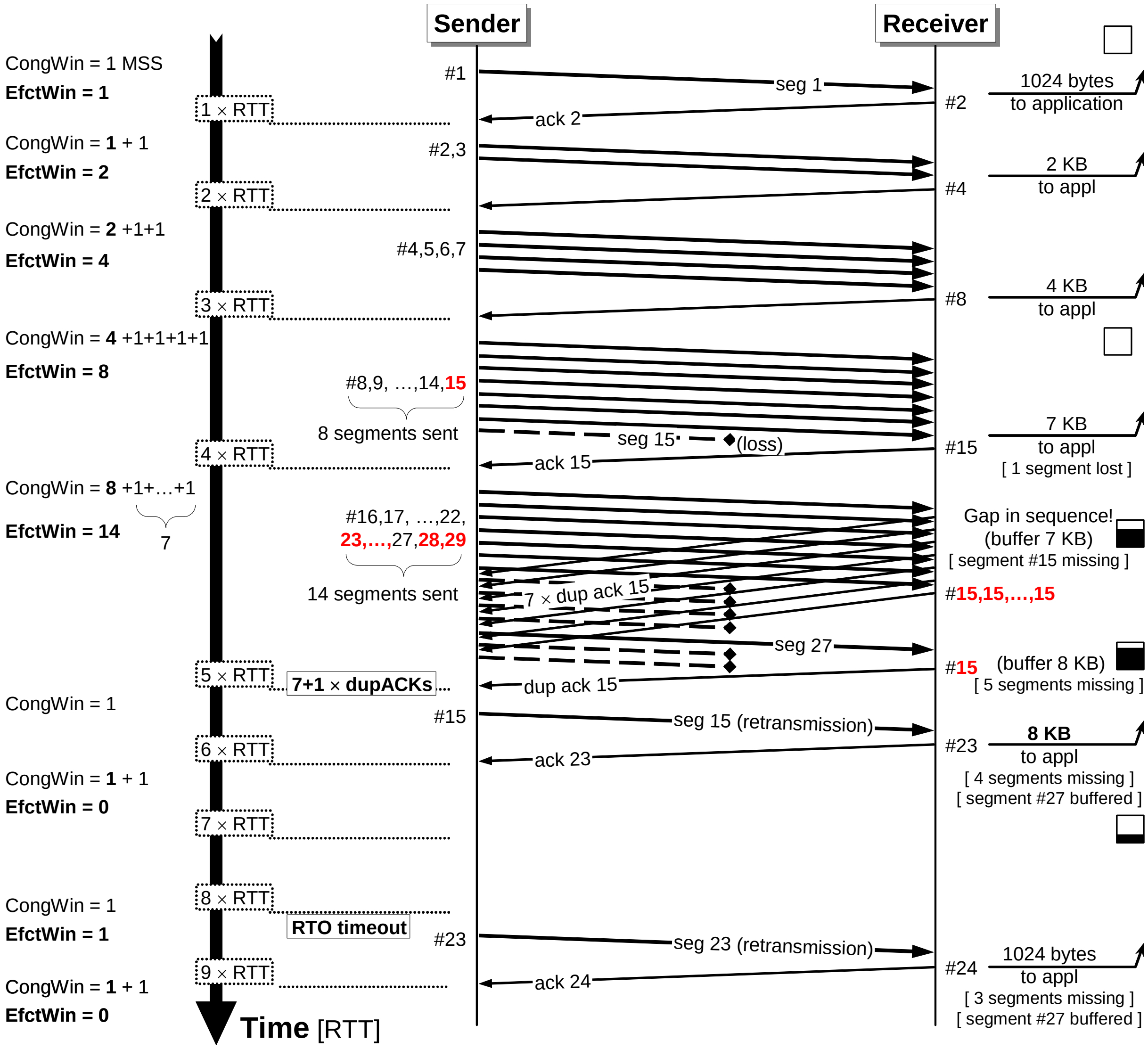


Figure 2-13: TCP Tahoe—partial timeline of segment and acknowledgment exchanges for Example 2.1. Shown on the sender's side are ordinal numbers of the sent segments and on the receiver's side are those of the ACKs (which indicate the next expected segment).

It is interesting to observe how the behavior of the retransmission timeout (RTO) timer. In our scenario of Example 2.1, the estimated RTT and the measured RTT are always the same and the variability of RTT (`DevRTT`(t)) is zero. Following Eq. (2.1), `TimeoutInterval` is always calculated as 2×RTT. Also, the backoff is still at its initial value of 1. Up to time = 4×RTT, acknowledgments arrive as expected and the RTO timer is always reset for the next burst of segments. However, at time = 4×RTT the timer is set for the 15th segment, which was sent at time = 3×RTT, and not for the 16th segment because the acknowledgment for the 15th segment is still missing. For the Tahoe sender, the RTO timer will expire first time at 9×RTT. The reader is encouraged to inspect the timer management for other segments in Figure 2-17.

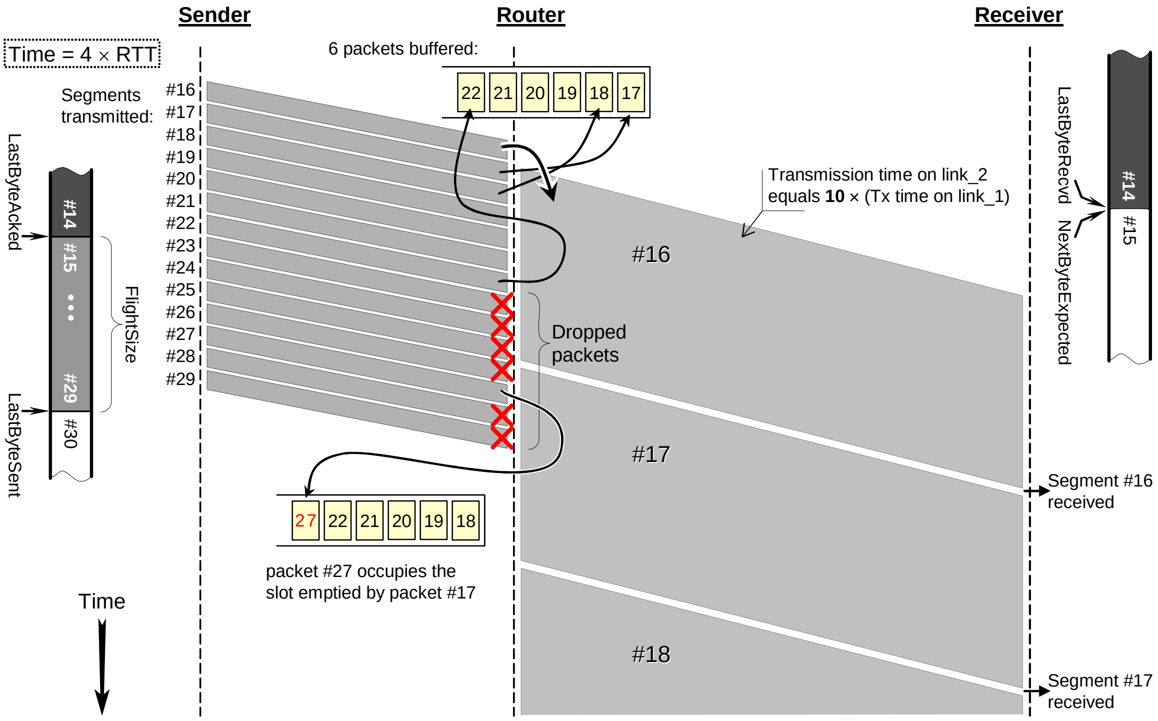


Figure 2-14: Detail from Figure 2-13 starting at time = 4×RTT. Mismatched transmission speeds of the two links result in packet loss at the bottleneck router. The sender and receiver byte streams are shown on the left and right side, respectively.

We next consider how different versions of TCP senders react once they detect a suspected segment loss.

2.2.1 TCP Tahoe

Problems related to this section: Problem 2.2 → Problem 2.7 and Problem 2.9 → Problem 2.12

The operation of the TCP Tahoe sender is shown in Figure 2-15. The sender can be in one of the two states: "slow start" or "congestion avoidance." The sender knows in which state it is by comparing the current values of the CongWin and SSThresh parameters. If CongWin < SSThresh, then the state is "slow start;" else, the state is "congestion avoidance."[18] The only difference between these states is in how the sender calculates the CongWin when it receives a "new ACK" (an acknowledgment for a segment that has not yet been acknowledged), as summarized in Figure 2-11 and detailed below.

TCP sender always begins in the slow start state, with a congestion window equal to one segment size and SSThresh = 65535 bytes. Every time the sender receives a "new ACK," it compares CongWin and SSThresh, and either remains in the current state or transitions to the other state. The sender does the following in its states (Figure 2-15):

[18] RFC-5681 states that when CongWin and SSThresh are equal, the sender may use either slow start or congestion avoidance. In this text, we will assume that it uses congestion avoidance.

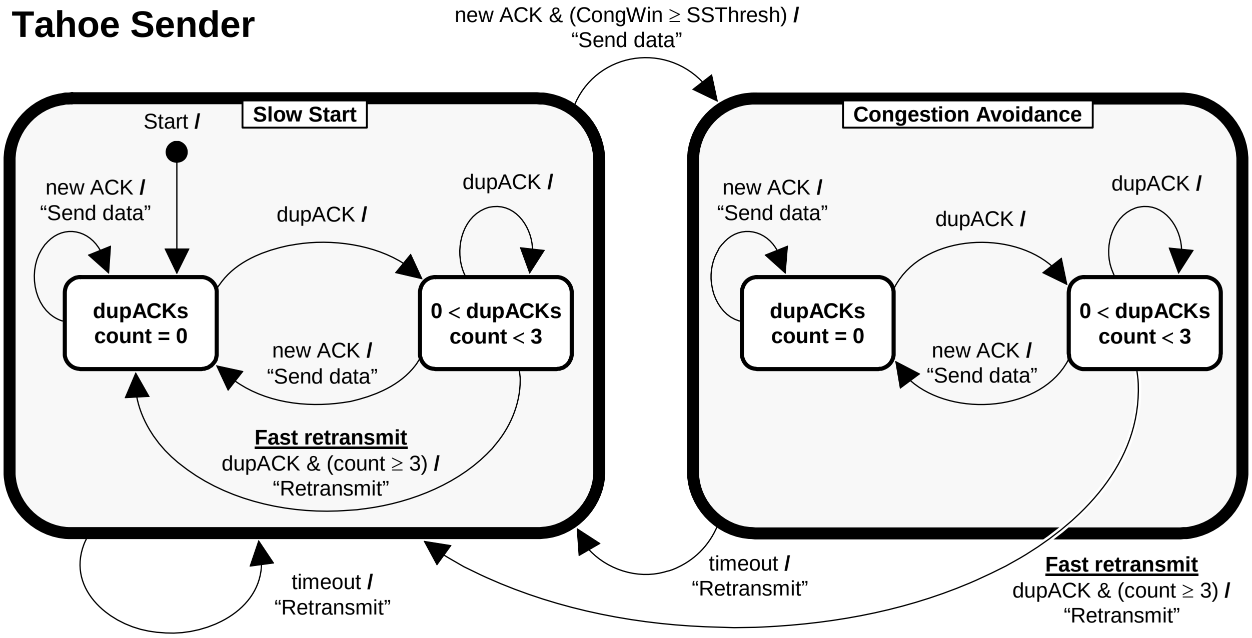


Figure 2-15: TCP Tahoe sender state diagram. "Send data" and "Retransmit" are defined the same as in Figure 2-11.

(T1) In **slow start**, the Tahoe sender does:

(T1.a) Calculate the `EffectiveWindow` using Eq. (2.3) and send as many segments as permitted. Restart the RTO timer and wait for an acknowledgment. Three types events may occur while waiting: a "new ACK" arrives, a "duplicate ACK" arrives; or, the RTO timer expires.

(T1.b) If a "new ACK" arrived, increment the congestion window size by as much as the ACK acknowledged. The new ACK may acknowledge a single segment, in which case `CongWin` is increased by one `MSS`, but it may also be a cumulative ACK acknowledging several segments. Update the `TimeoutInterval` using Eq. (2.1) and if there are remaining unacknowledged segments restart the RTO timer. If currently `CongWin` < `SSThresh`, go to back to step (T1.a); otherwise transition to the "congestion avoidance" state.

(T1.c) If a "duplicate ACK" arrived, increment the dupACKs counter by one. Of course, a duplicate ACK does not contribute to increasing the congestion window size. If the dupACKs counter becomes 3, perform **fast retransmit** to quickly retransmit the segment that is suspected lost (oldest unacknowledged segment) and set the congestion parameters as in Table 2-2. The Tahoe sender does not care about the number of dupACKs as long as it is at least three. This means that any dupACKs received after the first three are ignored. Go to back to step (T1.a).

(T1.c) If the RTO timer expired, retransmit the oldest unacknowledged segment (suspected lost) and set the parameters as in Table 2-2. Update the `TimeoutInterval` by performing exponential backoff using Eq. (2.2) and restart the RTO timer. Go to back to step (T1.a).

(T2) In **congestion avoidance**, the Tahoe sender does:

(T2.a) [same as in slow start] Send `EffectiveWindow` segments, if any, and wait for an acknowledgment. Three types events may occur while waiting:

(T2.b) If a "new ACK" arrived, increment the congestion window size by as much as the ACK acknowledged. The new ACK may acknowledge a single segment, in which case CongWin is increased by one MSS, but it may also be a cumulative ACK acknowledging several segments. Update the TimeoutInterval using Eq. (2.1) and if there are remaining unacknowledged segments restart the RTO timer. If currently CongWin < SSThresh, go to back to step (T1.a); otherwise transition to the "congestion avoidance" state.

(T2.c) [same as in slow start] If a "duplicate ACK" arrived, increment the dupACKs counter; if it becomes 3, perform **fast retransmit** and set the congestion parameters as in Table 2-2. Go to back to step (T2.a).

(T2.c) [same as in slow start] If the RTO timer expired, retransmit the lost segment and set the parameters as in Table 2-2. Update the TimeoutInterval and restart the RTO timer. Go to back to step (T2.a).

Consider again Example 2.1. As shown in both Figure 2-13 and Figure 2-16, the sender begins in slow start and grows CongWin exponentially. The congestion window size is measured in integer number of bytes, but for convenience we usually show it in multiples of segments. The first four CongWin values are: 1×MSS, 2×MSS, 4×MSS, 8×MSS, and 15×MSS. The last value is not 16×MSS as expected, because the router dropped segment #15 and the sender never received an ACK for it. At this time, using Eq. (2.3) the sender calculates EffectiveWindow = CongWindow − FlightSize = 14×MSS (assuming that RcvWindow is very large). The sender detects segment loss the first time in the fifth transmission round, i.e., at 5×RTT, by receiving eight duplicate ACKs. The congestion window size at this time equals to 15360 bytes or 15×MSS. At this time, the flight size is also 15360 bytes, because fifteen segments are unaccounted for: segment #15 from the third round and fourteen segments from the fourth round. After detecting a segment loss, the sender sharply reduces the congestion window size, which is known as **multiplicative decrease** behavior. The Tahoe sender resets CongWin to one MSS and reduces SSThresh to a half of CongWin[19] (see Table 2-2):

$$\texttt{SSThresh}(t) = \max\left\{\left\lfloor \tfrac{1}{2}\,\texttt{CongWindow}(t-1)\right\rfloor,\ 2\times\texttt{MSS}\right\} \qquad (2.4)$$

where the floor operation $\lfloor\cdot\rfloor$ indicates that SSThresh is rounded down to the integer number of MSS-size segments. Just before the moment the sender received eight dupACKs CongWin equaled 15, so the new value of SSThresh = 7×MSS is set.

Note that in TCP Tahoe, any additional dupACKs in excess of three are ignored—no new packet can be transmitted while additional dupACKs after the first three are received. TCP Reno sender differs from TCP Tahoe sender in that it starts *fast recovery* based on the additional dupACKs received after the first three (Section 2.2.2).

Upon completion of multiplicative decrease, TCP carries out **fast retransmit** to quickly retransmit the segment that is suspected lost, without waiting for the RTO timer timeout. (The dupACK counter is also reset to zero.) Note that at time = 5×RTT Figure 2-16 shows EffectiveWindow = 1×MSS. Obviously, this is not in accordance with Eq. (2.3b), because

[19] Note that this formula for setting the slow-start threshold, which appears in RFC-2001 as well as in [Stevens 1994] was used in TCP Tahoe and an early version of TCP Reno, but it was later modified in RFC-2581. (See details in Section 2.2.2.)

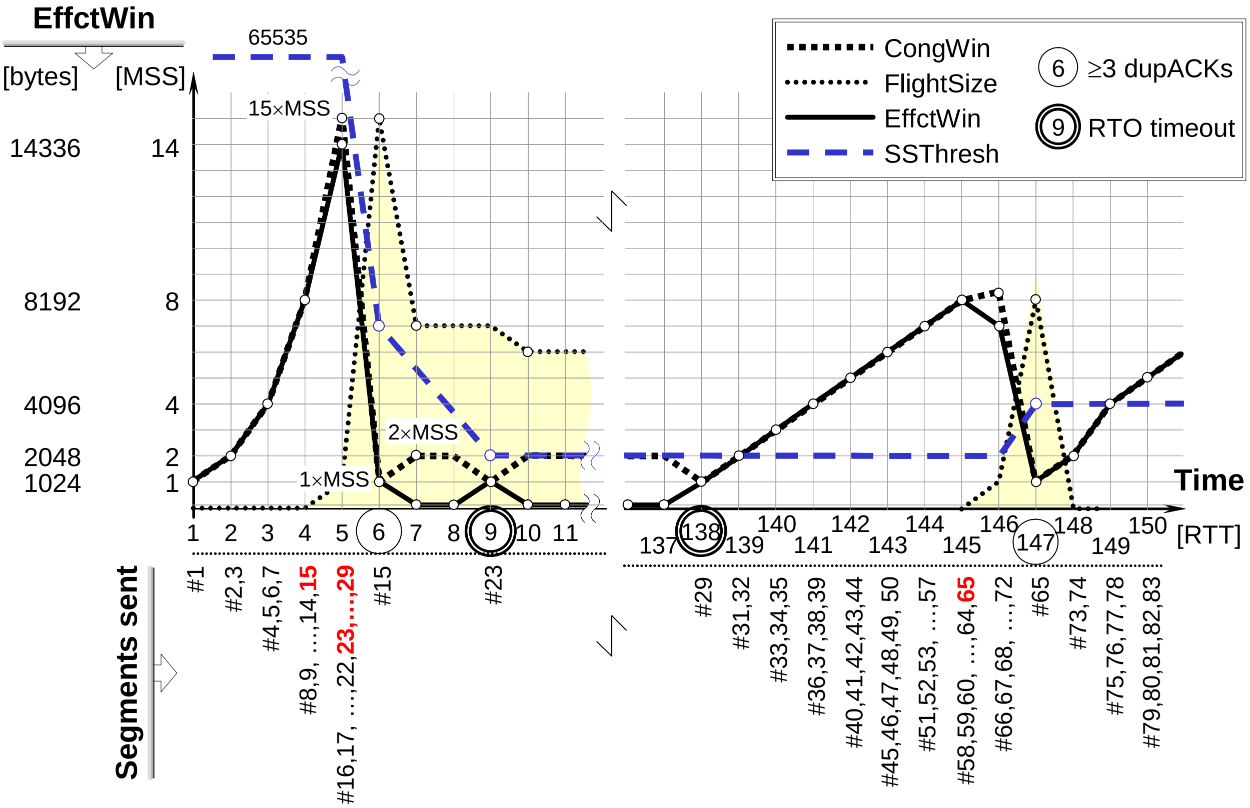


Figure 2-16: TCP Tahoe sender—the evolution of the effective and congestion window sizes for Example 2.1. Left chart: *Slow start*; Right chart: *Congestion avoidance*.
The window sizes are given on vertical axis (left) both in bytes and `MSS` units.

currently `CongWin` equals 1×`MSS` and `FlightSize` equals 15×`MSS`. This means that a sender in fast retransmit does not use Eq. (2.3b) to calculate the `EffectiveWindow` size but simply retransmits the segment suspected lost. The Tahoe sender now enters a new slow start cycle.

At time 6×RTT the sender will receive an ACK asking for the 23rd segment, thus cumulatively acknowledging all the previous segments. The sender does not immediately re-send #23 because it still has no indication of loss. The congestion window doubles to 2×`MSS` (because the sender is currently back in the *slow start* phase), but there is so much data in flight that `EffectiveWindow` = 0 and the sender is shut down. At the start of period 7×RTT, the sender restarts the RTO timer for the outstanding segments. Given that `TimeoutInterval`=2×RTT and backoff=1, the RTO timer will be scheduled to expire at the start of period 9×RTT.

During the recovery from a segment loss, the sender limits the number of segments sent in response to each ACK to two segments during slow-start. Therefore, cumulative ACKs for segments sent before the loss was detected count toward increasing `CongWin` the same as individual ACKs. (The limit during Reno-style fast recovery is one segment, Section 2.2.2.) That is why, although at time 6×RTT the ACK for #23 cumulatively acknowledged packets 15th to 22nd, `CongWin` grows only by 1×`MSS` although the sender is in slow start. Only for truly new segments (transmitted for the first time after the loss is detected), cumulative ACKs will grow the `CongWin` by the number of acknowledged segments (if the sender is still in slow start).

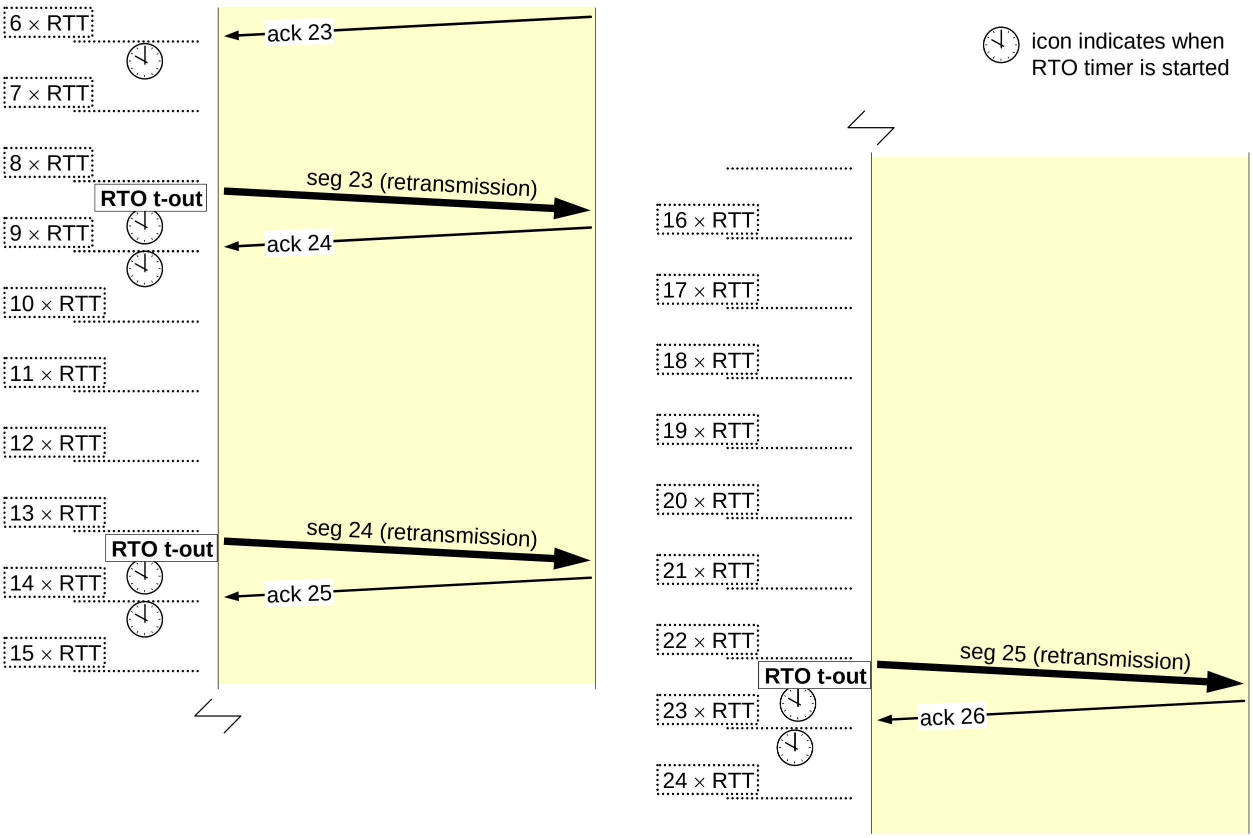


Figure 2-17: RTO timer timeout pattern for a Tahoe under the scenario of Example 2.1.

As Figure 2-13 and Figure 2-17 show, the RTO timer first expires at the start of period 9×RTT. The sender retransmits the oldest outstanding segment (#23) and performs RTO backoff. `TimeoutInterval` was previously calculated as 2×RTT and following Eq. (2.2) the RTO timer is started with 4×RTT. As the clock icon in Figure 2-17 indicates, the RTO timer is started at the start of period 9×RTT when segment #23 is retransmitted, but it is re-started at the start of period 10×RTT after ACK #24 is received. The RTO timer expires again at the start of period 14×RTT and the whole process is repeated for segment #24. The pattern of RTO backoff and segment retransmission repeats for segment #25 (retransmitted after RTO timeout at time 23×RTT,, as shown in Figure 2-17), and all the remaining lost segments, as shown in this table:

Time ACK recv'd	ACK number	RTO started	RTO value	RTO expired	Segment retransmitted
end of 6×RTT	ACK #23	start of 7×RTT	2×RTT	start of 9×RTT	#23
end of 9×RTT	ACK #24	start of 10×RTT	4×RTT	start of 14×RTT	#24
end of 14×RTT	ACK #25	start of 15×RTT	8×RTT	start of 23×RTT	#25
end of 23×RTT	ACK #26	start of 24×RTT	16×RTT	start of 40×RTT	#26
end of 40×RTT	ACK #28	start of 41×RTT	32×RTT	start of 73×RTT	#28
end of 73×RTT	ACK #29	start of 74×RTT	64×RTT	start of 138×RTT	#29

At time = 138×RTT, after the RTO expired, the sender retransmitted segment #29 and completely recovered from loss of segments at time 5×RTT. Because the congestion window now exceeds `SSThresh`, the sender enters the congestion avoidance state (Figure 2-15). The sender is in the

congestion avoidance (also known as *additive increase*) state when the current congestion window size is greater than the slow start threshold (SSThresh). RFC-5681 says that during congestion avoidance, TCP sender must not increase its congestion window by more than MSS bytes per round-trip time (RTT). There are several ways of how this can be achieved.[20] The recommended way to increase CongWin during congestion avoidance is to count the number of bytes that have been acknowledged by ACKs for new data. When the number of bytes acknowledged reaches CongWin, then CongWin can be incremented by up to MSS bytes. Another common method is to use the formula for every ACK for new data:

$$\text{CongWin}(t) = \text{CongWin}(t-1) + \text{MSS} \times \frac{\text{MSS}}{\text{CongWin}(t-1)} \quad \text{[bytes]} \tag{2.5}$$

where CongWin(t–1) is the congestion window size before receiving the current ACK. Note that the resulting CongWin is *not* rounded down to the next integer value of MSS as in other equations (but CongWin is always measured in integer number of bytes). Note also that for a connection where the receiver sends cumulative ACKS, Eq. (2.5) will lead to increasing CongWin by less than one full-sized segment per RTT. What matters is that during congestion avoidance, the congestion window does not increase by more than one segment per round-trip time (regardless of how many ACKs are received in that RTT). This results in a *linear increase* of the congestion window.

In the scenario of Example 2.1, the sender enters congestion avoidance at time 138×RTT (right chart in Figure 2-16). At time 145×RTT, the sender sends eight segments and the last (#65) is dropped in the router. At time 146×RTT, the sender receives three (or more) duplicate ACKs and performs fast retransmit of segment #65. Note that SSThresh is calculated using Eq. (2.4) as 4×MSS and the sender will be in the slow start state until this threshold is exceeded at time 149×RTT, at which time the sender enters congestion avoidance. The pattern from 142×RTT to 149×RTT is repeated until the sender has data to send. This behavior is known as the *saw-tooth pattern* of TCP traffic.

Figure 2-18 summarizes the key congestion avoidance and control mechanisms for TCP Tahoe. Note that the second slow-start phase, starting at 5×RTT, is immediately aborted due to the excessive amount of unacknowledged data. Thereafter, the TCP sender enters a prolonged phase of dampened activity until all the lost segments are retransmitted through “fast retransmits.”

It is interesting to note that TCP Tahoe in this example needs 150×RTT rounds to successfully transfer 83 segments. On the other hand, should the bottleneck bandwidth been known and constant, Go-back-7 ARQ would need just 12×RTT rounds to transfer 83 segments (assuming error-free transmission)! In this example, bottleneck resource uncertainty introduces delay greater than twelve times the minimum possible one. TCP Reno, although an improvement, does not perform much better under massive segment loss (Section 2.2.2). However, TCP NewReno manages much better the network resource uncertainty (Section 2.2.3).

[20] The formula remains the same for cumulative acknowledgments that acknowledge more than a single segment, which will lead to increasing CongWindow by less than one full-sized segment per RTT. The reader should check further discussion in RFC-5681 and [Stevens 1994]. Also, when CongWindow and SSThresh are equal, the sender may use either slow start or congestion avoidance (see Table 2-1).

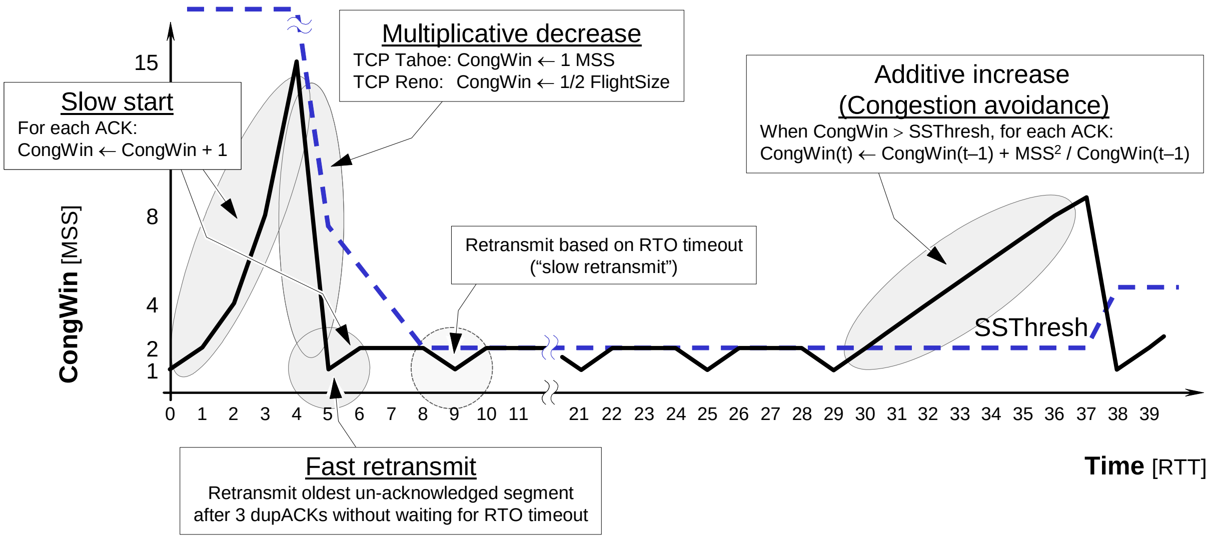


Figure 2-18: TCP Tahoe sender—highlighted are the key mechanisms for congestion avoidance and control; compare to Figure 2-16.

2.2.2 TCP Reno

Problems related to this section: Problem 2.8 → Problem 2.13

The observation that led to TCP Reno is that TCP connections often lost a *single segment*, but after this loss was discovered and Fast Retransmit performed, the communication path ("pipe") would become empty and Slow-Start would labor to re-fill the pipe. The TCP Reno retained the basic features of TCP Tahoe, but the key difference is that the Fast Retransmit is modified to include *Fast Recovery*. The key difference between TCP Tahoe and Reno senders is in their reaction to three duplicate ACKs (Table 2-2). The Tahoe sender treats triple duplicate ACKs the same as a timeout—it sets the slow-start threshold to half the current congestion window, reduces congestion window to 1×MSS, and resets to *slow start* state. Unlike this, the Reno sender upon three duplicate ACKs enters *fast recovery*. After the fast retransmit algorithm sends what appears to be the missing segment, the **fast recovery** algorithm governs the data transmission until a new acknowledgment arrives acknowledging any data that were not acknowledged before the three dupACKs (known as a "recovery ACK"). If there is no acknowledgment, TCP Reno experiences a timeout and enters the slow-start state. If a recovery ACK arrives, the sender transitions to the Congestion Avoidance state.

The Reno sender's state diagram is shown in Figure 2-19. During fast recovery, CongWindow is incremented by one MSS for each additional duplicate ACK received, including the first three dupACKs [Stevens 1994; Stevens 1997; Allman *et al.* 1999]. This procedure artificially inflates the congestion window in order to reflect that an additional segment has left the network. The fast recovery ends when either a retransmission timeout occurs or an ACK arrives that acknowledges any data not acknowledged before the fast recovery procedure began. It is important that Reno considers *any* new ACK as a recovery ACK. This is the key difference between Reno and NewReno (Section 2.2.3). After fast recovery is finished, the sender enters congestion avoidance.

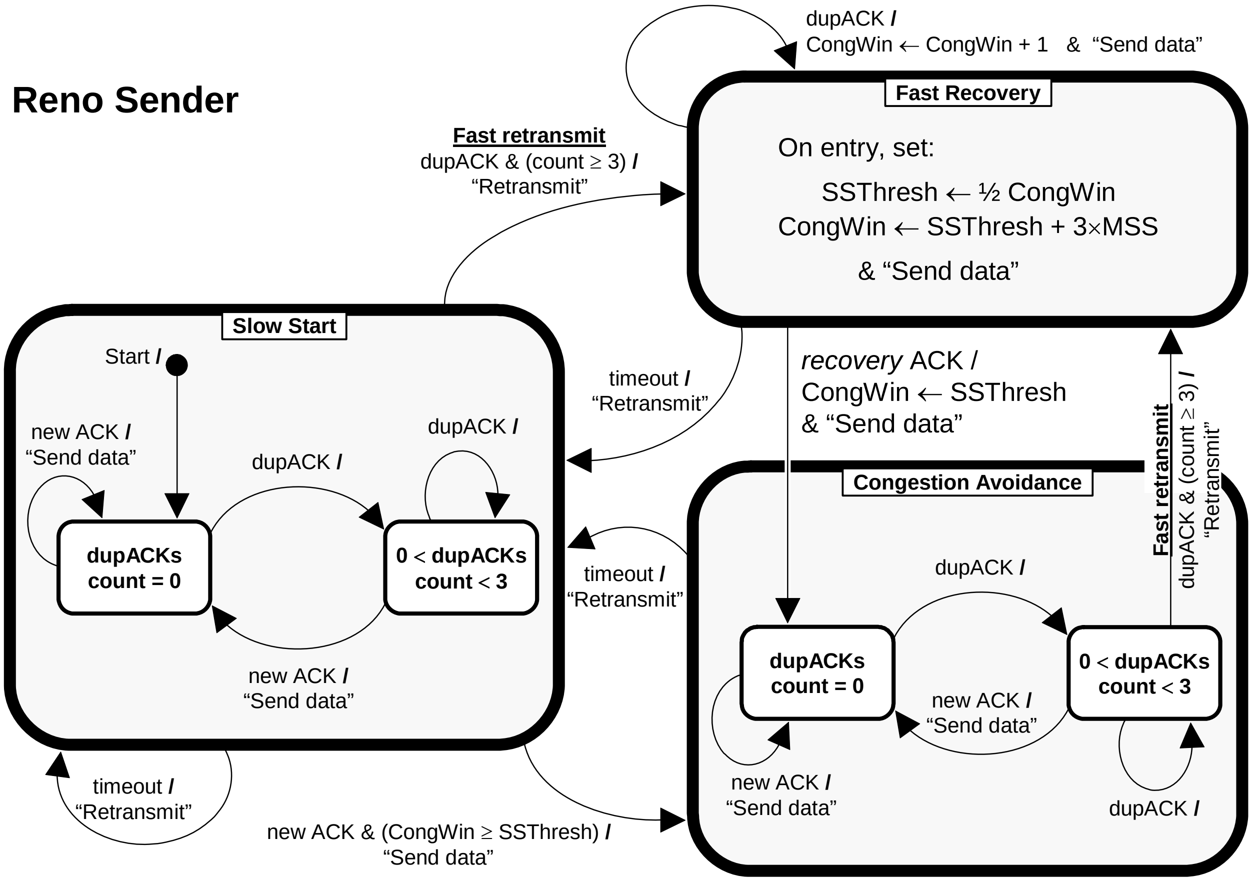


Figure 2-19: TCP Reno sender state diagram. Compare to Figure 2-11 and Figure 2-15.

Note that in the fast recovery state the Reno sender does not count the number of received dupACKs, and does not react specially to three consecutive dupACKs.

The reason for performing fast recovery rather than slow start is that the receipt of the dupACKs not only indicates that a segment has been lost, but also that segments are most likely leaving the network. (It is possible that a massive segment duplication by the network can invalidate this conclusion.) Because three dupACKs are received by the sender, this means that three segments have left the network and arrived successfully, but out-of-order, at the receiver. In other words, as the receiver can only generate a duplicate ACK when an error-free segment has arrived, that segment has left the network and is in the receive buffer, so we know it is no longer consuming network resources. Furthermore, because the ACK “clock” [Jacobson, 1988] is preserved, the TCP sender can continue to transmit new segments (although transmission must continue using a reduced `CongWindow`).

In addition to the fast recovery procedure, there are differences in how TCP Tahoe and Reno calculate different parameters (Table 2-2). When a Reno sender detects segment loss using the retransmission timer, the value of `SSThresh` *must* be set to no more than the value given as:

$$\texttt{SSThresh}(t) = \max\ \{½\ \texttt{FlightSize}(t–1),\ 2×\texttt{MSS}\} \tag{2.6}$$

where `FlightSize`(t–1) is the amount of outstanding data in the network (for which the sender has not yet received an acknowledgment) at the time of receiving the third dupACK. Compare this equation to (2.4), for computing `SSThresh` for TCP Tahoe. Note that some networking books and even TCP implementations confuse the Tahoe and Reno slow start threshold `SSThresh` calculation, which according to RFC-2581 is *incorrect*.[21] (Also see Table 2-2.)

TCP Reno sender retransmits the lost segment and sets congestion window to:

$$\texttt{CongWindow}(t) = \texttt{SSThresh}(t) + 3\times\texttt{MSS} \tag{2.7}$$

This artificially "inflates" the congestion window by the number of segments (three) that have left the network and which the receiver has buffered. In addition, for each additional dupACK received *after the third dupACK*, increment `CongWindow` by `MSS`. This artificially inflates the congestion window in order to reflect the additional segment that has left the network (the TCP receiver has buffered it, waiting for the missing gap in the data to arrive).

Reno significantly improves upon the behavior of Tahoe TCP when a single segment is dropped from a window of data. However, even Reno can suffer from performance problems in case of a "catastrophic loss" when multiple segments are dropped from a window of data. This is illustrated in the simulations for our default configuration, when several segments are dropped for a Reno TCP connection with a large congestion window after slow-starting in a network with drop-tail routers (or other routers that fail to monitor the average queue size). As a result, the sender needs to await a retransmission timer expiration before reinitiating data flow.

An example is illustrated in Figure 2-20, derived from Example 2.1. In Figure 2-20 at 5×RTT when the sender receives 3+5 dupACKs, `CongWindow` becomes equal to $\frac{15}{2}+3+1+1+1+1+1$ = 15.5 × `MSS`. The last five 1's are due to 7+1−3 = 5 dupACKs received after the initial 3 ones. At 6×RTT the receiver requests the 23rd segment (thus cumulatively acknowledging up to the 22nd). `CongWindow` grows slightly to 17.75, but because there are 14 segments outstanding (#23 → #37), the effective window is shut up. The sender arrives at standstill and thereafter behaves similar to the TCP Tahoe sender (Figure 2-16).

Note that, although at time = 10×RTT three dupACKs indicate that three segments that have left the network, these are only 1-byte segments, so it may be inappropriate to add 3×`MSS` as Eq. (2.6) postulates. RFC-2581 does not mention this possibility, so we continue applying Eq. (2.6) and because of this `CongWindow` converges to 6×`MSS` from above.

When a *full acknowledgment* arrives, it acknowledges all the intermediate segments sent after the original transmission of the lost segment until the loss is discovered (the sender received the third duplicate ACK). This does not mean that there are no more outstanding data (i.e., `FlightSize` = 0), because the sender might have sent some new segments after it discovered the loss (if its `EffectiveWin` permitted transmission of news segments). At this point, the sender sets its congestion window as:

$$\texttt{CongWindow} = \texttt{SSThresh} \tag{2.8}$$

[21] The formula `SSThresh` = ½ `CongWindow` is an older version for setting the slow-start threshold, which appears in RFC-2001 as well as in [Stevens 1994]. It was used in TCP Tahoe and an early version TCP Reno, but it was replaced in a later version of Reno with Eq. (2.4) in RFC-2581.

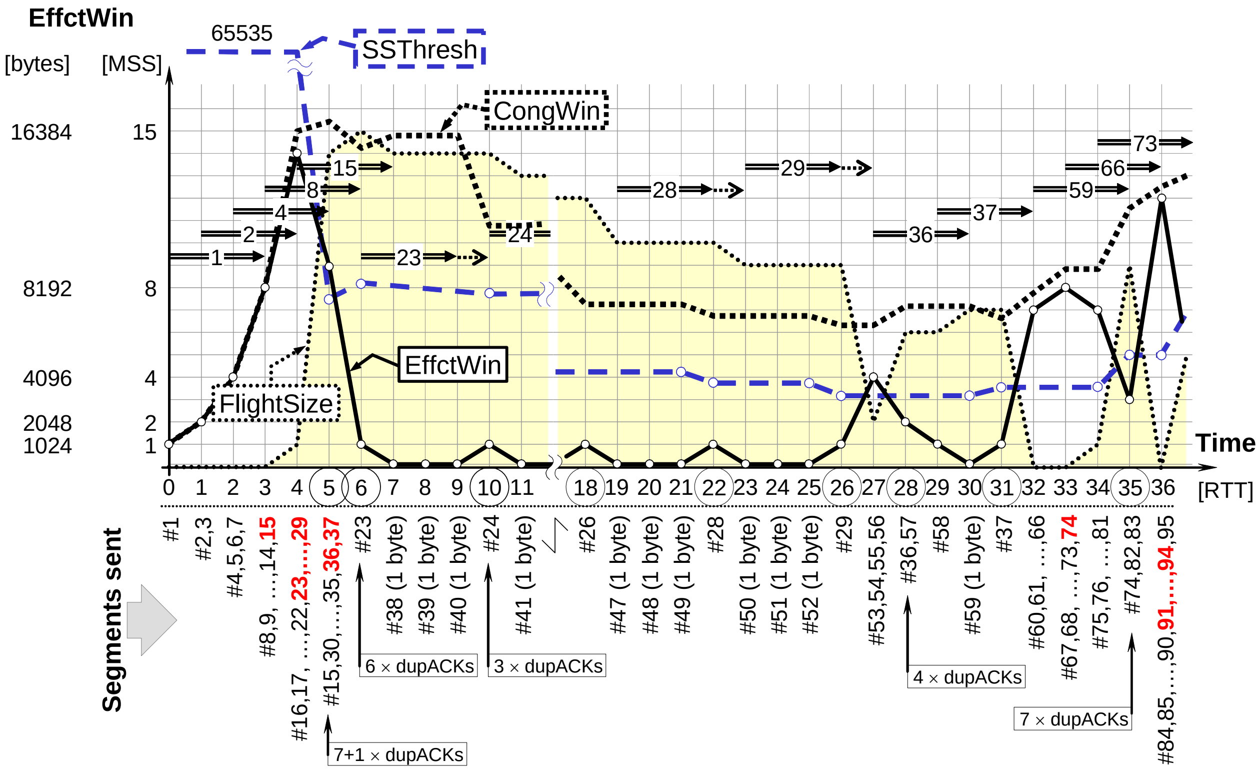


Figure 2-20: TCP Reno sender—the evolution of the effective and congestion window sizes for Example 2.1. The sizes are given on vertical axis (left) both in bytes and MSS units.

Recall that in TCP Reno SSThresh is computed using Eq. (2.6), where FlightSize is the amount of data outstanding when fast recovery was entered, *not* the current amount of data outstanding[22]. This reduction of the congestion window size is termed *deflating the window*. At this point, the TCP sender exits fast recovery and enters *congestion avoidance*.

Figure 2-21 shows partial timeline at the time when the sender starts recovering. After receiving the 29th segment, the receiver delivers it to the application along with the buffered segments #30 → #35 (a total of seven segments). At time = 27×RTT, a cumulative ACK arrives requesting the 36th segment (because segments #36 and #37 are lost at 5×RTT). Because CongWindow > 6×MSS and FlightSize = 2×MSS, the sender sends four new segments and each of the four makes the sender to send a dupACK. At 28×RTT, CongWindow becomes equal to $\frac{\dot{6}}{2}+3+1=\dot{7}$ × MSS and FlightSize = 6×MSS (we are assuming that the size of the unacknowledged 1-byte segments can be neglected).

[22] RFC-3782 suggests an alternative option to set CongWindow = min{SSThresh, FlightSize + MSS}, where FlightSize is the *current* amount of data outstanding. Check RFC-3782 for details.

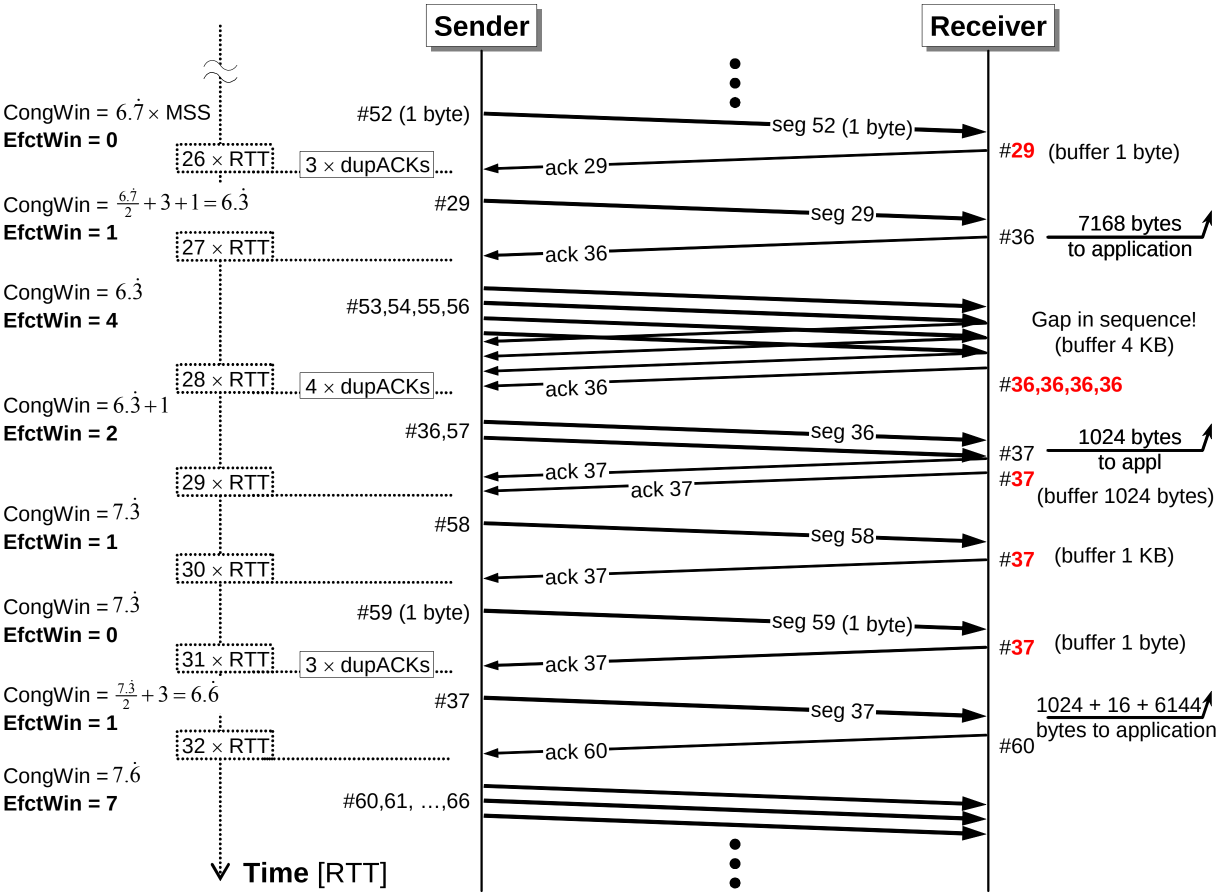


Figure 2-21: TCP Reno—partial timeline of segment and ACK exchanges for Example 2.1. (The slow start phase is the same as for Tahoe sender, Figure 2-13.)

Regarding the delay, TCP Reno in this example needs 37×RTT to successfully transfer 74 segments (not counting 16 one-byte segments to keep the connection alive, which makes a total of 90 segments—segment #91 and the consecutive ones are lost). This is somewhat better that TCP Tahoe and TCP Reno should better stabilize for a large number of segments.

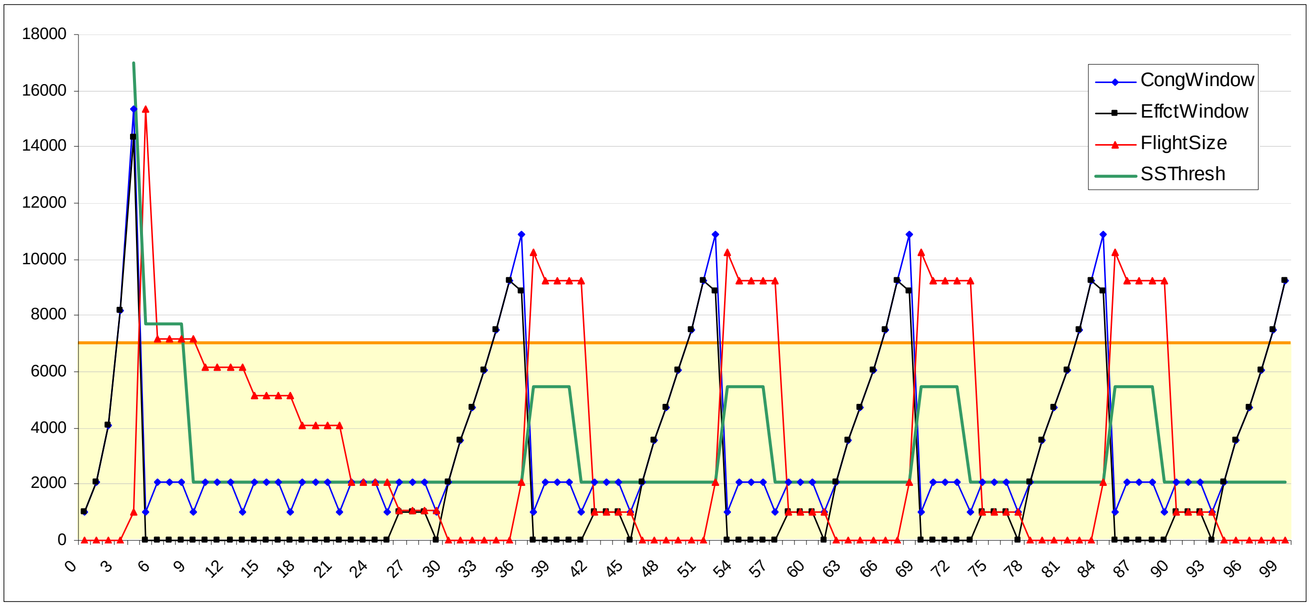


Figure 2-22: TCP Tahoe congestion parameters for Example 2.1 over the first 100 transmission rounds. The overall sender utilization comes to only 25 %. The lightly shaded background area shows the bottleneck router's capacity, which, of course, is constant.

2.2.3 TCP NewReno

Problems related to this section: Problem 2.15 → ??

The so-called **NewReno** version of TCP introduces a further improvement on fast recovery, which handles a case where two or more segments are lost within a single window. Same as the ordinary TCP Reno, the NewReno begins the **fast recovery** procedure when three duplicate ACKs are received, and ends it when either a retransmission timeout occurs or an ACK arrives that acknowledges all of the data up to and including the data that was outstanding when the fast recovery procedure began. After the presumably lost segment is retransmitted by fast retransmit, if the corresponding ACK arrives, there are two possibilities:

(4) The ACK specifies the sequence number at the end of the current window, in which case the retransmitted segment was the only segment lost from the current window. We call this acknowledgment a **full acknowledgment**.

(5) The ACK specifies the sequence number higher than the lost segment, but lower than the end of the window, in which case (at least) one more segment from the window has also been lost. We call this acknowledgment a **partial acknowledgment**.[23]

[23] Note that "partial ACK" or "full ACK" look just like any other ACKs—they are TCP segments that have the ACK flag set in their header (Figure 2-3). An acknowledgment is seen as "partial" by the sender because the sender would consider as "full" only an ACK that acknowledged up to what was the value of LastByteSent (Figure 2-5) at the time when three duplicate ACKs were received.

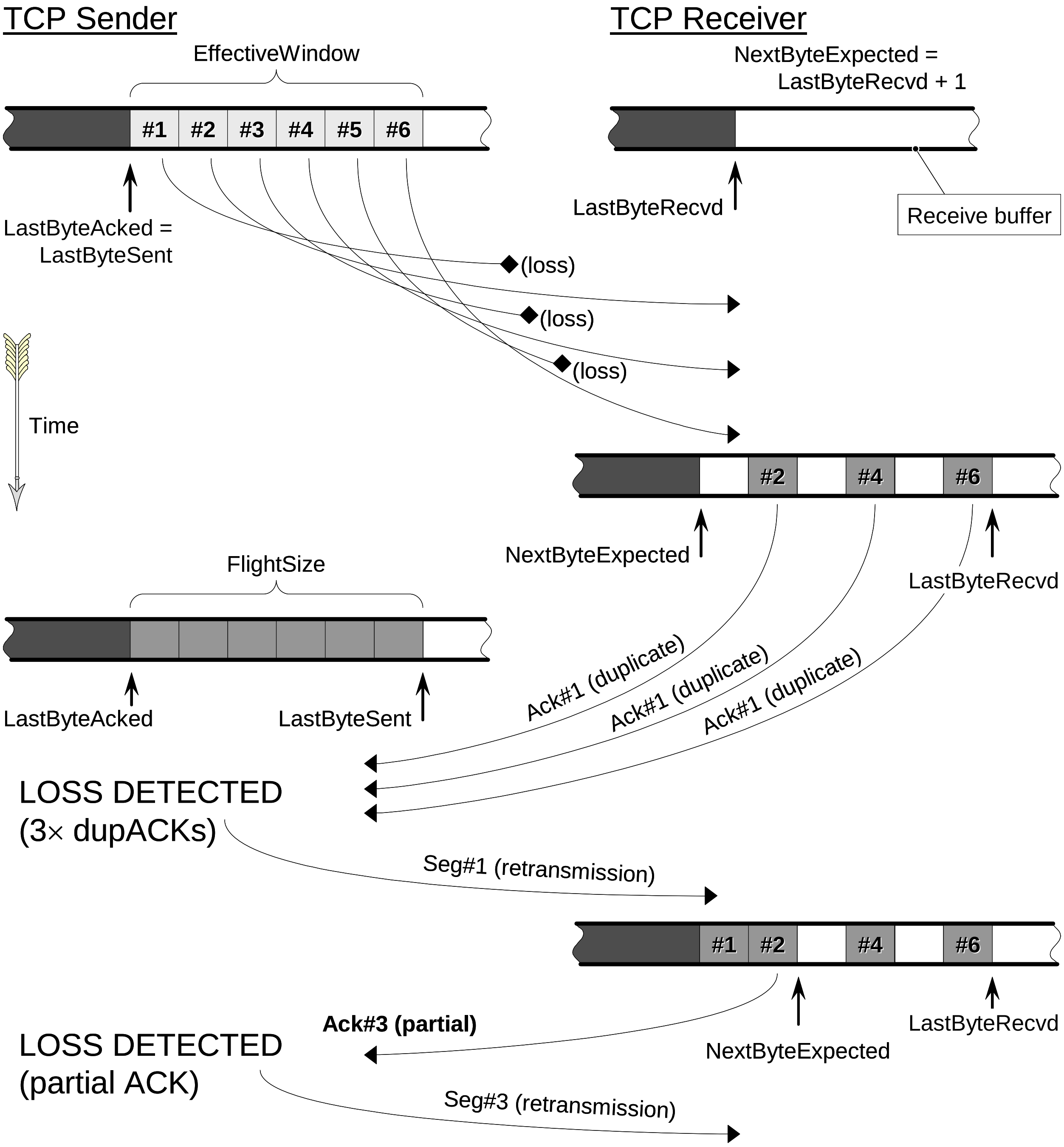


Figure 2-23: TCP NewReno partial acknowledgments.

As with the ordinary Reno, for each additional dupACK received while in fast recovery, NewReno increments `CongWindow` by `MSS` to reflect the additional segment that has left the network. Recall that in the fast recovery state the Reno sender does not count the number of received dupACKs, and does not react specially to three consecutive dupACKs.

The concept of partial acknowledgments is illustrated in Figure 2-23. In this scenario, the sender sends six segments, of which three are lost: segments #1, #3, and #5. The receiver buffers the three segments that arrive out of order and send three duplicate acknowledgments. Upon receiving the three dupACKs, the sender retransmits the oldest outstanding segment (#1) and waits. The receiver fills only the first gap and now sends acknowledgment asking for segment #3. This is a *partial acknowledgment*, because it does not acknowledge *all* segments that were outstanding at the time the loss was detected.

The key idea of TCP NewReno is, if the TCP sender receives a *partial acknowledgment* during fast recovery, the sender should respond to the partial acknowledgment by inferring that the next in-sequence packet has been lost, and retransmitting that packet. In other words, NewReno proceeds to retransmit the second missing segment, without waiting for three dupACKs or RTO timer expiration. This means that TCP NewReno adds "partial acknowledgment" to the list of events in Table 2-2 by which the sender detects segment loss. The sender also deflates its congestion window by the amount of new data acknowledged by the cumulative acknowledgment, that is:

$$\text{NewlyAcked} = \texttt{LastByteAcked}(t) - \texttt{LastByteAcked}(t-1)$$

$$\texttt{CongWindow}(t)' = \texttt{CongWindow}(t-1) - \text{NewlyAcked} \tag{2.9a}$$

If (NewlyAcked ≥ MSS), then add back MSS bytes to the congestion window:

$$\texttt{CongWindow}(t) = \texttt{CongWindow}(t)' + \texttt{MSS} \tag{2.9b}$$

As with duplicate acknowledgment, this artificially inflates the congestion window to reflect the additional segment that has left the network. This "partial window deflation" attempts to ensure that, when fast recovery eventually ends, approximately SSThresh amount of data will be outstanding in the network. Finally, the sender sends a new segment if permitted by the new value of EffectiveWin.

When a *full acknowledgment* arrives, it acknowledges all the intermediate segments sent after the original transmission of the lost segment until the loss is discovered (the sender received the third duplicate ACK). This does not mean that there are no more outstanding data (i.e., FlightSize = 0), because the sender might have sent some new segments after it discovered the loss (if its EffectiveWin permitted transmission of news segments). At this point, the sender sets its congestion window as:

$$\texttt{CongWindow} = \texttt{SSThresh} \tag{2.10}$$

Recall that in TCP Reno SSThresh is computed using Eq. (2.6), where FlightSize is the amount of data outstanding when fast recovery was entered, *not* the current amount of data outstanding[24]. This reduction of the congestion window size is termed *deflating the window*. At this point, the TCP sender exits fast recovery and enters *congestion avoidance*.

An example of NewReno behavior is given below. Example 2.2 works over a similar network configuration as the one in Example 2.1. Again, we have a high-speed link from the TCP sender to the bottleneck router and a low-speed link from the router to the TCP receiver. However, there are some differences in the assumptions. In Example 2.1, we assumed a very large RTT, so that all packet transmission times are negligible compared to the RTT. We also assumed that cumulative ACKs acknowledged all the segments sent in individual RTT-rounds (or, bursts).

[24] RFC-3782 suggests an alternative option to set CongWindow = min{SSThresh, FlightSize + MSS}, where FlightSize is the *current* amount of data outstanding. Check RFC-3782 for details.

Conversely, in Example 2.2, we will assume that the RTT is on the same order of magnitude as the transmission time on the *second* link. In addition, each segment is acknowledged individually. This results in a somewhat more complex, but also more accurate, analysis of the TCP behavior.

Example 2.2 Analysis of the Slow-Start Phase in TCP NewReno

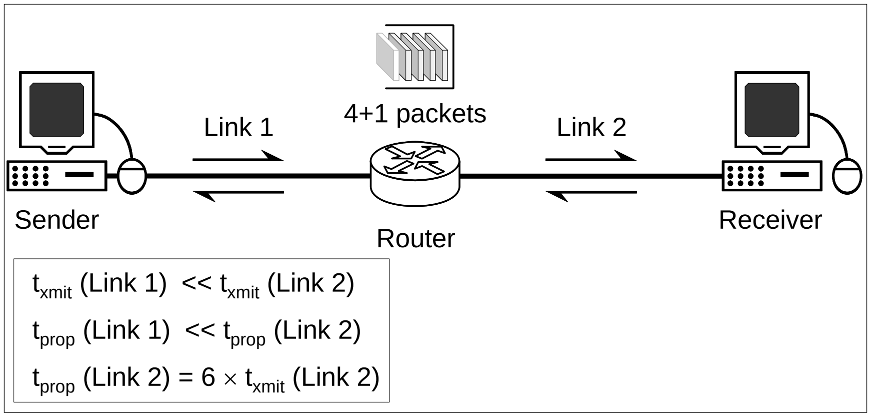
Consider an application that is engaged in a lengthy file transfer using the TCP NewReno protocol over the network shown in the figure.

The following assumptions are made:

A1. Full duplex links connect the router to each endpoint host so that simultaneous transmissions are possible in both directions on each link. The transmission rate of Link-1 is much greater than that of Link-2. One-way propagation delay on Link-1 is also negligible compared to the propagation delay of Link-2. Assume that all packet transmissions are error free.

A2. The propagation delay on Link-2 (from the router to the receiver equals six times the transmission delay for data packets on the same link. Also assume that the ACK packet size is negligible, i.e., their transmission delay is approximately zero.

A3. The router buffer can hold up to four packets plus one packet currently in transmission. The packets that arrive to a full buffer are dropped. However, this does *not* apply to ACK packets, i.e., ACKs do not experience congestion or loss.


4+1 packets
Link 1
Link 2
Sender
Router
Receiver
t_{xmit} (Link 1) << t_{xmit} (Link 2)
t_{prop} (Link 1) << t_{prop} (Link 2)
t_{prop} (Link 2) = 6 × t_{xmit} (Link 2)

A4. The receiver does not use delayed ACKs, i.e., it sends an ACK immediately after receiving a data segment.

A5. The receiver has set aside a large receive buffer for the received segments, so this will never be a limiting factor for the sender's window size.

Considering only the slow-start phase (until the first segment loss is detected), we would like to know the following:

(a) The evolution of the parameters such as congestion window size, router buffer occupancy, and how well the communication pipe is filled with packets.

(b) The ordinal packet numbers of all the packets that will be dropped at the router (due to the lack of router memory space).

(c) The maximum congestion widow size that will be achieved after the first packet loss is detected (but before the time *t* = 60).

(d) How many packets will be sent after the first packet loss is detected until the time *t* = 60? Explain the reason for transmission of each of these packets.

The solution is shown in Figure 2-25 and discussed in the following text.

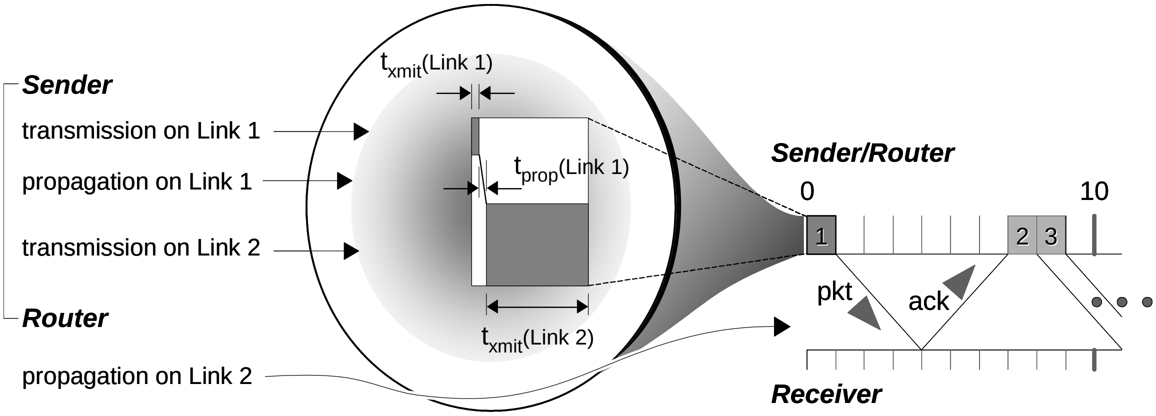


Figure 2-24: Timing diagram for segment transmissions in Example 2.2. Because the delays on the first link are negligible (shown in the zoomed-in detail), the sender and router are considered together, and the charts in Figure 2-25 show only the router (i.e., Link 2) delays.

Given very different delay characteristics of Link 1 and Link 2, for the sake of keeping diagrams simpler, the sender and router are considered together, as explained in Figure 2-24 which shows a detail from Figure 2-25. Figure 2-25 shows the evolution of four parameters over the first 20 time units of the slow-start phase. The four parameters are: (*i*) congestion window size; (*ii*) slow start threshold; (*iii*) current number of packets in the router, both in transmission or waiting for transmission; and (*iv*) current number of packets in flight on Link-2, that is the packets that neither are in the router nor acknowledged. Note that the last parameter is *not* the same as the `FlightSize` defined at the beginning of Section 2.2. `FlightSize` is maintained by the sender to know how many segments are outstanding. Unlike this, the current number of packets in flight on Link-2 (bottom chart in Figure 2-25) represents only the packets that were transmitted by the router, but for which the ACK has not yet arrived at the TCP sender.

The gray boxes on the top line symbolize packet transmissions on the Link-2. This is because the transmission delay on Link-1 is negligible, so any packet sent by the TCP sender immediately ends up on the router.

Recall that during the slow start, the sender increments its congestion window size by one MSS for each successfully received acknowledgment. We can see on the top of Figure 2-25 how the acknowledgment for packet #1 arrives at t = 7 and the congestion window size rises to 2. The sender sends two more segments (#2 and #3) and they immediately end up on the router. Note how packets that are sent back-to-back (in bursts) are separated by the packet transmission time on Link-2. When the ACK arrives for #2, the sender is in slow start, so `FlightSize` = 2 – 1 = 1 and `CongWin` = 2 + 1 = 3. According to equation (2.3),

$$\texttt{EffectiveWin} = \texttt{CongWin} - \texttt{FlightSize} = 3 - 1 = 2$$

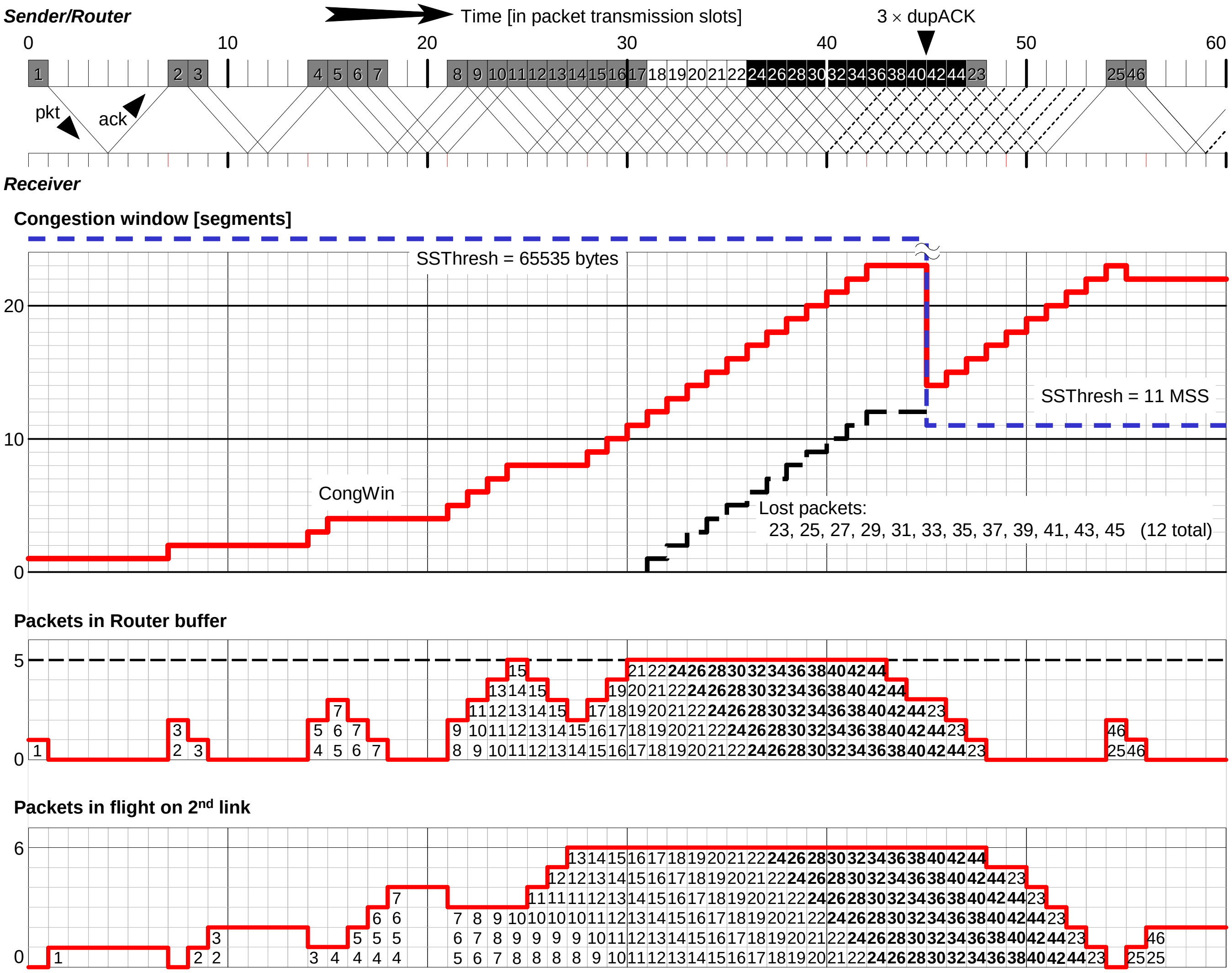


Figure 2-25: Evolution of the key parameters for the TCP NewReno sender in Example 2.2. (*Continued in* Figure 2-26.)

In other words, during slow start, every time the sender receives a non-duplicate acknowledgment for one segment, the sender can send two new segments. After sending segments #4 and #5, we have: `CongWin` = 3, `FlightSize` = 3, and `EffectiveWin` = 0. When the ACK for segment #3 arrives (delayed by t_x(Link-2) after the first ACK, because the segment #3 traveled back-to-back behind #2) we have: `FlightSize` = 3 – 1 = 2 and `CongWin` = 3 + 1 = 4. Therefore,

$$\texttt{EffectiveWin} = \texttt{CongWin} - \texttt{FlightSize} = 4 - 2 = 2$$

and the sender will send two new segments (#6 and #7). Now, we have `CongWin` = 4, `FlightSize` = 4, and `EffectiveWin` = 0. The sender is waiting for an acknowledgment for segment #4.

Note that when an in the chart for the current number of packets in the router shows below the curve the ordinal numbers of the packets. The bottommost packet is the one that is in transmission during the current time slot. The packets above it are currently waiting for transmission. For example, at time t = 7, packet #2 is currently in transmission and packet #3 is waiting in the router memory. At $t = 8$, packet #2 is traversing Link-2 (shown under the bottom curve) and packet #3 is in transmission on the router. The attentive reader might observe that the packet numbers in the bottommost row of the router buffer curve are identical to the ones at the top of Figure 2-25.

(a)

Because we are considering a single connection in slow start, packet arrivals on the router occur in bursts of exactly two packets. This is because for every received acknowledgment, `FlightSize` is reduced by 1 and `CongWin` is incremented by 1, which means that effectively the sender can send two new packets. The sender sends two new packets and they immediately end up on the router. Therefore, when a buffer overflow occurs, exactly one packet is dropped. This is because at the beginning of the preceding time slot, the buffer would have been full and the router would transmit one packet, therefore freeing up space for one new packet. When two packets arrive, the first is stored and the second is dropped.

The first loss happens at time $t = 31$. In the previous time slot (t =30) the router had five packets (#17, #18, #19, #20, and #21), of which packet #17 was transmitted by the router. At $t = 31$, the acknowledgment for packet #11 will arrive and it will increment congestion window by one, to the new value of 12×MSS. The sender sends packets #22 and #23 and they immediately arrive to the router. The router just transmitted packet #17 and has space for only one new packet. Packet #22 joins the tail of the waiting line and packet #23 is dropped.

The top row of Figure 2-25 shows white boxes for the transmission periods of the five packets that the router will transmit after the loss of packet #23. Black boxes symbolize packets that are sent out of order, for which the preceding packet was dropped at the router (due to the lack of router memory space). There will be a total of 12 packets fro which the preceding packet was lost, starting with packet #24 and ending with packet #44.

The TCP sender receives three duplicate acknowledgments (asking for packet #23) at time $t = 45$. Note that packet #23 was lost at the router at time $t = 31$, but the sender learned about the loss only at time $t = 45$ (by receiving three dupACKs)! When the sender discovers the loss, it sets the congestion window size to one half of the number of segments in flight, which is 23, plus 3 for three duplicate acknowledgments—recall equation (2.6) from Section 2.2.2. The slow-start threshold is set to one-half of the number of segments in flight so `SSThresh` becomes equal to 11. In addition, because this is a TCP Reno sender, the congestion window is incremented by one for each new duplicate acknowledgment that is received after the first three.

Upon detecting loss at time $t = 45$, the TCP sender immediately retransmits segment #23, but the packet joins the queue at the router behind packets #42 and #44, which arrived before #23. The router is not aware that these are TCP packets, lest that some of them are retransmitted, so it does not give preferential treatment to retransmitted packets. As seen in the top row of Figure 2-25, the router will transmit packet #23 over Link-2 during the time slot $t = 47$. Therefore, generally it takes longer than one RTT for the sender to receive the acknowledgment for a retransmitted segment.

The acknowledgment for the retransmitted segment #23 arrives at time $t = 54$ and it asks for segment #25 (because #24 was received correctly earlier). This is only a *partial acknowledgment* because a full acknowledgment would acknowledge segment #45. Therefore, the TCP NewReno sender immediately sends packet #25 without waiting for three duplicate acknowledgments. In addition, the sender adjusts its congestion window size according to Eq. (2.9):

$$\text{NewlyAcked} = \text{segment\#24} - \text{segment\#22} = 2\ \text{MSS}$$

Because (NewlyAcked ≥ `MSS`), the sender uses Eq. (2.9b):

$$\texttt{CongWindow}(54) = \texttt{CongWindow}(53) - \text{NewlyAcked} + \texttt{MSS} = 23 - 2 + 1 = 22 \times \texttt{MSS}$$

Because the current `FlightSize` = 21 × `MSS` (segments #25 through #45 are unacknowledged), one new segment can be sent. As a result, the sender transmits segment #46.

(b)

There will be a total of 12 packets lost during the considered period. The lost packets are (also indicated in Figure 2-25): 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, and 45.

TCP NewReno Fast Recovery Phase

Continuing with Example 2.2, the TCP NewReno sender will enter the fast recovery phase when it discovers the loss of packet #23 at time $t = 45$ (by receiving three dupACKs). The sender will exit the fast recovery phase when it receives a full acknowledgment. The sender originally transmits packet #23 at time $t = 31$ and it immediately arrives at the router where it is lost (Figure 2-25). From time $t = 31$ until the loss is discovered at time $t = 45$, the sender sends a total of 22 “intermediate segments” (segments #24, #25, …, #45). Therefore, the TCP sender will consider it a “full acknowledgment” when it receives an acknowledgment for packet #45. At this point, the sender will *exit fast recovery and enter congestion avoidance*. Note that segment #46, and any segments transmitted thereafter might still be outstanding.

Figure 2-26 shows the continuation of Figure 2-25 for the same Example 2.2. As seen, the sender has not yet received a “full acknowledgment” until time $t = 120$ (packet #45 has not been retransmitted); therefore, the sender is still in the fast recovery state.

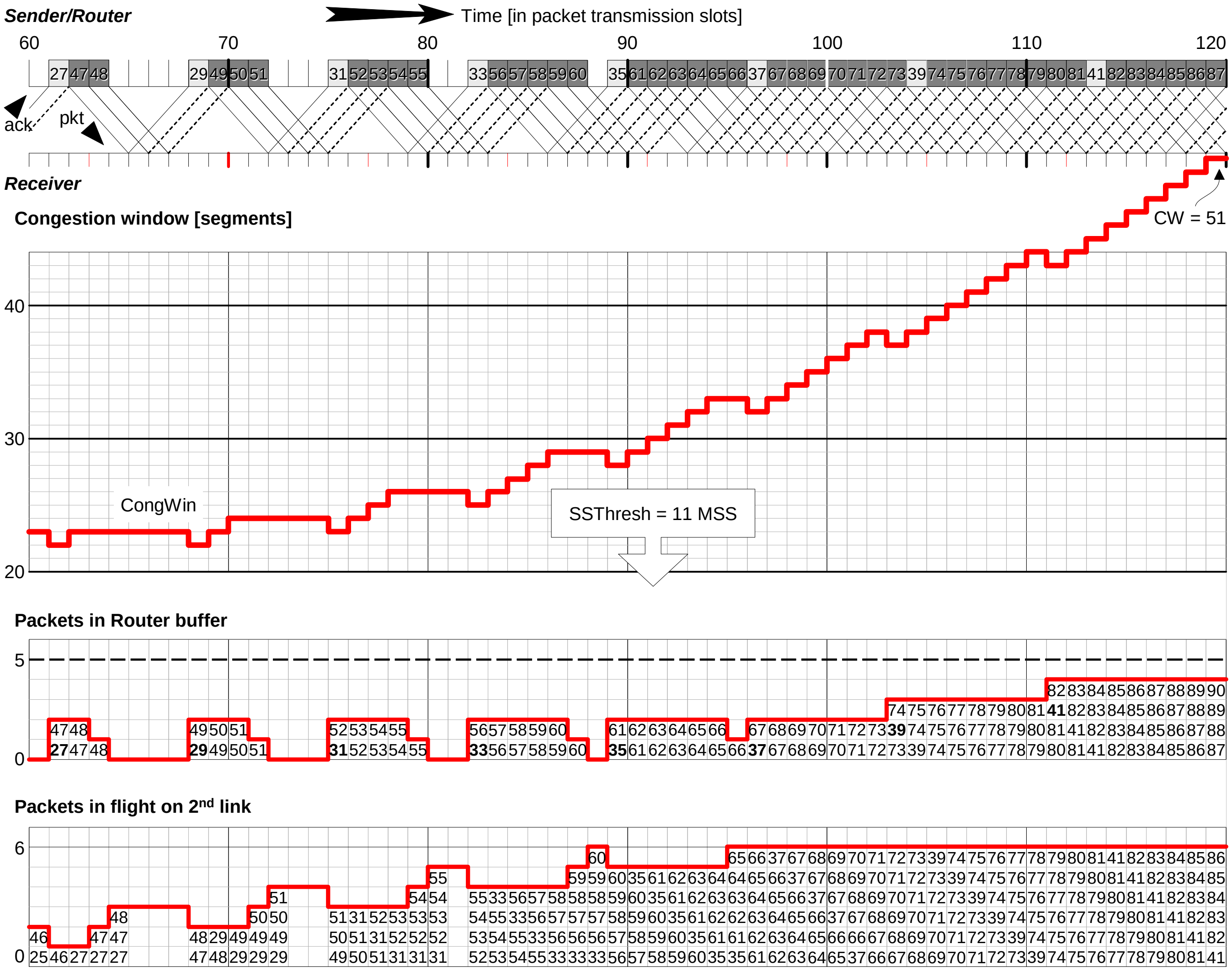


Figure 2-26: Evolution of the key parameters for the TCP NewReno sender in Example 2.2. (*Continued from* Figure 2-25.)

2.3 Fairness

In addition to preventing congestion collapse, there is another reason for employing congestion control: *fairness*. IP does not define any policies for sharing network capacity among flows, so the transport protocols that run above it need to address the sharing issue. The model adopted by TCP is to share the available capacity approximately evenly among all flows. Flows that use UDP as the transport-level protocol (usually multimedia applications, see Chapter 3) may employ different congestion control algorithms (or none). Such strategies upset TCP's fair sharing of resources, usually to the detriment of TCP (which is squeezed out by the more aggressive multimedia traffic).

Studies of the variation in TCP throughput and the behavior of competing TCP flows have shown that competing flows converge to approximately equal shares of the capacity on average, although the saw-tooth profile of TCP sender window means that their instantaneous share of the bandwidth is unlikely to be fair.

As seen in Section 2.2 and 2.4, the evolution of TCP has introduced several variants in the congestion control algorithms, each having slightly different behavior. These factors make it difficult to define fairness for TCP. Over what timescale is fairness measured—instantaneous or long-term average? Is it a problem that long-distance connections get less bandwidth than local connections? What about changes in the protocol that affect behavior in some conditions?

TCP adapts to network congestion on very short timescales, which introduces a systematic unfairness in TCP. Because the algorithm responds to feedback, connections with lower round-trip times return feedback faster and therefore achieve higher sender utilization. This behavior gives a somewhat higher share of the network bandwidth to connections with short network round-trip times. Therefore, connections with longer network round-trip time systematically receive a lower average share of the network capacity.

Therefore, depending on the criteria for comparison, TCP behavior may not be entirely fair. Over the short term, one flow will always win out over another. Over the long term, these variations mostly average out, except that connections with longer round-trip times generally achieve lower throughput on average. The different variants of TCP also have an effect—for example, SACK-TCP achieves better throughput with certain loss patterns. As a result, TCP is at best fair within a factor of 2 or 3, over the long term.

Because it is desirable for multimedia traffic to coexist peacefully with other traffic on the network, the multimedia traffic should employ a congestion control algorithm that is fair to that of TCP. Although it may be desirable to give multimedia traffic priority in some cases, it is not true that multimedia traffic is always more important than TCP traffic, and that it should always get a larger share of the network capacity. Consider a user browsing the Web while listening to an online radio station. In many cases, it is desirable to favor the streaming audio packets over Web browsing packets, so that the user hears uninterrupted music while Web content download may be slowed down. However, what if the Web page the user is submitting is a bid in an online auction, or a stock-trading request for which it is vital that the request be acted on immediately. In such cases, the user would be very unhappy if their online radio station delays the TCP traffic.

Therefore, it is not a proper solution that the application requirements are implied by the transport protocol that the application is using (such as UDP versus TCP). Instead, if an application needs preferential treatment over the regular traffic, this requirement should be explicitly signaled to the network. Such communication allows the network to make an intelligent choice between whether to permit or deny the higher priority on the basis of the available capacity and the user's preferences. Mechanisms such as Differentiated Services and Integrated Services allow for some flows to be given priority in a controlled way, based on application needs, rather than in an undifferentiated manner based on the transport protocol they employ. These service prioritization mechanisms are described in Chapter 3.

2.4 Recent TCP Versions

Early TCP versions, Tahoe and Reno, perform relatively simple system observation and control. They control congestion by first causing it and then controlling it. The TCP Tahoe performance is illustrated in Figure 2-22 over the first 100 transmission rounds. Although the obvious inefficiency (sender utilization is only 25 %) can be somewhat attributed to the contrived scenario of Example 2.1, this is not far from reality. By comparison, a simple Stop-and-Wait protocol would achieve the sender utilization of ?? %. TCP NewReno (Section 2.2.3) solved some problems with earlier versions of TCP. However, NewReno does not perform well when a large number of packets are dropped from a window of data. Both Reno and NewReno senders can retransmit at most one dropped packet per round-trip time, even if senders recover from multiple drops in a window of data without waiting for a retransmit timeout. An extension of TCP, called **TCP with selective acknowledgment (SACK)** allows receivers to additionally report non-sequential data they have received.

Recent TCP variants introduce sophisticated observation and control mechanisms to improve performance. They try to avoid congestion, instead of causing it (and then controlling it).

TCP Vegas [Brakmo & Peterson 1995] watches for the signs of incipient congestion—before losses occur—and takes actions to avert it.

TCP Westwood [Mascolo *et al*. 2001] uses bandwidth estimates to compute the congestion window and slow start threshold after a congestion episode.

FAST TCP [Jin *et al*. 2003] detects congestion by measuring packet delays.

CUBIC TCP [Ha *et al*. 2008] in which the window is a cubic function of time since the last congestion event, with the inflection point set to the window prior to the event.

2.5 TCP over Wireless Links

The TCP congestion control algorithms presented in Section 2.2 assume most packet losses are caused by routers dropping packets due to traffic congestion. However, packets may be also dropped if they are corrupted in their path to destination. In wired networks the fraction of packet loss due to transmission errors is generally low (less than 1 percent). Communication over wireless links is often characterized by sporadic high bit-error rates, and intermittent connectivity due to handoffs. TCP performance in such networks suffers from significant throughput degradation and very high interactive delays

Several factors affect TCP performance in mobile ad-hoc networks (MANETs):

- Wireless transmission errors
- Power saving operation

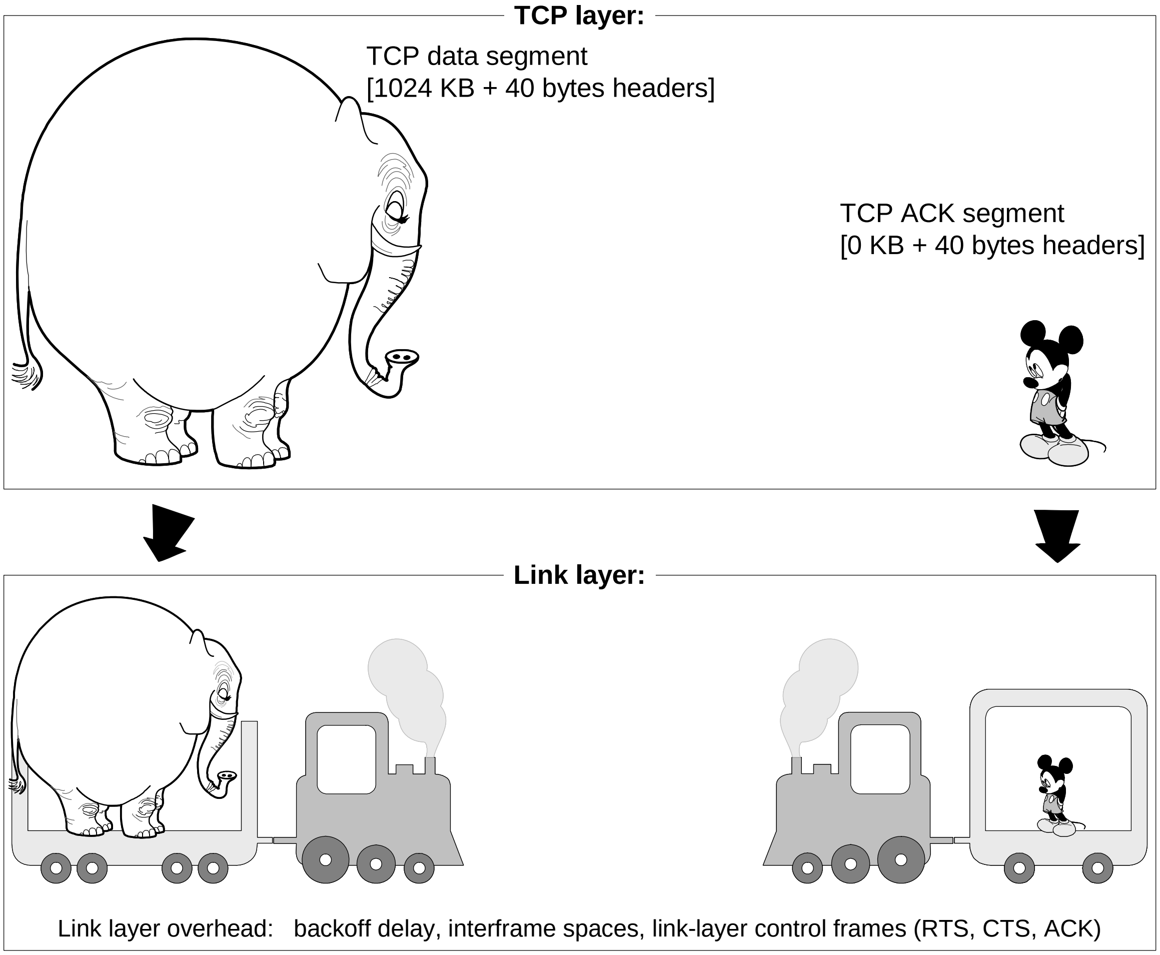

TCP layer:
TCP data segment
[1024 KB + 40 bytes headers]
TCP ACK segment
[0 KB + 40 bytes headers]
Link layer:
Link layer overhead: backoff delay, interframe spaces, link-layer control frames (RTS, CTS, ACK)

Figure 2-27: Due to significant wireless link-layer overhead, TCP data segments and TCP acknowledgments (which are of greatly differing sizes) appear about the same size at the link layer.

- Multi-hop routes on shared wireless medium (for instance, adjacent hops typically cannot transmit simultaneously)
- Route failures due to mobility

Figure 2-27

2.6 Summary and Bibliographical Notes

The TCP service model provides a communication abstraction that is reliable, ordered, point-to-point, duplex, byte-stream, and flow and congestion controlled. TCP’s notion of “duplex” is that the same logical connection handles reliable data delivery in both directions. Unlike ARQ protocols described in Section 1.3, which treat data packets as atomic units, TCP treats bytes as the fundamental unit of reliability.

The focus of this chapter is congestion avoidance and control. The interested reader should consult additional sources for other aspects of TCP, e.g., [Stevens 1994; Peterson & Davie 2007; Kurose & Ross 2010].

TCP sender uses the received cumulative acknowledgments to determine which packets have reached the receiver, and provides reliability by retransmitting lost packets. The sender detects the loss of a packet either by the arrival of several duplicate acknowledgments or the expiration of the timeout timer due to the absence of an acknowledgment for the packet. To accurately set the timeout interval, the sender maintains a running average of the estimated roundtrip delay and the mean linear deviation from it. The timeout interval is calculated as the sum of the smoothed round-trip delay plus four times its mean deviation. TCP reacts to packet losses by decreasing its transmission (congestion) window size before retransmitting packets, initiating congestion control or avoidance mechanisms (e.g., slow start), and backing off its retransmission timer (Karn's algorithm). These actions result in a reduction in the load on the intermediate links, thereby controlling the congestion in the network.

[Stevens, 1994] provides the most comprehensive coverage of TCP in a single book. It appears that this whole book is available online at http://www.uniar.ukrnet.net/books/tcp-ip_illustrated/.

[Comer, 2006] is also very good, although does not go in as much detail.

The UDP protocol was designed by David P. Reed in 1980 and formally defined in RFC-768.

The TCP protocol described in 1974 by Vinton Cerf and Robert Kahn in *IEEE Transactions on Communication*. Three-way handshake described by Raymond Tomlinson in SIGCOMM 1975.

TCP/IP initial standard appeared in 1982 (RFC-793 & RFC-791). BSD Unix 4.2 released in 1983 supported TCP/IP. Van Jacobson's algorithms for congestion avoidance and congestion control published in 1988 and described also in RFC-1122; most of it was implemented in 4.3 BSD Tahoe and described in [Leffler et al., 1989]. TCP Tahoe was also described in [Stevens 1994]. Some important IETF Requests for Comments (RFCs) related to TCP include:

- RFC-1191 (1990) — Path MTU Discovery
- RFC-1323 (1992) — TCP Extensions for High Performance (defines the options field, etc.)
- RFC-2018 (1996) — TCP Selective Acknowledgment Options
- RFC-2581 (1999) — TCP Congestion Control [Obsoleted by: RFC-5681]
- RFC-2988 (2000) — Computing TCP's Retransmission Timer [Obsoleted by: RFC-6298]
- RFC-5681 (2009) — TCP Congestion Control
- RFC-6298 (2011) — Computing TCP's Retransmission Timer

Section 2.1.3: TCP Retransmission Timer

In the original TCP specification (RFC-793), the retransmission timeout (RTO) was set as a multiple of a running average of the RTT. For example, it might have been set as `TimeoutInterval` = $\eta \times$ `EstimatedRTT`, with η set to a constant such as 2. However, this simple choice failed to take into account that at high loads round trip variability becomes high, leading to unnecessary retransmissions. The solution offered by Jacobson, see Eq. (2.1), factors in both average and standard deviation.

The computing and behavior of the RTO timer is specified in RFC-6298.

For details about how timestamps are carried in the Options field of the TCP header, the reader should check RFC-1323, Section 3.2 “TCP Timestamps Option.”

Section 2.2.2: TCP Reno

The TCP Reno was first implemented in the 1990 BSD Reno release and referred to as the Reno algorithm in [Stevens 1994; Fall & Floyd, 1996]. Reno was standardized in RFC-2001 – “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms.” RFC-2001 has since been replaced by “RFC-2581 – TCP Congestion Control.” RFC-2581 was then updated by RFC-3390 – “Increasing TCP’s Initial Window” and then replaced by RFC-5681 – “TCP Congestion Control.”

Section 2.2.3: TCP NewReno

In 1996, Janey Hoe [Hoe, 1996] proposed an enhancement to TCP Reno, which subsequently became known as *NewReno*. The main idea here is for the TCP sender to remain in fast recovery until all the losses in a window are recovered.

There have been other enhancements proposed to TCP over the past few years, such as TCP Vegas congestion control method [Brakmo & Peterson, 1995], various optimizations for wireless networks, optimizations for small windows (e.g., RFC-3042), etc.

RFC-3168, 3155, 3042, 2884, 2883, 2861, 2757, 2582 (NewReno)

The NewReno modification of TCP’s Fast Recovery algorithm is described in RFC-3782 [Floyd et al., 2004].

Section 2.5: TCP over Wireless Links

TCP over wireless:

[Balakrishnan, *et al*., 1997], [Holland & Vaidya, 1999], [Fu, *et al*., 2005].

TCP has supported ongoing research since it was written. As a result, the End-to-End research group has published a Roadmap for TCP Specification Documents [RFC-4614] which will guide expectations in that area.

SSFnet.org, “TCP Regression Tests,” Online at: http://www.ssfnet.org/Exchange/tcp/tcpTestPage.html

The SSFnet.org tests show the behavior of SSF TCP Tahoe and Reno variants for different networks, TCP parameter settings, and loss conditions.

NASA Jet Propulsion Laboratory (JPL) and Vinton Cerf recently jointly developed Disruption-Tolerant Networking (DTN) protocol to transmit images to and from a spacecraft more than 20 million miles from Earth. DTN is intended for reliable data transmissions over a deep space communications network, for which the TCP/IP protocol suite is unsuitable. An interplanetary Internet needs to be strong enough to withstand delays, disruptions, and lost connections that space can cause. For example, errors can happen when a spacecraft slips behind a planet, or when solar storms or long communication delays occur. Even traveling at the speed of light, communications sent between Mars and Earth take between three-and-a-half minutes to 20 minutes. Unlike TCP, DTN does not assume there will be a constant end-to-end connection. DTN is designed so that if a destination path cannot be found, the data packets are not discarded but are kept in a network node until it can safely communicate with another node. The interplanetary Internet could allow for new types of complex space missions that involve multiple landed, mobile, and orbiting spacecraft, as well as ensure reliable communications for astronauts on the surface of the moon.

http://www.jpl.nasa.gov/news/news.cfm?release=2008-216

Problems

Problem 2.1

Consider the TCP procedure for estimating RTT with $\alpha = 0.125$ and $\beta = 0.25$. Assume that the `TimeoutInterval` is initially set as 3 seconds. Suppose that **all** measured RTT values equal 5 seconds, no segment loss, and the segment transmission time is negligible. The sender starts sending at time zero.

(a) What values will `TimeoutInterval` be set to for the segments sent during the first 11 seconds?

(b) Assuming a TCP Tahoe sender, how many segments will the sender transmit (including retransmissions) during the first 11 seconds?

(c) Repeat steps (a) and (b) but this time around assume that the sender picked the initial `TimeoutInterval` as 5 seconds?

Show the work.

Problem 2.2

Consider two hosts connected by a local area network with a negligible round-trip time. Assume that one is sending to the other a large amount of data using TCP with RcvBuffer = 20 Kbytes and MSS = 1 Kbytes. Also assume error-free transmission, high-speed processors in the hosts, and reasonable values for any other parameters that you might need.

(a) Draw the congestion window diagram during the slow-start (until the sender enters congestion avoidance) for the network speed of 100 Mbps.

(b) How different the diagram becomes if the network speed is reduced to 10 Mbps? 1 Mbps?

(c) What will be the average throughput (amount of data transmitted per unit of time) once the sender enters congestion avoidance?

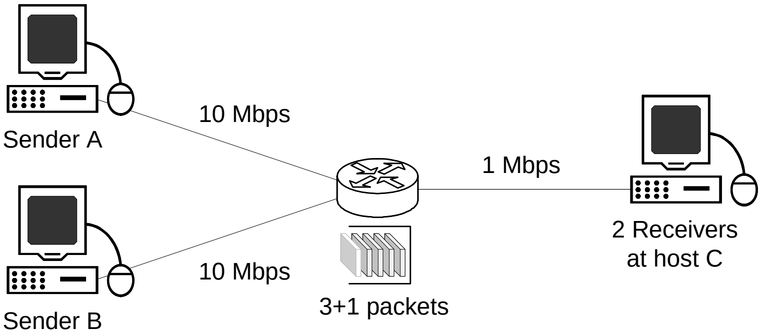
Explain your answers.

Problem 2.3

Suppose that the hosts from Problem 2.2 are connected over a satellite link with RTT = 20 ms (low earth orbit satellites are typically 850 km above the Earth surface). Draw the congestion window diagram during the slow-start for the network speed of 100 Mbps. Explain any similarities or differences compared to the one from Problem 2.2(a).

Problem 2.4

Consider the network shown in the figure. TCP senders at hosts *A* and *B* have 3.6 KB of data each to send to their corresponding TCP receivers, both running at host *C*. Assume MTU = 512 bytes for all the links and `TimeoutInterval` = 2×RTT = 2×1 sec. The router buffer size is 3 packets in addition to the packet currently being transmitted; should the router need to drop a packet, it drops the last arrived from the host which currently sent more packets. Sender *A* runs TCP Tahoe and sender *B* runs TCP Reno and assume that sender *B* starts transmission 2×RTTs after sender *A*.



(a) Trace the evolution of the congestion window sizes on both senders until all segments are successfully transmitted.

(b) What would change if `TimeoutInterval` is modified to 3×RTT = 3×1 sec?

Assume a large `RcvWindow` and error-free transmission on all the links. Finally, to simplify the graphs, assume that all ACK arrivals occur exactly at unit increments of RTT and that the associated `CongWindow` update occurs exactly at that time, too.

Problem 2.5

Consider a TCP Tahoe sender working on the network with RTT = 1 sec, `MSS` = 1 KB, and the bottleneck link bandwidth equal to 128 Kbps. Ignore the initial slow-start phase and assume that the sender exhibits periodic behavior where a segment loss is always detected in the congestion avoidance phase via duplicate ACKs when the congestion window size reaches `CongWindow` = 16×`MSS`.

(a) What is the min/max range in which the window size oscillates?

(b) What will be the average rate at which this sender sends data?

(c) Determine the utilization of the bottleneck link if it only carries this single sender.

[Hint: When computing the average rate, draw the evolution of the congestion window. Assume `RcvWindow` large enough not to matter.]

Problem 2.6

Specify precisely a system that exhibits the same behavior as in Problem 2.5:

- What is the buffer size at the bottleneck router?
- What is the minimum value of `TimeoutInterval`?

Demonstrate the correctness of your answer by graphing the last two transmission rounds before the segment loss is detected and five transmission rounds following the loss detection.

Problem 2.7

Consider two hosts communicating using the TCP-Tahoe protocol. Assume RTT = 1, `MSS` = 512 bytes, `TimeoutInterval` = 3×RTT, `SSThresh` = 3×`MSS` to start with, and `RcvBuffer` = 2 KB. Also, assume that the bottleneck router has available buffer size of 1 packet in addition to the packet currently being transmitted.

(a) Starting with `CongWindow` = 1×`MSS`, determine the congestion window size when the first packet loss will happen at the router (not yet detected at the sender).

(b) What will be the amount of unacknowledged data at the sender at the time the sender detects the loss? What is the total number of segments acknowledged by that time?

Assume that no cumulative ACKs are sent, i.e., each segment is acknowledged individually.

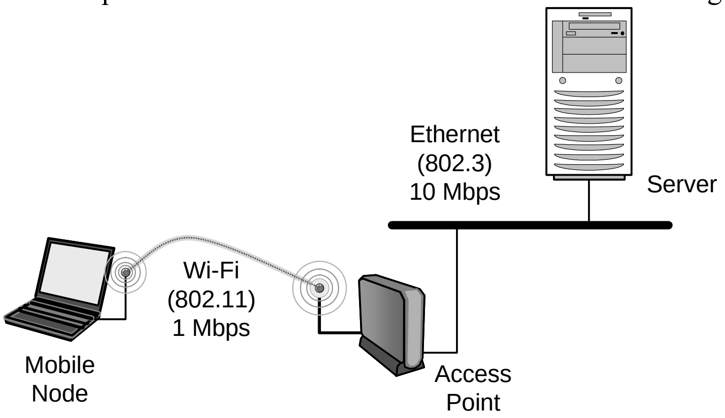
Problem 2.8

Consider two hosts communicating by TCP-Reno protocol. Assume RTT = 1, `MSS` = 256 bytes, `TimeoutInterval` = 3×RTT, `RcvBuffer` = 2 KB, and the sender has a very large file to send. Start considering the system at the moment when it is in slow start state, `CongWin` = 8×`MSS`, `SSThresh` = 10×`MSS` and the sender has just sent eight segments, each 1×`MSS` bytes long. Assume that there were no lost segments before this transmission round and currently there are no buffered segments at the receiver.

Assuming that, of the eight segments just sent, the fourth segment is lost, trace the evolution of the congestion window sizes for the subsequent five transmission rounds. Assume that no more segments are lost for all the considered rounds. For every step, indicate the transmitted segments and write down the numeric value of `CongWin` (in bytes). To simplify the charts, assume that ACK arrivals occur exactly at unit increments of RTT and that the associated `CongWin` update occurs exactly at that time, too.

Problem 2.9

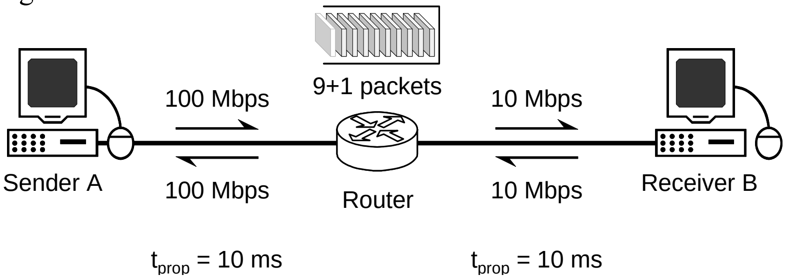
Consider the network configuration shown in the figure below. The mobile node connects to the server using the TCP protocol to download a large file. Assume MSS = 1024 bytes, error-free transmission, and sufficiently large storage spaces at the access point and the receiver. Assume that the Assuming that the TCP receiver sends only cumulative acknowledgments. Calculate how long time it takes to deliver the first 15 Kbytes of data from that moment the TCP connection is established. In addition, draw the timing diagram of data and acknowledgment transmissions. (You can exploit the fact that TCP sends cumulative acknowledgments.)



(In case you need these, assume the distance between the mobile node and the access point equal to 100 m, and the same from the access point to the server. Also, the speed of light in the air is 3×10^8 m/s, and in a copper wire is 2×10^8 m/s.)

Problem 2.10

Consider an application that is engaged in a lengthy file transfer using the TCP Tahoe protocol over the following network.



The following assumptions are made:

A1. Full duplex links connect the router to each endpoint host so that simultaneous transmissions are possible in both directions on each link. The link transmission rates are as indicated. One-way propagation delay on each link equals 10 ms. Assume that all packet transmissions are error free.

A2. Each data segment sent by the sender is 1250 bytes long. You can ignore all header overheads, so the transmission delay for data packets over a 100 Mbps link is exactly 0.1 ms and over 10 Mbps is exactly 1 ms. Also assume that the ACK packet size is negligible, i.e., their transmission delay is approximately zero.

A3. The router buffer can hold up to nine packets plus one packet currently in transmission. The packets that arrive to a full buffer are dropped. However, this does *not* apply to ACK packets, i.e., ACKs do not experience congestion or loss.

A4. The receiver does not use delayed ACKs, i.e., it sends an ACK immediately after receiving a data segment.

A5. The receiver has set aside a buffer of RcvBuffer = 64 Kbytes for the received segments.

Answer the following questions:

(a) What is the minimum possible time interval between receiving two consecutive ACKs at the sender?

(b) Write down the transmission start times for the first 7 segments.

(c) Write down the congestion widow sizes for the first 6 transmission rounds, i.e., the first 6 RTTs. (Hint: Try to figure out the pattern of packet arrivals and departures on the router, to understand how the queue of packets grows and when the buffer is fully occupied, so the next packet is dropped.)

(d) In which round will the first packet be dropped at the router? What is the ordinal number of the first dropped packet, starting with #1 for the first packet? Explain your answer.

(e) What is the congestion window size at the 11th transmission round?

(f) What is the long-term utilization of the TCP sender (ignore the initial period until it stabilizes)?

(g) What is the long-term utilization of the link between the router and the receiver (again, ignore the initial period until it stabilizes)?

(h) What will change if delayed ACKs are used to acknowledge cumulatively multiple packets?

(i) Estimate the sender utilization under the delayed ACKs scenario.

Problem 2.11

Consider a TCP Tahoe sender working with MSS = 1 KB, and the bottleneck link bandwidth equal to 1 Mbps. Ignore the initial slow-start phase and assume that the network exhibits periodic behavior where every tenth packet is lost. Consider three different scenarios where all parameters remain the same except for the round-trip time, which changes as: $\text{RTT}_1 = 0.01$ sec, $\text{RTT}_2 = 0.1$ sec, and $\text{RTT}_3 = 1$ sec.

What will be the average rate at which this sender sends data for the different scenarios? Provide an explanation in case you observe any differences between the three scenarios.

Problem 2.12

Calculate the total time required for transferring a 1-MB file from a server to a client in the following cases, assuming an RTT of 100 ms, a segment size of 1 KB, and an initial 2×RTT of "handshaking" (initiated by the client) before data is sent. Assume error-free transmission.

(a) The bottleneck bandwidth is 1.5 Mbps, and data packets can be sent continuously (i.e., without waiting for ACKs)

(b) The bottleneck bandwidth is 1.5 Mbps, but Stop-and-wait ARQ is employed

(c) The bandwidth is infinite, meaning that we take transmission time to be zero, and Go-back-20 is employed

(d) The bandwidth is infinite, and TCP Tahoe is employed

Problem 2.13

Consider a TCP Reno sender, which is in the middle of sending a large amount of data and assume that you are observing it at time t_i. Let t_i, t_{i+1}, t_{i+2}, …, t_{i+7} denote times when the TCP sender will send the subsequent 8 data segments, as governed by its congestion control algorithm. The following assumptions are made:

A1. The TCP sender's segment size equals `MSS` = 200 bytes. At time t_i, the sender is in the *slow start* phase and the congestion window size is already updated as `CongWin`(t_i) = 400 bytes. There are currently no unacknowledged segments. The slow start threshold `SSThresh`(t_i) = 64 Kbytes and the receiver's buffer size `RcvWindow`(t_i) = 1000 bytes.

A2. The sender's sequence number for the next segment that will be transmitted at time t_i equals 30. Assume that the sender transmits back-to-back all the segments that are permitted by its current `EffectiveWindow` size (i.e., the segments are sent in "bursts"). Assume that the segment transmission time is much smaller than the propagation time, i.e., $t_x << t_p$ and $t_p \approx ½$ RTT.

A3. The receiver does not use delayed ACKs, i.e., it sends an ACK immediately after receiving a data segment. All in-order segments are immediately delivered to the application and they never linger in the receive buffer.

A4. The estimated round-trip time at time t_{i-1} equals `EstimatedRTT`(t_{i-1}) = 100 milliseconds, the standard deviation equals `DevRTT`(t_{i-1}) = 10 milliseconds, and `SampleRTT`(t_i) = 106 ms.

Any subsequent transmissions will experience the following round-trip times (from the moment a data segment is transmitted from the sender until the corresponding ACK is received at the sender): `RTT`(t_i) = 105 ms, `RTT`(t_{i+1}) = 93 ms, `RTT`(t_{i+2}) = 179 ms, RTT(t_{i+3}) = 182 ms, RTT(t_{i+4}) = 165 ms, RTT(t_{i+5}) = 193 ms, RTT(t_{i+6}) = 154 ms, and RTT(t_{i+7}) = 171 ms.

Note: the above values RTT(t) are different from SampleRTT(t), which is the RTT value measured at time t.

Starting at time t_i, consider the subsequent 8 segment transmissions and do the following:

(a) Show the congestion window sizes CongWin(t) and the sequence numbers of the segments transmitted from the sender at times $t = t_i$, t_{i+1}, t_{i+2}, …, t_{i+7}.

(b) Show the sequence numbers of the corresponding acknowledgments and indicate the times when the ACKs will arrive. Also show the values of RcvWindow(t) as carried in each acknowledgment packet.

(c) Show the values of EstimatedRTT(t) and DevRTT(t) as measured by the TCP retransmission-timer management algorithm.

(d) Indicate the times when the TCP sender will set its retransmission timer, if any, as dictated by the TCP algorithm and write down the values of TimeoutInterval(t).

Problem 2.14

Problem 2.15

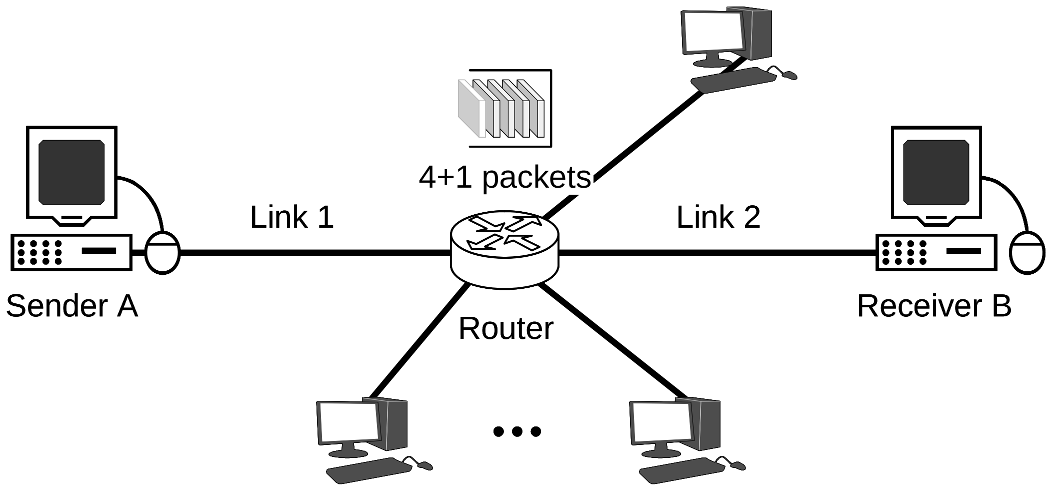
TCP NewReno RTO Timeout Timer Calculation

Consider the evolution of TCP NewReno parameters shown in Figure 2-26 for Example 2.2. Starting with time t = 89 when segment #35 is retransmitted, show the values of TimeoutInterval(t), calculated using Eq. (2.1). Stop when the ACK for the retransmitted #41 arrives, which will happen at t =120 and show the value of TimeoutInterval(120). Assume that at time t = 88, EstimatedRTT(88) = 6, DevRTT(88) = 0.05, and the values of the control parameters α = 0.125 and β = 0.25.

Follow the procedure for computing TimeoutInterval(t) explained in Section 2.1.3 (and summarized in the pseudocode at the end of this section) as closely as possible. Explain how you obtained every new value of TimeoutInterval(t).

Problem 2.16

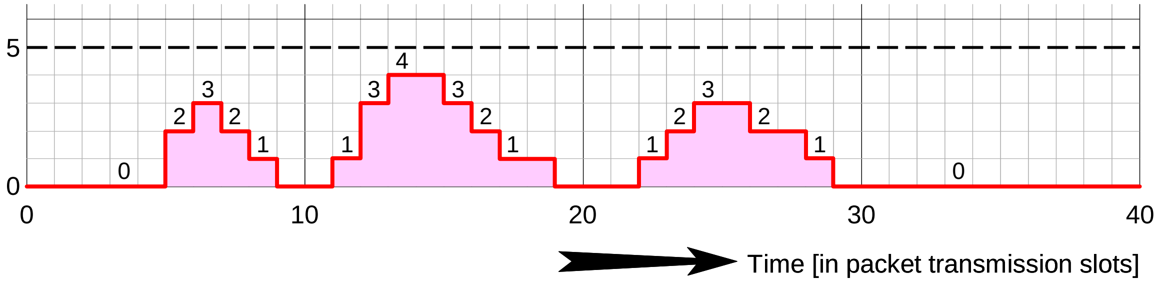
Consider a TCP NewReno connection described in Example 2.2, which is modified so other concurrent connections are competing for the router resources, as shown in the figure below. All other parameters remain the same as in Example 2.2.



The packets from our connection (sender *A* to receiver *B*) arriving to the router may find packets from other connections already waiting in the router buffer. Assume that our packets will at different times find the buffer occupancy as shown in the figure below. For example, if a packet from our flow arrive to the router at time *t* = 7, it would find 3 packets from *other* flows already waiting at the router. There may also be packets that have arrived previously from our flow, but they are *not* shown. It is your task to keep track of such packets and include them in the total count of packets in the router.

Also assume that packets from other connections are just competing for the memory space, but they are heading for a different output link. In other words, we assume that packets from our connection will never see packets from other flows waiting in front of them to be transmitted. The figure below shows only how many packets are already waiting in the buffer from other flows.

Packets already in Router Buffer from other concurrent connections



Starting at time 0 when our TCP connection from sender *A* to receiver *B* just became established, do the following:

(e) At what time the first segment from our flow will be lost at the router? At what time will the sender *A* detect this loss?

(f) Show the congestion window sizes `CongWin`(*t*) and the sequence numbers of the segments transmitted from our sender *A* until the time *t* = 30.

(g) Show the total buffer occupancy at the router from all flows (including ours), until the time *t* = 30.

Problem 2.17

Chapter 3
Multimedia and Real-time Applications

Contents

We first concentrate on the parameters of the network players, traffic characteristics of information sources, information needs of the receivers, and delay and loss introduced by the intermediaries. Then we review the techniques designed to mitigate the delay and loss to meet the receivers needs in the best possible way.

3.1 Application Requirements

People needs determine the system requirements. In some situations it is necessary to consider human users as part of an end-to-end system, treating them as active participants, rather than passive receivers of information. For instance, people have thresholds of boredom, and finite reaction times. A specification of user's perceptions is thus required, as it is the user that ultimately defines whether the result has the right quality level. Based on people's business needs, two general classes of applications are recognized:

- ***Elastic applications*** do not impose strict delay requirements, such as traditional data applications
- ***Real-time applications*** impose bounds on delay, jitter, and loss, such as:
  - *Loss*: intolerant or tolerant to some loss
  - *Delay*: not adaptive or adaptive (e.g., lengthening or shortening the silence between words, playing back video a little slower, etc.)

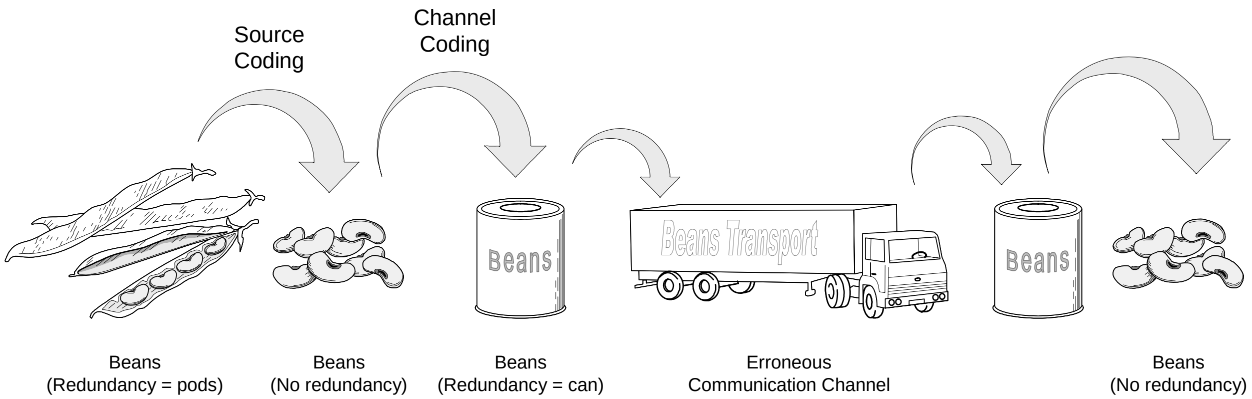


Figure 3-1. An analogy for comparing source coding and channel coding. Source coding removes bean pods because they are not the "useful" kind of redundancy. Channel coding envelops beans with metal cans to protect them in transport.

- *Data rate*: not adaptive or adaptive (e.g., reduce video quality by compressing video more)

A key principle is that different application requirements require different service classes, instead of only the "best effort" that we considered so far. A network that can provide these different levels of service is said to support *quality of service* (QoS).

An important concept for understanding network traffic sources is that of source coding. The purpose of *source coding* is to remove redundancy from source information and reduce the quantity of data that needs to be transmitted without losing valuable information. A related concept is *channel coding*, described in Section 1.2, which adds new redundancy to harden the signal against noise impairments in such a way that it guarantees achieving maximum transmission capacity in the presence of noise. An analogy in Figure 3-1 may help in understanding this dichotomy. Here, we remove the pods (source coding) to ship as many beans as possible. We then add cans to protect the beans from damage in transportation.

A *traffic model* summarizes the expected "typical" behavior of a source or an aggregate of sources. Of course, this is not necessarily the ultimate source of the network traffic. The model may consider an abstraction by "cutting" a network link or a set of links at any point in the network and considering the aggregate "upstream" system as the source(s).

Traffic models fall into two broad categories. Some models are obtained by detailed traffic measurements of thousands or millions of traffic flows crossing the physical link(s) over days or years. Others are chosen because they are amenable to mathematical analysis. Unfortunately, only a few models are both empirically obtained and mathematically tractable.

Figure 3-2 clarifies what represents a network traffic source that generates messages or data packets. Source is usually some kind of computing or sensory device, such as microphone, camera, etc. However, it may not be always possible to identify the actual traffic source. For example, the source could be within organizational boundaries, concealed for security or privacy reasons. In a global Internet with many autonomous systems, it is virtually impossible to identify the actual originators of network traffic. Therefore, the notion of traffic source depends on the

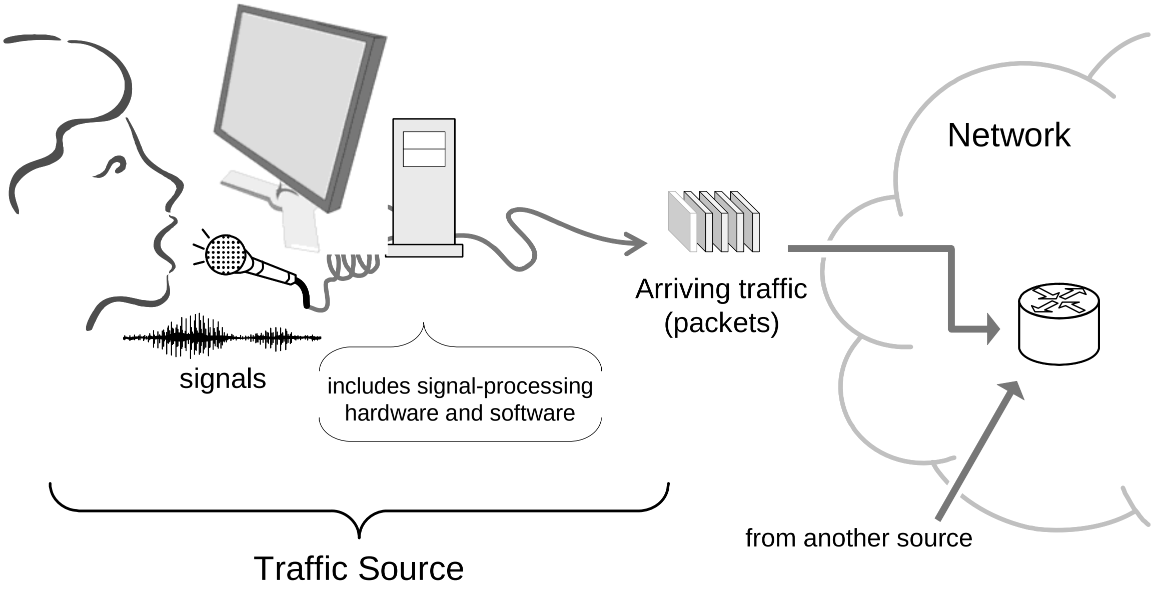

Network
Arriving traffic (packets)
signals
includes signal-processing hardware and software
from another source
Traffic Source

Figure 3-2: What is a traffic source.

observation cutpoint at any point within the network, and it is anything that generates incoming traffic at the cutpoint.

Quality of service can be applied to individual traffic "flows" or to flow aggregates, where a **flow** is identified by a source-destination pair. Hence, there are two other ways to characterize types of QoS:

- **Per Flow**: A "flow" is defined as an individual, unidirectional, data stream between two applications (sender and receiver), uniquely identified by a 5-tuple (transport protocol, source address, source port number, destination address, and destination port number).
- **Per Aggregate**: An aggregate is simply two or more flows. Typically, the flows will have something in common (e.g., any one or more of the 5-tuple parameters, a label or a priority number, or perhaps some authentication information).

Two key traffic parameters of traffic models are:

- Message arrival rate (at what rate messages/packets are arriving)
- Message servicing time (how long it takes to process the messages)

Message (packet) arrival rate specifies the average number of packets generated by a given source per unit of time. Message servicing time specifies the average time needed for processing of messages from a given source at a given server[18]. Within the network, packet-servicing time in routers comprises not much more than message-header inspection for correct forwarding plus the transmission time, which is directly proportional to the packet length. See Section 4.1 for details.

[18] We use the term "server" in a generic sense as any type of message processor. In data networks, "servers" are usually network routers.

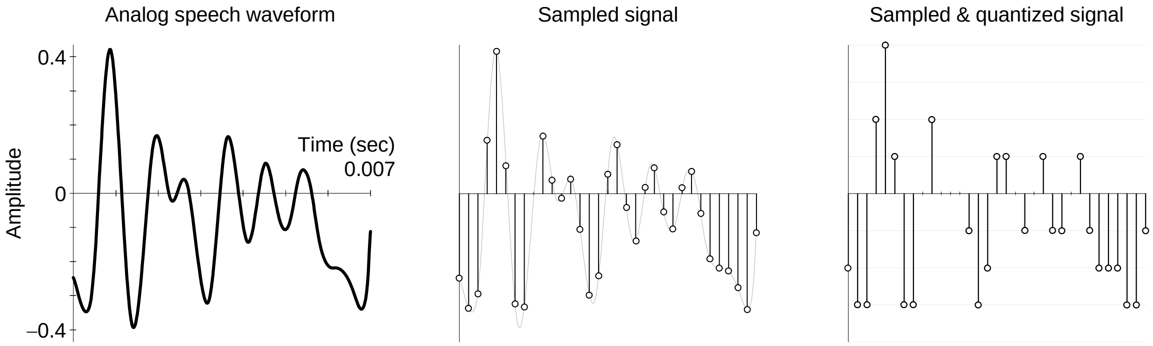


Figure 3-3: Analog speech signal original (a), sampling (b), and quantization to 4 bits (c).

In the following analysis, we will usually assume that the traffic source is infinite, because an infinite source is easier to describe mathematically. For a finite source, the arrival rate is affected by the number of messages already sent; indeed, if all messages are already sent, the arrival rate drops to zero. If the source sends a finite but large number of messages, we assume an infinite source to simplify the analysis.

The rate at which a traffic source sends packets (original as well as retransmitted, if any) into the network will be referred to as the **offered load** to the network, and denoted as λ_{in}'.

Traffic models commonly assume that packets arrive as a Poisson process, that is, the interarrival time between calls is drawn from an exponential distribution.

Packet servicing times have traditionally been modeled as drawn from an exponential distribution. That is, the probability that the servicing lasts longer than a given length *x* decreases exponentially with *x*. However, recent studies have shown servicing times to be *heavy-tailed*. Intuitively, this means that many packets are very long. More precisely, if T_p represents the packet servicing time, and $c(t)$ is defined to be a slowly varying function of time *t* when *t* is large, the probability that a packet will need to be serviced longer than *t* is given by:

$$P(T > t) = c(t) \cdot t^{-\alpha} \quad \text{as } t \to \infty,\ 1 < \alpha < 2$$

As Figure 3-xxx shows, a heavy-tailed distribution has a significantly higher probability mass at large values of *t* than an exponential function.

3.1.1 Application Types

Multimedia application bandwidth requirements range from 8Kbps for a G.729 speech codec and H.263 64Kbps video codec to 19.2 Mbps for MPEG2, P, 4:2:0 (US standard) based videoconferencing and 63Mbps SXGA 3D computer games [DuVal & Siep 2000]. In general, the higher the speech sampling rate, the better the potential call quality (but at the expense of more bandwidth being consumed). For example, G.711 encoding standard for audio provides excellent quality. Data is delivered at 64 Kbps, and the codec imposes no compression delay. Technically, G.711 delivers 8,000 bytes per second without compression so that full Nyquist-dictated samples are provided.

Applications may also have periodic traffic for real-time applications, aperiodic traffic for web browsing clients, aperiodic traffic with maximum response times for interactive devices like the mouse and keyboard, and non-real time traffic for file transfers. Thus, we see that the range of bandwidth and timeliness requirements for multimedia applications is large and diverse.

Table 3-1: Characteristics of traffic for some common sources/forms of information.

Source	Traffic type	Arrival rate/Service time	Size or Rate
Voice	CBR	Deterministic/ Deterministic	64 Kbps
Video	CBR	Deterministic/ Deterministic	64 Kbps, 1.5 Mbps
	VBR	Deterministic/Random	Mean 6 Mbps, peak 24 Mbps
Text	ASCII	Random/Random	2 KB/printed-page
	Fax	Random/ Deterministic	50 KB/printed-page
Picture	600 dots/in, 256 colors, 8.5 × 11 in	Random/ Deterministic	33.5 MB
	70 dots/in, b/w, 8.5 × 11 in	Random/ Deterministic	0.5 MB

Table 3-1 shows some characteristics of network traffic generated by common forms of information. Note that the bit streams generated by a video signal can vary greatly depending on the compression scheme used. When a printed page of text is encoded as a string of ASCII characters, it produces a 2-Kbyte string; when that page is digitized into pixels and compressed as in facsimile, it produces a 50-KB string. A high-quality digitization of color pictures (similar quality to a good color laser printer) generates a 33.5-MB string; a low-quality digitization of a black-and-white picture generates only a 0.5-MB string.

We classify all traffic into three types. A user application can generate a constant bit rate (CBR) stream, a variable bit rate (VBR) stream, or a sequence of messages with different temporal characteristics. We briefly describe each type of traffic, and then consider some examples.

Constant Bit Rate (CBR)

To transmit a voice signal, the telephone network equipment first converts it into a stream of bits with constant rate of 64 Kbps. Some video-compression standards convert a video signal into a bit stream with a constant bit rate (CBR). For instance, MPEG-1 is a standard for compressing video into a constant bit rate stream. The rate of the compressed bit stream depends on the parameters selected for the compression algorithm, such as the size of the video window, the number of frames per second, and the number of quantization levels. MPEG-1 produces a poor quality video at 1.15 Mbps and a good quality at 3 Mbps.

Voice signals have a rate that ranges from about 4 Kbps when heavily compressed and low quality to 64 Kbps. Audio signals range in rate from 8 Kbps to about 1.3 Mbps for CD quality.

Variable Bit Rate (VBR)

Some signal-compression techniques convert a signal into a bit stream that has variable bit rate (VBR). For instance, MPEG-2 is a family of standards for such variable bit rate compression of video signals. The bit rate is larger when the scenes of the compressed video are fast changing their content than when the content is slowly changing. Direct Broadcast Satellite (DBS) uses MPEG-2 with an average data rate of 4 Mbps.

To specify the characteristics of a VBR stream, the network engineer specifies the average bit rate and a statistical description of the fluctuations of that bit rate. More about such descriptions will be said later.

Messages

Many user applications are implemented as processes that exchange messages over a network. An example is Web browsing, where the user clicks on hyperlinks generating requests to a web server for Web pages with embedded multimedia information and the server replies with the requested items. The message traffic can have a wide range of characteristics. Some applications, such as email, generate isolated messages. Other applications, such as distributed computation, generate long streams of messages. The rate of messages can vary greatly across applications and devices.

To describe the traffic characteristics of a message-generating application, the network engineer may specify the average traffic rate and a statistical description of the fluctuations of that rate, in a way similar to the case of a VBR specification.

See definition of fidelity in:

B. Noble, "System support for mobile, adaptive applications," IEEE Personal Communications, 7(1), pp.44-49, February 2000.

E. de Lara, R. Kumar, D. S. Wallach, and W. Zwaenepoel, "Collaboration and Multimedia Authoring on Mobile Devices," *Proc. First Int'l Conf. Mobile Systems, Applications, and Services (MobiSys 2003)*, San Francisco, CA, pp. 287-301, May 2003.

In any scenario where information is communicated, two key aspects of information are fidelity and timeliness. Higher fidelity implies greater quantity of information, thus requiring more resources. The system resources may be constrained, so it may not be possible to transmit, store, and visualize at a particular fidelity. If memory and display are seen only as steps on information's way to a human consumer, then they are part of the communication channel. The user could experience pieces of information at high fidelity, sequentially, one at a time, but this requires time and, moreover, it requires the user to assemble in his or her mind the pieces of the puzzle to experience the whole. Some information must be experienced within particular temporal and or spatial (structural?) constraints to be meaningful. For example, it is probably impossible to experience music one note at a time with considerable intervals in between. Or, a picture cannot be experienced one pixel at a time. Therefore, the user has to trade fidelity for temporal or spatial capacity of the communication channel.

Information loss may sometimes be tolerable; e.g., if messages contain voice or video data, most of the time the receiver can tolerate some level of loss. Packets that do not confirm to the delay bound are considered to be lost. From a receiver's viewpoint, packets can be lost for two reasons:

(1) Router buffer overflows or noise corruption in links

(2) Late arrival at he the receiver

Shannon had to introduce fidelity in order to make problem tractable [Shannon & Weaver 1949].

Information can be characterized by fidelity ~ info content (entropy). The effect of a channel can be characterized as deteriorating information’s fidelity and increasing the latency:

$$\text{fidelity}_{\text{IN}} + \text{latency}_{\text{IN}} \rightarrow ()\_\_\_\_\_) \rightarrow \text{fidelity}_{\text{OUT}} + \text{latency}_{\text{OUT}}$$

Wireless channels in particular suffer from various limitations. Increasing the channel capacity to reduce latency is usually not feasible—either it is not physically possible or it is too costly.

Information qualities can be considered in many dimensions. We group them in two opposing ones:

- Those that tend to increase the information content
- Delay and its statistical characteristics

The computing system has its limitations as well. If we assume *finite buffer length*, then in addition to *delay* problem, there is a *random loss* problem. This further affects the fidelity. Fidelity has different aspects, such as:

- Spatial (sampling frequency in space and quantization – see Brown&Ballard CV book)
- Temporal (sampling frequency in time)
- Structural (topologic, geometric, …)

Delay or latency may also be characterized with more parameters than just instantaneous value, such as the amount of variability of delay, also called *delay jitter*. In real life both fidelity and latency matter and there are thresholds for each, below which information becomes useless. The system is forced to manipulate the fidelity in order to meet the latency constraints. A key question is, how faithful should signal be in order to be quite satisfactory without being too costly? In order arrive at a right tradeoff between the two, the system must know:

1. Current channel quality parameters, e.g., capacity, which constrain fidelity and latency
2. User’s tolerances for fidelity and latency

The former determines *what* can be done, i.e., what fidelity/latency can be achieved with the channel at hand, and the latter determines *how* to do it, i.e., what matters more or less to the user at hand. Of course, both channel quality and user preferences change with time.

Example with telephone: sound quality is reduced to meet the delay constraints, as well as reduce the costs.

Targeted reduction of information fidelity in a controlled manner helps meet the latency constraints and averts random loss of information. Common techniques for reducing information fidelity include:

- Lossless and lossy data compression
- Packet dropping (e.g., RED congestion-avoidance mechanism in TCP/IP, Section 5.3.1)
- …?

The above presentation is a simplification in order to introduce the problem. Note that there are many other relevant parameters, such as security, etc., that characterize the communicated information and will be considered in detail later.

Organizational concerns:

- Local traffic that originates at or terminates on nodes within an organization (or an autonomous system, Section 1.4.5)
- Transit traffic that passes through an autonomous system

3.1.2 Standards of Information Quality

In text, the entropy per character depends on how many values the character can assume. Because a continuous signal can assume an infinite number of different values at a sample point, we are led to assume that a continuous signal must have entropy of an infinite number of bits per sample. This would be true if we required absolutely accurate reproduction of the continuous signal. However, signals are transmitted to be heard, seen, or sensed. Only a certain degree of fidelity of reproduction is required. Thus, in dealing with the samples that specify continuous signals, Shannon introduces *fidelity criterion*. To reproduce the signal in a way meeting the fidelity criterion requires only a finite number of binary digits per sample per second, and hence we say that, within the accuracy imposed by a particular fidelity criterion, the entropy of a continuous source has a particular value in bits per sample or bits per second.

Standards of information quality help perform ordering of information bits by importance (to the user).

Man best handles information if encoded to his abilities. (Pierce, p.234)

In some cases, we can apply common sense in deciding user's servicing quality needs. For example, in applications such as voice and video, users are somewhat tolerable of information loss, but very sensitive to delays. Conversely, in file transfer or electronic mail applications, the users are expected to be intolerable to loss and tolerable to delays. Finally, there are applications where both delay and loss can be aggravating to the user, such as in the case of interactive graphics or interactive computing applications.

For video, expectations are low

For voice, ear is very sensitive to jitter and latencies, and loss/flicker

Voice communication requires a steady, predictable packet delivery rate in order to maintain quality. Jitter, which is variation in packet delivery timing, is the most common culprit that reduces call quality in Internet telephony systems. Jitter causes the audio stream to become broken, uneven or irregular. As a result, the listener's experience becomes unpleasant or intolerable.

The end results of packet loss are similar to those of jitter but are typically more severe when the rate of packet loss is high. Excessive latency can result in unnatural conversation flow where there is a delay between words that one speaks versus words that one hears. Latency can cause callers to talk over one another and can also result in echoes on the line. Hence, jitter, packet loss and latency can have dramatic consequences in maintaining normal and expected call quality.

Human users are not the only recipients of information. For example, network management system (Section 9.5) exchanges signaling packets that may never reach human user. These packets normally receive preferential treatment at the intermediaries (routers), and this is particularly required during times of congestion or failure.

It is particularly important during periods of congestion that traffic flows with different requirements be differentiated for servicing treatments. For example, a router might transmit higher-priority packets ahead of lower-priority packets in the same queue. Or a router may maintain different queues for different packet priorities and provide preferential treatment to the higher priority queues.

User Studies

User studies uncover the degree of service degradation that the user is capable of tolerating without significant impact on task-performance efficiency. A user may be *willing* to tolerate inadequate QoS, but that does not assure that he or she will be *able* to perform the task adequately.

Psychophysical and cognitive studies reveal population levels, not individual differences. Context also plays a significant role in user's performance.

The human senses seem to perceive the world in a roughly logarithmic way. The eye, for example, cannot distinguish more than six degrees of brightness; but the actual range of physical brightness covered by those six degrees is a factor of 2.5 × 2.5 × 2.5 × 2.5 × 2.5 × 2.5, or about 100. A scale of a hundred steps is too fine for human perception. The ear, too, perceives approximately logarithmically. The physical intensity of sound, in terms of energy carried through the air, varies by a factor of a trillion (10^{12}) from the barely audible to the threshold of pain; but because neither the ear nor the brain can cope with so immense a gamut, they convert the unimaginable multiplicative factors into comprehensible additive scale. The ear, in other words, relays the physical intensity of the sound as logarithmic ratios of loudness. Thus a normal conversation may seem three times as loud as a whisper, whereas its measured intensity is actually 1,000 or 10^3 times greater.

Fechner's law in psychophysics stipulates that the magnitude of sensation—brightness, warmth, weight, electrical shock, any sensation at all—is proportional to the logarithm of the intensity of the stimulus, measured as a multiple of the smallest perceptible stimulus. Note that this way the stimulus is characterized by a pure number, instead of a number endowed with units, like seven pounds, or five volts, or 20 degrees Celsius. By removing the dependence on specific units, we have a general law that applies to stimuli of different kinds. Beginning in the 1950s, serious departures from Fechner's law began to be reported, and today it is regarded more as a historical curiosity than as a rigorous rule. But even so, it remains important approximation …

Define j.n.d. (just noticeable difference)

Studies of telephone conversations subject to various round-trip delays have shown that people do not notice delays of less than approximately 300 milliseconds (ms). Note that this number is approximate and depends on the listener and the task being performed. ITU-T Rec. G.114 (05/2003) recommends that, regardless of the type of application, a one-way delay not exceed 400 ms for general network planning. Delay does not affect the intelligibility but rather the character of a conversation. This is because delay between the moment a person speaks and the other person hears the speech impairs the turn-taking perception of both persons.

For the voice or video application to be of an acceptable quality, the network must transmit the bit stream with a short delay and corrupt at most a small fraction of the bits (i.e., the BER must be small). The maximum acceptable BER is about 10^{-4} for audio and video transmission, in the absence of compression. When an audio and video signal is compressed, however, an error in the compressed signal will cause a sequence of errors in the uncompressed signal. Therefore, the tolerable BER is much less than 10^{-4} for transmission of compressed signals.

The end-to-end delay should be less than 200 ms for real-time video and voice conversations, because people find larger delay uncomfortable. That delay can be a few seconds for non-real-time interactive applications such as interactive video and information on demand. The delay is not critical for non-interactive applications such as streaming of stored video or audio programs, or television broadcast.

Typical acceptable values of delays are a few seconds for interactive services (e.g., Web browsing), and many seconds for non-interactive services such as email. The acceptable fraction of messages that can be corrupted ranges from 10^{-8} for data transmissions to much larger values for noncritical applications such as junk mail distribution.

Among applications that exchange sequences of messages, we can distinguish those applications that expect the messages to reach the destination in the correct order and those that do not care about the order.

3.1.3 User Models

User Preferences

User Utility Functions

Example: Augmented Reality

{PROBLEM STATEMENT}

Inaccuracy and delays on the alignment of computer graphics and the real world are one of the greatest constrains in scene registration for augmented reality. Even with current motion tracking algorithms, it is still necessary to use software to minimize misalignments of virtual and real objects.

{CONSTRAINS}

Augmented reality scene-registration systems are constrained by perception issues in the human vision system.

An important parameter of continuous signals is the acceptable frame rate. For virtual reality applications, it has been found that the acceptable frame rate is 20 frames per second (fps), with periodical variations of up to 40% [Watson, et al., 1997], and maximum delays of 10 milliseconds [Azuma, 1995]. The perception of misalignment by the human eye is also restrictive. Azuma found experimentally that it is about 2-3 mm of error at the length of the arm (with an arm length of about 70 cm) is acceptable [Azuma, 1995]. However, the human eye can detect even smaller differences as of one minute of arc [Doenges, 1985]. Current (2006) commercially available head-mounted displays used for augmented reality cannot provide more than 800 × 600 pixels, this resolution makes impossible to provide an accuracy of one minute of arc.

{SOURCES OF ERROR}

Errors can be classified as static and dynamic. Static errors are intrinsic to the scene-registration system and are present even if there is no movement of the head or tracked features.

Most important static errors are optical distortions and mechanical misalignments on head-mounted displays, errors in the tracking devices (magnetic, differential, optical trackers), incorrect viewing parameters as field of view, tracker-to-eye position and orientation. If computer vision is used to track moving objects, the optical distortion of the camera also has to be added to the error model.

Dynamic errors are caused by delays in the scene-registration system. If information is transmitted over the network, dynamic changes of throughput and latencies become an additional source of error.

{OUR AUGMENTED REALITY SYSTEM}

Research projects have addressed some solutions for scene registration involving predictive tracking [Azuma, 1995] [Chai, et al., 1999]. It is necessary to have accurate registration most of its usage however it is created for task where there is limited movement of the user, as in mechanical repair tasks. Network delays should be added to the model if processing is performed on a different machine. Also there is the necessity of having a user interface that can adapt to registration changes or according to the user's task, for example removing or adding information only when necessary to avoid occluding the view of the augmented reality user.

{PROPOSED SOLUTION}

We mention two approaches: predictive scene registration and adaptive user interfaces. Predictive registration allows saving processing time, or in case of a networked system, it can provide better registration in presence of latency and jitter. With predictive registration, delays as long as 80ms can be tolerated [Azuma & Bishop, 1994]. A statistical model of Kalman filters and extended Kalman filters can be used to optimize the response of the system when multiple tracking inputs as video and inertial trackers are used [Chai, et al., 1999].

Adaptive user interfaces can be used to improve the view of the augmented world. This approach essentially takes information form the tracking system to determine how the graphics can be gracefully degraded to match the real world. Estimation of the errors was used before to get and approximated shape of the 3D objects being displayed [MacIntyre, et al., 2000]. In addition, some user interface techniques based on heuristics where used to switch different representations of the augmented world [Höllerer, et al., 2001]. The first technique has a strict model to get an approximated augmented-reality view but it degrades the quality of the graphics, particularly affecting 3D models. The second technique degrades more gracefully but the heuristics used are not effective for all the augmented reality systems. A combination would be desirable.

{TRACKING PIPELINE}

This is a primary description of our current registration pipeline

Image Processing

[Frame capture] => [Image threshold] => [Subsampling] => [Features Finding] =>

[Image undistortion] => [3D Tracking information] => [Notify Display]

Video Display

[Get processed frame] => [Frame rendering in a buffer] => [3D graphics added to Buffer]

=> [Double buffering] => [Display]

These processes are executed by two separated threads for better performance and resource usage.

{REFERENCES}

[Azuma 94]

R. Azuma and G. Bishop, “Improving Static and Dynamic Registration in an Optical See-Through HMD,” Proceedings of SIGGRAPH '94 (Orlando, FL, 24-29 July 1994), Computer Graphics, Annual Conference Series, 1994, 197-204. http://www.cs.unc.edu/~azuma/sig94paper.pdf

[Azuma 95]

R. Azuma, "Predictive Tracking for Augmented Reality," UNC Chapel Hill Dept. of Computer Science Technical Report TR95-007 (February 1995), 262 pages. http://www.cs.unc.edu/~azuma/dissertation.pdf

[Chai 99]

L. Chai, K. Nguyen, W. Hoff, and T. Vincent, "An adaptive estimator for registration in augmented reality," Proc. of 2nd IEEE/ACM Int'l Workshop on Augmented Reality, San Franscisco, Oct. 20-21, 1999. http://egweb.mines.edu/whoff/projects/augmented/iwar1999.pdf

[Doenges 85]

P. K. Doenges, "Overview of Computer Image Generation in Visual Simulation," SIGGRAPH '85 Course Notes #14 on High Performance Image Generation Systems, San Francisco, CA, 22 July 1985.

[Hollerer 01]

T. Höllerer, D. Hallaway, N. Tinna, S. Feiner, "Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system," In Proc. 2nd Int. Workshop on Artificial Intelligence in Mobile Systems (AIMS '01), pages 31-37, 2001. http://monet.cs.columbia.edu/publications/hollerer-2001-aims.pdf

[MacIntyre 00]

B. MacIntyre, E. Coelho, S. Julier, "Estimating and Adapting to Registration Errors in Augmented Reality Systems," In IEEE Virtual Reality Conference 2002 (VR 2002), pp. 73-80, Orlando, Florida, March 24-28, 2002. http://www.cc.gatech.edu/people/home/machado/papers/vr2002.pdf

[Watson 97]

Watson, B., Spaulding, V., Walker, N., Ribarsky W., "Evaluation of the effects of frame time variation on VR task performance," IEEE VRAIS'96, 1996, pp. 38-52. http://www.cs.northwestern.edu/~watsonb/school/docs/vr97.pdf

3.1.4 Performance Metrics and Network Service Models

Delay (the average time needed for a packet to travel from source to destination), statistics of delay (variation, jitter), packet loss (fraction of packets lost, or delivered so late that they are considered lost) during transmission, packet error rate (fraction of packets delivered in error).

The round-trip time is important in interactive applications because delay hinders interactivity (Section 3.1.2).

An example calculation is the packet loss rate over the interval between two observations. The difference in the cumulative number of packets lost yields the number lost during that interval. The difference in the last received sequence numbers gives the number of packets expected during the interval. The ratio of these two is the packet loss fraction over the interval. This ratio should equal the fraction lost field if the two observations are consecutive, but otherwise it may not. The loss rate per second can be obtained by dividing the loss fraction by the difference in

NTP timestamps, expressed in seconds. The number of packets received is the number of packets expected minus the number lost. The number of packets expected may also be used to judge the statistical validity of any loss estimates. For example, 1 out of 5 packets lost has a lower significance than 200 out of 1000.

The interarrival jitter provides a second short-term measure of network congestion. Packet loss tracks persistent congestion while the jitter measure tracks transient congestion. The jitter measure may indicate congestion before it leads to packet loss.

Bounded delay packet delivery ratio (BDPDR): Ratio of packets forwarded between a mobile node and an access point that are successfully delivered within some pre-specified delay constraint. The delay measurement starts at the time the packet is initially queued for transmission (at the access point for downstream traffic or at the originating node for upstream traffic) and ends when it is delivered successfully at either the mobile node destination (downstream traffic) or AP (upstream traffic).

Quality of Service

QoS, Keshav p.154

Cite Ray Chauduhuri's W-ATM paper {cited in Goodman}

Quality of Service (QoS)

Performance measures

Throughput

Latency

Real-time guarantees

Other factors

Reliability

Availability

Security

Synchronization of data streams

Etc.

A networking professional may be able to specify what quality-of-service metrics are needed, and can specify latency, packet loss and other technical requirements. However, the consumer or independent small-office-home-office (SOHO) user would more easily understand service classifications such as "High-Definition Movie Tier" or an "Online Gamer Tier." Few consumers will be able to specify service-level agreements, but they may want to know if they are getting better services when they pay for them, so a consumer-friendly reporting tool would be needed. In addition, although enterprises are increasingly likely to buy or use a premise-based session border controller to better manage IP traffic, service providers will need to come up with an easier and less expensive alternative to classify consumer IP packets based on parameters such as user profiles and service classes.

3.2 Source Characteristics and Traffic Models

Different media sources have different traffic characteristics.

3.2.1 Traffic Descriptors

Some commonly used traffic descriptors include peak rate and average rate of a traffic source.

Average Rate

The average rate parameter specifies the average number of packets that a particular flow is allowed to send per unit of time. A key issue here is to decide the interval of time over which the average rate is calculated and then will be regulated for compliance. If the interval is longer, the flow can generate much greater number of packets over a short period than if the interval is short. In other words, a shorter averaging interval imposes greater constraints. For example, average of 100 packets per second is different from an average of 6,000 packets per minute, because in the latter case the flow is allowed to generate all 6,000 over one 1-second interval and remain silent for the remaining 59 seconds.

Researchers have proposed two types of average rate definitions. Both […]

Burst size: this parameter constrains the total number of packets (the "burst" of packets) that can be sent by a particular flow into the network over a short interval of time.

Peak Rate

The peak rate is the highest rate at which a source can generate data during a communication session. Of course, the highest data rate from a source is constrained by the data rate of its outgoing link. However, by this definition even a source that generates very few packets on a 100-Mbps Ethernet would be said to have a peak rate of 100 Mbps. Obviously, this definition does not reflect the true traffic load generated by a source. Instead, we define the **peak rate** as the maximum number of packets that a source can generate over a very short period of time. In the above example, one may specify that a flow be constrained to an average rate of 6,000 packets/minute and a peak rate of 100 packets/second.

Primitive traffic characterization is given by the source entropy.

See also MobiCom'04, p. 174: flow characterization

For example, image transport is often modeled as a two state ON-OFF process. While ON, a source transmits at a constant rate. Telephone conversation can also be modeled as an ON-OFF process if "silence suppression" is employed (ON for talk and OFF for silence). For more complex media sources such as variable bit rate (VBR) video coding algorithms, more states are often used to model the video source. The state transitions are often assumed Markovian, but it is well known that non-Markovian state transitions could also be well represented with several Markovian states. Therefore, it is common to adopt a general Markovian structure, for which a deterministic traffic rate is assigned for each state. This is the well-known Markovian fluid flow model [Anick *et al*. 1982], where larger communication entities, such as an image or a video frame, is in a sense "fluidized" into a fairly smooth flow of very small information entities called "cells." Under this fluid assumption, let $X_i(t)$ be the rate of cell emission for a connection i at the time t, for which this rate is determined by the state of the source at time t.

The most common modeling context is queuing, where traffic is offered to a queue or a network of queues and various performance measures are calculated. More about queuing in Section 4.2.

Simple traffic consists of single arrivals of discrete entities (packets, frames, etc.). It can be mathematically described as a point process, consisting of a sequence of arrival instants T_1, T_2, …, T_n, … measured from the origin 0; by convention, $T_0 = 0$. There are two additional equivalent descriptions of point processes: *counting processes* and *interarrival time processes*. A counting process $\{N(t)\}_{t=0}^{\infty}$ is a continuous-time, non-negative integer-valued stochastic process, where $N(t) = \max\{n : T_n \le t\}$ is the number of (traffic) arrivals in the interval (0, t]. An interarrival time process is a non-negative random sequence $\{A_n\}_{n=1}^{\infty}$, where $A_n = T_n - T_{n-1}$ is the length of the time interval separating the n-th arrival from the previous one. The equivalence of these descriptions follows from the equality of events:

$$\{N(t) = n\} = \{T_n \le t < T_{n+1}\} = \left\{\sum_{k=1}^{n} A_k \le t < \sum_{k=1}^{n+1} A_k\right\}$$

because $T_n = \sum_{k=1}^{n} A_k$. Unless otherwise stated, we assume throughout that $\{A_n\}$ is a stationary sequence and that the common variance of the A_n is finite.

3.2.2 Self-Similar Traffic

3.3 Approaches to Quality-of-Service

This section reviews several approaches and architectures for providing end-to-end quality-of-service (QoS), and hints at the mechanisms used in routers. Some approaches rely on network-based mechanisms, where the network elements are designed to provide improved quality of service. Other approaches assume that the network offers the regular "best-effort" service, and implement quality-of-service mechanisms in the endpoints. The first approach (Section 3.3.1) is endpoint-based and relies on having the receiver to pick the time to play out the content, known as the "playout point." By adjusting the playout point, the receiver can conceal the undesirable network effects such as packet reordering and variable delays. Another endpoint-based approach is peer-to-peer (Section 3.3.3). Peer-to-peer connects many end-to-end connections into an "overlay network," which then "routes" packets between the endpoints. The overlay network is an abstraction implemented on top of a physical network, which provides the physical connectivity between the endpoint nodes.

The remaining three solutions are network-based and require network mechanisms to support improved quality of service. The multicast architecture (Section 3.3.2) relies on the idea that large broadcasts, such as radio and television shows, will be a driver for network applications.

The last two network architectures (Sections 3.3.4 and 3.3.5) rely on network support for quality of service. Differentiated services (DiffServ, Section 3.3.4) and Integrated services (IntServ, Section 3.3.5) approach this problem differently. DiffServ supports priority treatment at individual routers for specially marked packets. In other words, it offers different "per-hop-behaviors," where the marked packets receive preferential treatment on each hop of the end-to-end path. However, DiffServ does not guarantee that end-to-end delays or loss rates of packet will remain within firm bounds.

IntServ relies on making advance reservation of network resources along the path that the future data packets will travel, and admission control to prevent resource overbooking. In this way, IntServ can guarantee that end-to-end delays or loss rates of packet will remain within firm bounds. Although IntServ offers hard guarantees, it is also more complex to implement and deploy, and for this reason it is much less widespread in practice than DffServ. Chapter 5 details how routers implement QoS mechanisms to support network-based QoS solutions (IntServ and DiffServ).

3.3.1 Removing Jitter by Adjusting the Playout Point

Problems related to this section: Problem 3.3 → Problem 3.5

Consider the example shown in Figure 3-4 where a source sends audio signal to a receiver for playout. Let us assume that the source segments the speech stream every 20 milliseconds and creates data packets. The source outputs packets with uniform time spacing between them, but they arrive at the receiver at irregular times due to random network delays for individual packets.

Speech packetization at the intervals of 20 ms seems to be a good compromise. If the interval were longer, flicker due to lost or late packets would be more noticeable; conversely, if the interval were shorter, the packet-header overhead would be too high, with the header size possibly exceeding the speech-segment payload.

As Figure 3-4 shows, the network introduces random delays for different packets, and they arrive at the receiver not uniformly spaced as they left the source. Playing out the speech packets as they

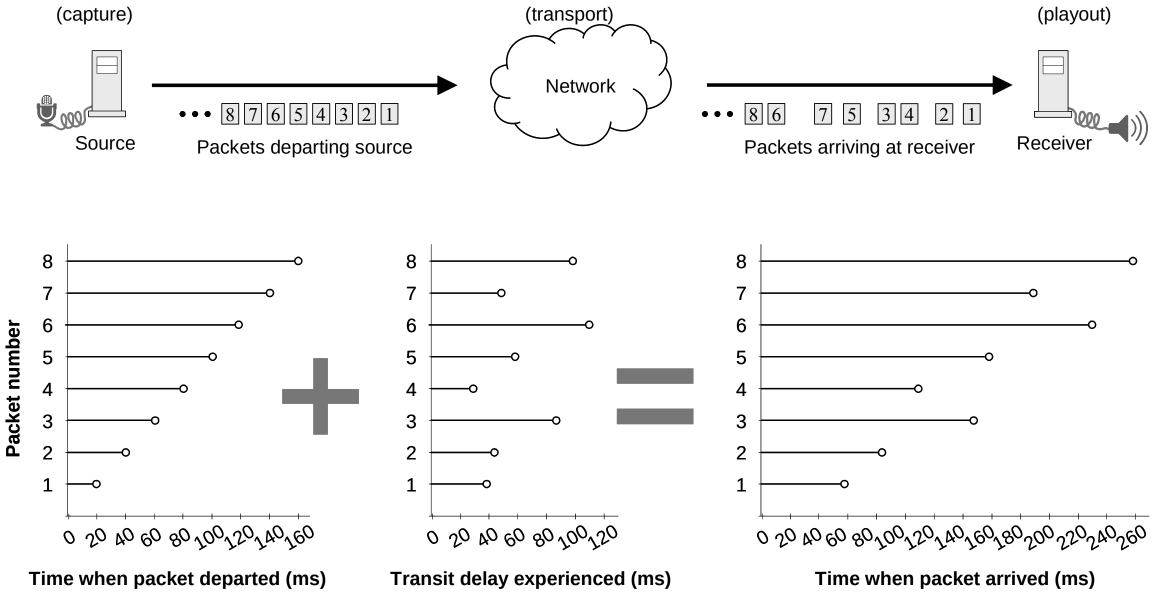


Figure 3-4: Packets depart with a uniform spacing, but they experience variable amount of delay (jitter) and arrive at the receiver irregularly (packets #3 and #6 arrive out of order).

arrive (with random delays) would create significant distortions and impair the user experience. One way to deal with this is to buffer the packets at the receiving host to smooth out the jitter. Packets are buffered for variable amounts of time in an attempt to play out each speech segment with a constant amount of delay relative to the time when it was pacektized and transmitted from the source. We introduce the following notation (Figure 3-5):

t_i = the time when the i^{th} packet departed its source

d_i = the amount of delay experienced by the i^{th} packet while in transit across the network

r_i = the time when the i^{th} packet is received by the receiver (note that $r_i = t_i + d_i$)

p_i = the time when the i^{th} packet is played at the receiver

To smoothen out the playout times, we introduce a constant delay q between packet transmission from the source and its playout at the receiver (see Figure 3-5). Then $p_i = t_i + q$. The time difference between the i^{th} packet's playout time and the time it is received equals $\Delta_i = (t_i + q) - r_i$. If $\Delta_i \geq 0$, the i^{th} packet should be buffered for Δ_i duration before it is played out. If $\Delta_i < 0$, the i^{th} packet should be discarded because it arrived too late for playout. Figure 3-5 illustrates jitter removal for the example given in Figure 3-4. In this case, the constant playout delay of q = 100 ms is selected. With this choice, the sixth packet does not arrive by its scheduled playout time, and the receiver considers it lost.

We could try selecting a large q so that all packets will arrive by their scheduled playout time. However, for applications such as Internet telephony, delays greater than 400 ms are not acceptable because of human psychophysical constraints (Section 3.1). Ideally, we would like to keep the playout delay at less than 150 ms. Larger delays become annoying and it is difficult to maintain a meaningful conversation. We know from the discussion in Section 2.1.3 that average end-to-end network delays can change significantly during day or even during short periods. Therefore, the best strategy is to adjust the playout delay adaptively, so that we select the

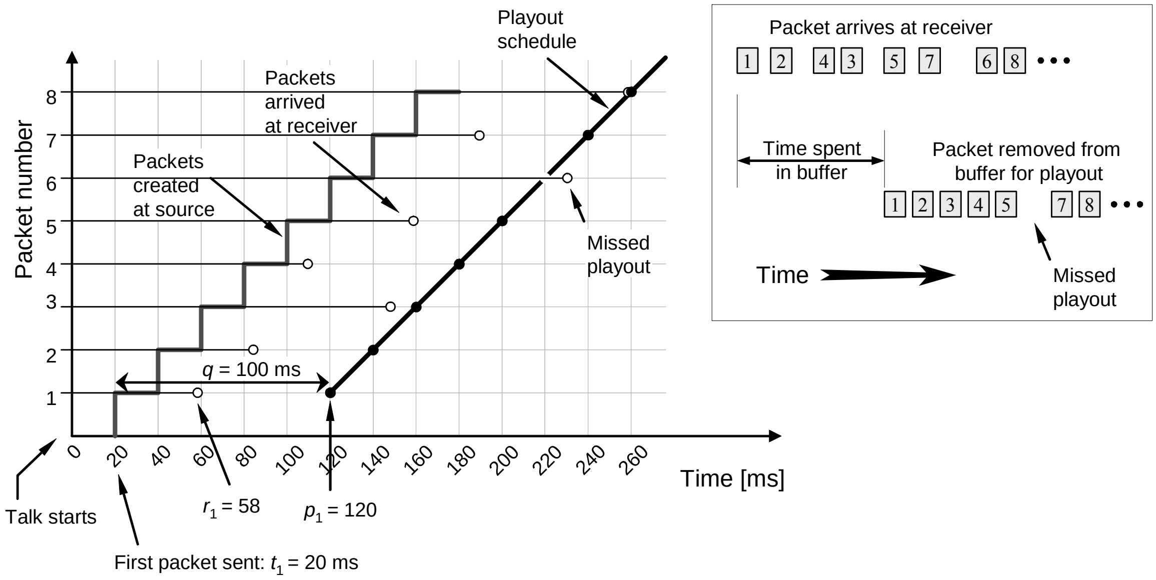


Figure 3-5: Removing jitter at receiver by adjusting the playout point.

minimum possible delay for which the fraction of missed playouts is kept below a given threshold.

We can again use the approach described in Section 2.1.3 and estimate the average end-to-end network delay using *Exponential Weighted Moving Average* (EWMA). Similar to Eq. (2.2), we estimate the average network delay $\hat{\delta}_i$ upon reception of the i^{th} packet as

$$\hat{\delta}_i = (1-\alpha)\cdot\hat{\delta}_{i-1} + \alpha\cdot(r_i - t_i) \tag{3.1}$$

where α is a fixed constant, say, $\alpha = 0.001$. We also estimate the average standard deviation $\hat{\upsilon}_i$ of the delay as

$$\hat{\upsilon}_i = (1-\alpha)\cdot\hat{\upsilon}_{i-1} + \alpha\cdot|\, r_i - t_i - \hat{\delta}_i \,| \tag{3.2}$$

Note that the playout delay q is measured relative to packet's departure time (Figure 3-5). Therefore, we cannot adjust q for each packet individually, because this would still result in a distorted speech. An option is to set the playout delay constant for an interval of time, but the question is when this interval should start and how long it should last. It turns out that humans do not notice if the periods of silence between the utterances are stretched or compressed. This fact is used to adjust the playout delay adaptively: the playout delay q is adjusted only at the start of an utterance (or, "talk spurt") and it is maintained constant until the next period of silence. The receiver maintains average delay $\hat{\delta}_i$ and average standard deviation $\hat{\upsilon}_i$ for each received packet. During a period of silence, the receiver calculates the playout delay for the subsequent talk spurt as follows. If packet k is the first packet of the next talk spurt, k^{th}'s playout delay is computed as

$$q_k = \hat{\delta}_k + K\cdot\hat{\upsilon}_k \tag{3.3}$$

where K is a positive constant, for example we can set $K = 4$ following the same reasoning as in Section 2.1.3 for Eq. (2.1). Then, the playout time of the k^{th} packet and all the remaining packets of the next spurt is computed as $p_i = t_i + q_k$.

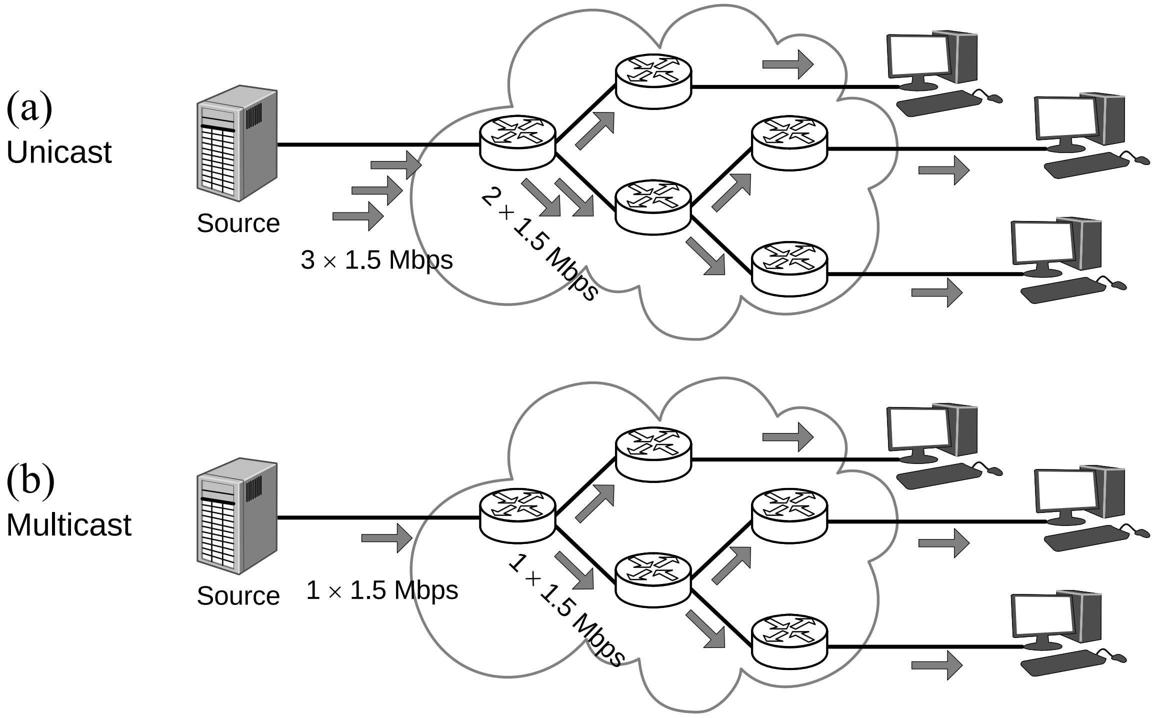


Figure 3-6: Unicast vs. multicast routing.

3.3.2 Multicast Routing

Problems related to this section: Problem 3.7 → ??

When multiple receivers are required to get the same data at approximately the same time, multicast routing is a more efficient way of delivering data than unicast. A unicast packet has a single source IP address and a single destination IP address. Data are delivered to a single host. A multicast packet has a single IP source, but it has a multicast destination IP address that will be delivered to a group of receivers. (Recall the multicast address class D in Figure 1-49.) The advantage is that multiple hosts can receive the same multicast stream (instead of several individual streams), thereby saving network bandwidth. In general, the bandwidth saving with multicast routing becomes more substantial as the number of destinations increases.

Figure 3-6 shows an example where three users are simultaneously watching the same video that is streamed from the same media server (“source”). In Figure 3-6(a), all users receive their individual streams via a unicast delivery. As illustrated, the source must send a total of 3 unicast streams, each targeted to a different user. Obviously, this is a waste of bandwidth. If the compressed video takes approximately 1.5 Mbps of bandwidth per stream, then the first link must support data rate of at least 3 × 1.5 = 4.5 Mbps. Similarly, the lower link from the first router relays two streams which consume 2 × 1.5 = 3 Mbps of bandwidth. If, on the other hand, the network supports multicast routing as in Figure 3-6(b), the source sends only a single stream and all links would be using 1.5 Mbps for this video. Of course, this kind of resource saving is possible only if multiple users simultaneously need to receive the same video from the same source.

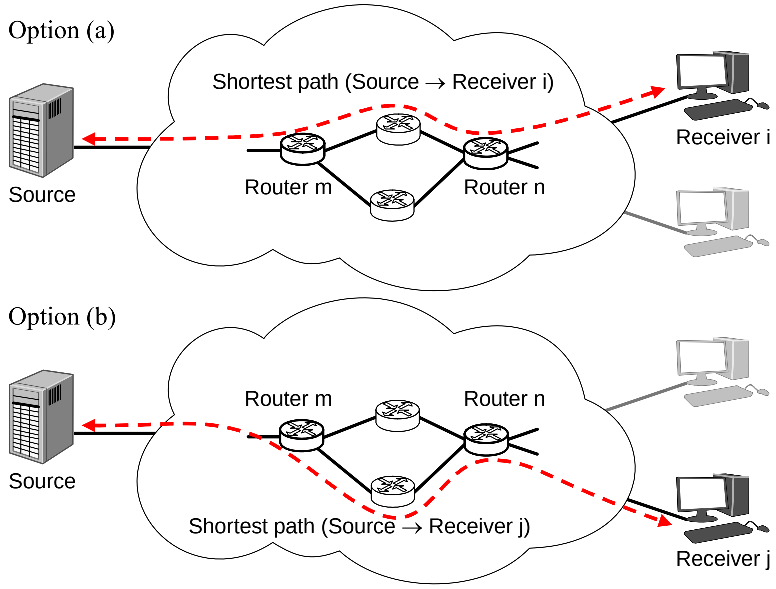


Figure 3-7: The superimposed shortest paths must form a tree rooted in the source.

There are two key issues for multicast routing protocols:

1. **Multicast Group Management:** identifying which hosts are members of a multicast group and supporting dynamic changes in the group membership; multiple sources and multiple receivers may need to be supported
2. **Multicast Route Establishment:** setting up the (shortest path) route from each source to each receiver

A *multicast group* mutually relates a set of sources and receivers, but conceptually exists independently of them. That is, the group continues to exist even if some nodes leave the group or new nodes join it. Such a group is identified by a unique IP multicast address of Class D (Figure 1-49). It is created either when a source starts sending to the group address (even if no receivers are present) or when a receiver expresses its interest in receiving packets from the group (even if no source is currently active).

To establish the multicasting routes, we start by superimposing all the shortest paths connecting the source with all the receivers. The result will be a tree, i.e., it cannot be a graph with cycles. To see why, consider a contrary possibility, illustrated in Figure 3-7, where the shortest paths from the source to two receivers *i* and *j* share two intermediary nodes (routers *m* and *n*), but not the nodes between these two nodes. We know from Section 1.4 that the shortest path between two intermediate points does not depend on where this path extends beyond those points. In other words, it does not depend on the endpoints. Hence, if we superimpose all the shortest paths from the source host to any destination, we will obtain a tree structure, for which the source host is the root node. (Note that alternative paths between *m* and *n* could be equally long, but there should be a uniform policy to resolve the tied cases.) The next issue is, how the multicast-capable routers should construct this tree.

Reverse Path Forwarding (RPF) Algorithm

We assume that all routers in the network are running a unicast routing protocol (described in Section 1.4) and maintain unicast routing tables independently of the multicast protocol. Thus, the routers already know either the shortest unicast paths to all nodes in the network, or at least the next hop on the shortest path to any other node in the network. Multicast routing algorithms then build on top of the unicast routing infrastructure. One such algorithm is reverse path forwarding.

In **reverse path forwarding (RPF)** algorithm, when a router receives a packet, it forwards the packet to all outgoing links (except the one on which it was received) *only if* the packet arrived on the link that is on this router's shortest unicast path back to the source. Otherwise, the router simply discards the incoming packet without forwarding it on any of its outgoing links. (A tie between two routers is broken by selecting the router with the smallest network address.)

A problem with RPF is that it essentially *floods* every router in the network, regardless of whether it has hosts attached to it that are interested in receiving packets from the multicast group. To avoid these unnecessary transmissions, we perform **pruning**: when a router no longer has attached hosts interested in receiving multicast packets from a particular source, it informs the next-hop router on the shortest path to the source that it is not interested.

Here is an example:

Example 3.1 Multicast Using Reverse Path Forwarding (RPF) Algorithm

Consider the network shown in Figure 3-8, in which a radio broadcast source *A* distributes a radio program to a multicast group *Γ*, whose members are all the shown hosts. Assume that all link costs are equal to 1, including the Ethernet links, i.e., any two adjacent nodes that are separated by one hop.

(a) Draw the shortest path multicast tree for the group *Γ*.

(b) Assuming that the reverse path forwarding (RPF) algorithm is used, how many packets are forwarded in the entire network per every packet sent by the source *A*? To avoid ambiguities, describe how you counted the packets.

(c) Assuming that the RPF algorithm uses pruning to exclude the networks that do not have hosts that are members of *Γ*, how many packets are forwarded in the entire network per every packet sent by the source *A*?

The solutions for (a) and (b) are shown in Figure 3-9. The shortest path multicast tree is drawn by thick lines in Figure 3-9(a). Router R3 is two hops from R2 (the root of the tree) both via R1 and via R4. The tie is resolved and router R1 is selected because it has a smaller address than R4. (Therefore, the shortest path from R3 to R2 is R3→R1→R2 and the link R4–R3 is not part of the tree.) Note that router R6 is not connected to R3 (via the multihomed host *E*), because multihomed hosts do *not* participate in routing or forwarding of transit traffic.

The source router R2 sends packets to routers R1, R4, and R5, which in turn forward them to all outgoing links (except the one on which it was received) *only if* the packet arrived on the link that is on this router's shortest unicast path back to the source. Otherwise, the router simply discards the incoming packet without forwarding it. In Figure 3-9(b), R3 will receive the packets from both R1 and R4, but it will forward only the one from R1 and discard the one from R4. The way we count packets is how many packets leave the router, and the router has to forward a different packet on each different outgoing link. Therefore, the number of forwarded packets in the entire network is 8 per each source packet. If we include the six packets forwarded to end hosts, the total is 8 + 6 = 14 (shown in Figure 3-9(b)).

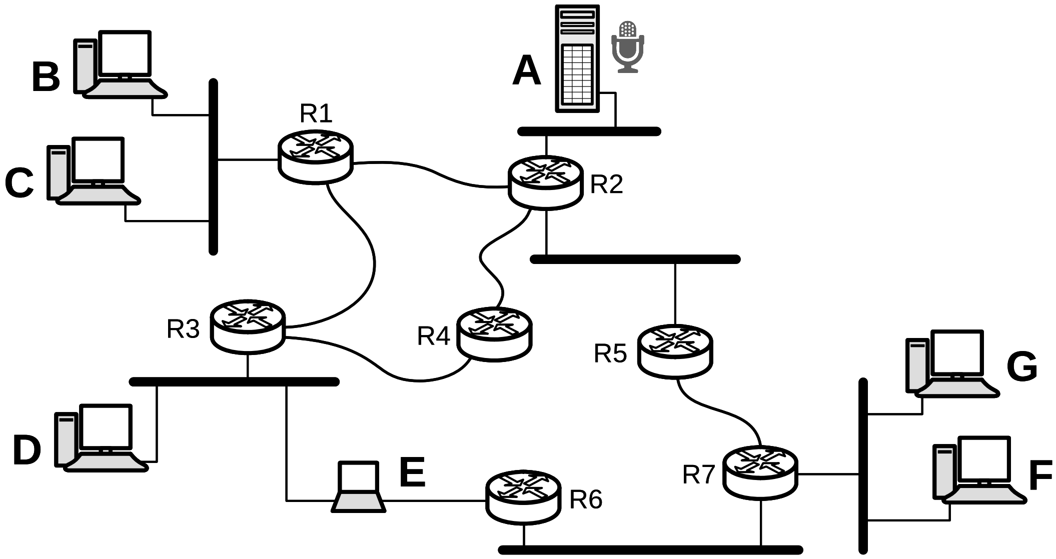


Figure 3-8: Example network used in the multicast Example 3.1.

> As for part (c), routers R4 and R6 do not have any host for which either one is on the shortest path to the source *A*. The shortest path for host *E* is via R3–R1–R2, selected by the routing algorithm as explained above. Therefore, R4 and R6 should send a *prune* message to R2 and R7, respectively, to be removed from the multicast tree. This reduces the number of forwarded packets by 4 (remove p_1'', p_2, p_2', p_3'' in Figure 3-9(b)), so the total number is 4 per each sent packet, or 4 + 6 = 10, if the end hosts are counted.

What if a router is pruned earlier in the session, but later it discovers that some of its hosts wish to receive packets from that multicast group? One option is that the router explicitly sends a **graft message** to the next-hop router on the shortest path to the source. Another option is for the source(s) and other downstream routers to flood packets periodically from the source in search for receivers that may wish to join the group later in the session. This extended version of RPF is called **flood-and-prune** approach to multicast-tree management.

A key property of RPF is that routing loops are automatically suppressed and each packet is forwarded by a router exactly once. The basic assumption underlying RPF is that the shortest path is symmetric in both directions. That is, the shortest path from the source to a given router contains the same links as the shortest path from this router back to the source. This assumption requires that each link is symmetric (roughly, that each direction of the link has the same cost). If links are not symmetric, then the router must compute the shortest path from the source to itself, given the information from its unicast routing tables. Note that this is possible only if a link-state protocol (Section 1.4.2) is used as the unicast routing algorithm.

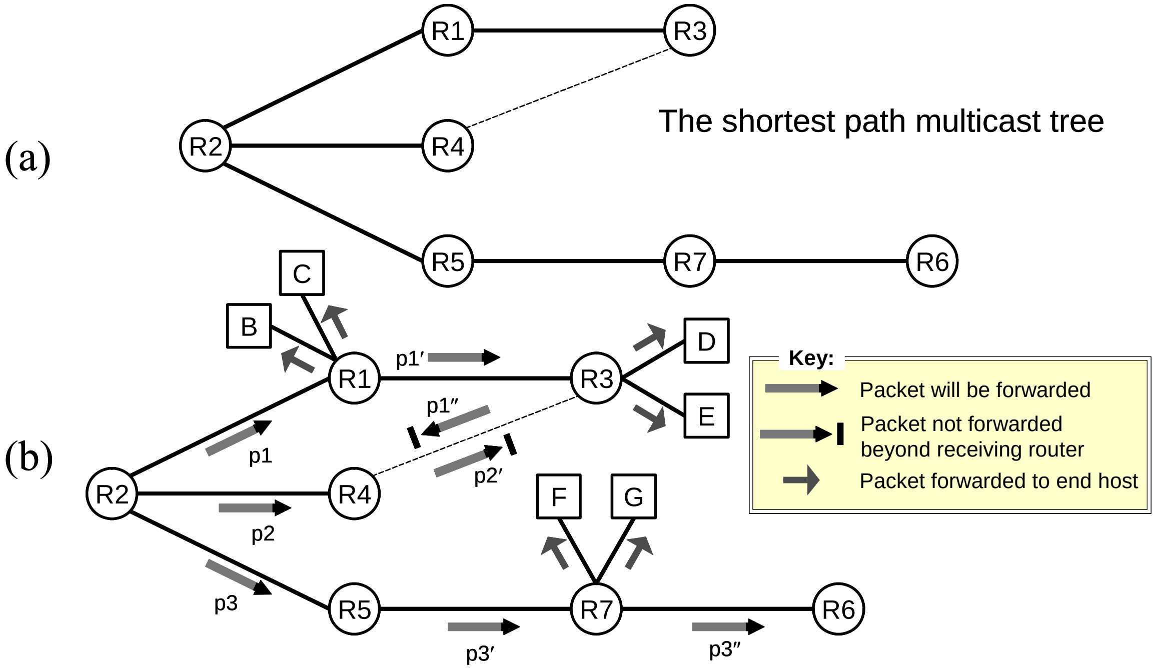

(a)
R1
R3
R2
R4
The shortest path multicast tree
R5
R7
R6
(b)
C
B
R1
p1′
R3
D
E
p1″
p1
p2′
R2
R4
p2
F
G
p3
R5
R7
R6
p3′
p3″
Key:
Packet will be forwarded
Packet not forwarded beyond receiving router
Packet forwarded to end host

Figure 3-9: The shortest path multicast tree for the example network in Figure 3-8.

Spanning Tree Algorithms

The reverse path forwarding algorithm, even with pruning, does not completely avoid transmission of redundant multicast packets. Consider the network in Figure 3-10(a), which is similar to Figure 3-8 but slightly more complex. The shortest-path multicast tree is shown in Figure 3-10(b). Router R4 can be pruned because it does not have attached hosts that are interested in multicast packets from the source *A*. (Router R5 is relaying packets for R6 and R7, so it stays.) As seen, routers R3, R5, R6, R7, and R8 will receive either one or two redundant packets. Ideally, every node should receive only a single copy of the multicast packet. This would be the case if the nodes were connected only by the thick lines in Figure 3-10(b). The reason is that the thick lines form a *tree* structure, so there are no multiple paths for the packet to reach the same node. A tree obtained by removing alternative paths, while keeping connected the nodes that were originally connected, is called a **spanning tree**. If a multicast packet were forwarded from the root of the tree to all other nodes, every node would receive exactly one copy of the packet. If links have associated costs and the total cost of the tree is the sum of its link costs, then the spanning tree that has the minimum total cost is called the **minimum spanning tree**.

Therefore, an alternative to RPF is to construct a spanning tree and have each source send the packets out on its incident link that belongs to the spanning tree. Any node that receives a multicast packet then forwards it to all of its neighbors in the spanning tree (except the one where the packet came from). Multicasting on a spanning tree requires a total of only $N - 1$ packet transmissions per packet multicast, where N is the number of nodes. Note that a single spanning tree is sufficient for any number of sources. This is true because any node of a tree can serve as its root. To convince yourself about this, take an arbitrary tree and select any of its nodes. Now imagine that you pull this node up and the other nodes remain hanging from the selected node. What you get is a tree rooted in the selected node.

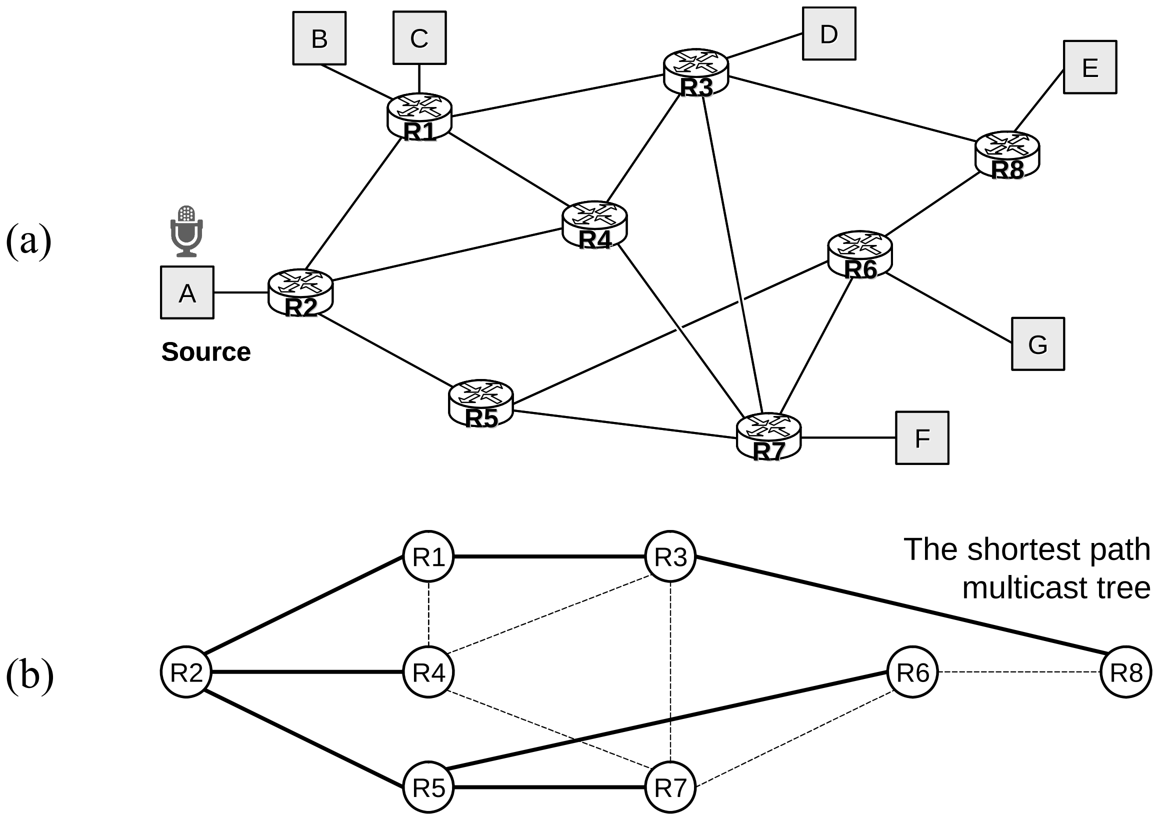


Figure 3-10: (a) Example network for multicast routing. (b) The shortest path multicast tree determined by RPF.

The main complexity of the spanning-tree multicasting lies in the creation and maintenance of the spanning tree, as sources and receivers dynamically join or leave the multicast group or the network topology changes. (Note that RPF does not have this problem, because it relies on flooding.) One algorithm that builds and maintains the spanning-tree efficiently is known as **core-based trees (CBTs)**. With CBTs, the spanning tree is formed starting from a "core router" (also known as a "rendezvous point" or a "center node"), which can be statically configured or automatically selected. Other routers are added by growing "branches" of the tree, consisting of a chain of routers, away from the core router out towards the routers directly adjoining the multicast group members. The core router is also known as a "center" and CBT is sometimes called *center-based tree*.

The tree building process starts when a host joins the multicast group. The host sends a join-request packet addressed to the core router. The information about the core router is statically configured. This join-request packet travels hop-by-hop towards the target core, forwarded using unicast routing. The process stops when the packet either arrives at an intermediate router that already belongs to the spanning tree or arrives at the destination (the core router). In either case, the path that the join-request packet has followed defines the branch of the spanning tree between the leaf node that originated the join-request and the core router. The node at which the message terminated confirms the packet by sending a join-acknowledgement message. The join-acknowledgement message travels in the opposite direction the same route the join-request message traveled earlier.

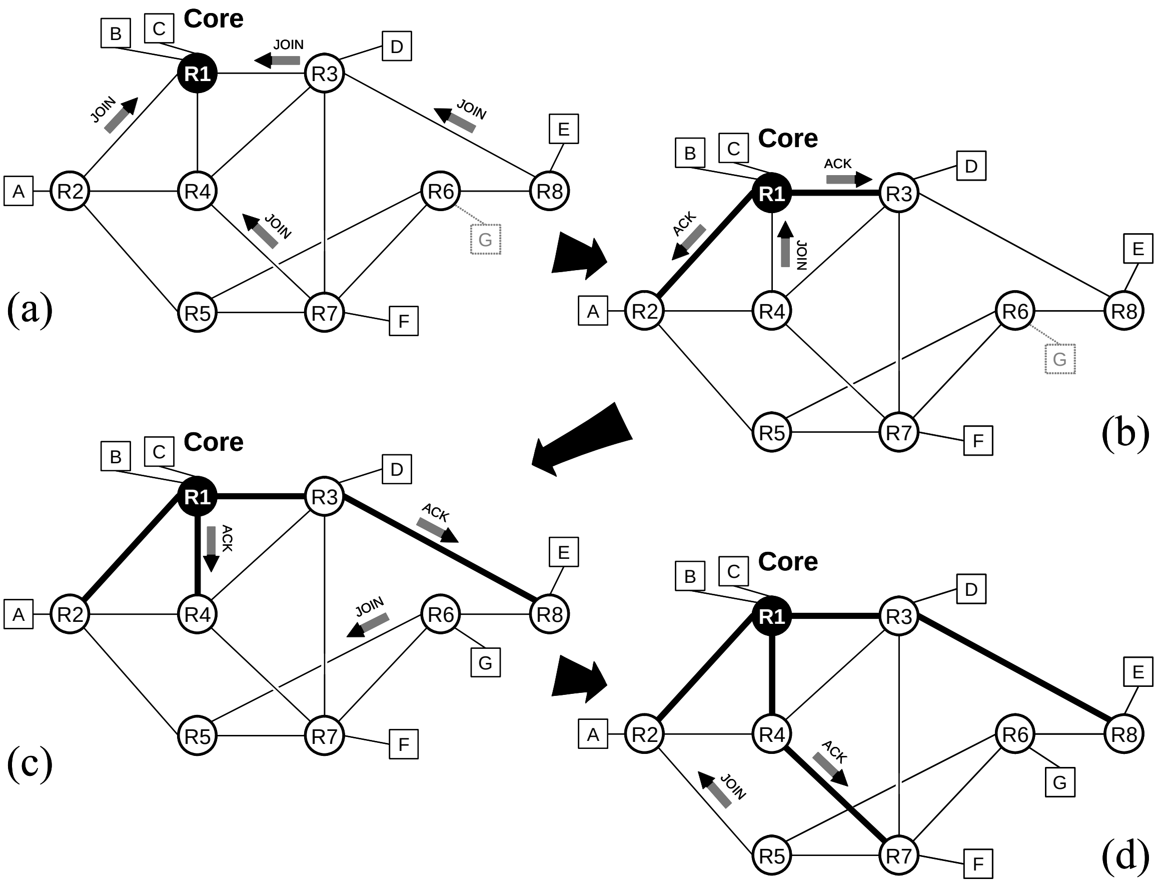


Figure 3-11: Forming the spanning tree by CBTs approach for the example in Figure 3-10. Thick lines represent the spanning tree, as its branches are grown.

Figure 3-11 illustrates the CBT process for the network in Figure 3-10(a), assuming that R1 is configured as the core router. Suppose that the source *A* and receivers *B*, *C*, *D*, *E*, and *F* join simultaneously. (Receiver *G* is shaded because it will join later.) Figure 3-11(a) shows how each router that has a group member attached, unicasts the join-request message to the next hop on the unicast path to the group's core. In Figure 3-11(b), the core R1 sends join-acknowledgement messages to R2 and R3, and R4 relays R7's join request. Note that R3 does not forward the join request by R8. This is because R3 has already sent its own join request. Subsequent join requests received for the same group are cached until this router has received a join acknowledgement for the previously sent join, at which time any cached joins can also be acknowledged. This happens in Figure 3-11(c), where after receiving a join acknowledgement, R3 in turn acknowledges R8's join. We assume that at this time receiver *G* decides to join the multicast group and R6 sends a join request on its behalf. There are three shortest paths from R6 to R1: paths R6-R5-R2-R1, R6-R7-R4-R1, and R6-R8-R3-R1; we assume that the tie was broken by the unicast routing algorithm and R6-R5-R2-R1 was selected. (The reader may observe that there were several other shortest-path ties, which again we assume were broken by the unicast algorithm.) Figure 3-11(d) shows that the branch from the core to R7 is established, and at the same time, the join request from R6 reaches R2. R2 will not propagate R6's join request because it is already on the spanning tree for the same group. Therefore, R2 will respond with a join acknowledgement, which will travel opposite the join request until it reaches R6 (not shown in Figure 3-11).

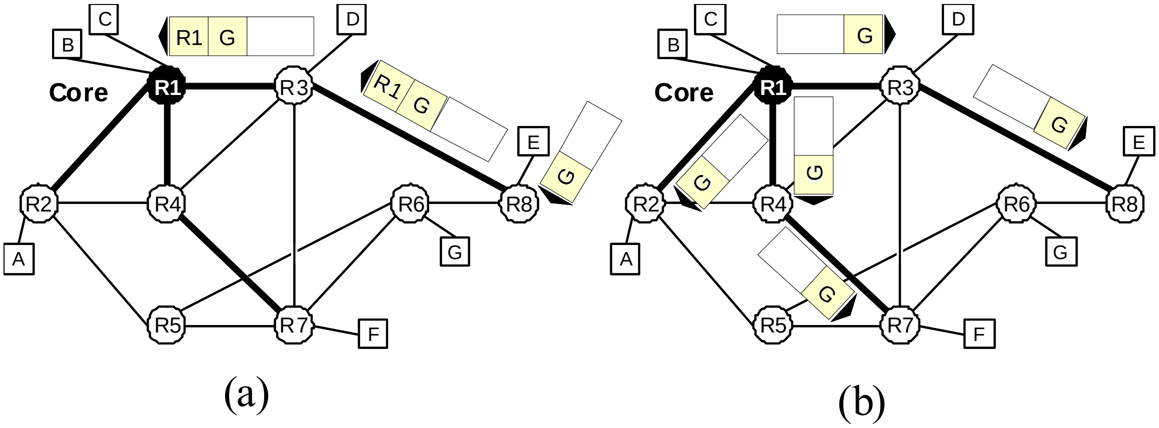


Figure 3-12: Packet forwarding from source *E* to the multicast group *G* in Figure 3-11. (a) A multicast packet from *E* is encapsulated by R8 in a unicast IP packet addressed to the core router R1, which is why *E*'s packet is shown with two IP headers. (b) The core R1 multicasts *E*'s packet to the multicast group.

The resulting spanning tree is known as a **group-shared multicast tree** because any multicast source in the multicast group Γ can use this tree. All routers that are part of the spanning tree create a forwarding table entry for the shared tree, called $\langle *, \Gamma\rangle$ entry, where the wildcard $*$ stands for "any source" (within the group Γ). The outgoing network port for this entry is the network interface on which the Join message arrived during the spanning tree construction. All data packets that arrive for group Γ will be forwarded out on this port. Each source first sends its traffic to the core router, which then multicasts it down the spanning tree. Consider the example in Figure 3-11 and assume that host *E* multicasts a message to the group. Host *E* constructs an IP packet using the destination IP address of group Γ (Figure 1-49). *E* sends the packet to a router on its local network known as the *designated router*, in our case R8. R8 encapsulates the packet into a unicast IP packet and sends it to the core R1 (Figure 3-12). When the packet reaches R1, the core removes the unicast IP header and forwards it down the tree. A version of the basic algorithm modified for performance optimization allows that packets destined for the group do not need to reach the core before they are multicast. As soon as a packet reaches the tree (at the "designated router"), it can be forwarded upstream toward the root, as well as downstream to all other branches.

If any router or a link goes down, the downstream router that used this router as the next hop towards the core will have to rejoin the spanning tree individually on behalf of each group present on their outgoing interfaces. Further, during reconfiguring a new router as the core a situation can occur where a leaf router finds that the next hop towards the new core is the router that is downstream to it relative to the previous core. Such a situation is depicted in Figure 3-13. Here, after reconfiguration, router R7 finds that in order to join the new core it has to send a join request towards R5, which is downstream to it (i.e., R7 is still the next-hop router for R5 toward the old core). To deal with this situation, R7 sends a "flush-tree" message downstream to teardown the old tree, i.e., to break the spanning-tree branch from R7 to R5. The downstream routers then perform explicit Rejoin if they have group members attached to them.

CBTs has several advantages over RPF's flood-and-prune approach when the multicast group is sparse (i.e., relatively few routers in the network have group members attached). First, routers

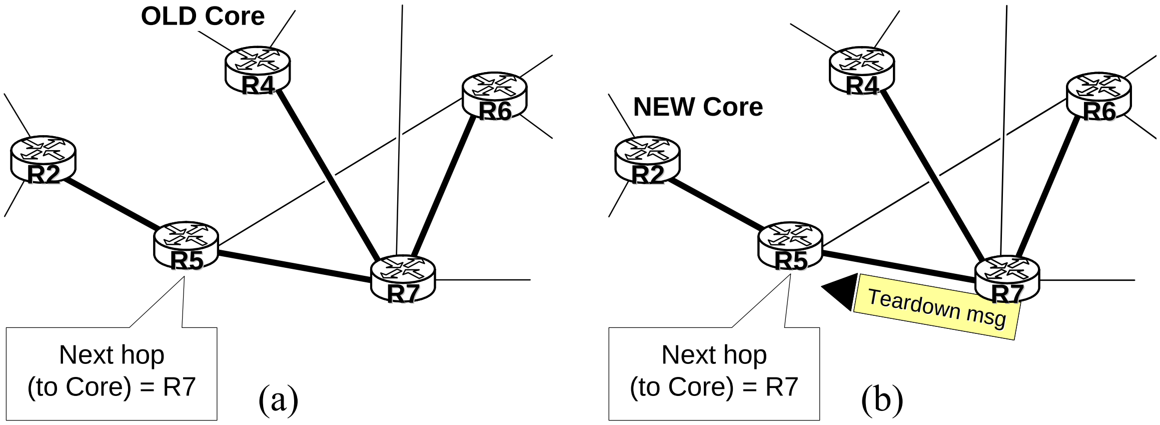


Figure 3-13: Reconfiguration of a CBTs core router from (a) to (b) requires the spanning-tree teardown and rebuilding a new spanning-tree.

that are not members of the multicast group will never know of its existence, so we avoid the overhead of flooding. Second, join and leave messages are explicit, so the hosts can join or leave without waiting for the next flooded packet. Third, each router needs to store only one record per group (the interfaces on which to forward packets for that group). It does not need to store per-source prune information or compute a shortest path tree explicitly.

However, CBTs has several issues of its own. All traffic for the group must pass through the core router, which can become a bottleneck. A shared spanning tree is not the most efficient solution for different sources, as discussed below. Additionally, there is a reliability issue: if the core router goes down, every multicast group that goes through it also goes down. Other issues include: unidirectional vs. bidirectional shared trees; core placement and selection; multiple cores; and, dynamic cores.

A shared spanning tree based on a core router is not optimal for all sources. For example in Figure 3-12, a packet from R8 will reach R7 in four hops (via the core R1), instead of two hops (via R6). In general, the path from a source to receiver via the core might be significantly longer than the shortest possible path. The degree of inefficiency depends on where the core and sources are located relative to each other. If the core is in the "middle," the inefficiency is reasonably small. A possible optimization is to build a **source-specific tree**. Instead of sending a wildcard Join message to join the group *G*, a receiver router sends a source-specific Join towards the source. As this message follows the shortest path towards the source *S*, the routers along the way create an $\langle S, G\rangle$ entry for this tree in their forwarding table. The resulting tree has the root at the source *S* rather than the core router, which may not be part of the new source-specific tree at all. However, the group-shared tree rooted in the core should remain untouched so that other nodes in the group *G* may become sources at a later point.

Core-based tree approach to building and maintaining spanning trees is implemented in the Internet multicast protocol called Protocol-Independent Multicast (PIM), in the variation called Sparse Mode (PIM-SM). See Section 9.2.4 for more information.

3.3.3 Peer-to-Peer Routing

Skype, etc.

3.3.4 Traffic Classes and Differentiated Services

Two complementary network architecture approaches have been devised to guarantee quality-of-service (QoS) in data networks. *Differentiated Services (DiffServ)* aims to guarantee QoS per traffic classes. *Integrated Services (IntServ)*, described in Section 3.3.5, aims to guarantee QoS per individual connections, using resource reservation and connection admission. IntServ requires that every router in the system implements IntServ.

Consider this analogy. If you traveled by car to another city, you have no guarantees about the arrival time. Online map services may predict the travel time based on past traffic statistics, but they cannot predict what will actually happen. Suppose now that we introduce a new class of vehicles, called "emergency vehicles," such as ambulance cars, fire trucks, or police cars. Travelers belonging to this class may receive preferential treatment at intersections (or, "routers"). Even for them, there are no guarantees on their origin-to-destination (i.e., end-to-end) delays. In this sense, their preferential treatment is defined only in terms of "per-hop behaviors" and not "end-to-end." On the other hand, if you traveled by train, the transportation resources are reserved ahead of time. Your origin-to-destination (i.e., end-to-end) travel time is predictable—train timetables are fixed and widely publicized.

DiffServ (Differentiated Services) is an IETF model for QoS provisioning. There are different DiffServ proposals, and some simply divide traffic types into two classes. The rationale behind this approach is that, given the complexities of the best effort traffic, it makes sense to add new complexity in small increments.

The idea of packet marking to inform the network about the different importance and needs of different packets. The source knows whether packets are generated by a delay-intolerant application differently than those generated by an "elastic" application. Based on this knowledge, the source marks the packets for priority treatment or leaves them unmarked for regular best-effort service. Network elements (routers and switches) then identify the packet type by its marking and provide different quality of service to these different packet types. Network elements use scheduling mechanisms to decide the order in which packets are transmitted or when a packet will be dropped. A key problem is how to prevent the "tragedy of the commons," where all sources act greedily and mark their packets with the highest priority. DiffServ relies on traffic policing on the edges of the network, based on service agreements between the customers and the provider.

Suppose that we have enhanced the best-effort service model by adding just one new class, which we call "premium." A small bit-pattern in each packet, in the IPv4 Differentiated Services field (Figure 1-39) or the IPv6 Traffic Class byte (Figure 9-1), is used to mark a packet to receive a particular forwarding treatment at each network node/router (known as **per-hop behavior**).

A router can offer to packets two aspects of preferential treatment:

(1) Expedited forwarding, by moving priority packets at the head of the queue for transmission on an outgoing link. Or, stated differently, given the number of packets that a router can transmit on

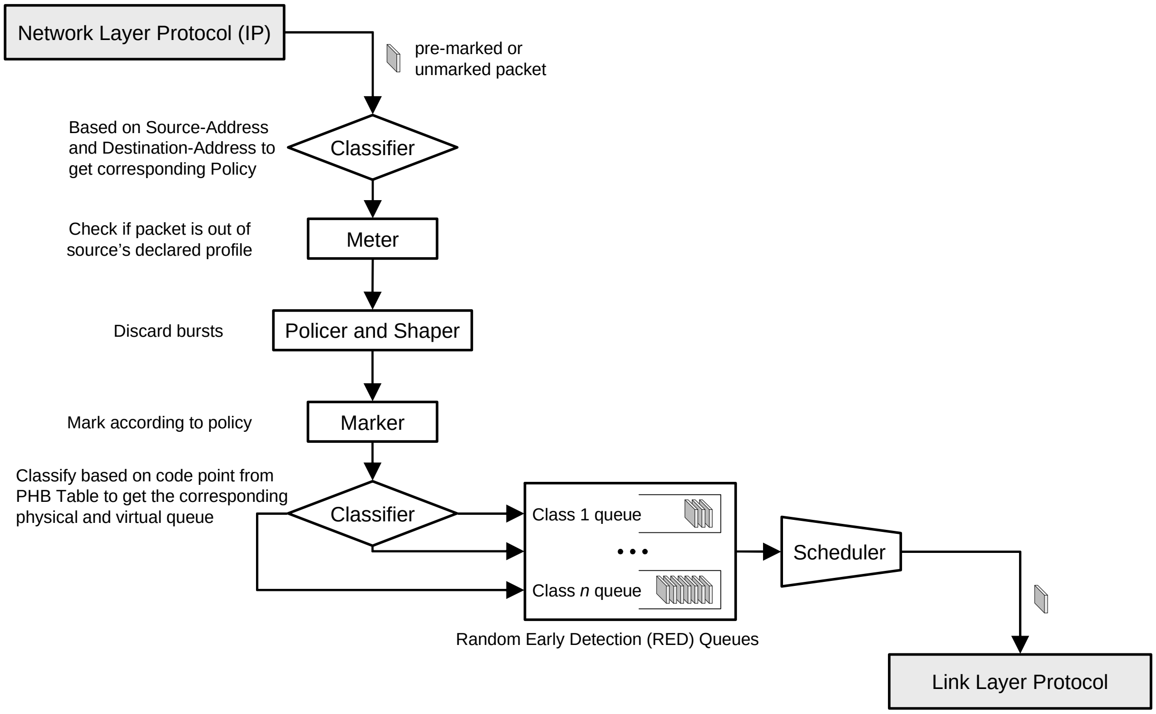

Network Layer Protocol (IP)
pre-marked or unmarked packet
Based on Source-Address and Destination-Address to get corresponding Policy
Classifier
Check if packet is out of source's declared profile
Meter
Discard bursts
Policer and Shaper
Mark according to policy
Marker
Classify based on code point from PHB Table to get the corresponding physical and virtual queue
Classifier
Class 1 queue
• • •
Class *n* queue
Random Early Detection (RED) Queues
Scheduler
Link Layer Protocol

Figure 3-14: DiffServ architecture.

an outgoing link, what fraction will be allocated to different flows. The result is a reduced queuing delay for priority packets.

(2) Reduced packet loss, when router memory space becomes overbooked. The packets belonging to preferred flows are given priority access to the queue space; less favored packets are dropped.

Assuming that packets have been marked in some way, we need to specify the router behavior on encountering a packet with different markings. This can be done in different ways and IETF is standardizing a set of router behaviors to be applied to marked packets. These are called "per-hop behaviors" (PHBs), a term indicating that they define the behavior of individual routers rather than end-to-end services.

DiffServ mechanisms (Figure 3-14):

* DiffServ lies between the network layer and the link layer

* Traffic *marked*, *metered*, *policed*, and *shaped* at source

* Packets *queued* for preferential forwarding, based on:

  - Delay bounds marking

  - Throughput guarantees marking

* Queue for each class of traffic, varying parameters

* Weighted *scheduling* preferentially forwards packets to link layer

DiffServ Service Classes: Expedited Forwarding (EF) and Assured Forwarding (AF)

One PHB is "expedited forwarding" (EF), which states that the packets marked as EF should be forwarded by the router with minimal delay and loss. Of course, this is only possible if the arrival rate of EF packets at the router is always less than the rate at which the router can forward EF packets.

Another PHB is known as "assured forwarding" (AF). Assured forwarding defines several classes of traffic based on two parameters: queuing delay and packet loss. These two parameters take different values for packets belonging to different service classes.

3.3.5 Resource Reservation and Integrated Services

The IntServ model for IP QoS architecture provides two QoS mechanisms: (*i*) different classes of service among which applications can choose to meet their QoS needs; and (*ii*) a protocol to reserve network resources in support of the chosen class of service. Once control packets find and reserve a path from the source to the receiver, the subsequent data packets must follow the established path. This is unlike ordinary IP protocol, where routers may forward different packets of the same flow to different output ports (along different paths).

The classes of service are defined based on applications' requirements described in Section 3.1. The three classes of service that are currently standardized are (from highest performance to lowest):

- ***Guaranteed service class*** provides firm bounds on end-to-end delays and bandwidth, and no-loss guarantees

- ***Controlled load service class*** provides QoS closely approximating the QoS that the same flow would receive in a lightly loaded network, but uses capacity (admission) control to assure that this service is received even when the network is overloaded. Nodes providing this service class use queue scheduling (Section 5.1) to isolate the controlled load traffic from other traffic, and admission control to limit the total amount of controlled load traffic.

- ***Best-effort service class*** is similar to that which the Internet already offers, which is further partitioned into three categories: interactive burst (e.g., Web); interactive bulk (e.g., FTP); and asynchronous (e.g., e-mail).

The main point is that the guaranteed service and controlled load classes are based on quantitative service requirements, and both require signaling and admission control in network nodes. These services can be provided either per-flow or per-flow-aggregate, depending on flow concentration at different points in the network.

An application that requires some kind of guarantees has to make an individual reservation of resources on all routers along the source-destination path. Flow specifications (**Flowspecs**) describe describes the service requested from the network and the traffic that will be injected into the network. This flow specification is communicated to the network by using a signaling protocol, such as RSVP described below. The two parts of a Flowspec are:

(i) What the input traffic will look like, specified in the **Traffic SPECification** or T_{spec} part.

(ii) What guarantees are needed, given in the service **Request SPECification** or R_{spec} part.

T_{spec} parameters include token bucket algorithm parameters (Section 5.2). The idea is that there is a token bucket which slowly fills up with tokens, arriving at a constant rate. Every packet that is sent requires a token, and if there are no tokens, then it cannot be sent. Thus, the rate at which tokens arrive dictates the average rate of traffic flow, while the depth of the bucket dictates how "bursty" the traffic is allowed to be.

T_{spec}s typically just specify the token rate and the bucket depth. For example, a video with a refresh rate of 75 frames per second, with each frame taking 10 packets, might specify a token rate of 750Hz, and a bucket depth of only 10. The bucket depth would be sufficient to accommodate the "burst" associated with sending an entire frame all at once. On the other hand, a conversation would need a lower token rate, but a much higher bucket depth. This is because there are often pauses in conversations, so they can make do with fewer tokens by not sending the gaps between words and sentences. However, this means the bucket depth needs to be increased to compensate for the traffic being burstier.

R_{spec}s specify what requirements there are for the flow: it can be normal Internet "best effort," in which case no reservation is needed. This setting is likely to be used for Web browsing, FTP, and similar applications. The "controlled load" setting mirrors the performance of a lightly loaded network: there may be occasional glitches when two people access the same resource by chance, but generally both delay and drop rate are fairly constant at the desired rate. This setting is likely to be used by soft QoS applications. The "guaranteed" setting gives an absolutely bounded service, where the delay is promised to never go above a desired amount, and packets never dropped, provided the traffic stays within the specification.

Resource Reservation Protocol (RSVP)

Although the IntServ architecture need not be tied to any particular signaling protocol, RSVP is often regarded as the signaling protocol in IntServ. The RSVP protocol (*Resource ReSerVation Protocol)* is a transport layer protocol designed to reserve resources across a network for an integrated services Internet. RSVP defines how applications place reservations for network resources and how they can relinquish the reserved resources once they are not need any more. It is used by a host to request specific qualities of service from the network for particular application data streams or flows. RSVP is also used by routers to deliver quality-of-service (QoS) requests to all nodes along the path(s) of the flows and to establish and maintain state to provide the requested service. RSVP requests will generally result in resources being reserved in each node along the data path.

RSVP is a *signaling protocol*, which means that it is not used to transport application data but rather to control the network, similar to routing protocols. A host uses RSVP to request a specific QoS from the network, on behalf of an application data stream. RSVP carries the request through the network, visiting each node the network uses to carry the stream. At each node, RSVP attempts to make a resource reservation for the stream.

RSVP is a set-up protocol providing a receiver-based, guaranteed, end-to-end QoS pipe. A reserved pipe is created in the following manner: First, PATH messages flow from the sender downstream to discover the data path to the receiver. An RESV message, originating from a receiver and traveling in the reverse direction of the PATH messages, attempts to set local IntServ reservation for the flow. Each node along the path may either admit or reject the reservation subject to capacity or policy admission controls.

To make a resource reservation at a node, the RSVP daemon communicates with two local decision modules, admission control and policy control. Admission control determines whether the node has sufficient available resources to supply the requested QoS. Policy control determines whether the user has administrative permission to make the reservation. If either check fails, the RSVP program returns an error notification to the application process that originated the request. If both checks succeed, the RSVP daemon sets parameters in a packet classifier and packet scheduler to obtain the desired QoS. The packet classifier determines the QoS class for each packet and the scheduler orders packet transmission to achieve the promised QoS for each stream. The routers between the sender and receiver have to decide if they can support the reservation being requested, and, if they cannot, they send a reject message to let the receiver know about it. Otherwise, once they accept the reservation they have to carry the traffic.

The routers store the nature of the flow, and then police it. This is all done in soft state. A *soft state* is a temporary state governed by the periodic expiration of resource reservations, so that no explicit path tear down request is required. Soft states are refreshed by periodic RSVP messages. If nothing is heard for a certain length of time, then the router will time out and the reservation will be cancelled. This solves the problem if either the sender or the receiver crash or are shut down incorrectly without first canceling the reservation. The individual routers have an option to police the traffic to ascertain that it conforms to the flowspecs.

Summary of the key aspects of the RSVP protocol:

1. Shortest-path multicast group/tree

* Require a shortest-path multicast group/tree to have already been created.

* Tree created by Dijkstra algorithm (Section 1.4.2) for link state routing protocols, or via reverse path broadcast procedure (Section 3.3.2), for distance vector routing protocols.

2. PATH message

* Source sends a PATH message to group members with T_{spec} info describing its traffic flow requirements

3. Router inspection of PATH message

* Each router receiving the PATH message inspects it and determines the reverse path to the source.

* Each router also may include a QoS advertisement, which is sent downstream so that the receiving hosts of the PATH message might be able to more intelligently construct, or dynamically adjust, their reservation request.

4. RESV message

* Receiver sends RESV message “back up the tree” to the source.

* RESV message contains the (T_{spec}, R_{spec}) info (the FlowSpec pair) and Filter spec.

* R_{spec} = Description of service requested from the network (i.e., the receiver’s requirements)

* Thus, the FlowSpec (T_{spec}, R_{spec}) specifies a desired QoS.

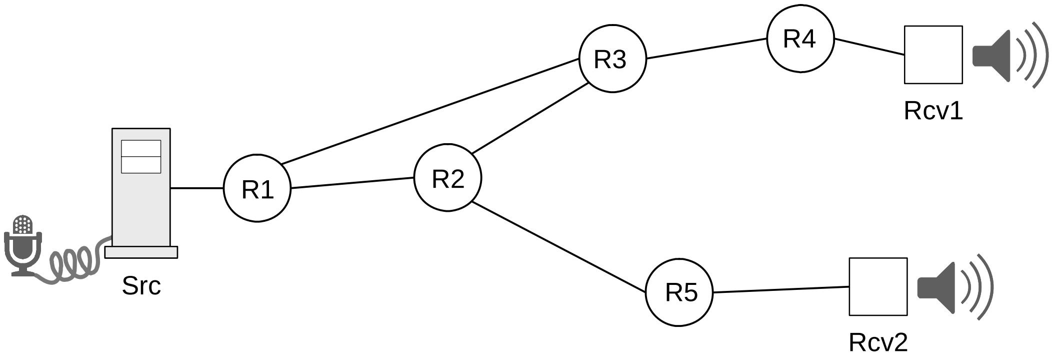


Figure 3-15: Example network for RSVP session setup.

* The Filter spec, together with a session specification, defines the set of data packets—the "flow"—to receive the QoS defined by the FlowSpec.

5. Router inspection of RESV message

* Each router inspects the (T_{spec}, R_{spec}) requirements and determines if the desired QoS can be satisfied.

* If yes, the router forwards the RESV message to the next node up the multicast tree towards the source.

* If no, the router sends a rejection message back to the receiving host.

6. RSVP session

* If the RESV message makes its way up the multicast tree back to the source, the reservation flow request has been approved by all routers in the flow path, and transmission of the application data can begin.

* PATH/RESV messages are sent by source/receiver every 30 seconds to maintain the reservation.

* When a timeout occurs while routers await receipt of a RESV message, then the routers will free the network resources that had been reserved for the RSVP session.

RSVP runs over IP, both IPv4 and IPv6. Among RSVP's other features, it provides opaque transport of traffic control and policy control messages, and provides transparent operation through non-supporting regions.

Consider an example network with a source and two receivers shown in Figure 3-15. Assume that the cost of each link is one hop and both receivers have the same R_{spec} requirements. Figure 3-16 shows all the messages that RSVP will send during the initial session setup phase. Note that PATH message will not be sent on the link R2-R3 because it is not part of the multicast tree. The RESV message stops at the router that is attached to the source host.

Assume now that receiver 2 decides to leave the session. Describe how RSVP will react to this event.

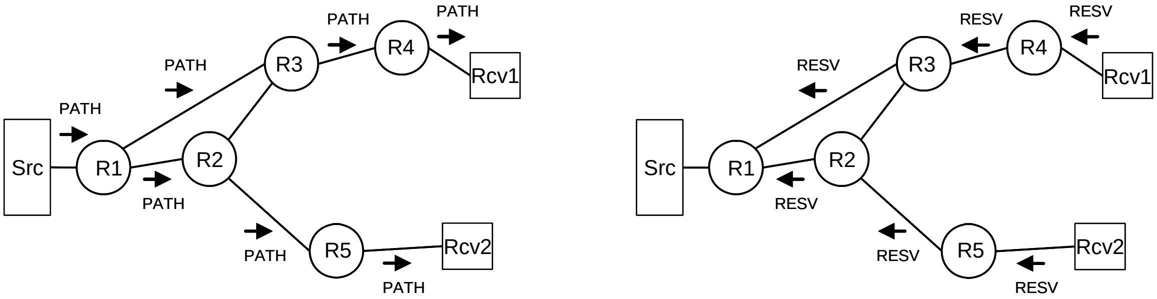


Figure 3-16: RSVP messages during session setup for the network in Figure 3-15.

Merits and Limitations of Integrated Services

IntServ specifies a fine-grained QoS system, which is often contrasted with DiffServ's coarse-grained control system (Section 3.3.4). The major advantage of IntServ is that it provides service classes, which closely match the different application types and their requirements. For example, the guaranteed service class is particularly well suited to support critical, intolerant applications. On the other hand, critical, tolerant applications and some adaptive applications can generally be efficiently supported by the controlled load service class. Other adaptive and elastic applications are accommodated in the best-effort service class.

The problem with IntServ is that many states must be stored in all nodes along the route. Therefore, scalability is a key architectural concern for IntServ, since it requires end-to-end signaling and must maintain a per-flow soft state at every router along the path. Other concerns are, how to authorize and prioritize reservation requests, and what happens when signaling is not deployed end-to-end. Because of these issues, it is generally accepted that IntServ is a better candidate for enterprise networks (i.e., for access networks), which are relatively small and user flows can be managed at the desktop user level, than for large service provider backbones where it is difficult to keep track of all of the reservations. As a result, IntServ is not very popular.

One way to solve this problem is by using a multi-level approach, where per-microflow resource reservation (i.e., resource reservation for individual users) is done in the edge network, while in the core network resources are reserved for aggregate flows only. The routers that lie between these different levels must adjust the amount of aggregate bandwidth reserved from the core network so that the reservation requests for individual flows from the edge network can be better satisfied. See RFC-3175.

3.4 Media Transport Protocols

Multimedia protocols are often designed according to the principle of **application level framing**, which states that only the application has sufficient knowledge of its data to make an informed decision about how data should be transported. In other words, the application better knows its communication needs than does any general-purpose transport protocol. The implication is that a transport protocol should accept data in application-meaningful units (*application data units*, ADUs) and expose the details of their delivery as much as possible, so that the application can appropriately respond to errors in transport. The application cooperates with the transport to achieve the desired characteristics of data delivery.

Application-level framing derives from an insight that there are many ways in which an application can recover from network problems, and that the correct approach depends on both the application and the scenario where it is being used. In some cases, an exact copy of the lost data should be retransmitted. In others, a lower-fidelity copy may be used, or the data may have been superseded, so the replacement is different from the original. If the data was of only transient interest, then the loss can be ignored. The right choice can be made only if the application itself make the choice and interacts closely with the transport to carry it out.

An example of this new style of protocol following the principles of application level framing is RTP (Section 3.4.1). RTP is flexible to provide the information required by a particular application and will often be integrated into the application processing rather than being implemented as a separate layer. RTP is a protocol framework that is deliberately not complete. Unlike conventional protocols in which additional functions might be accommodated by making the protocol more general or by adding an option mechanism that would require parsing, RTP is intended to be tailored through modifications or additions to the headers as needed.

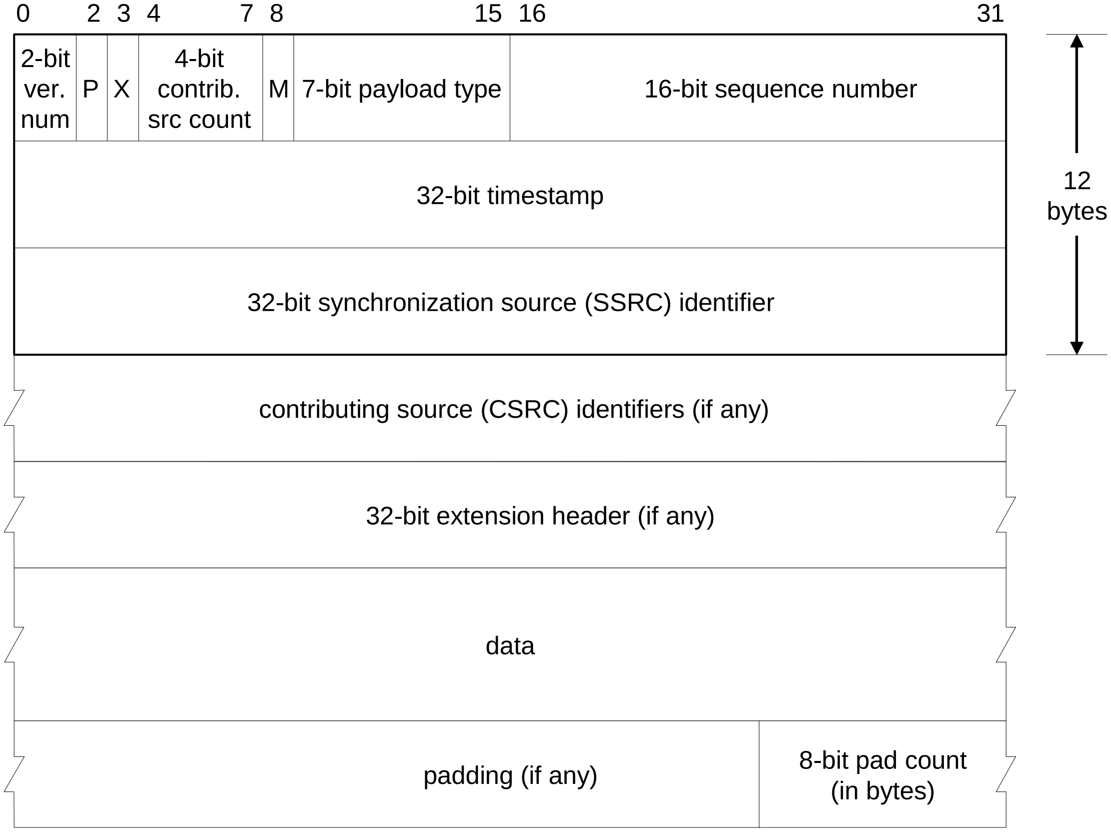
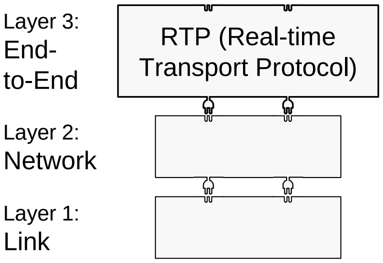

0
2
3
4
7
8
15
16
31
2-bit ver. num
P
X
4-bit contrib. src count
M
7-bit payload type
16-bit sequence number
32-bit timestamp
32-bit synchronization source (SSRC) identifier
12 bytes
contributing source (CSRC) identifiers (if any)
32-bit extension header (if any)
data
padding (if any)
8-bit pad count (in bytes)

Figure 3-17: Real-time Transport Protocol (RTP) packet format.

3.4.1 Real-Time Transport Protocol (RTP)

The *Real-time Transport Protocol* (RTP) provides the transport of real-time data packets. To accommodate new real-time applications, the protocol specifies only the basics and it is somewhat incomplete. Unlike conventional protocols, RTP can be tailored to specific application needs through modifications and additions to headers. This allows the protocol to adapt easily to new audio and video standards.



RTP implements the end-to-end layer (or, transport-layer in the OSI model) features needed to provide synchronization of multimedia data streams. Figure 3-17 shows the header format used by RTP.

The first two bits indicate the RTP version.

The "*padding*" (P) bit is set when the packet contains a set of padding bytes that are not part of the payload. For example, RTP data might be padded to fill up a block of a certain size as required by some encryption algorithms.

The *extension bit* (X) is used to indicate the presence of an extension header, which can be defined for some application's special needs. Such headers are rarely used, because a payload-specific header can be defined as part of the payload format definition used by the application.

The 4-bit *contributing-sources count* represents the number of contributing source (CSRC) identifiers, if any are included in the header.

The *marking bit* M allows significant events to be marked in the packet stream (that is, frame boundaries).

The 7-bit *payload type* specifies the format of the payload in the RTP packet. An RTP sender emits a single RTP payload type at any given time. An RTP packet can contain portions of either audio or video data streams. To differentiate between these streams, the sending application includes a payload type identifier within the RTP header. This identifier indicates the specific encoding scheme used to create the payload.

The *sequence number* is used by the receiver when removing jitter to restore the packet ordering and detect loss (Section 3.3.1). The sequence number is incremented by one for each RTP data packet sent. The initial value of the sequence number is randomly determined. This makes hacking attacks on encryption more difficult. A random number is used even if the source device does not encrypt the RTP packet. The packets can flow through a translator host or router that does provide encryption services.

The *timestamp* is used along with the sequence number to detect gaps in a packet sequence, as described in Section 3.3.1. Timestamps are also used in RTP to synchronize packets from different sources, such as video and its associated audio. The timestamp represents the sampling (creation) time of the first byte in each RTP data packet. However, this is not the actual time of day when this packet was generated (known as "wall-clock time"). In other words, the timestamp is not a reading of the sender's system clock. Instead, the initial value of the time stamp is randomly chosen and increments monotonically and linearly. The resolution of the timer depends on the desired synchronization accuracy required by the application. It is possible that several consecutive RTP packets have the same timestamp. For example, this can occur when a single video frame is transmitted in multiple RTP packets. Because the payloads of these packets were logically generated at the same instant, their time stamps remain constant. The wall-clock time is transmitted in RTCP sender reports, as described in Section 3.4.2.

The *synchronization source* (SSRC) identifier is a randomly chosen identifier for an RTP host. All packets from the same source contain the same SSRC identifier. Each device in the same RTP session must have a unique SSRC identifier. This enables the receiver to group packets for playback.

The *contributing source* (CSRC) identifiers field contains a list of the sources for the payload in the current packet. This field is used when a mixer combines different streams of packets. The information contained in this field allows the receiver to identify the original senders.

3.4.2 RTP Control Protocol (RTCP)

Problems related to this section: Problem 3.11 → ?

The *RTP Control Protocol* (RTCP) provides a control stream that is used to monitor the quality of service provided to its associated RTP stream. RTCP also supports diagnosis of faults in the multicast distribution tree, congestion control, third party performance monitoring and logging, and (simple) conference control. The four basic functions of RTCP are:

(1) The primary function of RTCP is to provide feedback about the delay and loss of the RTP data distribution. This is comparable to the flow and congestion control functions provided by other transport protocols, such as TCP (Chapter 2). The feedback provided by each receiver is used to diagnose stream-distribution faults. By sending feedback to all participants in a session, the device observing problems can determine if the problem is local or remote. The feedback may also be used for control of adaptive source coding. In addition, the network service provider (not a participant in the session) may receive the feedback information. The network provider can then act as a third-party monitor to diagnose network problems.

(2) RTCP carries a persistent transport-level identifier for an RTP source called the *canonical name or CNAME*. Because the SSRC identifier (Figure 3-17) may change if a conflict is discovered or a program is restarted, receivers require the CNAME to keep track of each participant. Receivers may also require the CNAME to associate different media streams (e.g., audio and video) from a given participant that are distributed over different RTP sessions. Recall that there is no requirement in RTP that an audio and video stream from the same node use the same SSRC. Different media may also use different timestamps, which in turn are different from the global clock. RTCP packets by data senders carry timestamps needed for inter-media synchronization.

(3) To support the first two functions, it is necessary that all participants send RTCP packets. Their sending rate must be controlled to avoid network overload for large numbers of participants. By having each participant send its control packets to all the others, each can independently observe the number of participants. This number is used to calculate the rate at which the packets are sent, as described later in this section.

(4) A fourth (optional) function is to convey minimal session-control information, for example participant identification to be displayed in the user interface. This is most likely to be useful in “loosely controlled” sessions where participants enter and leave without membership control or parameter negotiation. RTCP serves as a convenient channel to reach all the participants, but it is not necessarily expected to support all the control communication requirements of an application. A higher-level session control protocol may be needed for this purpose.

RTCP Packet Types

There are five types of RTCP packets:

- Sender Report (SR) contains transmission and reception statistics, sent out periodically by each active sender. It essentially carries cumulative frame and byte counts, and information about the actual time of day on a reference clock (“wall-clock”) to help interpret the timestamp values in RTP packets.

- Receiver Report (RR) contains reception statistics, sent out periodically by passive participants that do not transmit RTP packets. RR carries packet loss and packet delivery rate.

- Source Description (SDES) contains sending participant identifiers, including CNAME (user@host), email, telephone, location, etc.

- End Participation (BYE) indicates end of participation for a participant. It is used by the remaining participants to update their SSRC tables.

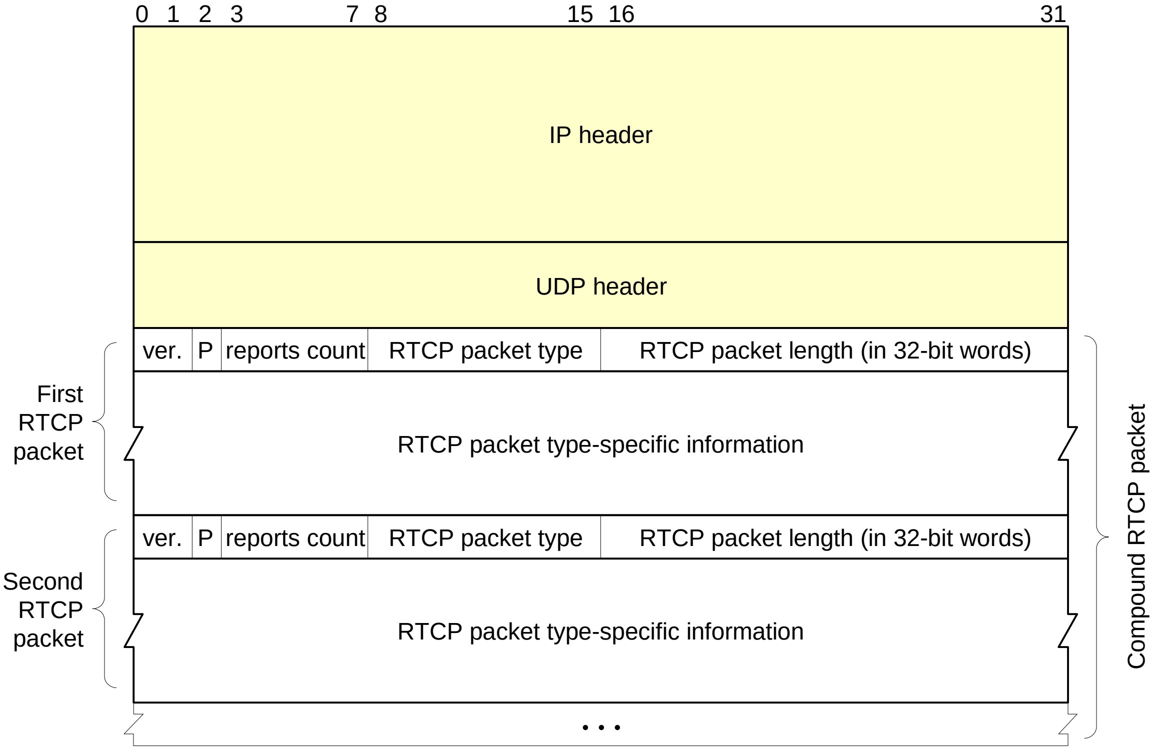


Figure 3-18: Format of a Compound RTCP packet.

• Application Specific (APP) carries application-defined functions.

Each sender and receiver report should contain a reception report block for each synchronization source heard from since the last RTCP report. Each SR or RR packet can report on no more than 31 active senders. If there are more than 31 active sources, the reporter should send multiple RR packets in a *compound packet* (Figure 3-18). Because RTCP packets are "stackable," this helps to amortize the packet header overhead.

An **RTCP Receiver Report (RR)** packet consists of the header, the sender information section, a variable number of receiver report blocks and, optionally, profile-specific extensions (Figure 3-19). The meaning of each field is as follows:

• *Version* (2 bits)—The version number identifies the version of RTP, which is the same in RTCP packets as in RTP data packets (Section 3.4.1, Figure 3-17).

• *Padding bit* (P): Same as for RTP data packets. Padding may be needed by some encryption algorithms with fixed block sizes. (Check the details in RFC-3550 [Schulzrinne, et al., 2003]).

• *Report Count* (RC, 5 bits)—The number of reception report blocks contained in this packet. A value of zero is valid, in which case this RTCP RR packet will contain only the header.

• *Packet Type* (8 bits)—The packet type constant 201 designates identify this as an RTCP RR packet.

• *Length* (16 bits)—The length of this RTCP packet minus one (in 32-bit words), including the header and padding, if any. If the list of report blocks is empty, then RC=0 and Length=1, corresponding to the first 32-bit word of the header plus the reporter SSRC word. The length

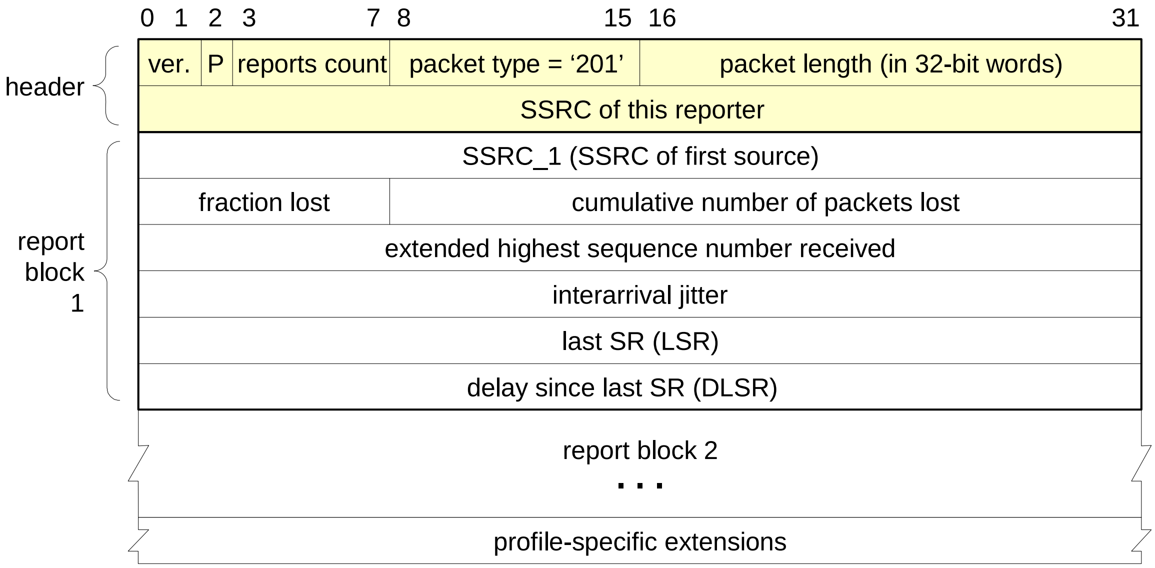

0 1 2 3 7 8 15 16 31
header
ver.
P
reports count
packet type = ‘201’
packet length (in 32-bit words)
SSRC of this reporter
report block 1
SSRC_1 (SSRC of first source)
fraction lost
cumulative number of packets lost
extended highest sequence number received
interarrival jitter
last SR (LSR)
delay since last SR (DLSR)
report block 2
. . .
profile-specific extensions

Figure 3-19: RTCP Receiver Report (RR) packet format.

offset of one makes zero a valid length and avoids a possible infinite loop in scanning a compound RTCP packet, while counting 32-bit words avoids a validity check for a multiple of 4.

- *Reporter SSRC* (32 bits)—The synchronization source identifier for the participant who is sending this RR packet.

Each receiver report block contains these fields:

- *SSRC_n* (32 bits)—The SSRC identifier of the *n*-th sender whose reception is reported in this block. This is one of the synchronization sources from which this reporter has received RTP packets during the current reporting interval.

- *Fraction Lost* (8 bits)—The estimated fraction of the RTP data packets from source SSRC_*n* that is assumed to be lost since the previous SR or RR packet. The reporter estimates the loss as the number of packets lost in this reporting interval, divided by the expected number of packets (defined below). The result is expressed as a fixed-point number with the binary point at the left edge of the field. (That is the integer part after multiplying the loss fraction by 256. For example, if 1/4 of the packets were lost, the loss fraction would be 1/4 × 256 = 64.) If the loss is negative due to duplicates, this field is set to zero. Note that a receiver cannot tell whether any packets were lost after the last one received, and that there will be no reception report block issued for a source if all packets from that source sent during the last reporting interval have been lost.

- *Cumulative Number of Packets Lost* (24 bits)—The total number of RTP data packets from source SSRC_*n* that have been lost since the beginning of reception. This number is defined as the number of packets expected less the number of packets actually received, where the number of packets received includes any which are late or duplicates. Thus, packets that arrive late are not counted as lost, and the loss may be negative if there are duplicates. The number of packets expected is defined to be the extended last sequence number received, as defined next, less the initial sequence number received.

- *Extended Highest Sequence Number Received* (EHSN, 32 bits)—The low 16 bits of the extended highest sequence number contain the highest sequence number received in an RTP data

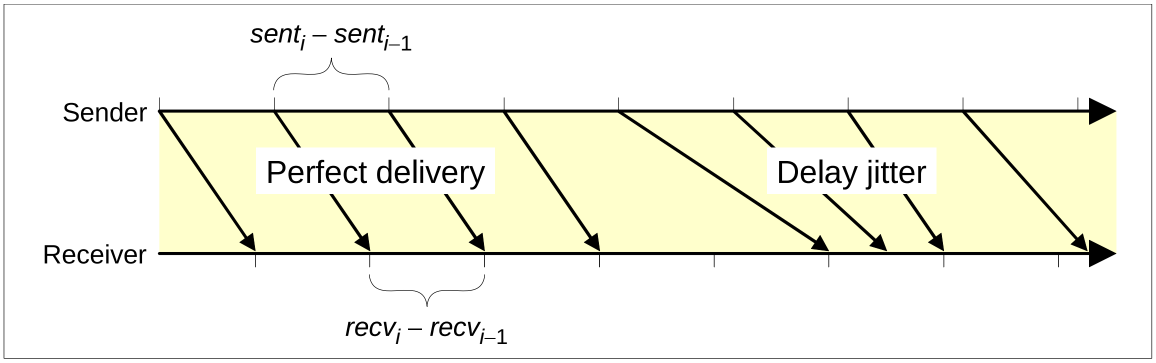


Figure 3-20: Inter-arrival jitter defined as the deviation of the difference in packet spacing at the receiver compared to the sender. For a perfect delivery, the jitter is zero.

packet from source SSRC_*n*, and the most significant 16 bits extend that sequence number with the corresponding count of sequence number cycles.

• *Inter-arrival Jitter* (32 bits)—An estimate of the statistical variance of the RTP data packet inter-arrival times. It is measured in RTP timestamp units and expressed as an unsigned integer. The inter-arrival jitter *J* is defined as the mean deviation (smoothed absolute value) of the difference *D* in packet spacing at the receiver compared to the sender, for a pair of packets (Figure 3-20). If S_i is the RTP timestamp from packet *i*, and R_i is the time of arrival in RTP timestamp units for packet *i*, then for two packets *i* and *j*, *D* may be expressed as:

$$D(i,j) = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i) \tag{3.4}$$

This value is equivalent to the difference in the “relative transit time” for the two packets; the relative transit time is the difference between a packet’s RTP timestamp and the receiver’s clock at the time of arrival, measured in the same units.

The inter-arrival jitter should be calculated continuously as each data packet *i* is received from source SSRC_*n*, using this difference *D* for that packet and the previous packet *i*–1 in order of arrival (not necessarily in sequence), according to the formula:

$$\text{jitter}_{\text{new}} = \text{jitter}_{\text{old}} + \frac{\text{instantaneous jitter} - \text{jitter}_{\text{old}}}{16}$$

or more formally:

$$J(i) = J(i-1) + \frac{|D(i-1,i)| - J(i-1)}{16} \tag{3.5}$$

Whenever a reception report is issued, the current value of *J* is sampled.

The jitter calculation must conform to the formula specified here in order to allow profile-independent monitors to make valid interpretations of reports coming from different implementations. This algorithm is the optimal first-order estimator and the gain parameter 1/16 gives a good noise reduction ratio while maintaining a reasonable rate of convergence. See RFC-3550, Section 6.4.4, for a discussion of the effects of varying packet duration and delay before transmission.

• *Last SR Timestamp* (LSR, 32 bits)—The middle 32 bits out of 64 in the NTP timestamp (described below for the SR packet type) received as part of the most recent RTCP sender report

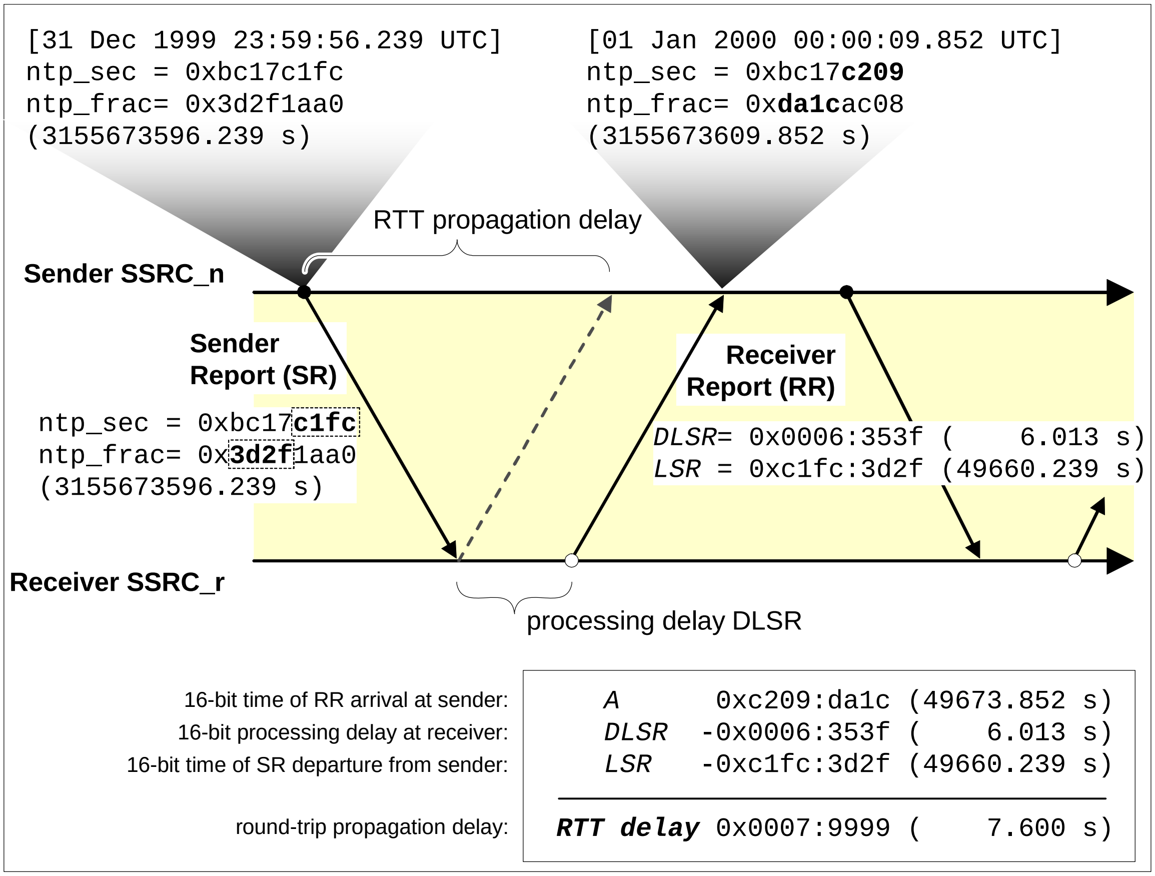

[31 Dec 1999 23:59:56.239 UTC]
ntp_sec = 0xbc17c1fc
ntp_frac= 0x3d2f1aa0
(3155673596.239 s)
[01 Jan 2000 00:00:09.852 UTC]
ntp_sec = 0xbc17c209
ntp_frac= 0xda1cac08
(3155673609.852 s)
RTT propagation delay
Sender SSRC_n
Sender Report (SR)
ntp_sec = 0xbc17c1fc
ntp_frac= 0x3d2f1aa0
(3155673596.239 s)
Receiver Report (RR)
DLSR= 0x0006:353f (6.013 s)
LSR = 0xc1fc:3d2f (49660.239 s)
Receiver SSRC_r
processing delay DLSR
16-bit time of RR arrival at sender:
16-bit processing delay at receiver:
16-bit time of SR departure from sender:
round-trip propagation delay:
A 0xc209:da1c (49673.852 s)
DLSR -0x0006:353f (6.013 s)
LSR -0xc1fc:3d2f (49660.239 s)
RTT delay 0x0007:9999 (7.600 s)

Figure 3-21: Example for round-trip time computation: RTT delay = *A* – *LSR* – *DLSR*.

(SR) packet from source SSRC_*n*. In other words, LSR represents the low 16 bits of the integer seconds part and the high 16 bits of the fractional seconds part of the NTP timestamp (Figure 3-19). If no SR has been received yet, the field is set to zero.

- *Delay Since Last SR* (DLSR, 32 bits)—The delay, expressed in units of 1/65536 seconds, between receiving the last SR packet from source SSRC_*n* and sending this reception report block. If no SR packet has been received yet from SSRC_*n*, the DLSR field is set to zero.

Let SSRC_*r* denote the receiver issuing this Receiver Report. Source SSRC_*n* can compute the round-trip propagation delay to SSRC_*r* by recording the time *A* when this reception report block is received. It calculates the total round-trip time *A*–*LSR* using the last SR timestamp (*LSR*) field, and then subtracting this field to leave the *round-trip propagation delay* as (*A* – *LSR* – *DLSR*). This calculation is illustrated in Figure 3-21. Times are shown in both a hexadecimal representation of the 32-bit fields and the equivalent floating-point decimal representation. Colons indicate a 32-bit field divided into a 16-bit integer part and 16-bit fraction part.

An **RTCP Sender Report (SR)** packet consists of the header, the sender information section, a variable number of receiver report blocks and, optionally, profile-specific extensions (Figure 3-22). The format of the SR packet is essentially the same as that of the RR packet (Figure 3-19) except that the packet type field contains the constant 200 and the section on sender information (containing the NTP and RTP timestamps and sender's packet and byte counts). The remaining fields have the same meaning as for the RR packet.

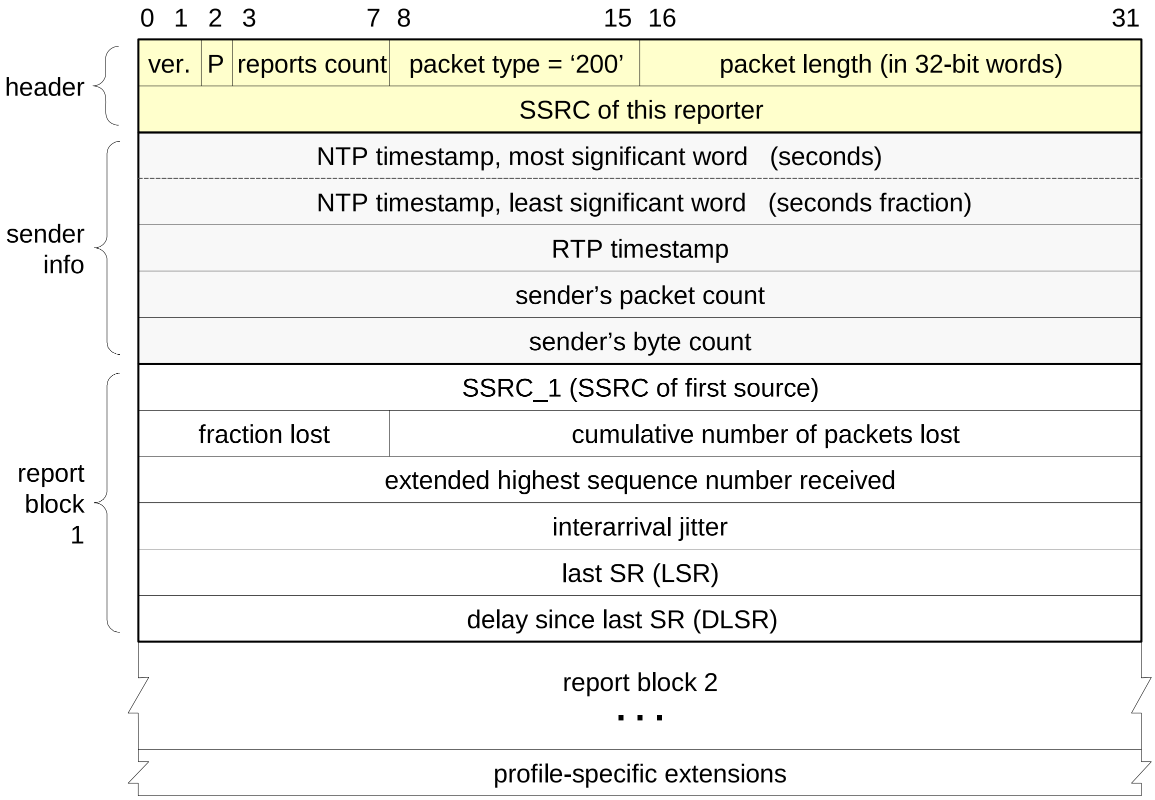

0 1 2 3 7 8 15 16 31
header
ver.
P
reports count
packet type = '200'
packet length (in 32-bit words)
SSRC of this reporter
sender info
NTP timestamp, most significant word (seconds)
NTP timestamp, least significant word (seconds fraction)
RTP timestamp
sender's packet count
sender's byte count
report block 1
SSRC_1 (SSRC of first source)
fraction lost
cumulative number of packets lost
extended highest sequence number received
interarrival jitter
last SR (LSR)
delay since last SR (DLSR)
report block 2
. . .
profile-specific extensions

Figure 3-22: RTCP Sender Report (SR) packet format.

The second section, the sender information, is 20 bytes long and is present in every sender report packet. It summarizes the data transmissions from this sender, as follows:

- *NTP Timestamp* (64 bits)—The NTP timestamp indicates the point of time measured in reference wall-clock time when this report was sent. NTP timestamps are represented as a 64-bit unsigned fixed-point number, in seconds relative to 0 hours on 1 January 1900. The integer part is in the first 32 bits and the fraction part in the last 32 bits. The 32-bit fraction field determines the clock resolution as 2^{-32}, in seconds, which equals 232 picoseconds.

The reference clock is shared by all media to be synchronized. The NTP timestamp can be used to estimate the round-trip propagation time to and from the receivers, in combination with timestamps returned in reception reports from other receivers. Receivers should expect that the measurement accuracy of the timestamp may be limited to much less than the resolution of the NTP timestamp. The measurement uncertainty of the timestamp is not indicated as it may not be known. On a system that has no notion of wall-clock time but does have some system-specific clock such as "system uptime," a sender may use that clock as a reference to calculate relative NTP timestamps. It is important to choose a commonly used clock so that if separate implementations are used to produce the individual streams of a multimedia session, all implementations will use the same clock. Until the year 2036, relative and absolute timestamps will differ in the high bit so (invalid) comparisons will show a large difference; by then one hopes relative timestamps will no longer be needed. A sender that has no notion of wall-clock or elapsed time may set the NTP timestamp to zero.

• RTS (32 bits)—The RTP timestamp resembles the same time as the NTP timestamp (above), but is measured in the same units and with the same random offset as the RTP timestamps in data packets. This correspondence may be used for intra- and inter-media synchronization for sources whose NTP timestamps are synchronized, and may be used by media-independent receivers to estimate the nominal RTP clock frequency.

• SPC (32 bits)—The sender's packet count totals up the number of RTP data packets transmitted by the sender since joining the RTP session. This field can be used to estimate the average data packet rate.

• SOC (32 bits)—The total number of payload octets (i.e., not including the header or any padding) transmitted in RTP data packets by the sender since starting up transmission. This field can be used to estimate the average payload data rate.

RTCP Transmission Interval

RTP allows multimedia applications to scale automatically from a few participants to thousands. In an audio conference, the data traffic is inherently self-limiting because only one or two people will speak at a time. Even in broadcasting sessions with thousands of listeners, if data are distributed using multicast then the data rate on any given link will remain relatively constant independent of the number of listeners (see Figure 3-6(b)). However, the RTCP control traffic is not self-limiting. If each participant sent control reports at a constant rate, the control traffic would grow linearly with the number of participants. Therefore, the reporting rate of each participant must be reduced as the number of participants grows.

The rules for dynamically adjusting the interval between RTCP packet transmissions are somewhat complex, but the basic goal is simple—to limit the total amount of bandwidth consumed by control messages a fraction (usually 5%) of the RTP data bandwidth. As a result, although RTP data packets are typically sent every few milliseconds, the RTCP control protocol sends reports on the scale of seconds. To accomplish this goal, the participants need to know how much data bandwidth is likely to be in use (e.g., the amount to send two audio streams) and the number of participants. They learn the data bandwidth from means outside RTP (known as *session management*, Section 3.4.3), and they learn the number of participants from their RTCP reports. Because RTCP reports might be sent at a very low rate, it might be possible to know only an approximate count of current participants, but that is typically sufficient. In addition, it is recommended to allocate more RTCP bandwidth to active senders, on the assumption that most participants would like to see reports from them.

To maintain scalability, the average interval between packets from a session participant should scale with the group size. The average time each participant waits between sending RTCP packets is known as the **transmission interval** (or *reporting interval*). This interval is calculated by combining a several factors:

• RTCP bandwidth is limited to a fraction of session bandwidth, typically 5%. The session bandwidth is the expected data rate for the RTP session. Typically, this is the bit rate of a single stream of audio or video data, multiplied by the typical number of concurrent senders. All members of a session must use the same fraction. The session bandwidth is fixed for the duration of a session, and given as a configuration parameter to the RTP application when it starts.

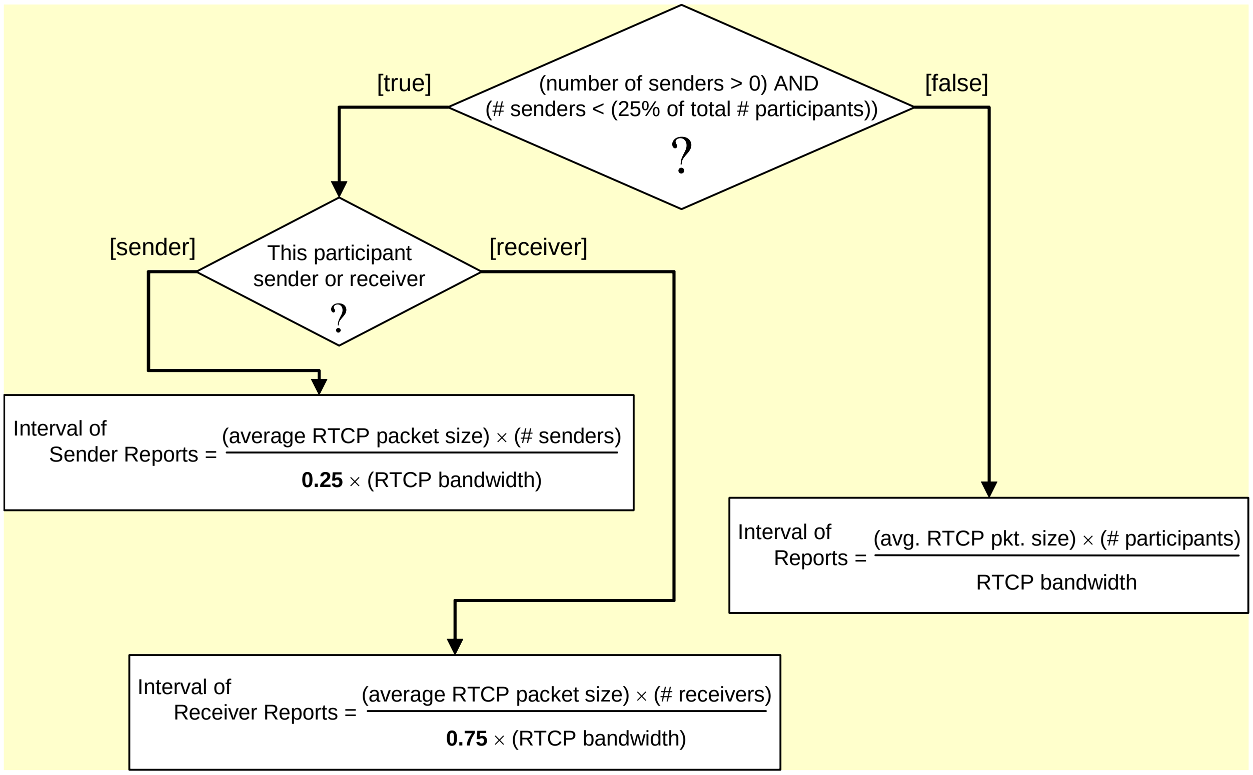

[true]
(number of senders > 0) AND
(# senders < (25% of total # participants))
?
[false]
[sender]
This participant
sender or receiver
?
[receiver]
Interval of Sender Reports = (average RTCP packet size) × (# senders) / 0.25 × (RTCP bandwidth)
Interval of Reports = (avg. RTCP pkt. size) × (# participants) / RTCP bandwidth
Interval of Receiver Reports = (average RTCP packet size) × (# receivers) / 0.75 × (RTCP bandwidth)

Figure 3-23: Algorithm for computing the RTCP reporting interval.

• 25% of RTCP bandwidth should be allocated to senders. This requires the application to maintain a database of all participants, noting whether they are senders (that is, if RTP data packets or RTCP SR packets have been received from them) or receivers (if only RTCP RR, SDES, or APP packets have been received).

The algorithm for computing the RTCP transmission interval is shown in Figure 3-23. Note that the average size of RTC packets includes not just the RTCP data, but also the UDP and IP header sizes (that is, add 28 bytes per packet for a typical IPv4 implementation, see Figure 3-18). As the right branch in Figure 3-23 shows, if the number of senders is greater than 25%, senders and receivers are treated together. This algorithm ensures that senders have a significant fraction of the RTCP bandwidth, sharing at least one-quarter of the total RTCP bandwidth. The RTCP packets required for lip synchronization and identification of senders can therefore be sent comparatively quickly, while still allowing reports from receivers.

The challenge is that each session participant must independently compute an RTCP transmission interval that ensures transmission guidelines are met. General principles for computing the RTCP transmission interval include:

• Participants keep a running estimate of the number of participants in the session

• For a given session bandwidth, session size, and RTCP packet size, calculate a transmission interval

• Randomize the calculated interval to avoid synchronization effects: set to a number uniformly distributed between 0.5 and 1.5 times the deterministic calculated interval. The resulting value of is divided by *e*–3/2=1.21828 to compensate for the fact that the timer reconsideration algorithm converges to a value of the RTCP bandwidth below the intended average

- As other participants join and leave the group (detected by the receipt of BYE messages), adjust the transmission interval accordingly

This procedure results in an interval which is random, but which, on average, gives at least 25% of the RTCP bandwidth to senders and the rest to receivers. If the senders constitute more than one quarter of the membership, this procedure splits the bandwidth equally among all participants, on average.

3.4.3 Real-Time Conferencing

A *multimedia session* is a set of concurrent RTP sessions among a common group of participants. For example, a videoconference (which is a multimedia session) may contain an audio RTP session and a video RTP session.

An *RTP session* is an association among a set of participants communicating with RTP. A participant may be involved in multiple RTP sessions at the same time. In a multimedia session, each medium is typically carried in a separate RTP session with its own RTCP packets unless the encoding itself multiplexes multiple media into a single data stream. A participant distinguishes multiple RTP sessions by reception of different sessions using different pairs of destination transport addresses, where a pair of transport addresses comprises one network address plus a pair of ports for RTP and RTCP. All participants in an RTP session may share a common destination transport address pair, as in the case of IP multicast, or the pairs may be different for each participant, as in the case of individual unicast network addresses and port pairs. In the unicast case, a participant may receive from all other participants in the session using the same pair of ports, or may use a distinct pair of ports for each.

The distinguishing feature of an RTP session is that each maintains a full, separate space of SSRC identifiers (defined in Section 3.4.1). The set of participants included in one RTP session consists of those that can receive an SSRC identifier transmitted by any one of the participants either in RTP as the SSRC or a CSRC (also defined in Section 3.4.1) or in RTCP. For example, consider a three-party conference implemented using unicast UDP with each participant receiving from the other two on separate port pairs. If each participant sends RTCP feedback about data received from one other participant only back to that participant, then the conference is composed of three separate point-to-point RTP sessions. If each participant provides RTCP feedback about its reception of one other participant to both of the other participants, then the conference is composed of one multi-party RTP session. The latter case simulates the behavior that would occur with IP multicast communication among the three participants.

Figure 3-24

RTP relies on the underlying protocol(s) to provide demultiplexing of RTP data and RTCP control streams. For UDP and similar protocols, RTP should use an even destination port number and the corresponding RTCP stream should use the next higher (odd) destination port number. For applications that take a single port number as a parameter and derive the RTP and RTCP port pair from that number, if an odd number is supplied then the application should replace that number with the next lower (even) number to use as the base of the port pair. For applications in

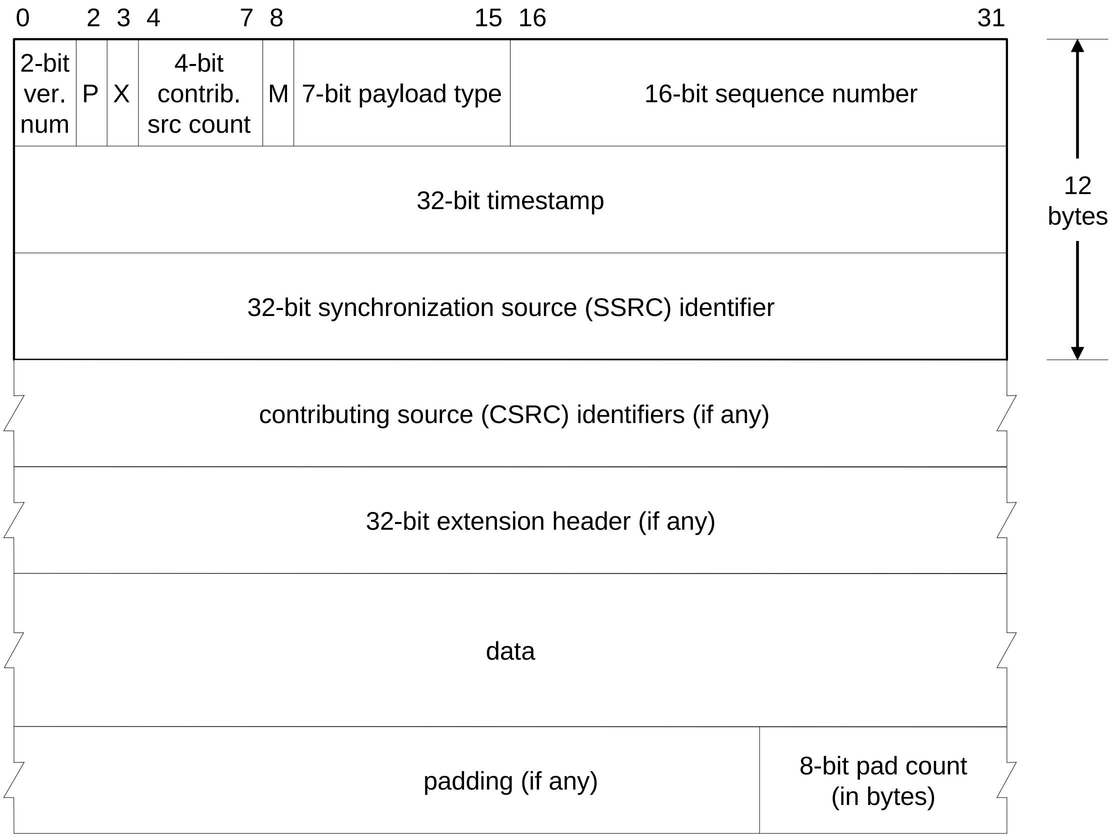

0
2
3
4
7
8
15
16
31
2-bit ver. num
P
X
4-bit contrib. src count
M
7-bit payload type
16-bit sequence number
32-bit timestamp
32-bit synchronization source (SSRC) identifier
12 bytes
contributing source (CSRC) identifiers (if any)
32-bit extension header (if any)
data
padding (if any)
8-bit pad count (in bytes)

Figure 3-24: Logical channels for a real-time conferencing session.

which the RTP and RTCP destination port numbers are specified via explicit, separate parameters (using a signaling protocol or other means), the application may disregard the restrictions that the port numbers be even/odd and consecutive although the use of an even/odd port pair is still encouraged. The RTP and RTCP port numbers must not be the same since RTP relies on the port numbers to demultiplex the RTP data and RTCP control streams.

In a unicast session, both participants need to identify a port pair for receiving RTP and RTCP packets. Both participants may use the same port pair. A participant must not assume that the source port of the incoming RTP or RTCP packet can be used as the destination port for outgoing RTP or RTCP packets. When RTP data packets are being sent in both directions, each participant's RTCP SR packets must be sent to the port that the other participant has specified for reception of RTCP. The RTCP SR packets combine sender information for the outgoing data plus reception report information for the incoming data. If a side is not actively sending data, an RTCP RR packet is sent instead.

3.5 QoS in Wireless Networks

3.6 Interprovider QoS

QoS based on the Differentiated Services architecture (Section 3.3.4) is now widely deployed within the networks of individual Internet service providers. Some providers are now beginning to interconnect with each other via “QoS-enabled peering” in an attempt to offer QoS that spans the networks of multiple providers.

The commercial challenges include the lack of QoS settlement among ISPs. To make the QoS effect perceivable by end users, service differentiation needs to be end-to-end, which requires cooperation of all ISPs along the communication path. To enable such cooperation, all the involved ISPs must be able to benefit monetarily from providing QoS to a particular application session.

Obviously, it may be difficult to keep accounting on a per-session basis, so that each benefits from each individual session. Instead, a more feasible option is to have agreements and accounting so that the involved ISPs just care that the aggregate QoS provided to their customers is approximately equal. This kind of arrangement is what ISPs already do for the amounts of transit traffic that cross their borders (Section 1.4.5). The next step would be to design such agreements for QoS service classes, which brings us to technological challenges of inteprovider QoS.

In addition to commercial challenges, there are also technological challenges regarding how ISPs cooperate. One challenge is that ISPs are likely in different stages of supporting QoS. If and ISP does not have deployed DiffServ, it has no means to honor the QoS traffic by incident ISPs that do support DiffServ. This deficiency will reduce the end-to-end differentiation.

Another technical challenge is that different ISPs may offer different number of traffic classes, and there is no standardized inter-mapping between different ISPs traffic classes. Recall that IETF only standardized Per Hop Behaviors (PHBs) that can be used to construct different traffic classes (Section 3.3.4). The traffic classes are not standardized and are left to ISPs to define. Suppose that one ISP offers classes CoS1, CoS2, and CoS3, and another ISP offers classes CoS1 and CoS3. If the second ISP maps CoS2 of the first ISP to its class CoS1, the distinction between that ISP’s CoS1 and CoS2 will become unclear. Similar problem will occur if ISP-2 maps ISP-1’s CoS2 to its CoS3. When the distinction between two traffic classes becomes fuzzy, the ISP will have difficulty selling the more expensive class of service.

Yet another challenge is evaluating whether an ISP that is involved in the end-to-end path delivers the promised QoS. This task may involve allocating the delay, delay jitter, and packet loss ratio (PLR) budgets. For example, in order for a telephony call to work, the delay, delay jitter, and PLR must be within certain ranges (Section 3.1). Suppose that a call packets travel over three ISPs and the delay budget is 150 ms. Should the 150 ms budget be divided equally among the three ISPs, each getting 50 ms? This may not be fair if different ISPs cover different geographical areas. More broadly, the issue is if the budget allocation should be application-specific or ISP-dependent so that a given ISP will have certain budgets assigned regardless of how many ISPs are traveled over by application's packets.

3.7 Summary and Bibliographical Notes

Latency, jitter and packet loss are the most common ills that plague real-time and multimedia systems. The remedy is in the form of various quality-of-service (QoS) provisions. Chapter 4 analyzes store-and-forward and queuing congestion in switches and routers. Congestion can lead to packets spacing unpredictably and thus resulting in jitter. The more hops a packet has to travel, the worse the jitter. Latency due to distance (propagation delay) is due to the underlying physics and nothing can be done to reduce propagation delay. However, devices that interconnect networks (routers) impose latency that is often highly variable. Jitter is primarily caused by these device-related latency variations. As a device becomes busier, packets must be queued. If those packets happen to be real-time audio data, jitter is introduced into the audio stream and audio quality declines.

Chapter 5 describes techniques for QoS provisioning.

The material presented in this chapter requires basic understanding of probability and random processes. [Yates & Goodman 2004] provides an excellent introduction and [Papoulis & Pillai 2001] is a more advanced and comprehensive text.

Section 3.1: Application Requirements

For video, expectations are low

For voice, ear is very sensitive to jitter and latencies, and loss/flicker

Discussion of the role of round-trip time in telephone conversations is available in:

ITU-T Recommendation G.114. "One-way Transmission Time," May 2003. Online at: http://www.itu.int/rec/T-REC-G.114-200305-I/en

Section 3.3: Approaches to Quality-of-Service

QoS: [Wang, 2001]

→ Multicast Routing

Bertsekas and Gallager [1992] describe several algorithms for spanning-tree construction. Ballardie, *et al*., [1993] introduced *core based trees* (CBT) algorithm for forming the delivery tree—the collection of nodes and links that a multicast packet traverses. Also see RFC-2189.

Gärtner [2003] reviews several distributed algorithms for computing the spanning tree of a network. He is particularly focusing on *self-stabilizing* algorithms that are guaranteed to recover from an arbitrary perturbation of their local state in a finite number of execution steps. This means that the variables of such algorithms do not need to be initialized properly.

→ DiffServ

The IETF Working Group on Differentiated Services has standardized a common layout for a six-bit field of both bytes, called the "DS field." RFC-2474 and RFC-2475 define the architecture, and the general use of bits within the Differentiated Services field (superseding the IPv4 Type of Service definitions of RFC-1349). A small number of specific per-hop behaviors (PHBs) are defined and a particular bit pattern or "code-point of the DS field recommended for each one, in RFC-2474, RFC-2597, and RFC-2598.

→ IntServ

RSVP by itself is rarely deployed in data networks as of this writing (2013), but the traffic engineering extension of RSVP, called RSVP-TE [RFC-3209], is becoming accepted recently in many QoS-oriented networks. See Section 5.4 on MPLS and particularly Section 5.4.3 on traffic engineering.

As an important research topic: show that multihop can or cannot support multiple streams of voice.

RFC-2330 [Paxson, *et al*., 1998] defines a general framework for performance metrics being developed by the IETF's IP Performance Metrics effort, by the IP Performance Metrics (IPPM) Working Group.

RFC-3393 [Demichelis & Chimento, 2002] defines one-way delay jitter across Internet paths.

RFC-2205: Resource ReSerVation Protocol (RSVP) -- Version 1 Functional

There was no notion of QoS in Ethernet until 1998 when IEEE published 802.1p as part of the 802.1D-1998 standard. 802.1p uses a three-bit field in the Ethernet frame header to denote an eight-level priority. One possible service-to-value mapping is suggested by RFC-2815, which describes Integrated Service (IntServ) mappings on IEEE 802 networks.

[Thomson, *et al*., 1997]

Section 3.4: Media Transport Protocols

Application level framing and integrated layer processing was proposed by Clark and Tennenhouse [1990] as a new way to design protocols for emerging multimedia applications. They recognized that these new applications were unlikely to be well served by existing protocols, such as TCP (Chapter 2), and that they might not be well served by any general transport protocol. The essence of the application level framing principle is that application understands its own communication needs best.

RTP and RTCP are defined in RFC-3550 [Schulzrinne, et al., 2003]. A comprehensive reference for RTP and RTCP is [Perkins, 2003].

Problems

Problem 3.1

Problem 3.2

Problem 3.3

Consider an internet telephony session over a network where the observed propagation delays vary between 50–200 ms. Assume that the session starts at time zero and both hosts use pulse code modulation to encode speech, where voice packets of 160 bytes are sent every 20 ms. Also, both hosts use a fixed playout delay of q = 150 ms.

(a) Write down the playout times of the packets received at one of the hosts as shown in the table below.

(b) What size of memory buffer is required at the destination to hold the packets for which the playout is delayed?

Packet sequence number	Arrival time r_i [ms]	Playout time p_i [ms]
#1	95	
#2	145	
#3	170	
#4	135	
#6	160	
#5	275	
#7	280	
#8	220	
#9	285	
#10	305	

Problem 3.4

Consider the same internet telephony session as in Problem 3.2, but this time the hosts use adaptive playout delay. Assume that the first packet of a new talk spurt is labeled k, the current estimate of the average delay is $\hat{\delta}_k$ = 90 ms, the average deviation of the delay is $\hat{\upsilon}_k$ = 15 ms, and the constants α = 0.01 and K = 4.

The table below shows how the packets are received at the host B. Write down their playout times, keeping in mind that the receiver must detect the start of a new talk spurt.

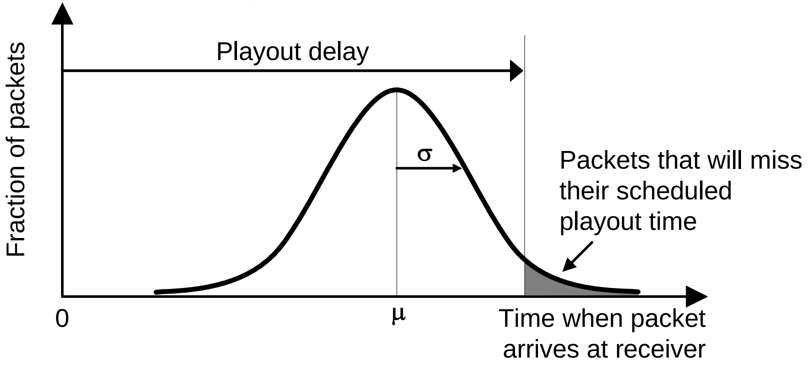
Packet seq. #	Timestamp t_i [ms]	Arrival time r_i [ms]	Playout time p_i [ms]	Average delay $\hat{\delta}_i$ [ms]	Average deviation $\hat{\upsilon}_i$

k	400	480			
k+1	420	510			
k+2	440	570			
k+3	460	600			
k+4	480	605			
k+7	540	645			
k+6	520	650			
k+8	560	680			
k+9	580	690			
k+10	620	695			
k+11	640	705			

Problem 3.5

Consider an internet telephony session using adaptive playout delay, where voice packets of 160 bytes are sent every 20 ms. Consider one of the receivers during the conversation. Assume that at the end of a previous talk spurt, the current estimate for the average delay is $\hat{\delta}_k$ = 150 ms, the average deviation of the delay is $\hat{\upsilon}_k$ = 50 ms. Because of high delay and jitter, using the constant K = 4 would produce noticeable playout delays affecting the perceived quality of conversation. If the receiver decides to maintain the playout delay at 300 ms, what will be the percentage of packets with missed playouts (approximate)? Explain your answer.

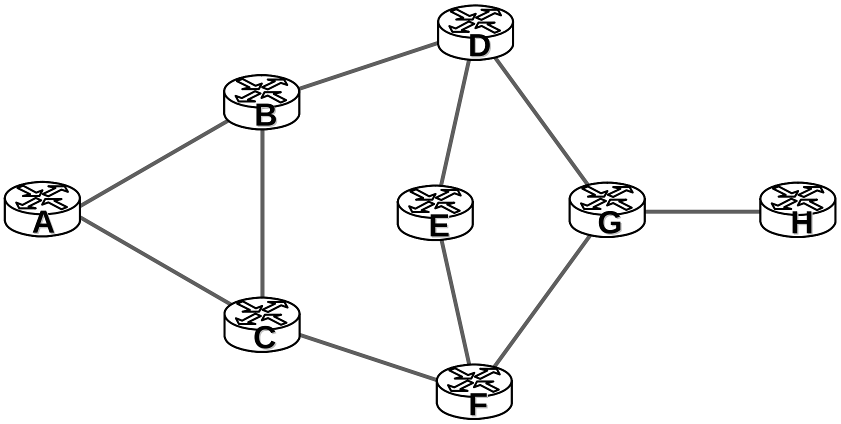
(*Hint*: Use the chart shown in the figure below to derive the answer.)



Problem 3.6

Problem 3.7

Consider the following network where router *A* needs to multicast a packet to all other routers in the network. Assume that the cost of each link is 1 and that the routers are using the reverse path forwarding (RPF) algorithm.



Do the following:

(a) Draw the shortest path multicast tree for the network.

(b) How many packets are forwarded in the entire network per every packet sent by the source *A*?

(c) Assuming that RPF uses pruning and routers *E* and *F* do not have attached hosts that are members of the multicast group, how many packets are forwarded in the entire network per every packet sent by *A*?

For each item, show the work, not only the final result.

Problem 3.8

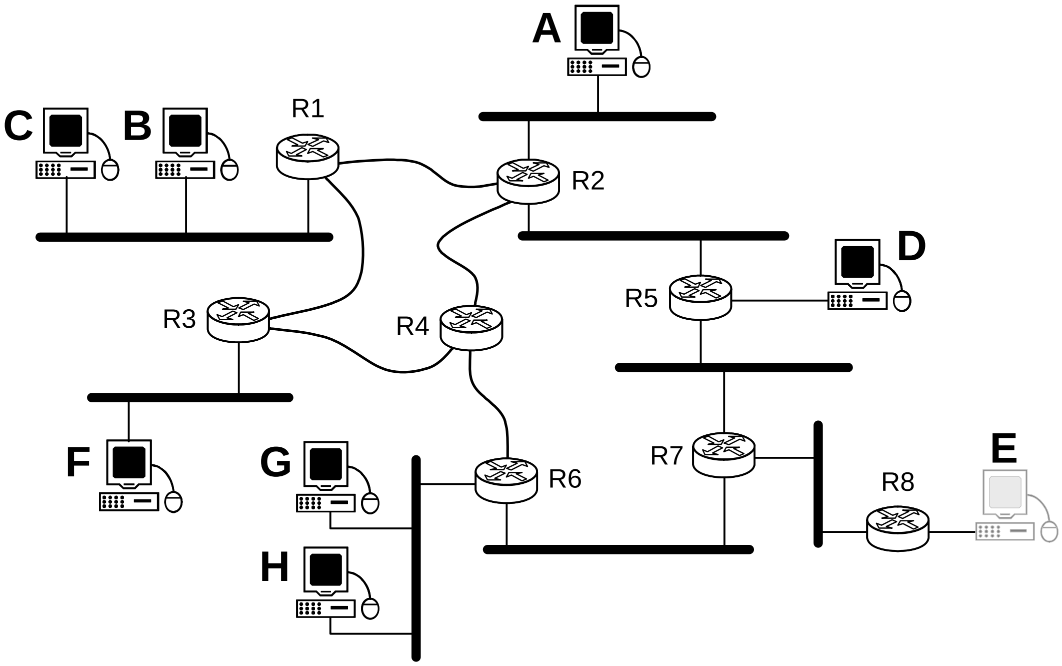
Consider the network shown in the figure below, in which source *D* sends packets to multicast group *Γ*, whose members are all the shown hosts, *including E*. Assume that all links between the connected nodes have cost = 1 (one hop), regardless of whether they are point-to-point or Ethernet links. For example, the link between host *B* and router *R*1 has the cost equal 1. Show step-by-step how to build:

(a) RPF shortest-path multicast tree for the source *D*

Suppose that, after the shortest-path multicast tree for source *D* is already built, host ***E*** leaves the group. For the multicast routing using RPF, how many total packets are generated in the entire network per every packet sent by the source *D* if:

(b) RPF is used without pruning

(c) RPF is used with pruning



Show the work (tracing the packets), not only the final result.

Problem 3.9

Consider the network shown in Problem 3.8, but now using the spanning tree multicast algorithm. Solve the following:

(a) Show step-by-step how to build the group-shared multicast tree assuming that *R*6 is the core node.

(b) How many total packets are forwarded in the entire network per each packet sent by the source *D*?

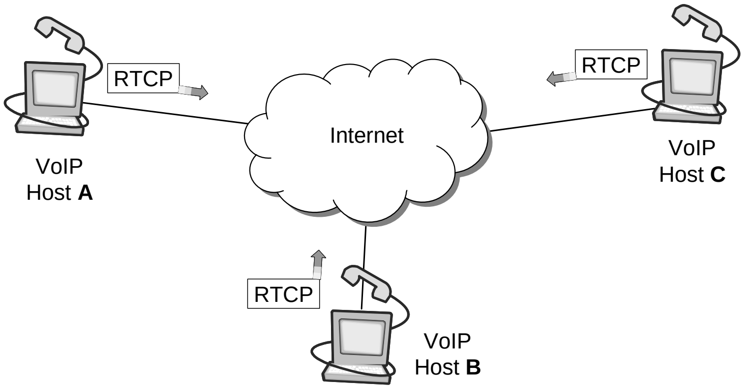
(c) How many packets are forwarded per each packet sent by *D* if host *E* leaves the group *Γ*?

(d) Assume now that *R*6 crashes and *R*1 becomes the new core. Show all the messages that are sent during the old spanning tree teardown and rebuilding a new spanning tree.

Problem 3.10

Problem 3.11

Consider the Internet telephony (VoIP) conferencing session shown in the figure. Assume that each audio stream is using PCM encoding at 64 Kbps. The voice data size is 200 bytes per packet.



Determine the intervals for transmitting RTCP packets for all senders and receivers in the session. Assume that participants will send compound RTCP reports, containing both their Sender Report and Receiver Report.

Problem 3.12

Chapter 4

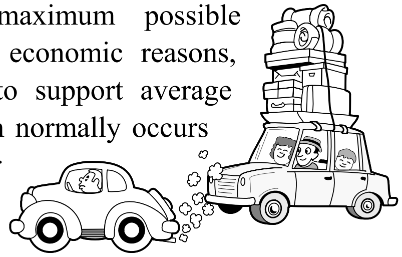
Switching and Queuing Delay Models

Contents

Data packets need two types of services on their way from the source to the final destination that are provided by intermediate nodes, known as *switches* or *routers*:

- *Computation* (or *processing*), which involves formatting packets for the given protocol, i.e., adding guidance information (or headers) to packets, and looking up this information to deliver the packet to its correct destination
- *Communication* (or *transmission*) of packets as bit streams over communication links

A key problem that switches and routers must deal with are the finite physical resources. Network nodes and links are never provisioned to support the maximum traffic rates because that is not economical. We all know that highway networks are not designed to support the maximum possible vehicular traffic. Because of economic reasons, highway networks are built to support average traffic rates. Traffic congestion normally occurs during the rush hours and later subsides. If congestion lasts too long, it can cause a major disruption of travel in the area. Although this is very unpleasant for the travelers and can cause economic loss, it is simply too expensive to provision road networks to avoid such situations altogether. Similar philosophy guides the design of communication networks.



We know from Chapter 1 that using routers helps us avoid the need for total network connectivity with a direct link from every node to every other node. Routers also provide route redundancy, in case of node or link failures. However, there is a tradeoff because of using routers. Figure 4-1 compares the total delay a packet will experience if the source and destination are directly connected against the total delay when an intermediate router relays packets. As seen, the *router introduces both processing and transmission delays* (because of an additional transmission on the

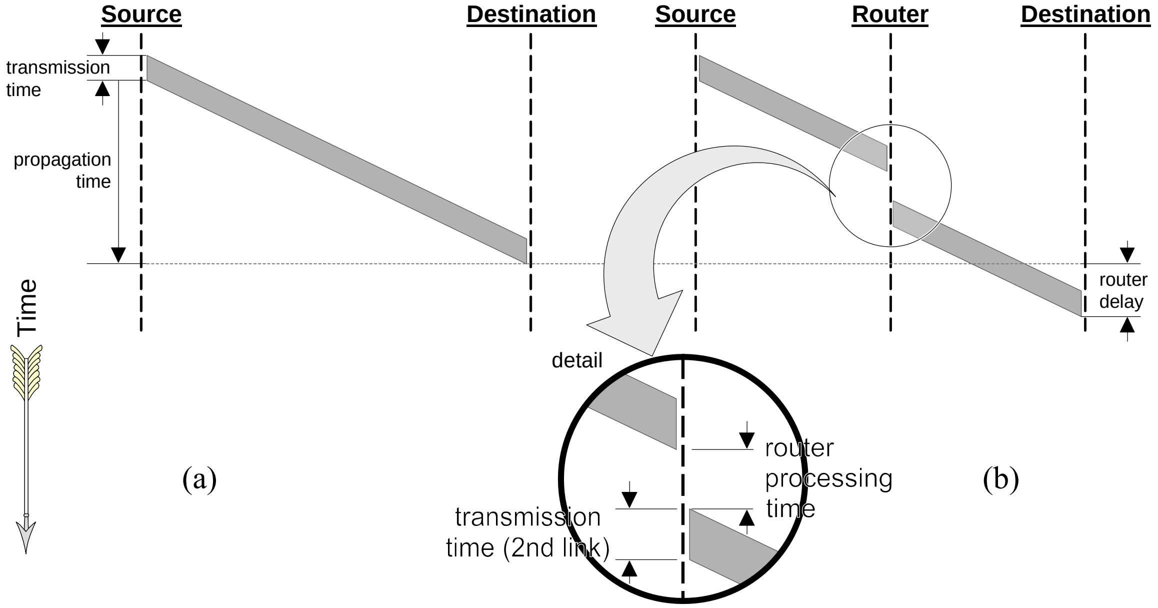


Figure 4-1: Delays in datagram packet switching (or, forwarding, or, routing). (a) Single hop source-to-destination connection without intermediary nodes. (b) Intermediary nodes introduce additionally processing and transmission delays.

second link). Both of these services are offered by physical servers (processing units and communication links) which have limited servicing capacity. When a packet arrives to a server that is already servicing packets that arrived previously, then we have a problem of *contention* for the service. The new packet is placed into a *waiting line* (or *queue*) to wait for its turn. The delay experienced while waiting in line before being serviced is part of the total delay experienced by packets when traveling from source to destination. Generally, packets in a networked system experience these types of delays:

processing + transmission + propagation + queuing

The first three types of delays are described in Section 1.3. This chapter focuses on the last kind of delay, **queuing delays**. Queuing models are used as a prediction tool to estimate this waiting time:

- Computation (queuing delays while waiting for processing)
- Communication (queuing delays while waiting for transmission)

This chapter studies what contributes to routing delays and presents simple analytical models of router queuing delays. Chapter 5 describes techniques to reduce routing delays or redistribute them in a desired manner over different types of data traffic.

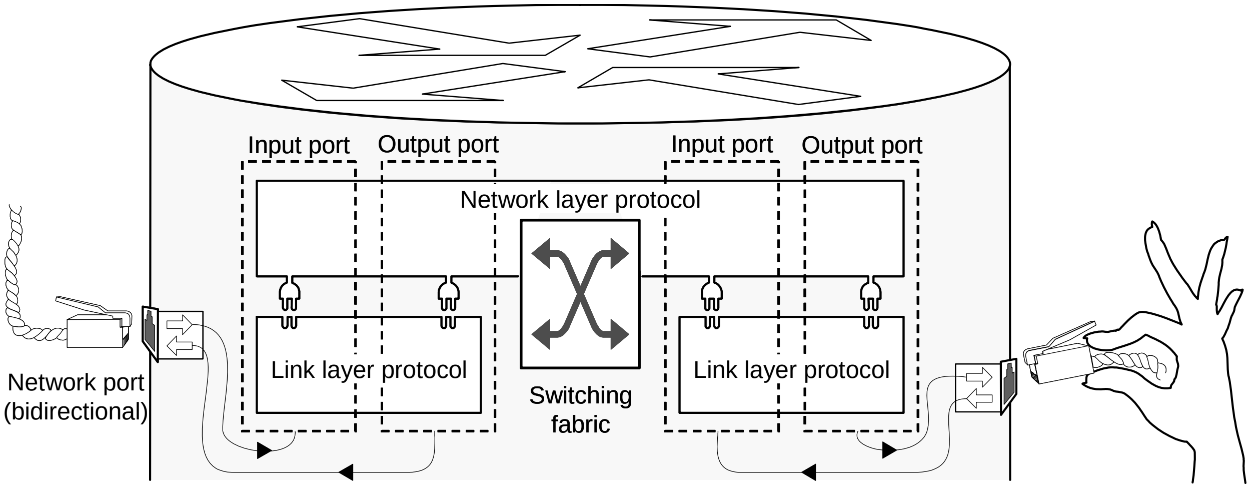


Figure 4-2: Hardware and software components of a router.

4.1 Packet Switching in Routers

Section 1.4 introduced routers and mainly focused on their *control function*, which is building the routing tables and exchanging route advertisements to keep track of a dynamically changing network topology. This section describes the *datapath function* of routers, which is forwarding data packets efficiently toward their destination. A key problem to deal with is the resource limitation. If the router is too slow to move the incoming packets, the packets will experience large delays and may need to be discarded if the router runs out of the memory space (known as buffering space). When packets are discarded too frequently, the router is said to be *congested*. The ability of a router to handle successfully the resource contention is a key aspect of its performance.

Figure 4-2 illustrates key hardware and software components of a router. Section 4.1.1 describes how these components function to forward data packets. Although network ports are bidirectional, it is useful to logically separate input and output ports. Router implements only the bottom two layers of the software protocol stack: link and network layer. Each network port has an associated link-layer protocol, but the network-layer protocol is common for all ports. This property will be explained later with Figure 4-4.

4.1.1 How Routers Forward Packets

Routers have two main functions:

1. **Forwarding or switching packets** (*datapath functions*) that pass through the router. One could think of these functions as using maps to direct the packets to their destinations. These operations are performed very frequently and are most often implemented in special purpose hardware.

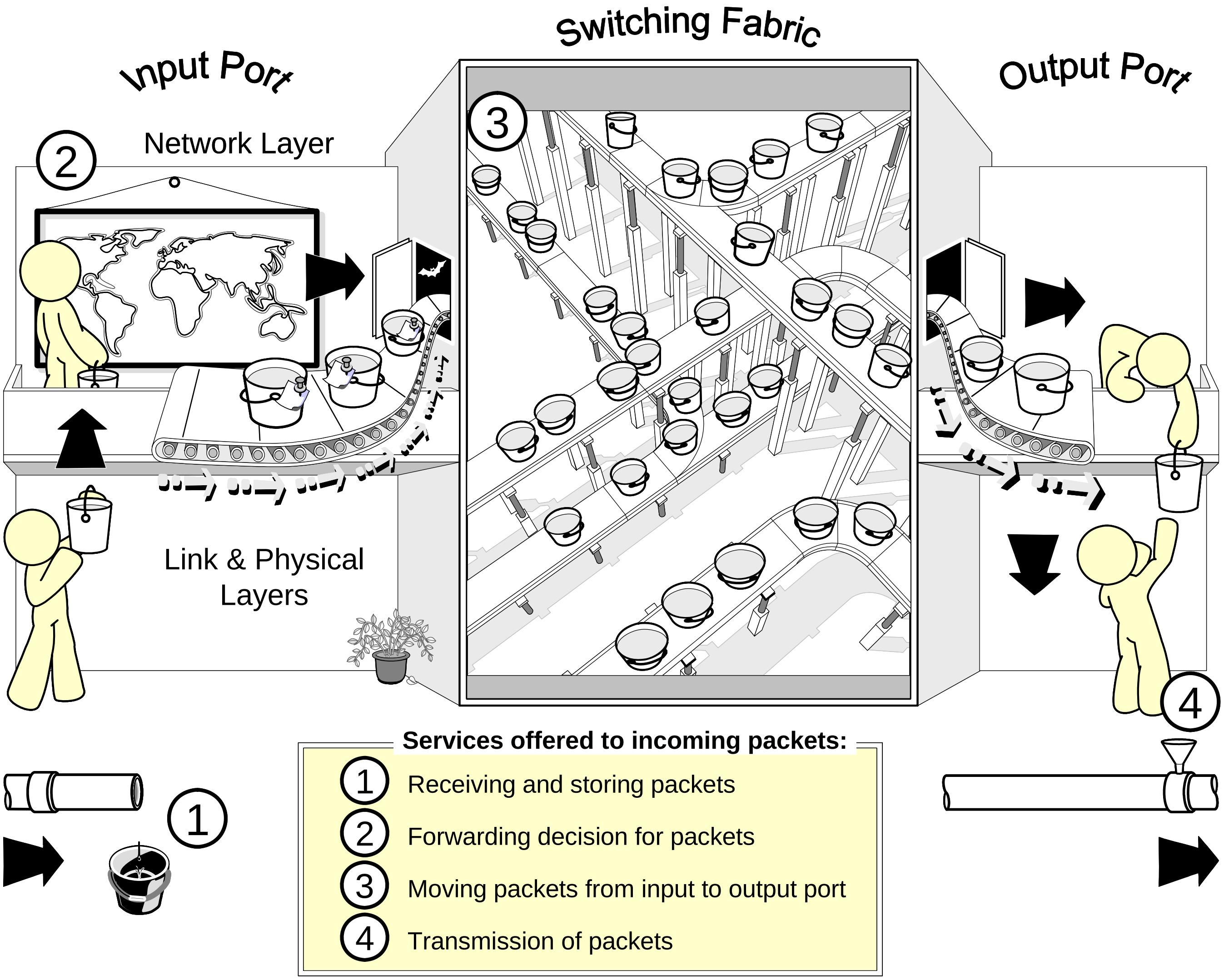
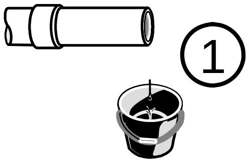

Switching Fabric
Input Port
Output Port
Network Layer
Link & Physical Layers
1
2
3
4
Services offered to incoming packets:
1 Receiving and storing packets
2 Forwarding decision for packets
3 Moving packets from input to output port
4 Transmission of packets

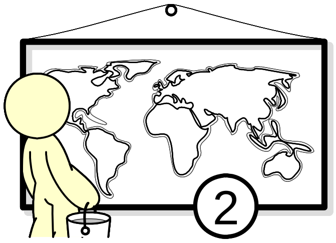
Figure 4-3: How router forwards packets: illustrated are four key services offered to incoming data packets.

2. **Maintaining routing tables** (*control functions*) by exchanging network connectivity information with neighboring routers, as well as system configuration and management. One could think of these functions as surveying and cartography to build the road maps that are used for packet forwarding. These operations are performed relatively infrequently and are invariably implemented in software.

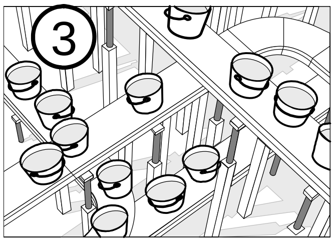
When trying to improve the per-packet performance of a router, we focus on the datapath functions because they must be fast. Routing table maintenance is described in Section 1.4. This chapter focuses on the datapath functions of packet forwarding (or, switching).

The datapath architecture consists of three main units: (1) *input network ports* where incoming packets are received; (2) *switching fabric* that moves packets from input to output ports; and, (3) *output network ports* that transmit packets to an outgoing communication link. Routers offer four key functions to incoming data packets (illustrated in Figure 4-3):

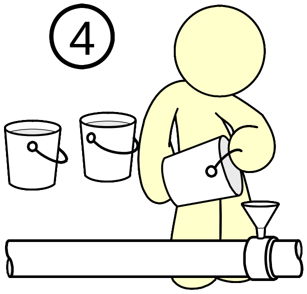




1. A packet is **received and stored** in the local memory on the input port at which the packet arrived.
2. The packet guidance information (destination address, stored in the packet header) is looked up and the **forwarding decision** is made as to which output port the packet should be transferred to.



3. The packet is **moved** across the switching fabric (also known as the *backplane*) to the appropriate output port, as decided in Step 2.
4. The packet is **transmitted** on the output port over the outgoing communication link.



4.1.2 Router Architecture

Figure 1-12 shows protocol layering in end systems and intermediate nodes (switches or routers). Figure 4-4 shows the same layering from a network perspective. Routers run an independent layer-1 (Link layer) protocol for each communication line. A single Network layer (layer-2) is common for all link layers of a router.

The key architectural question in router design is about the implementation of the (shared) Network layer of the router's protocol stack. The Network layer binds together the Link layers and performs packets switching. Link layers are terminating different communication links at the router and they essentially provide data input/output operations. They function independently of one another and are implemented using separate hardware units. They may even run different link-layer protocols on their networks, e.g., PPP or Ethernet (Section 1.5).

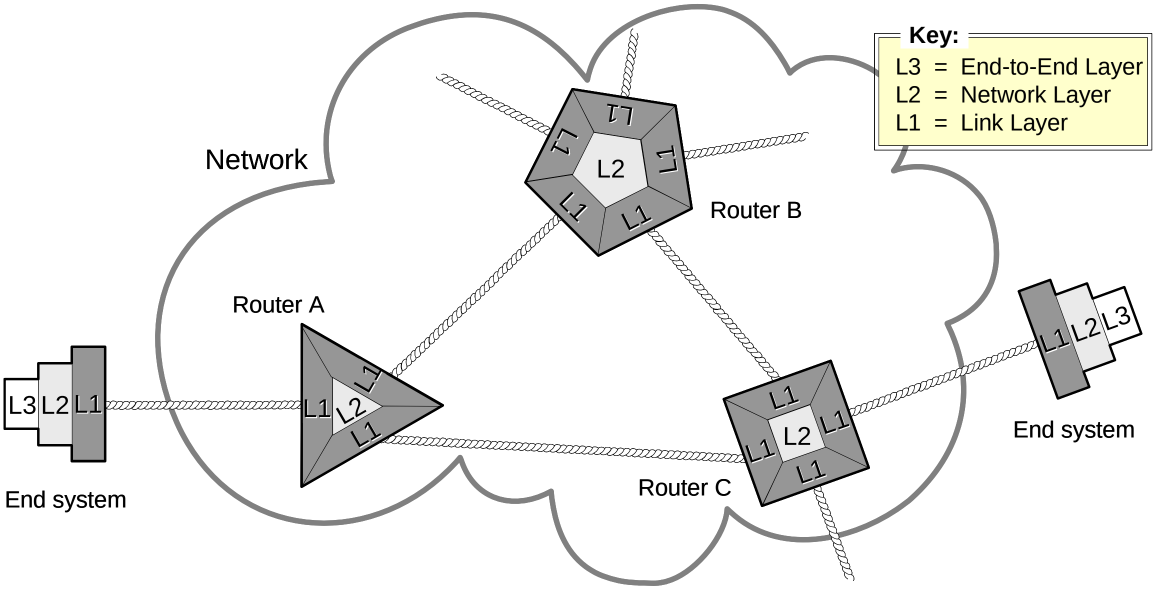


Figure 4-4: Distribution of protocol layers in routers and end systems. Note that end-to-end layer L3 exists only on end nodes, and on routers network layer L2 is *shared* by multiple link layers L1. (Compare to Figure 1-12.)

The router's Network layer must deal simultaneously with many parallel data streams. To achieve high performance, the Network layer could be implemented in parallel hardware. The issues that make the router's network layer design difficult include:

- *Maintaining consistent forwarding tables*: If the Network layer is distributed over parallel hardware units, each unit must maintain its own copy of the forwarding table. Because the forwarding table dynamically changes (as updated by the routing algorithm), all copies must be maintained consistent.
- *Achieving high-speed packet switching*: Once a packet's outgoing port is determined, the packet must be moved as quickly as possible from the incoming to the outgoing port.
- *Reducing queuing and blocking*: If two or more packets cross each other's way, they must be ordered in series, where they move one-by-one while others are waiting for their turn (like cars on a road intersection). The pathways inside the router should be designed to minimize chances for queuing and blocking to occur.

Over the years, different architectures have been used for routers. Particular architectures have been selected based on a number of factors, including cost, number of network ports, required performance, and currently available technology. Detailed implementations of individual commercial routers have generally remained proprietary, but in broad terms, all routers have evolved in similar ways. The evolution in the architecture of routers is illustrated in Figure 4-5.

First-Generation Routers: Central CPU Processor. The original routers in 1980s were built around a conventional computer architecture, as shown in Figure 4-5(a): a shared central bus, with a central CPU, memory, and peripheral *Line Cards* (also known as *Network Interface Cards* or NICs). Each Line Card performs the link-layer function, connecting the router to each of the communication links. The central CPU performs the network-layer function. Packets arriving

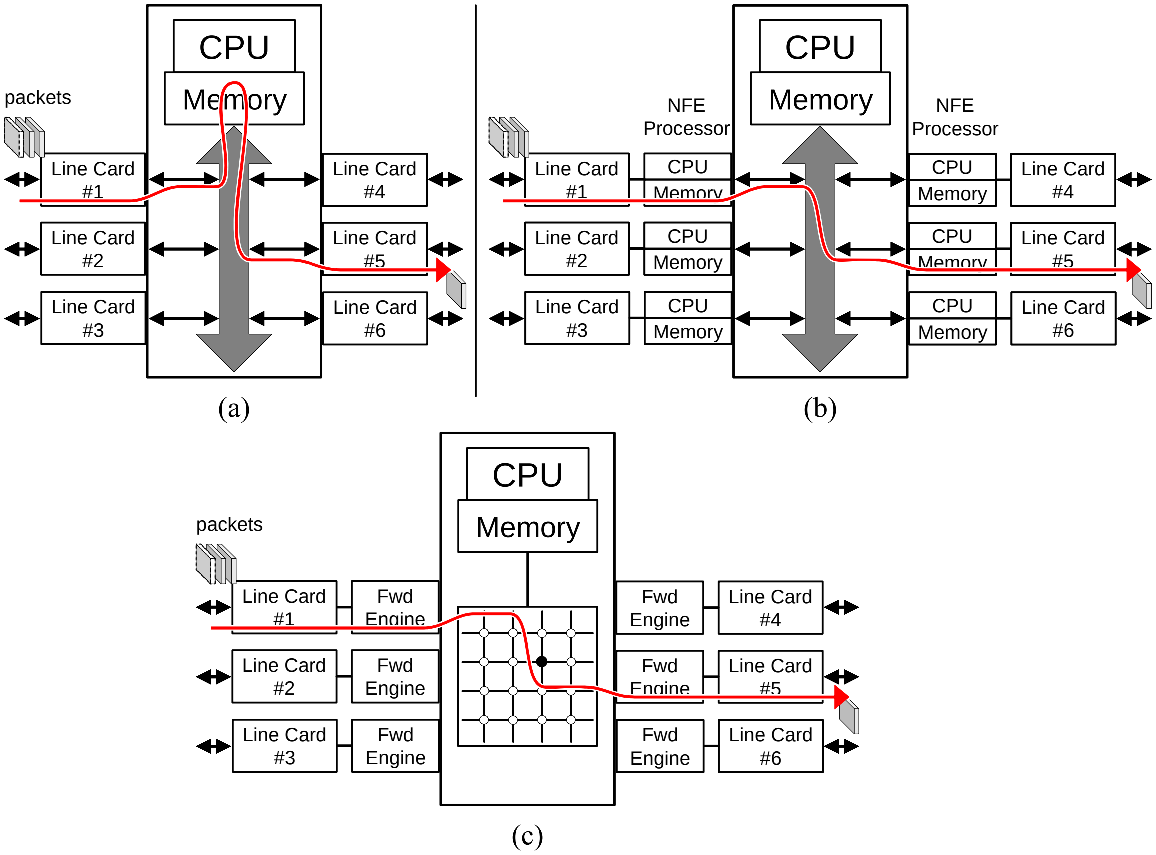


Figure 4-5: The basic architectures of packet-switching processors. (a) Central CPU processor; (b) Parallel network-front-end processors; (c) Switching fabric. The curved line indicates the packet path, from input to output port.

from a link are transferred across the shared bus to the CPU, where a forwarding decision is made. The packet is then transferred again across the bus to its outgoing Line Card, and onto the communication link. Figure 4-6 highlights the datapath of first-generation routers and shows Figure 4-5(a) redrawn in a slightly different way.

Second-Generation Routers: Network Front-end (NFE) Processors. The main limitation of the architecture in Figure 4-5(a) is that the central CPU must process every packet, ultimately limiting the throughput of the system. To increase the system throughput, the architecture in Figure 4-5(b) implements parallelism by placing a separate CPU at each network interface. That is, the link-layer is still implemented in individual Line Cards, but the network-layer function is distributed across several dedicated CPUs, known as *network front-end* (NFE) *processor*s. A local forwarding decision is made in a NFE processor, and the packet is immediately forwarded to its outgoing interface. The central CPU is needed to run the routing algorithm and for centralized system management functions. It also computes the forwarding table and distributes it to the NFE processors.

The architecture in Figure 4-5(b) has higher performance than a first-generation design because the network-layer function is distributed over several NFE processors that run in parallel; and because each packet needs to travel over the bus only once, thus increasing the system throughput. Figure 4-7 highlights the datapath of second-generation routers. However, the

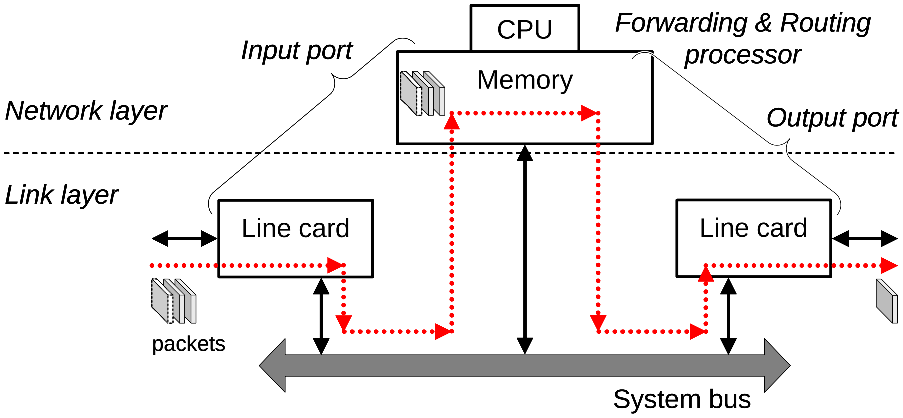


Figure 4-6: Packet datapath for switching via memory. Also shown in Figure 4-5(a).

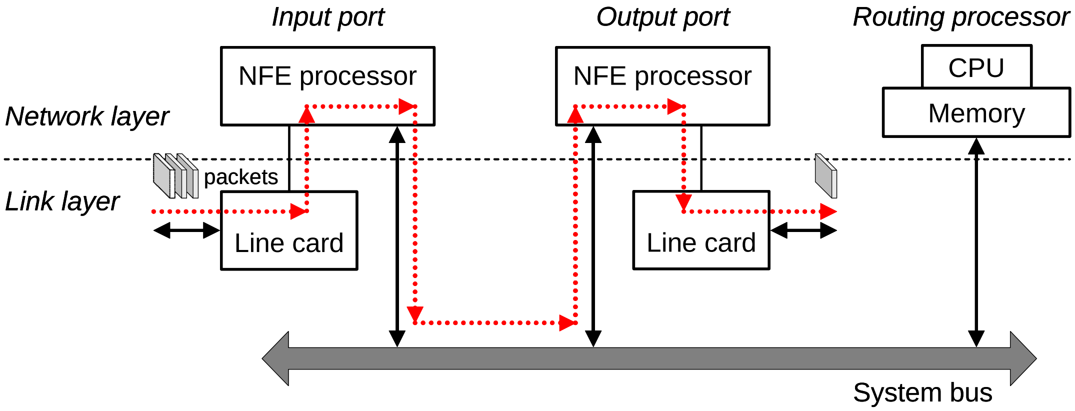


Figure 4-7: Packet datapath for switching via bus. Also shown in Figure 4-5(b).

performance is still limited by two factors. First, forwarding decisions are made in software, and so are limited by the speed of the NFE processor, which is a general purpose CPU. But general purpose CPUs are not well suited to applications in which the data (packets) flow through the system; CPUs are better suited to applications in which data is examined multiple times, thus allowing the efficient use of a cache. Carefully designed, special purpose ASICs can readily outperform a CPU when making forwarding decisions, managing queues, and arbitrating access to the bus. Hence, increasingly CPUs are being replaced by specialized ASICs. The second factor that limits the performance is the use of a shared bus—only one packet may traverse the bus at a time between two Line Cards. Performance can be increased if multiple packets can be transferred across the bus simultaneously. This is the reason that a switch fabric is used in high-end routers.

Third-Generation Routers: Switching Fabric. By introducing a hardware-forwarding engine and replacing the bus with an interconnection network, we reach the architecture shown in Figure 4-5(c). In an interconnection network, multiple Line Cards can communicate with each other simultaneously, greatly increasing the system throughput. Today, the highest performance routers are designed according to this architecture. Switching fabric design is described in Section 4.1.4.

Visit http://en.wikipedia.org/wiki/ASIC for information about Application-Specific Integrated Circuits

Input Ports

The key functions of the input component of network ports are to receive packets and make the forwarding decision. This spans both link and network layers of the protocol stack.

A network port is not the same as a Line Card. A Line Card supports the link-layer functionality of a network port, which is receiving and transmitting packets. A Line Card may also support the network-layer functionality, if the Network Front-End Processor is located on a Line Card as in second-generation routers. However, it may not have any of the network-layer functionality, which is the case in first-generation routers where all network-layer functionality is supported by the central processor and memory.

Output Ports

The key function of output ports is to transmit packets on the outgoing communication links. If packets are arriving at a greater rate than the output port is able to transmit, some packets will be enqueued into waiting lines. The output port may also need to manage how different types of packets are lined up for transmission. This is known as *scheduling* and several scheduling techniques are described in Chapter 5. As with input ports, these functions span both link and network layers of the protocol stack.

4.1.3 Forwarding Table Lookup

We know from Section 1.4.4 that routers use *destination address prefixes* to identify a contiguous range of IP addresses in their routing messages. A destination prefix is a group of IP addresses that may be treated similarly for packet forwarding purposes. Based on its routing table, the router derives its forwarding table, also known as FIB (Forwarding Information Base), and uses it for making forwarding decisions for data packets. Each entry in a forwarding table/FIB represents a mapping from an IP address prefix (a range of addresses) to an outgoing link, with the property that packets from any destination with that prefix may be sent along the corresponding link. Forwarding table entries are called **routes**.

Routing function	Unicast routing	Unicast routing with Types of Service	Multicast routing
Forwarding algorithm	Longest prefix match on destination address	Longest prefix match on destination address **+** exact match on Type of Service	Longest match on source address **+** exact match on source address, destination address, and incoming interface

Figure 4-8: Forwarding algorithms for different routing functions.

The algorithm used by the forwarding component of a router to make a forwarding decision on a packet uses two sources of information: (1) the forwarding table or FIB, and (2) the packet header. Although IP addresses are always the same length, IP prefixes are of variable length. The IP destination lookup algorithm needs to find the *longest prefix match*—the longest prefix in the FIB that matches the high-order bits in the IP address of the packet being forwarded. Longest prefix match used to be computationally expensive. The advances that have been made in longest-match algorithms in recent years have solved the problem of matching.

Packet forwarding decision depends on several parameters, depending on the routing function that needs to be supported (Figure 4-8), such as unicast routing, multicast routing, or unicast routing with Types of Service. Therefore, in addition to the information that controls where a packet is forwarded (next hop), an entry in the forwarding table may include the information about what resources the packet may use, such as a particular outgoing queue that the packet should be placed on (known as *packet classification*, to be described later). Forwarding of unicast packets requires longest prefix match based on the network-layer destination address. Unicast forwarding with Types of Service requires the longest match on the destination network-layer address, plus the exact match (fixed-length match) on the Differentiated Services bits carried in the network-layer header (Figure 1-39). Forwarding of multicast packets requires longest match on the source network-layer address, plus the exact match (fixed-length match) on both source and destination addresses, where the destination address is the multicast group address.

For the purposes of multicast forwarding, some entries in the forwarding table/FIB may have multiple subentries. In multicast, a packet that arrives on one network interface needs to be sent out on multiple outgoing interfaces that are identified in subentries of a FIB record.

4.1.4 Switching Fabric Design

Problems related to this section: ?? → Problem 4.7

Switch Design Issues:

• Switch contention occurs when several packets are crossing each other's path – switch cannot support arbitrary set of transfers;

• Complex rearranging of the timetable for packet servicing (known as *scheduling*) is needed to avoid switch contention;

• High clock/transfer rate needed for bus-based design (first- and second-generation routers);

• Packet queuing (or, buffering) to avoid packet loss is needed when the component that provides service (generally known as "server") is busy;

Example switch fabrics include:

• System bus (first- and second-generation routers)

• Crossbar

• Banyan network

Banyan networks and other interconnection networks were initially developed to connect processors in multiprocessor computers. They typically provide lower capacity than a complete crossbar.

Switching fabric may introduce different types of packet blocking. For example, if two or more packets at different inputs want to cross the switching fabric simultaneously towards the same output, then these packets experience *output blocking*. When one packet is heading for an idle port, but in front of it (in the same waiting line/queue) is another packet headed for a different output port that is currently busy, and the former packet must wait until the latter departs, then the former packet experiences *head-of-line blocking*. Find more information about packet blocking in Section 4.1.5.

Crossbar

The simplest switch fabric is a crossbar, which is a matrix of pathways that can be configured to connect any input port to any output port. An $N \times N$ crossbar has N input buses, N output buses, and N^2 crosspoints, which are either ON or OFF. If the (i, j) crosspoint is on, the i^{th} input port is connected to the j^{th} output port.

A crossbar needs a switching timetable, known as *switching schedule*, that tells it which inputs to connect to which outputs at a given time. If packets arrive at fixed intervals then the schedule can be computed in advance. However, in the general case, the switch has to compute the schedule while it is operating.

If packets from all N inputs are all heading towards different outputs then crossbar is N times faster than a bus-based (second-generation) switch. However, if two or more packets at different inputs want to go to the same output, then crossbar suffers from "output blocking" and as a result, it is not fully used. In the worst-case scenario, each output port must be able to accept packets from all input ports at once. To avoid output blocking, each output port would need to have a memory bandwidth equal to the total switch throughput, i.e., $N \times$ input port datarate. In reality, sophisticated designs are used to address this issue with lower memory bandwidths.

Banyan Network

Banyan network is a so-called *self-routing switch fabric*, because each switching element forwards the incoming packets based on the packet's tag that represents the output port to which

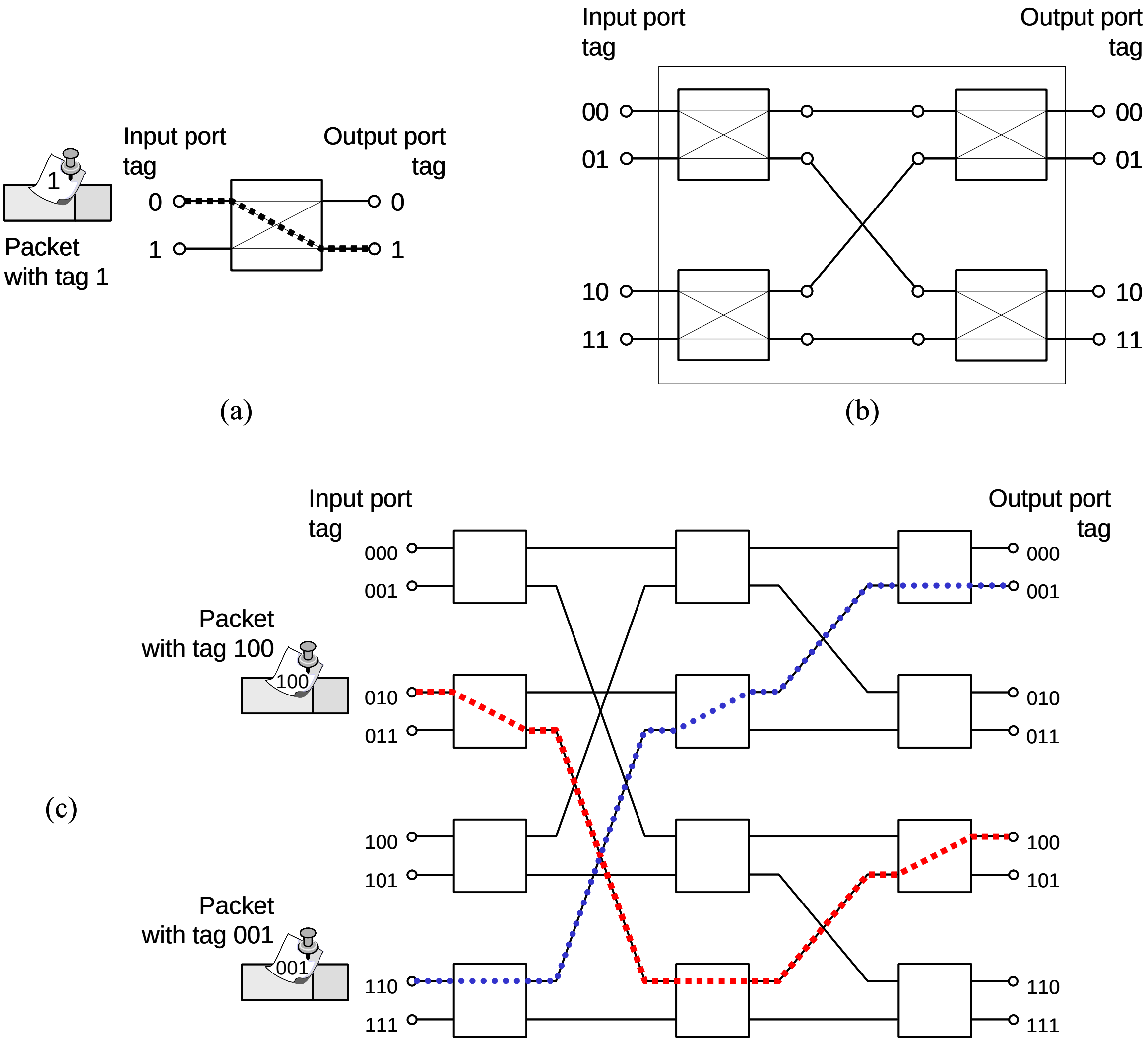


Figure 4-9: Banyan switch fabric. (a) A Banyan switching element is a 2 × 2 switch that moves packets either to output port 0 (upper port) or output port 1 (lower port), depending on the packet's tag. (b) A 4 × 4 Banyan network is composed from four 2 × 2 switching elements. (c) An 8 × 8 Banyan network is composed from twelve 2 × 2 switching elements.

this packets should go. The input port looks up the packet's outgoing port in the forwarding table based on packet's destination, and *tags* the packet with a binary representation of the output port. A Banyan switch fabric is organized in a hierarchy of switching elements. A switching element at level *i* checks the *i*th bit of the tag; if the bit is 0, the packet is forwarded to the upper output, and otherwise to the lower output. Therefore, the tag can be considered a self-routing header of the packet, for routing the packet inside the switching fabric. The tag is removed at the output port before the packet leaves the router.

The building block of a Banyan network is a 2 × 2 switch, i.e., a switch with two inputs and two outputs. The upper input and output ports are labeled with 0 and the lower input and output ports are labeled with 1. This switch moves packets based on a single-bit tag. For example, in Figure 4-9(a) a packet labeled with tag "1" arrives at input port 0 of a 2 × 2 switch. The switch directs the packet to the lower output port (the output port 1). To create a 4 × 4 switch, we need four

2 × 2 switching elements placed in a grid as in Figure 4-9(b). First we take a pair of 2 × 2 switches and label them 0 and 1. Then we take another pair of 2 × 2 switches and place them before the first two. When a packet enters a switching element in the first stage, it is sent to the 2 × 2 switch labeled 0 if the first bit of the packet's tag is 0. Otherwise, it is sent to the switch labeled 1. To create an 8 × 8 switch, we need two 4 × 4 switches and four 2 × 2 switching elements placed in a grid as in Figure 4-9(c). Again, we label the two 4 × 4 switches as 0 and 1. The four 2 × 2 switching elements are placed before the 4 × 4 switches, and they send the packets to the corresponding 4 × 4 switch based on the first bit of the packet's tag. (Note that there are several equivalent 8 × 8 Banyan switches, only one of which is shown in Figure 4-9(c).)

If two incoming packets on any 2 × 2 switching element are heading to the same output port of this element, they *collide* and *block* at this element. For example, if two packets with tags "000" and "010" arrived at the input ports "000" and "001" in Figure 4-9(c), then they will collide at the first stage (in the upper-left corner 2 × 2 switching element), because they both need to go to the same output of this switching element. The switching element discards both of the colliding packets. Because packet loss is not desirable, we need to either prevent collisions or deal with them when they happen. In either case, instead of loss, the packet experiences delay while waiting for its turn to cross the switching fabric.

One option is to deal with collisions when they happen. This option requires a memory buffer for storing packets within the switching element. One of the colliding packets is transferred to the requested direction, while the other is stored in the buffer and sent in the subsequent cycle. This design is called *internal queuing* in Section 4.1.5. In the worst-case scenario for input-traffic pattern, the internal buffer size must be large enough to hold several colliding packets.

Another option is to prevent collisions from happening. One way of preventing collisions is to, before sending a packet from the input port, *check* whether its path across the switching fabric is available.

An alternative way of preventing collisions is by choosing the order in which packets appear at the input of the switching fabric. Obviously, the router cannot choose the input port at which a particular packet will arrive—packets arrive along the links depending on the upstream nodes that transmitted them. What can be done is to insert an additional network (known as *sorting network*) before a Banyan network, which rearranges the packets so that they are presented to the Banyan network in an order that avoids collisions. This is what a Batcher network does.

Batcher-Banyan Network

A Batcher network is a hardware network that takes a list of numbers and sorts them in the ascending order. Again, we assume that packets are tagged at the input ports at which they arrive. A tag is a binary representation of the output port to which the packet needs to be moved. A Batcher network sorts the packets that are currently at input ports into a non-decreasing order of their tags.

Figure 4-10 shows several Batcher sorting networks. To get an intuition why this network will correctly sort the inputs, consider Figure 4-10(b). The first four comparators will "sink" the largest value to the bottom and "lift" the smallest value to the top. The final comparator simply sorts out the middle two values.

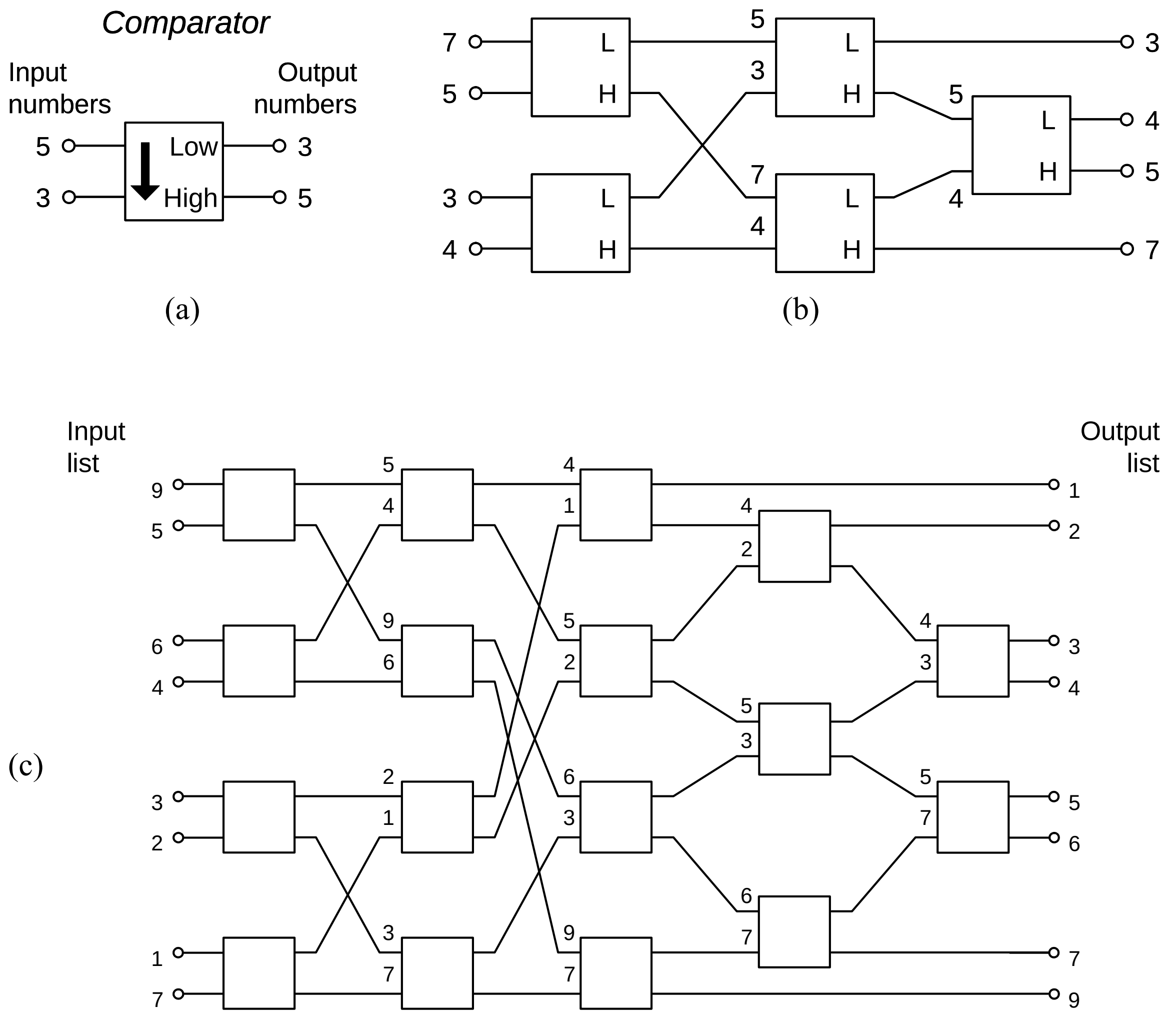


Figure 4-10: Batcher network. (a) A comparator element output the smaller input value at the upper output and the greater input value at the lower output. (b) A 4 × 4 Batcher network is composed five comparators. (c) An 8 × 8 Batcher network is composed from twelve comparators.

A combined Batcher-Banyan network is collision-free only when there are no packets heading for the same output port. Because there may be duplicates (tags with the same output port number) or gaps in the sequence, an additional network or special control sequence is required. Figure 4-11 shows a trap network and a shuffle-exchange network (or, concentrator) which serve to remove duplicates and gaps. In order to eliminate the packets for the same output, a *trap network* is required at the output of the Batcher sorter. Because packets are sorted by the Batcher sorter, the packets for the same output can be checked by comparison with neighboring packets. The trap network performs this comparison, and selects only one packet per output. Duplicates are trapped and dealt with separately.

One way to deal with the trapped duplicates is to store them and recirculate back to the entrance of the Batcher network, so in the next cycle they can compete with incoming packets. This requires that at least half the Batcher's inputs be reserved for recirculated packets (to account for the worst-case scenario).

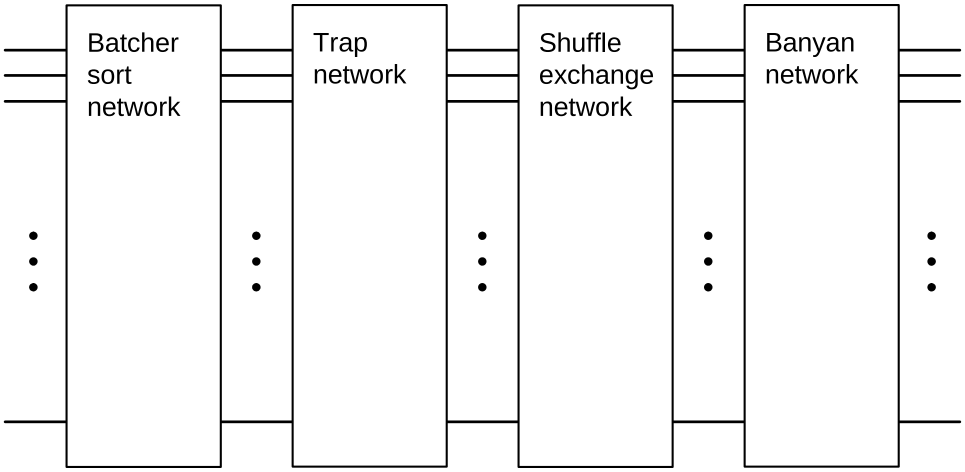


Figure 4-11: Batcher-Banyan network. Internal blocking is avoided by sorting the inputs to a Banyan network. A trap network and a shuffle-exchange network remove duplicates and gaps.

An alternative is to take the duplicates through multiple Banyan networks, where each packet that wants to go to the same output port is presented to a separate Banyan network.

Even when duplicates are removed, unselected and eliminated packets (gaps in the input sequence) generate empty inputs for the Banyan network. These gaps cause collisions in the Banyan network even if all packets are heading to different outputs. For example, consider an 8 × 8 Batcher-Banyan network with four packets heading to outputs 0, 0, 0, and 1, respectively. Although two of the three packets for the output 0 are trapped, the remaining two packets still collide in the second stage of the Banyan. To solve this problem, a *shuffle-exchange network* (or, *concentrator*) is required (Figure 4-11). In order to eliminate empty inputs, packets are shifted and the conflict is avoided. Although the role of the concentrator is just shifting the packets and eliminating empty inputs, the implementation is difficult because the number of packets that will be trapped cannot be predicted. Usually, special control sequence is introduced, or a Batcher sorter is used again as the concentrator.

4.1.5 Where and Why Queuing Happens

Problems related to this section: Problem 4.9 → ??

Queuing happens when customers are arriving at a rate that is higher than the server is able to service. In routers, waiting lines (queues) may be formed for any of the three services shown in illustrated in Figure 4-3, except for receiving packets at the input port. Packet queuing in routers is also known as **switch buffering**. The router needs memory allow for buffering, i.e., storing packets while they are waiting for service. By adopting different designs, the router architect can control where the buffering will occur.

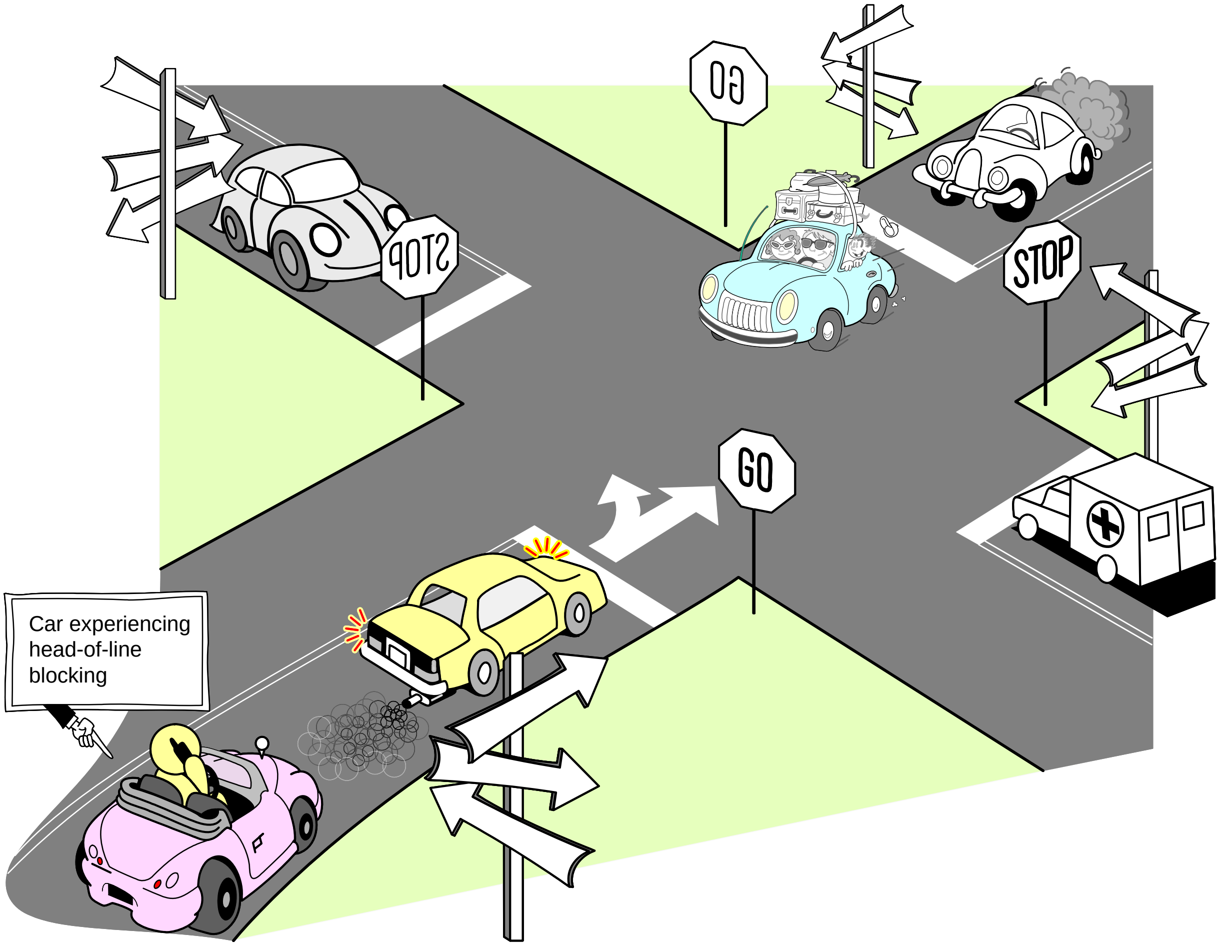


Figure 4-12: Illustration of forwarding issues by a road intersection analogy.

Before considering how queuing occurs in packet switches, let us consider the analogy with vehicular traffic, illustrated in Figure 4-12. The car in the lower left corner could go if it were not for the car in front of it that wishes to make the left turn but cannot because of the cars arriving in the parallel lane from the opposite direction. We say that the lower-left-corner car experiences **head-of-line** (HOL) **blocking**. A queue of cars will be formed at an entrance to the road intersection (or, “input port”) because the front car cannot cross the intersection area (or, “switching fabric”).

Another cause for queuing occurs the access to the intersection area (or, “switching fabric”) has to be serialized. In Figure 4-12, the cars crossing from upper left corner to lower right corner and vice versa must wait for their turn because the intersection area is busy. Note that we need an “arbiter” to serialize the access to the intersection area, and STOP/GO signals in Figure 4-12 serve this purpose. The corresponding queuing occurs in a router when the switching fabric has insufficient capacity to support incoming packet traffic. The queuing occurs at the input port and it is known as *input queuing* (or, *input buffering*) or in the switch fabric (*internal queuing*).

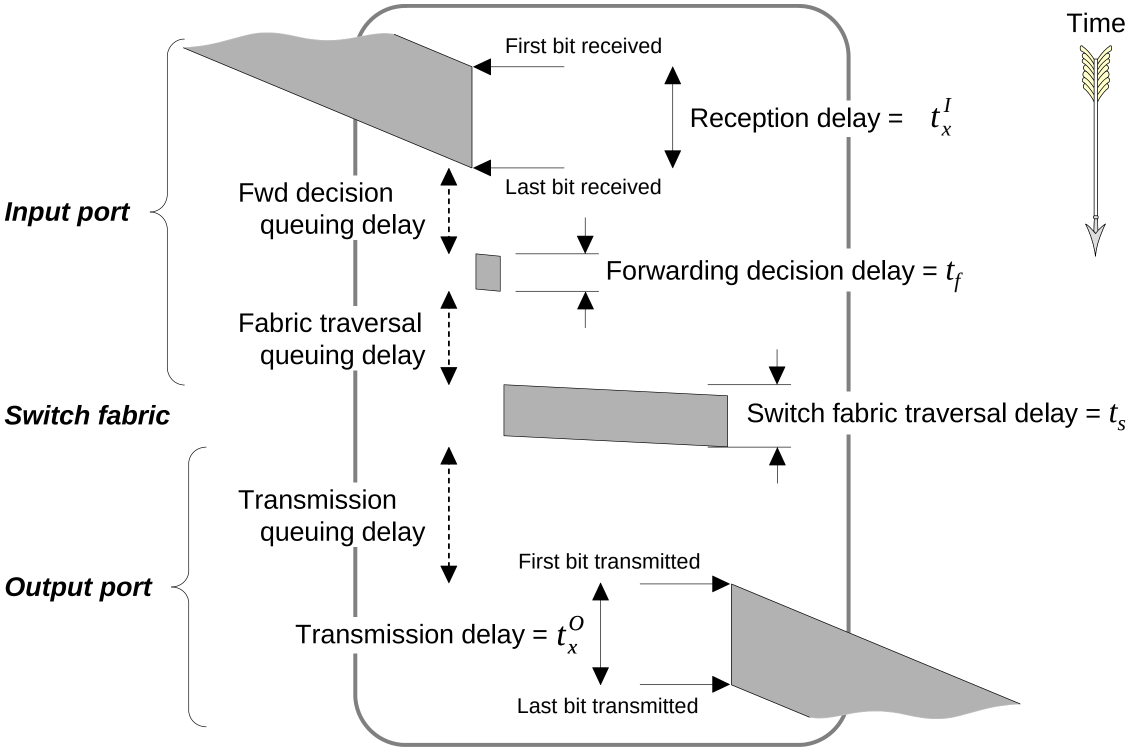

Time
First bit received
Reception delay = t_x^I
Last bit received
Input port
Fwd decision queuing delay
Forwarding decision delay = t_f
Fabric traversal queuing delay
Switch fabric
Switch fabric traversal delay = t_s
Transmission queuing delay
Output port
First bit transmitted
Transmission delay = t_x^O
Last bit transmitted

Figure 4-13: Components of delay in data packet forwarding.

Yet another reason for queuing occurs when an outgoing road cannot support the incoming traffic and the corresponding intersection exit becomes congested. In this case, even if an incoming car does not experience head-of-line blocking and it has a GO signal, it still bust wait because the exit is congested. The corresponding queuing occurs in a router when the outgoing communication line has insufficient capacity to support incoming packet traffic. The queuing may occur at the input port (*input queuing*), in the switch fabric (*internal queuing*), or at the output port (*output queuing*).

Figure 4-13 summarizes the datapath delays in a router. Some of these delays may be negligible, depending on the switch design. As already noted, queuing may also occur in the switch fabric (*internal queuing*), which is not shown in Figure 4-13.

Input Queuing

In switches with pure *input queuing* (or, *input buffering*), packets are stored at the input ports and released when they win access to both the switching fabric and the output line. An arbiter decides the timetable for accessing the fabric depending on the status of the fabric and the output lines (Figure 4-14). Because the packets leaving the input queues are guaranteed access to the fabric and the output line, there is no need for an output queue.

The key advantage of input queuing is that links in the switching fabric (and the input queues themselves) need to run at the speed of the input communication lines. For a router with *N* input ports and *N* output ports, only the arbiter needs to run *N* times faster than the input lines.

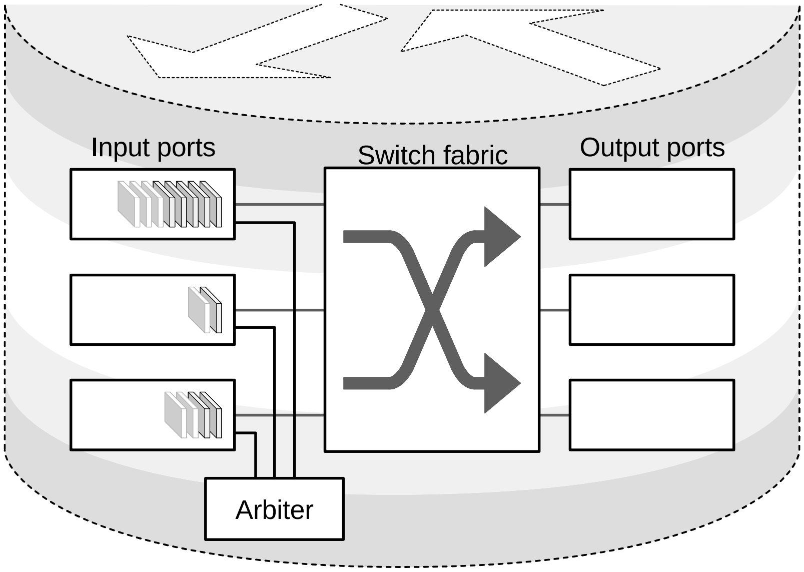


Figure 4-14: A router with input queuing. The arbiter releases a packet from an input queues when a path through the switch fabric and the output line is available.

The problem with input-queued routers is that if input queues are served in a first-come-first-served (FCFS) order, then a head-of-line packet destined for a busy output blocks the packets in the queue behind it. This is known as **head-of-line** (HOL) **blocking**. HOL blocking can be prevented if packets are served according to a timetable different from FCFS. Scheduling techniques that prevent HOL blocking are described in Chapter 5.

Output Queuing

In switches with pure *output queuing* (or, *output buffering*), packets are stored at the output ports. Incoming packets immediately proceed through the switching fabric to their corresponding output port. Because multiple packets may be simultaneously heading towards the same output port, the switch must provide a switching-fabric speedup proportional to the number of input ports. In a switch with *N* input ports, each output port must be able to store *N* packets in the time it takes a single packet to arrive at an input port. This makes the hardware for the switching fabric and output queues more expensive than in input queuing.

Note that output queue do not suffer from head-of-line blocking—the transmitter can transmit the packets in the packets according to any desired timetable. The output port may rearrange its output queue to transmit packets according to a timetable different from their arrival order. We will study scheduling disciplines in Chapter 5.

Internal Queuing

- Head of line blocking
- What amount of buffering is needed?

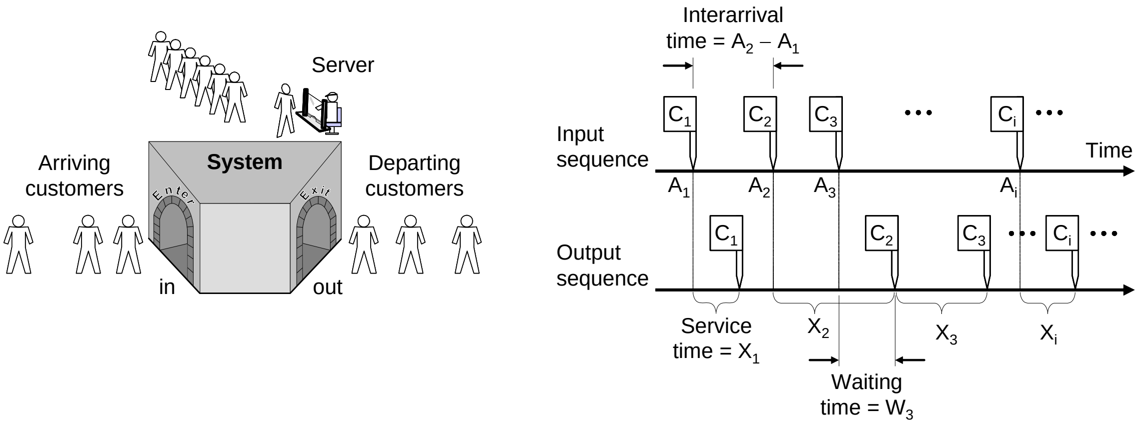


Figure 4-15: General service delay model: customers are delayed in a system for their own service time plus a possible waiting time. Customer 3 has to wait in line because a previous customer is being serviced at customer 3's arrival time.

4.2 Queuing Models

Queuing introduces latency, and the potential for packet loss if a queue overflows. When traffic patterns are bursty, the queuing-induced latency varies unpredictably from packet to packet, manifesting itself as jitter (delay variability) in the affected traffic streams. Modeling queuing processes is important for understanding the problem and designing the solutions.

General Server

A general service model is shown in Figure 4-15. Customers arrive in the system at a certain rate. It is helpful if the arrival times happened to be random and independent of the previous arrivals, because such systems can be well modeled. The server services customers in a certain order, the simplest being their order of arrival, also called *first-come-first-served* (FCFS). Every physical processing takes time, so a customer *i* takes a certain amount of time to service, the service time denoted as X_i.

Most commonly used performance measures are: (1) *the average number of customers in the system*; and, (2) *average delay per customer*. A successful method for calculating these parameters is based on the use of a *queuing model*. Figure 4-16 shows a simple example of a queuing model, where the system is represented by a single-server queue. The *queuing time* is the time that a customer waits before it enters the service. Figure 4-17 illustrates queuing system parameters on an example of a bank office with a single teller.

Why Queuing Happens?

Queuing occurs because of the server's inability to process the customers at the rate at which they are arriving. When a customer arrives at a busy server, it enters a waiting line (queue) and waits on its turn for processing. The critical assumption here is the following:

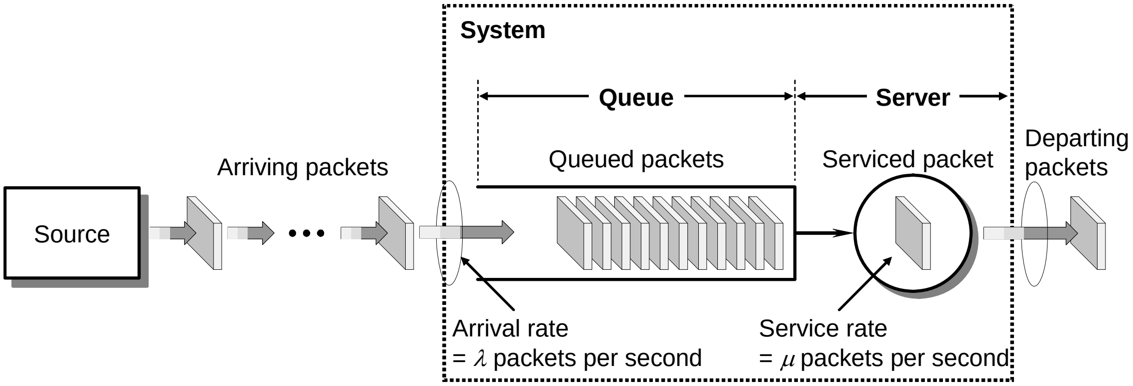


Figure 4-16: Simple queuing system with a single server.

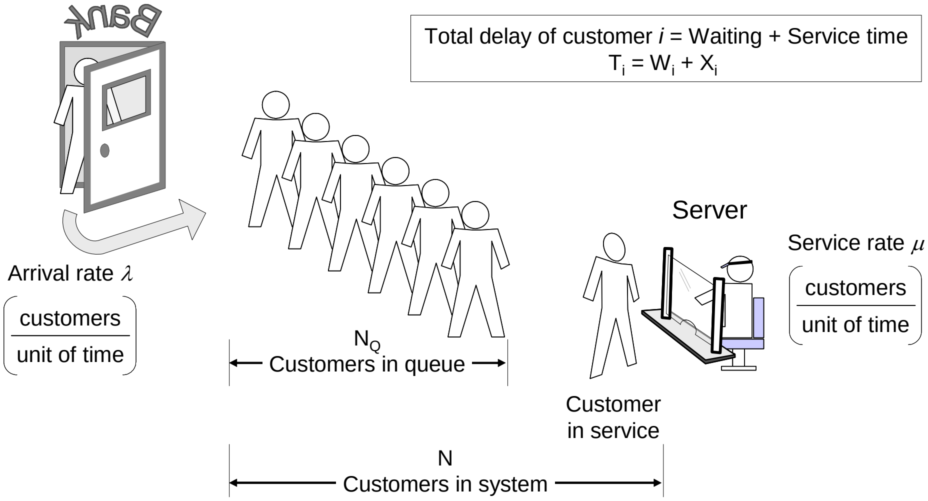


Figure 4-17: Illustration of queuing system parameters.

$$\text{Average arrival rate} \le \text{Maximum service rate}$$

Otherwise, the queue length would grow unlimited and the system would become meaningless because some customers would have to wait infinite amount of time to be serviced. A corollary of this requirement is that queuing is an artifact of irregular customer arrival patterns, sometimes being too many, sometimes very few. Customers arriving in groups create queues. Had they been arriving "individually" (well spaced), allowing the server enough time to process the previous one, there would be no queuing. The arrival pattern where the actual arrival rate is equal to the average one would incur no queuing delays on any customer.

This is illustrated in Figure 4-18 where we consider a bank teller that can service five customers per hour, $\mu = 5\,\dfrac{\text{customers}}{\text{hour}}$, on average. This means, serving one customer takes 12 minutes, on average. Assume that for a stretch of time all arriving customers take 12 minutes to be served and that three customers arrive as shown in the figure. Although the server capacity is greater than the arrival rate, the second and third customers still need to wait in line before being served, because their arrivals are too closely spaced. If the customers arrived spaced according to their departure times at the same server, there would be no queuing delay for any customer. However, if this

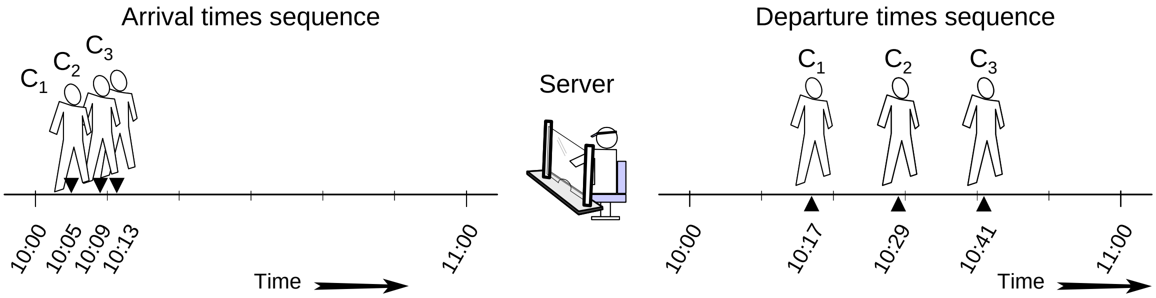


Figure 4-18: Illustration of how queues are formed. The server can serve 5 customers per hour and only 3 customers arrive during an hour period. Although the server capacity is greater than the arrival rate, some customers may still need to wait before being served, because their arrivals are too closely spaced.

sequence arrived at a server that can service only four customers per hour, again there would be queuing delays. Thus, having a server with service rate greater than the arrival rate is no guarantee that there will be no queuing delays. In summary, queuing results because packet arrivals cannot be preplanned and provisioned for—it is too costly or physically impossible to support peak arrival rates.

Note also that in the steady state, the average departure rate equals the average arrival rate. Server utilization = (arrival rate / max. service rate)

Communication Channel

Queuing delay is the time it takes to transmit the packets that arrived earlier at the network interface. Packet's service time is its transmission time, which is equal to L/C, where L is the packet length and C is the server capacity. In case of packet transmission, "server capacity" is the outgoing channel capacity. The average queuing time is typically a few transmission times, depending on the load of the network.

→()_____)→ delay $\propto$ capacity $^{-1}$

Another parameter that affects delay is *error rate*—errors result in retransmissions, which significantly influence the delay. Reliable vs. unreliable (if error correction is employed + Gaussian channel)

We study what are the sources of delay and try to estimate its amount. In a communication system, main delay contributors are (see Section 1.3):

- Processing (e.g., conversion of a stream of bytes to packets or packetization, compression/fidelity reduction, encryption, switching at routers, etc.)
- Queuing, due to irregular packet arrivals, sometimes too many, sometimes just few
- Transmission, converting the digital information into analog signals that travel the medium
- Propagation, signals can travel at most at the speed of light, which is finite
- Errors or loss in transmission or various other causes (e.g., insufficient buffer space in routers, recall Figure 2-14 for TCP), resulting in retransmission

Errors result in retransmission. For most links, error rates are negligible, but for multiaccess links, particularly wireless links, they are significant.

Processing may also need to be considered to form a queue if this time is not negligible.

Give example of how delay and capacity are related, see Figure from Peterson & Davie, or from [Jeremiah Hayes 1984].

Notation

Some of the symbols that will be used in this chapter are defined as follows (see also Figure 4-17):

$A(t)$ Counting process that represents the total number of tasks/customers that arrived from 0 to time t, i.e., $A(0) = 0$, and for $s < t$, $A(t) - A(s)$ equals the number of arrivals in the time interval $(s, t]$

λ Arrival rate, i.e., the average number of arrivals per unit of time, in steady state

$N(t)$ Number of tasks/customers in the system at time t

N Average number of tasks/customers in the system (this includes the tasks in the queue and the tasks currently in service) in steady state

N_Q Average number of tasks/customers waiting in queue (but not currently in service) in steady state

μ Service rate of the server (in customers per unit time) at which the server operates when busy

X_i Service time of the i^{th} arrival (depends on the particular server's service rate μ and can be different for different servers)

T_i Total time the i^{th} arrival spends in the system (includes waiting in queue plus service time)

T Average delay per task/customer (includes the time waiting in queue and the service time) in steady state

W Average queuing delay per task/customer (not including the service time) in steady state

ρ Rate of server capacity utilization (the fraction of time that the server is busy servicing a task, as opposed to idly waiting)

4.2.1 Little's Law

Imagine that you perform the following experiment. You are frequently visiting your local bank office and you always do the following:

1. As you walk into the bank, you count how many customers are in the room, including those waiting in the line and those currently being served. Let us denote the average count as *N*. You join the queue as the last person; there is no one behind you.

2. You will be waiting W time, on average, and then it will take X time, on average, for you to complete your job. The expected amount of time that has elapsed since you joined the queue until you are ready to leave is $T = W + X$. During this time T new customers will arrive at an arrival rate λ.
3. At the instant you are about to leave, you look over your shoulder at the customers who have arrived after you. These are all new customers that have arrived while you were waiting or being served. You will count, on average, $\lambda \cdot T$ customers in the system.

If you compare the average number of customers you counted at your arrival time (N) and the average number of customers you counted at your departure time ($\lambda \cdot T$), you will find that they are equal. This is called *Little's Law* and it relates the average number of tasks in the system, the average arrival rate of new tasks, and the average delay per task:

Average number of tasks in the system = Arrival rate × Average delay per task

$$N = \lambda \cdot T \tag{4.1a}$$

I will not present a formal proof of this result, but the reader should glean some intuition from the above experiment. For example, if customers arrive at the rate of 5 per minute and each spends 10 minutes in the system, Little's Law tells us that there will be 50 customers in the system on average.

The above observation experiment essentially states that the number of customers in the system, on average, does not depend on the time when you observe it. A stochastic process is **stationary** if *all* its statistical properties are invariant with respect to time.

Another version of Little's Law is

$$N_Q = \lambda \cdot W \tag{4.1b}$$

The argument is essentially the same, except that the customer looks over her shoulder as she enters service, rather than when completing the service. A more formal discussion is available in [Bertsekas & Gallager, 1992].

Little's Law applies to any system in equilibrium, as long as nothing inside the system is creating new tasks or destroying them. Of course, to reach an equilibrium state we have to assume that the traffic source generates infinite number of tasks.

Using Little's Law, given any two variables, we can determine the third one. However, in practice it is not easy to get values that represent well the system under consideration. The reader should keep in mind that N, T, N_Q, and W are random variables; that is, they are not constant but have probability distributions. One way to obtain those probability distributions is to observe the system over a long period of time and acquire different statistics, much like traffic observers taking tally of people or cars passing through a certain public spot. Another option is to make certain assumptions about the statistical properties of the system. In the following, we will take the second approach, by making assumptions about statistics of customer arrivals and service times. From these statistics, we will be able to determine the expected values of other parameters needed to apply Little's Law.

Kendall's notation for queuing models specifies six factors:

Arrival Process / Service Proc. / Num. Servers / Max. Occupancy / User Population / Scheduling Discipline

1. *Arrival Process* (first symbol) indicates the statistical nature of the arrival process. The letter *M* is used to denote pure random arrivals or pure random service times. It stands for Markovian, a reference to the memoryless property of the exponential distribution of interarrival times. In other words, the arrival process is a Poisson process. Commonly used letters are:
   M – for exponential distribution of interarrival times
   G – for general independent distribution of interarrival times
   D – for deterministic (constant) interarrival times
2. *Service Process* (second symbol) indicates the nature of the probability distribution of the service times. For example, *M*, *G*, and *D* stand for exponential, general, and deterministic distributions, respectively. In all cases, successive interarrival times and service times are assumed to be statistically independent of each other.
3. *Number of Servers* (third symbol) specifies the number of servers in the system.
4. *Maximum Occupancy* (fourth symbol) is a number that specifies the waiting room capacity. Excess customers are blocked and not allowed into the system.
5. *User Population* (fifth symbol) is a number that specifies the total customer population (the "universe" of customers)
6. *Scheduling Discipline* (sixth symbol) indicates how the arriving customers are scheduled for service. Scheduling discipline is also called *Service Discipline* or *Queuing Discipline*. Commonly used service disciplines are the following:
   FCFS – first-come-first-served, also called first-in-first-out (FIFO), where the first customer that arrives in the system is the first customer to be served
   LCFS – last-come-first served (like a popup stack)
   FIRO – first-in-random-out

Service disciplines will be covered later in Chapter 5, where fair queuing (FQ) service discipline will be introduced. Only the first three symbols are commonly used in specifying a queuing model, although sometimes other symbols will be used in the rest of this chapter.

4.2.2 *M* / *M* / 1 Queuing System

Problems related to this section: Problem 4.16 → Problem 4.20

A correct notation for the system we consider is *M*/*M*/1/∞/∞/FCFS. This system can hold unlimited (infinite) number of customers, i.e., it has an unlimited waiting room size or the maximum queue length; the total customer population is unlimited; and, the customers are served in the FCFS order. It is common to omit the last three items and simply use *M*/*M*/1.

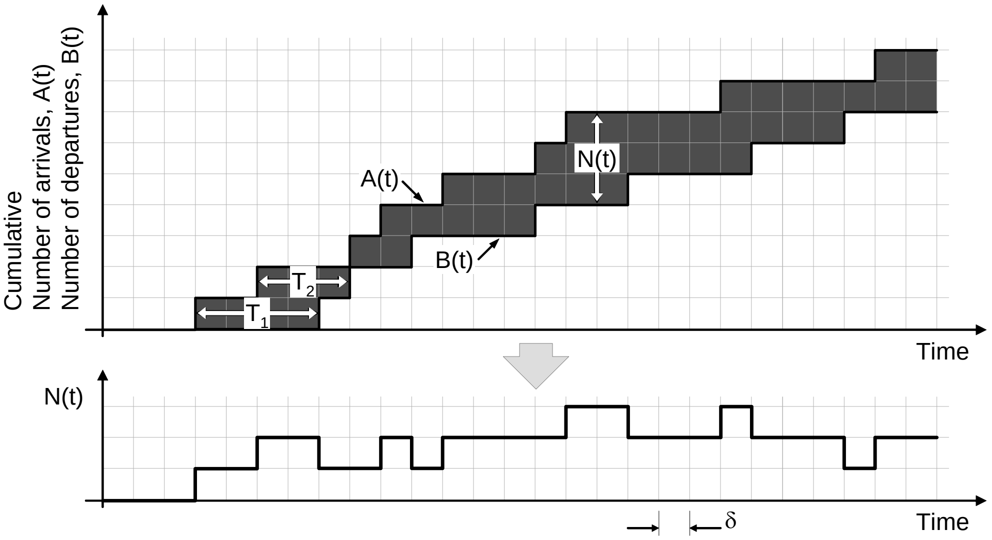


Figure 4-19: Example of birth and death processes. Top: Arrival and departure processes; Bottom: Number of customers in the system.

Figure 4-19 illustrates an $M/M/1$ queuing system, for which the process $A(t)$, total number of customers that arrived from 0 to time t, has a Poisson distribution. A Poisson process is generally considered a good model for the aggregate traffic of a large number of similar and independent customers. Then, $A(0) = 0$, and for $s < t$, $A(t) - A(s)$ equals the number of arrivals in the interval (s, t). The intervals between two arrivals (interarrival times) for a Poisson process are independent of each other and exponentially distributed with the parameter λ. If t_n denotes the time of the n^{th} arrival, the interarrival intervals $\tau_n = t_{n+1} - t_n$ have the probability distribution

$$P\{\tau_n \leq s\} = 1 - e^{-\lambda \cdot s}, \qquad s \geq 0$$

It is important that we select the unit time period δ in Figure 4-19 small enough so that it is likely that at most one customer will arrive during δ. In other words, δ should be so small that it is unlikely that two or more customers will arrive during δ.

The process $A(t)$ is a pure *birth process* because it monotonically increases by one at each arrival event. So is the process $B(t)$, the number of departures up until time t. The process $N(t)$, the number of customers in the system at time t, is a *birth and death process* because it sometimes increases and at other times decreases. It increases by one at each arrival and decreases by one at each completion of service. We say that $N(t)$ represents the ***state*** of the system at time t. Note that the state of this particular system (a birth and death process) can either increases by one or decreases by one—there are no other options. The intensity or rate at which the system state increases is λ and the intensity at which the system state decreases is μ. This means that we can represent the rate at which the system changes the state by the diagram in Figure 4-21.

Now suppose that the system has evolved to a steady-state condition. That means that the state of the system is independent of the starting state. The sequence $N(t)$ representing the number of customers in the system at different times does not converge. This is a random process taking unpredictable values. What does converge are the probabilities p_n that at any time a certain number of customers n will be observed in the system

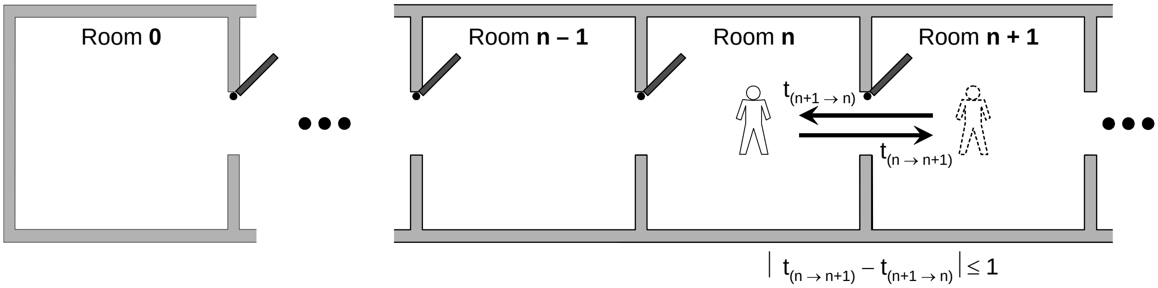


Figure 4-20: Intuition behind the balance principle for a birth and death process.

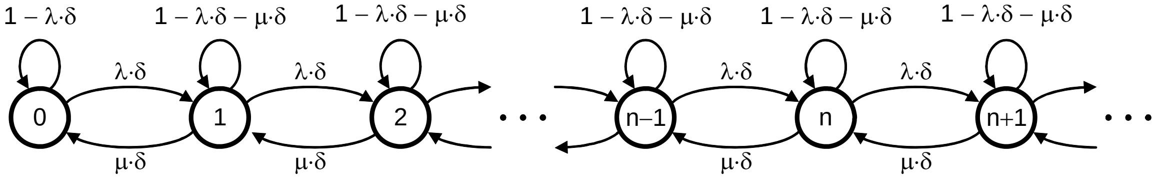


Figure 4-21: Transition probability diagram for the number of customers in the system.

$$\lim_{t \to \infty} P\{N(t) = n\} = p_n$$

Note that during any time interval, the total number of transitions from state n to $n + 1$ can differ from the total number of transitions from $n + 1$ to n by at most 1. Thus asymptotically, the frequency of transitions from n to $n + 1$ is equal to the frequency of transitions from $n + 1$ to n. This is called the *balance principle*. As an intuition, each state of this system can be imagined as a room, with doors connecting the adjacent rooms. If you keep walking from one room to the adjacent one and back, you can cross at most once more in one direction than in the other. In other words, the difference between how many times you went from $n + 1$ to n vs. from n to $n + 1$ at any time can be no more than one.

Given the stationary probabilities and the arrival and service rates, from our rate-equality principle we have the following *detailed balance equations*

$$p_n \cdot \lambda = p_{n+1} \cdot \mu, \qquad n = 0, 1, 2, \ldots \tag{4.2}$$

These equations simply state that the rate at which the process leaves state n equals the rate at which it enters that state. The ratio $\rho = \lambda/\mu$ is called the *utilization factor* of the queuing system, which is the long-run proportion of the time the server is busy. With this, we can rewrite the detailed balance equations as

$$p_{n+1} = \rho \cdot p_n = \rho^2 \cdot p_{n-1} = \ldots = \rho^{n+1} \cdot p_0 \tag{4.3}$$

If $\rho < 1$ (service rate exceeds arrival rate), the probabilities p_n are all positive and add up to unity, so

$$1 = \sum_{n=0}^{\infty} p_n = \sum_{n=0}^{\infty} \rho^n \cdot p_0 = p_0 \cdot \sum_{n=0}^{\infty} \rho^n = \frac{p_0}{1-\rho} \tag{4.4}$$

by using the well-known summation formula for the geometric series (see the derivation of Eq. (1.8) in Section 1.3.1). Combining equations (4.3) and (4.4), we obtain the probability of finding n customers in the system

$$p_n = P\{N(t) = n\} = \rho^n \cdot (1-\rho), \qquad n = 0, 1, 2, \ldots \tag{4.5}$$

The average number of customers in the system in steady state is $N = \lim_{t\to\infty} E\{N(t)\}$. Because (4.5) is the p.m.f. for a geometric random variable, meaning that $N(t)$ has a geometric distribution, checking a probability textbook for the expected value of geometric distribution quickly yields

$$N = \lim_{t\to\infty} E\{N(t)\} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda} \tag{4.6}$$

It turns out that for an $M/M/1$ system, by knowing only the arrival rate λ and service rate μ, we can determine the average number of customers in the system. From this, Little's Law (4.1a) gives the average delay per customer (waiting time in queue plus service time) as

$$T = \frac{N}{\lambda} = \frac{1}{\mu-\lambda} \tag{4.7}$$

The average waiting time in the queue, W, is the average delay T less the average service time $1/\mu$, like so

$$W = T - \frac{1}{\mu} = \frac{1}{\mu-\lambda} - \frac{1}{\mu} = \frac{\rho}{\mu-\lambda}$$

and by using the version (4.1b) of Little's Law, we have $N_Q = \lambda \cdot W = \rho^2 \big/ (1-\rho)$.

4.2.3 *M / M /* 1 / m Queuing System

Problems related to this section: Problem 4.22 → Problem 4.23

Now consider the $M/M/1/m$ system that is the same as $M/M/1$ except that the system can be occupied by up to m customers, which implies a finite waiting room or maximum queue length. The customers arriving when the queue is full are *blocked* and not allowed into the system. We have $p_n = \rho^n \cdot p_0$ for $0 \le n \le m$; otherwise $p_n = 0$. Using the relation $\sum_{n=0}^{m} p_n = 1$ we obtain

$$p_0 = \frac{1}{\sum_{n=0}^{m} \rho^n} = \frac{1-\rho}{1-\rho^{m+1}}, \quad 0 \le n \le m$$

From this, the steady-state occupancy probabilities are given by (cf. Eq. (4.5))

$$p_n = \frac{\rho^n \cdot (1-\rho)}{1-\rho^{m+1}}, \quad 0 \le n \le m \tag{4.8}$$

Assuming again that $\rho < 1$, the expected number of customers in the system is

$$N = E\{N(t)\} = \sum_{n=0}^{m} n \cdot p_n = \frac{1-\rho}{1-\rho^{m+1}} \cdot \sum_{n=0}^{m} n \cdot \rho^n = \frac{1-\rho}{1-\rho^{m+1}} \cdot \rho \cdot \sum_{n=0}^{m} n \cdot \rho^{n-1} = \frac{\rho \cdot (1-\rho)}{1-\rho^{m+1}} \cdot \frac{\partial}{\partial \rho}\left(\sum_{n=0}^{m} \rho^n\right)$$

$$= \frac{\rho \cdot (1-\rho)}{1-\rho^{m+1}} \cdot \frac{\partial}{\partial \rho}\left(\frac{1-\rho^{m+1}}{1-\rho} \right) = \frac{\rho}{1-\rho} - \frac{(m+1)\cdot \rho^{m+1}}{1-\rho^{m+1}} \tag{4.9}$$

Thus, the expected number of customers in the system is always less than for the unlimited queue length case, Eq. (4.6).

It is also of interest to know the probability of a customer arriving to a full waiting room, also called *blocking probability* p_B. Generally, the probability that a customer arrives when there are n customers in the queue is (using Bayes' formula)

$$P\{N(t) = n \mid \text{a customer arrives in } (t, t+\delta)\} = \frac{P\{\text{a customer arrives in}(t, t+\delta) \mid N(t) = n\} \cdot P\{N(t) = n\}}{P\{\text{a customer arrives in}(t, t+\delta)\}}$$

$$= \frac{(\lambda \cdot \delta) \cdot p_n}{\lambda \cdot \delta} = p_n$$

because of the memoryless assumption about the system. Thus, the blocking probability is the probability that an arrival will find m customers in the system, which is (using Eq. (4.8))

$$p_\mathrm{B} = P\{N(t) = m\} = p_m = \frac{\rho^m \cdot (1-\rho)}{1-\rho^{m+1}} \tag{4.10}$$

4.2.4 *M* / *G* / 1 Queuing System

We now consider a class of systems where arrival process is still memoryless with rate λ. However, the service times have a general distribution—not necessarily exponential as in the *M*/*M*/1 system—meaning that we do not know anything about the distribution of service times. Suppose again that the customers are served in the order they arrive (FCFS) and that X_i is the service time of the i^{th} arrival. We assume that the random variables (X_1, X_2, …) are independent of each other and of the arrival process, and identically distributed according to an unspecified distribution function.

The class of *M*/*G*/1 systems is a superset of *M*/*M*/1 systems. The key difference is that in general there may be an additional component of memory. In such a case, one cannot say as for *M*/*M*/1 that the future of the process depends only on the present length of the queue. To calculate the average delay per customer, it is also necessary to account for the customer that has been in service for some time. Similar to *M*/*M*/1, we could define the state of the system as the number of customers in the system and use the so called moment generating functions to derive the system parameters. Instead, a simpler method from [Bertsekas & Gallager, 1992] is used.

Assume that upon arrival the i^{th} customer finds N_i customers waiting in queue and one currently in service. The time the i^{th} customer will wait in the queue is given as

$$W_i = \sum_{j=i-N_i}^{i-1} X_j + R_i \tag{4.11}$$

where R_i is the *residual service time* seen by the i^{th} customer. By this we mean that if customer j is currently being served when i arrives, R_i is the remaining time until customer j's service is

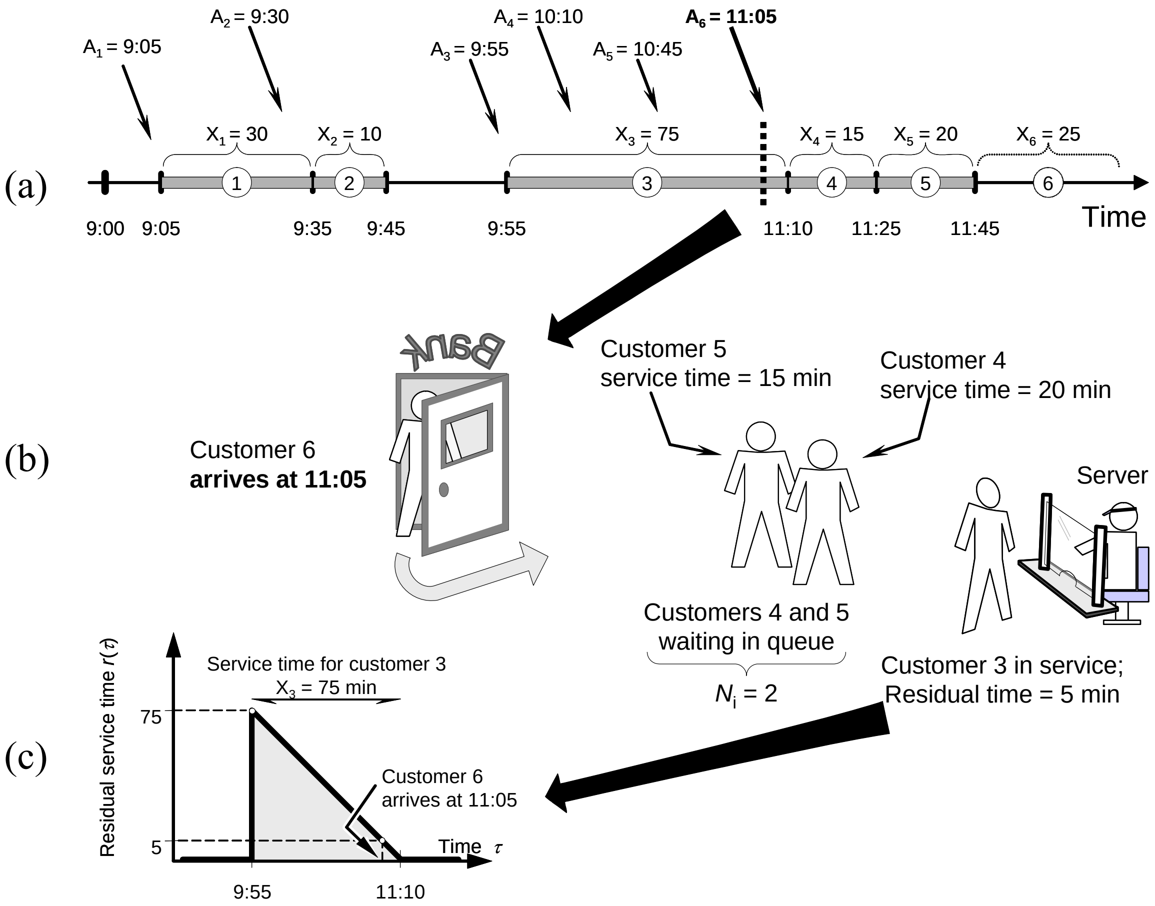


Figure 4-22: (a) Example of customer arrivals and service times; see Example 4.1 for details. (b) Detail of the situation found by customer 6 at his/her arrival. (c) Residual service time for customer 3 at the arrival of customer 6.

completed. The residual time's index is *i* (not *j*) because this time depends on *i*'s arrival time and is not inherent to the served customer. If no customer is served at the time of *i*'s arrival, then R_i is zero.

Example 4.1 Delay in a Bank Teller Service

An example pattern of customer arrivals to the bank from Figure 4-17 is shown in Figure 4-22. Assume that nothing is known about the distribution of service times. In this case, customer $k = 6$ will find customer 3 in service and customers 4 and 5 waiting in queue, i.e., $N_6 = 2$. The residual service time for 3 at the time of 6's arrival is 5 min. Thus, customer 6 will experience the following queuing delay:

$$W_6 = \sum_{j=4}^{5} X_j + R_6 = \left(15 + 20\right) + 5 = 40\,\text{min}$$

This formula simply adds up all the times shown in Figure 4-22(b). Note that the residual time depends on the arrival time of customer $i = 6$ and not on how long the service time of customer $(i - N_i - 1) = 3$ is.

The total time that 6 will spend in the system (the bank) is $T_6 = W_6 + X_6 = 40 + 25 = 65$ min.

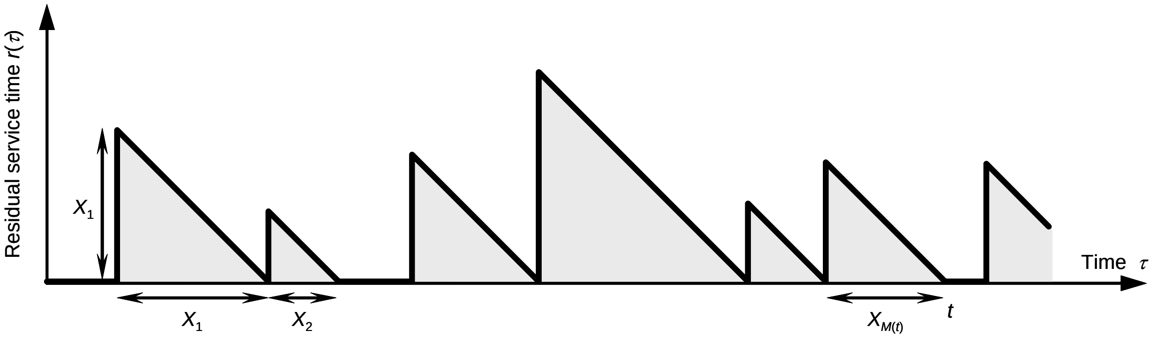


Figure 4-23: Expected residual service time computation. The time average of *r*(τ) is computed as the sum of areas of the isosceles triangles over the given period *t*.

By taking expectations of Eq. (4.11) and using the independence of the random variables N_i and $X_{i-1}, X_{i-2}, \ldots, X_{i-Ni}$ (which means that how many customers are found in the queue is independent of what business they came for), we have

$$E\{W_i\} = E\left\{\sum_{j=i-N_i}^{i-1} E\{X_j \mid N_i\}\right\} + E\{R_i\} = E\{X\} \cdot E\{N_i\} + E\{R_i\} \;=\; \frac{1}{\mu} \cdot N_Q + E\{R_i\} \tag{4.12}$$

Throughout this section all long-term average quantities should be viewed as limits when time or customer index converges to infinity. We assume that these limits exist, which is true for most systems of interest provided that the utilization $\rho < 1$.The second term in the above equation is the *mean residual time*, $R = \lim_{t \to \infty} E\{R_i\}$, and it will be determined by a graphical argument. The residual service time $r(\tau)$ can be plotted as in Figure 4-22(c). The general case is shown in Figure 4-23. Every time a new customer enters the service, the residual time equals that customer's service time. Then it decays linearly until the customer's service is completed. The time average of $r(\tau)$ in the interval [0, t] is

$$\frac{1}{t}\int_0^t r(\tau)d\tau = \frac{1}{t}\sum_{i=1}^{M(t)} \frac{1}{2} X_i^2$$

where $M(t)$ is the number of service completions within [0, t]. Hence, we obtain

$$R = \lim_{i \to \infty} E\{R_i\} = \lim_{i \to \infty}\left(\frac{1}{t}\int_0^t r(\tau)d\tau\right) = \frac{1}{2}\lim_{i \to \infty}\left(\frac{M(t)}{M(t)} \cdot \frac{1}{t}\sum_{i=1}^{M(t)} X_i^2\right) = \frac{1}{2}\lim_{i \to \infty}\left(\frac{M(t)}{t}\right) \cdot \lim_{i \to \infty}\left(\frac{\sum_{i=1}^{M(t)} X_i^2}{M(t)}\right) = \frac{1}{2}\lambda \cdot \overline{X^2}$$

where $\overline{X^2}$ is the second moment of service time, computed as

$$E\{X^n\} = \begin{cases} \sum_{X \,:\, p_i > 0} p_i \cdot (X_i)^n & \text{if } X \text{ is discrete r.v.} \\ \int_{-\infty}^{\infty} f(x) \cdot x^n \cdot dx & \text{if } X \text{ is continuous r.v.} \end{cases}$$

By substituting this expression in the queue waiting time, Eq. (4.12), we obtain the so called Pollaczek-Khinchin (P-K) formula

$$W = \frac{\lambda \cdot \overline{X^2}}{2 \cdot (1-\rho)} \tag{4.13}$$

The P-K formula holds for any distribution of service times as long as the variance of the service times is finite.

Example 4.2 Queuing Delays of an Go-Back-*N* ARQ

Consider a Go-Back-*N* ARQ such as described in Section 1.3.2. Assume that packets arrive at the sender according to a Poisson process with rate λ. Assume also that errors affect only the data packets, from the sender to the receiver, and not the acknowledgment packets. What is the expected queuing delay per packet in this system?

Note that the expected service time per packet equals the expected delay per packet transmission, which is determined in the solution of Problem 1.11 at the back of this text as follows

$$\overline{X} = E\{T_{\text{total}}\} = t_{\text{succ}} + \frac{p_{\text{fail}}}{1 - p_{\text{fail}}} \cdot t_{\text{fail}}$$

The second moment of the service time, $\overline{X^2}$, is determined similarly as:

Finally, Eq. (4.13) yields

$$W = \frac{\lambda \cdot \overline{X^2}}{2(1-\rho)} = \frac{\lambda \cdot \overline{X^2}}{2(1-\lambda/\mu)} = \frac{\lambda \cdot \overline{X^2}}{2(1-\lambda \cdot \overline{X})}$$

4.3 Networks of Queues

4.4 Summary and Bibliographical Notes

Section 4.1 focuses on one function of routers—forwarding packets—but this is just one of its many jobs. Section 1.4 describes another key function: building and maintaining the routing tables. In addition, more and more applications, such as firewalls, VPN concentration, voice gateways and video monitoring, are being implemented in routers. Cisco's Integrated Services Router (ISR), for example, even includes an optional application server blade for running various Linux and open source packages.

[Keshav & Sharma, 1998]

Kumar, *et al*. [1998] provide a good overview of router architectures and mechanisms.

James Aweya, "IP router architectures: An overview"

The material presented in this chapter requires basic understanding of probability and random processes. [Yates & Goodman, 2004] provides an excellent introduction and [Papoulis & Pillai, 2001] is a more advanced and comprehensive text.

[Bertsekas & Gallager, 1992] provides a classic treatment of queuing delays in data networks. Most of the material in Sections 4.2 and 4.3 is derived from this reference.

Problems

Problem 4.1

Problem 4.2

Consider a second-generation router with 4 network ports and each link is full-duplex with data rate of 1 Mbps. Assume that all packets in the network have the same size of 1 KBytes. The total memory capacity of each NFE is 2 packets (combined for its input and output ports). The system bus data rate is 4 Mbps. Solve the following:

(a) Describe the worst-case scenario for packet loss at each NFE.
(b) When will the first packet be lost under the worst-case scenario?
(c) What is the maximum average fraction of packets that can be lost at each NFE?

Problem 4.3

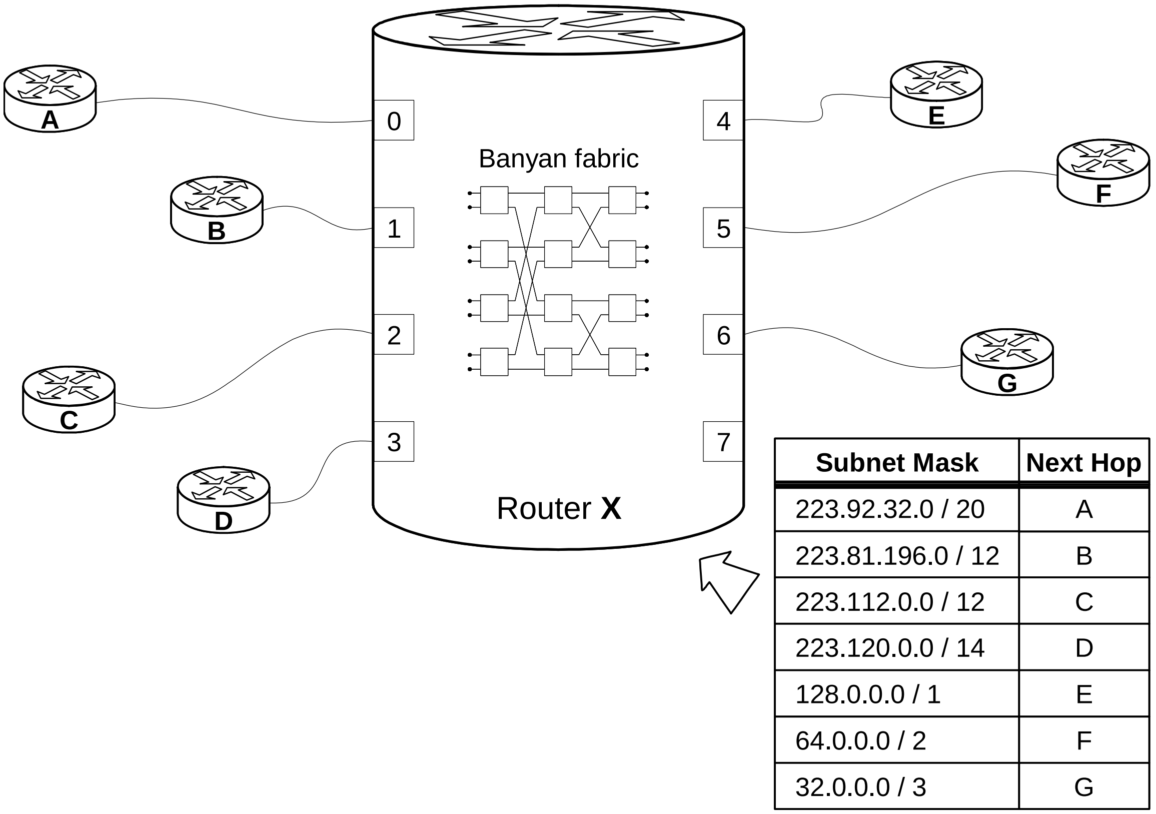
Consider a regular PC that is used as a router, i.e., this is first-generation router architecture. The router has 4 network ports, each on its own line card. All four links have the same data rate of *R* bits/sec. The system bus operates at a four times higher data rate, i.e., 4×*R* bps. Consider a scenario where steady traffic is arriving on all four ports and all packets are of the same length *L*.

(a) What is the worst-case delay that a packet can experience in this router?
(b) Will there be any head-of-line or output blocking observed?

Problem 4.4

Problem 4.5

Consider a router *X* that uses Banyan switching fabric. The figure below shows the router's connectivity to the adjacent routers, as well as the forwarding table of router *X*.



Subnet Mask	Next Hop
223.92.32.0 / 20	A
223.81.196.0 / 12	B
223.112.0.0 / 12	C
223.120.0.0 / 14	D
128.0.0.0 / 1	E
64.0.0.0 / 2	F
32.0.0.0 / 3	G

Assume that the following packets arrived simultaneously on router *X*:

Packet arrived from	Packet destination IP address
B	63.67.145.18
C	223.123.59.47
G	223.125.49.47

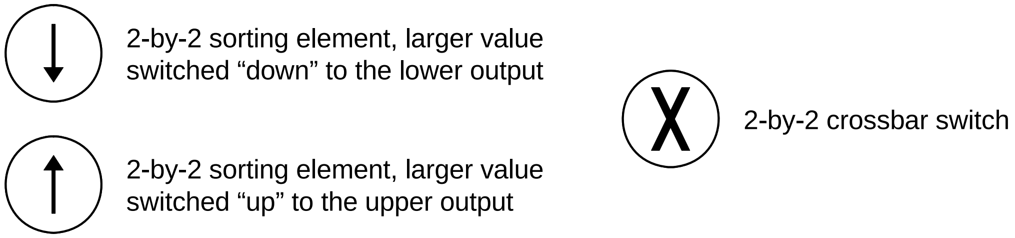
Draw and explain a diagram that shows how these packets will traverse the switching fabric.

Note: Check Problem 1.33 in Chapter 1 to see how the next hop for a packet is decided.

Problem 4.6

Problem 4.7

Using the switch icons shown below, sketch a simple 4×4 Batcher-Banyan network (without a trap network and a shuffle-exchange network). Label the input ports and the output ports of the fabric, from top to bottom, as 0, 1, 2 and 3.


2-by-2 sorting element, larger value switched “down” to the lower output
2-by-2 sorting element, larger value switched “up” to the upper output
2-by-2 crossbar switch

Suppose four packets are presented to the input ports of the Batcher-Banyan fabric that you sketched. Suppose further that the incoming packets are heading to the following output ports:

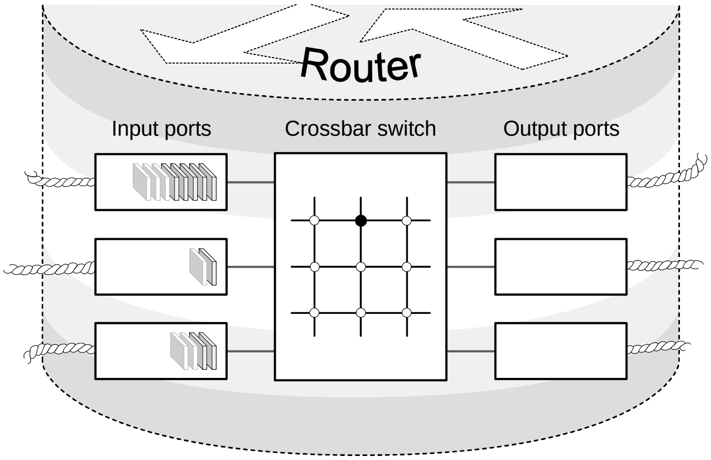
- Packet at input port 0 is heading to output port 1
- Packet at input port 1 to output port 0
- Packet at input 2 to output port 0
- Packet at input port 3 to output port 2

Show on the diagram of the fabric the switching of these packets through the fabric from the input ports to the output ports. Will any collisions and/or idle output lines occur?

Problem 4.8

Problem 4.9

Consider the router shown below where the data rates are the same for all the communication lines. The switch fabric is a crossbar so that at most one packet at a time can be transferred to a given output port, but different output ports can simultaneously receive packets from different input ports. Assume that the fabric moves packets two times faster than the data rate of the communication lines. Packets at each port are moved in a first-come-first-served (FCFS) order. If two or more packets arrive simultaneously at different input ports and are heading towards the same output port, their order of transfer is decided so that the lower index port wins (e.g., if ports 2 and 3 contend at the same time to the same output port, port 2 goes first). If a packet at a lower-index input port arrives and goes to the same output port for which a packet at a higher-index port is already waiting, then the higher-index port wins.



Consider the following traffic arrival pattern:

Input port 1: packet of length 2 received at time $t = 0$ heading to output port 2; packet of length 4 at time 4 to output 2

Input port 2: packet of length 8 at time 0 to output 2; packet of length 2 at time 2 to output 1

Input port 3: packet of length 8 at time 2 to output 2; packet of length 2 at time 4 to output 1

Draw the timing diagram for transfers of packets across the switching fabric. Will there be any head-of-line or output blocking observed? Explain your answer.

Problem 4.10

Problem 4.11

Problem 4.12

Problem 4.13

[Little's Law] Consider a system with a single server. The arrival and service times for the first 10 customers are as follows: ($A_1 = 0$, $X_1 = 3$); (2, 4); (3, 5); (4, 2); (6, 5); (7, 2); (10, 4); (11, 3); (12, 5); and ($A_{10} = 13$, $X_{10} = 3$).

(a) Draw the arrivals as a birth-death process similar to Figure 4-19.

(b) What is the average number of customers in the system N and the average delay T per customer in this system during the observed period? Assuming that the arrival rate is λ = 1 customer/unit-of-time, does the system satisfy the Little's Law over the observed period?

Problem 4.14

Problem 4.15

Problem 4.16

Consider a router that can process 1,000,000 packets per second. Assume that the load offered to it is 950,000 packets per second. Also assume that the interarrival times and service durations are exponentially distributed.

(a) How much time will a packet, on average, spend being queued before being serviced?

(b) Compare the waiting time to the time that an average packet would spend in the router if no other packets arrived.

(c) How many packets, on average, can a packet expect to find in the router upon its arrival?

Problem 4.17

Consider an $M/G/1$ queue with the arrival and service rates λ and μ, respectively. What is the probability that an arriving customer will find the server busy (i.e., serving another customer)?

Problem 4.18

Messages arrive at random to be sent across a communications link with a data rate of 9600 bps. The link is 70% utilized, and the average message length is 1000 bytes. Determine the average waiting time for exponentially distributed length messages and for constant-length messages.

Problem 4.19

A facility of m identical machines is sharing a single repairperson. The time to repair a failed machine is exponentially distributed with mean $1/\lambda$. A machine, once operational, fails after a time that is exponentially distributed with mean $1/\mu$. All failure and repair times are independent. What is the steady-state proportion of time where there is no operational machine?

Problem 4.20

Imagine that K users share a link (e.g., Ethernet or Wi-Fi) with throughput rate R bps (i.e., R represents the actual number of file bits that can be transferred per second, after accounting for overheads and retransmissions). User's behavior is random and we model it as follows. Each user requests a file and waits for it to arrive. After receiving the file, the user sleeps for a random time, and then repeats the procedure. Each file has an exponential length, with mean $A \times R$ bits. Sleeping times between a user's subsequent requests are also exponentially distributed but with a mean of B seconds. All these random variables are independent. Write a formula that estimates the average time it takes a user to get a file since completion of his previous file transfer.

Problem 4.21

Problem 4.22

Consider a single queue with a constant service time of 4 seconds and a Poisson input with mean rate of 0.20 items per second.

(a) Find the mean and standard deviation of queue size

(b) Find the mean and standard deviation of the time a customer spends in system.

Problem 4.23

Consider the Go-back-N protocol used for communication over a noisy link with the probability of packet error equal to p_e. Assume that the link is memoryless, i.e., the packet error events are independent from transmission to transmission. Also assume that the following parameters are given: the round-trip time (RTT), packet size L, and transmission rate R. What is the average number of successfully transmitted packets per unit of time (also called *throughput*), assuming that the sender always has a packet ready for transmission?
Hint: Recall that the average queuing delay per packet for the Go-back-N protocol is derived in Example 4.2 (Section 4.2.4).

Problem 4.24

Problem 4.25

Chapter 5

Mechanisms for Quality-of-Service

Contents

This chapter reviews mechanisms used in network routers to provide quality-of-service (QoS). Section 3.3 reviewed the requirements and some end-to-end mechanisms for providing quality of service, and hinted at mechanisms used in routers. This chapter details the router-based QoS mechanisms.

End-to-end QoS is built from the concatenation of edge-to-edge QoS from each network domain (or Autonomous System) through which traffic passes, and ultimately depends on the QoS characteristics of the individual hops along any given route. Networking solutions for end-to-end QoS are usually broken into three parts: per-hop QoS, traffic engineering, and signaling/provisioning. This chapter starts with mechanisms used to provide per-hop QoS in individual routers, and describes traffic-engineering solutions in Section 5.4.3. Signaling/provisioning was already considered in Section 3.3.5 and will be considered further here.

The goal of per-hop QoS is to enable congestion-point (or, "bottleneck") routers and switches to provide predictable differentiated loss, latency, and jitter characteristics to traffic classes of interest to the service provider or its customers. This goal is achieved in part by the scheduling mechanism which:

- Prevents aggressive or misbehaving sources from overtaking network resources
- Provides preferential service to packets from preferred sources.

5.1 Queue Scheduling

The queuing models in Chapter 4 considered delays and blocking probabilities under the assumption that tasks/packets are served on a first-come-first-served (FCFS) basis and that a task is blocked if it arrives at a full queue (if the waiting room capacity is limited). A queue scheduler employs two strategies:

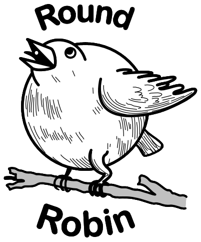
1. ***Scheduling discipline*** that decides which packet to serve (transmit) next; it is also called *service discipline* or *queuing discipline*, see Section 4.2
2. ***Drop policy*** that decides which packet to drop next (when required); it is also called *blocking policy* or *packet-discarding policy*

The simplest combination is *FCFS with tail drop*, i.e., always service head of the line and, when the queue is full, drop the last arriving packet, regardless of its flow or importance. This is what we considered in Section 4.2.3. Scheduling has direct impact on a packet's queuing delay and hence on its total delay. Dropping decides whether the packet will arrive at all to the destination.

FCFS does not make any distinction between packets. A single FCFS queue cannot simultaneously support QoS-sensitive and QoS-insensitive traffic. Although a large buffer is less likely to overflow during a traffic burst (thus reducing the packet loss probability), it potentially increases the queuing delay for non-dropped packets. A short queue reduces this delay, but conversely increases the probability of packet loss for bursty traffic.

The solution is to split traffic into multiple queues at each congestion point, assigning different classes of traffic to queues sized for each class's desired loss, latency, and jitter characteristics, usually expressed as *service-level agreements* (SLAs). Access to the resource (e.g., outbound link) is mediated by a scheduler, which empties each queue in proportion to its allocated resource share or priority. Therefore, the system that wishes to make distinction between packets (QoS-enabled router or switch) must (1) classify packets, (2) differentially queue packets per class, and (3) provide controllable and predictable scheduling of packet transmissions from each class (queue) onto the outbound link. This approach is often referred to as a **classify, queue, and schedule (CQS) architecture** and it comprises three components (Figure 5-1):

1. ***Classifier****—decides which waiting line the arriving packet will join*. The criteria for sorting packets into different lines include: priority, source and/or destination network address, application port number, etc.
2. ***Queues****—form different waiting lines for different packet types*. The queues have limited capacities and implement packet-dropping policies.
3. ***Scheduler****—calls packets from waiting lines for service* in such a way that satisfies certain commitments made to the packet senders, described by service-level agreements (SLAs). Options for the rules of calling the packets for service (*scheduling discipline*) include: (*i*) first serve all the packets waiting in the high-priority line, if any; then go to the next lower priority class, etc.; (*ii*) serve the lines in round-robin manner by serving one or more packets from one line (but not necessarily all that are currently waiting), then go to serve few from the next waiting line, etc., then repeat the cycle.



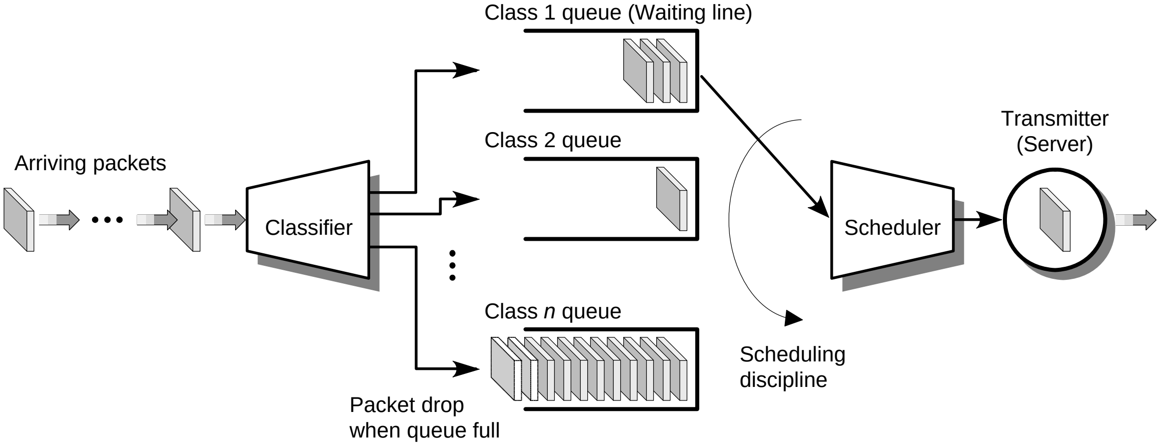


Figure 5-1: Components of a scheduler. Classifier sorts the arriving packets into different queues based on one or more criteria, such as priority or source identity. Scheduler places the packets into service based on its scheduling discipline. A single server serves all queues.

There are different design choices and business policies that need to be decided when implementing each of these components. We consider some options next.

5.1.1 Scheduler Design Concerns

We already mentioned the issue of queue lengths that depends on buffer capacity. A large buffer gives a lower probability of packet loss because it is less likely to overflow during traffic bursts, but it potentially increases the queuing delay for non-dropped packets. A short queue does not allow much delay, but has a higher probability of packet loss for traffic bursts. The design decision can be taken based on a queue model (Section 4.2), and acceptable delays and drop rates that depend on the application type.

Additional concerns may compel the network designer to consider discriminating the packets, and design more complex scheduling disciplines and dropping policies. Such concerns include:

- *Prioritization*, where different tasks/packets can have assigned different priorities, so that the delay time for certain packets is reduced (at the expense of other packets)
- *Fairness*, so that different flows (identified by source-destination pairs) are offered equitable access to system resources (i.e., comparable waiting delay and probability of dropped packets)
- *Protection*, so that misbehavior of some flows (by sending packets at a rate faster than their fair share) should not affect the performance achieved by other flows

Prioritization and fairness are complementary, rather than mutually exclusive. Fairness ensures that traffic flows of equal priority receive equitable service and that flows of lower priority are not excluded from receiving any service because all of it is consumed by higher priority flows. Fairness and protection are related so that ensuring fairness automatically provides protection, because it limits a misbehaving flow to its fair share. However, the converse need not be true. For example, if flows are *policed* at the entrance to the network, so that they are forced to confirm to a predeclared traffic pattern, they are protected from each other, but their resource shares may not be fair. Policing will be considered later in Section 5.2.

FCFS places indiscriminately all the arriving packets at the tail of a single queue. The idea with *prioritization* is that the packets with highest priority, upon arrival to the system, are placed at the head-of-the line, so they bypass waiting in the line. They may still need to wait if there is another packet (perhaps even of a lower priority) currently being transmitted. *Non-preemptive scheduling* is the discipline under which the ongoing transmission of lower-priority packets is not interrupted upon the arrival of a higher-priority packet. Conversely, *preemptive scheduling* is the discipline under which lower-priority packet is bumped out of service (back into the waiting line or dropped from the system) if a higher-priority packet arrives at the time a lower-priority packet is being transmitted.

Packet *priority* may be assigned simply based on the packet type, or it may be result of applying a complex set of *policies*. For example, the policies may specify that a certain packet type of a certain user type has high priority at a certain time of the day and low priority at other times.

Although priority scheduler does provide different performance characteristics to different classes, it still has shortcomings. For example, it does not deal with fairness and protection. An aggressive or misbehaving high-priority source may take over the communication line and elbow out all other sources. Not only the flows of the lower priority will suffer, but also the flows of the same priority are not protected from misbehaving flows.

A *round robin scheduler* alternates the service among different flows or classes of packets. In the simplest form of round robin scheduling, the head of each queue is called, in turn, for service. That is, a class-1 packet is transmitted, followed by a class-2 packet, and so on until a class-*n* packet is transmitted. The whole round is repeated forever or until there are no more packets to transmit. If a particular queue is empty, because no packets of such type arrived in the meantime, the scheduler has two options:

1. Keep unused the portion of service (or work) allocated for that particular class and let the server stay idle (*non-work-conserving scheduler*)
2. Let the packet from another queue, if any, use this service (*work-conserving scheduler*)

A work-conserving scheduler will never allow the link (server) to remain idle if there are packets (of any class or flow) waiting for service. When such scheduler looks for a packet of a given class but finds none, it will immediately check the next class in the round robin sequence.

One way to achieve control of channel conditions (hence, performance bounds) is to employ time division multiplexing (TDM) or frequency division multiplexing (FDM). TDM/FDM maintains a separate channel for each traffic flow and never mixes packets from different flows, so they never interfere with each other. TDM and FDM are non-work-conserving. Statistical multiplexing is work-conserving and that is what we consider in the rest of this section.

5.1.2 Fair Queuing

Problems related to this section: Problem 5.2 → Problem 5.8

Suppose that a system, such as transmission link, has insufficient resource to satisfy the demands of all users, each of whom has an equal right to the resource, but some need less than others. The

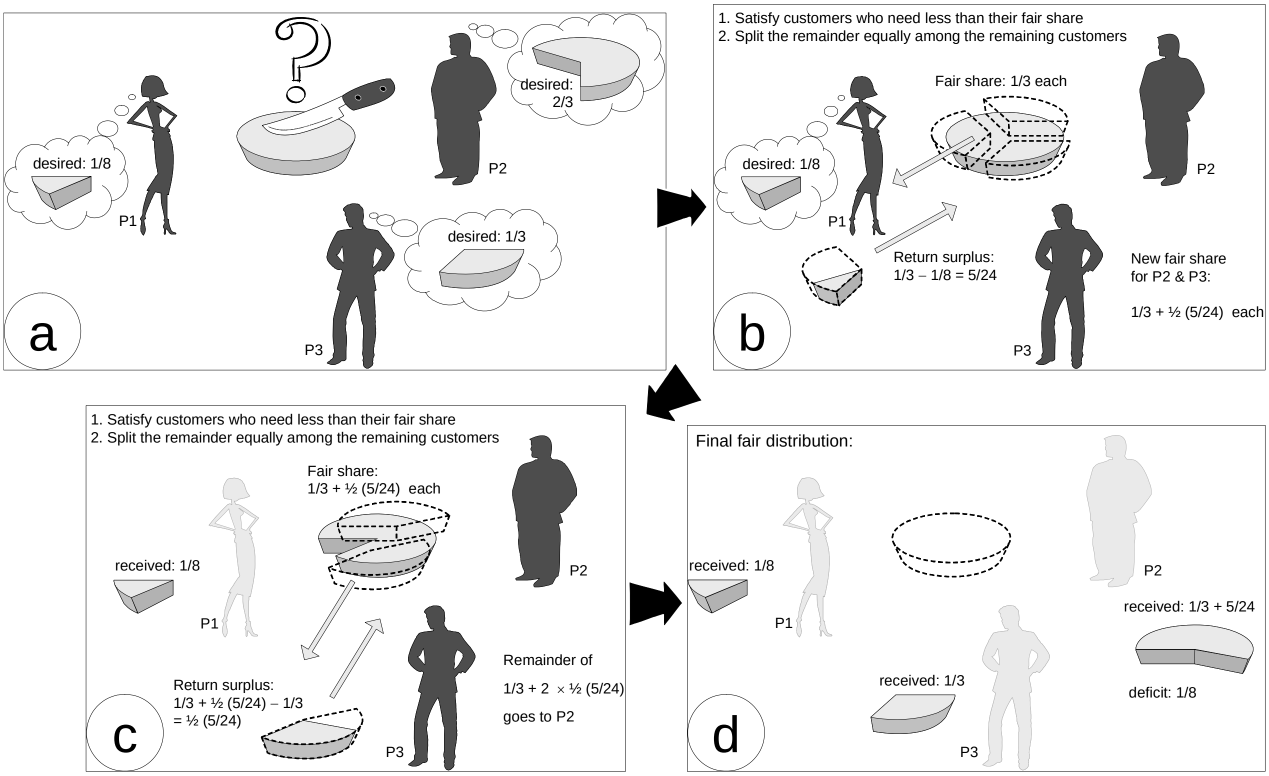


Figure 5-2: Illustration of max-min fair share algorithm; see text for details.

problem is conceptually represented in Figure 5-1. Our goal is to design a *scheduling algorithm* that places the packets into service so that the server resource (transmission line) is shared *fairly*. We start by determining how to divide the resource fairly in a general case. This general technique will then be adapted to packet transmission, because packets must be transmitted as a whole, rather than partially as needed to fit in their fair share.

A generic sharing technique for fair distribution of a limited resource that is widely used in practice is called **max-min fair share** (MMFS). We first check whether the sum of all demands exceeds the available capacity. If not, there is no need to apply MMFS—the resource can be divided just as requested by each user. If Σ(*demands*) > *capacity*, then we apply MMFS. Intuitively, a fair share first fulfils the demand of users who need less than they are entitled to, and then evenly distributes unused resources among the "big" users (Figure 5-2). Formally, we define max-min fair share allocation as follows:

- Resources are allocated in order of increasing demand
- No source obtains a resource share larger than its demand
- Sources with unsatisfied demands obtain an equal share of the resource

There will be no surplus remaining after this process, because we do not do it if Σ(*demands*) ≤ *capacity*. This formal definition corresponds to the following operational definition. Consider a set of sources 1, ..., *n* that have resource demands r_1, r_2, ..., r_n. Without loss of generality, order the source demands so that $r_1 \leq r_2 \leq \ldots \leq r_n$. Assume that the server has capacity *C* and all sources are equally entitled to the resource, although they may need more or less than they are entitled to. Then, we initially assign *C*/*n* of the resource to the source with the smallest demand, r_1. This may

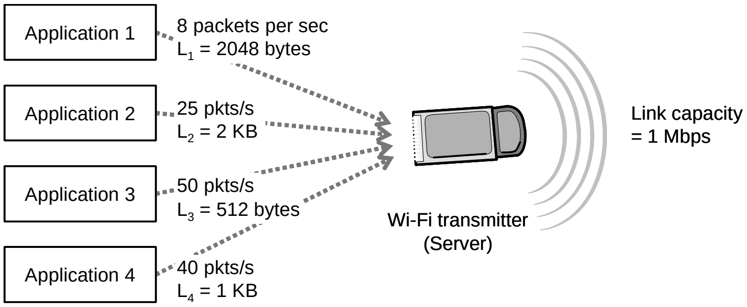


Figure 5-3: Example of a server (Wi-Fi transmitter) transmitting packets from four sources (applications) over a wireless link; see text for details.

be more than what source 1 needs, perhaps, so we can continue the process. The process ends when each source receives no more than what it asks for, and, if its demand was not satisfied, no less than what any other source with a higher index (i.e., demand) received. We call such an allocation a *max-min fair allocation*, because it maximizes the minimum share of sources whose demand cannot be fully satisfied.

Example 5.1 Max-Min Fair Share

Consider the server in Figure 5-3 where packets are arriving from $n = 4$ sources of equal priority to be transmitted over a wireless link. (Assume that the full link bandwidth is available and ignore the link-layer overhead due to interfame spaces, backoff, collisions, etc.) The total required link capacity is:

$$\underbrace{8 \times 2048}_{\text{Appl. A}} + \underbrace{25 \times 2048}_{\text{Appl. B}} + \underbrace{50 \times 512}_{\text{Appl. C}} + \underbrace{40 \times 1024}_{\text{Appl. D}} = \underbrace{134{,}144 \text{ bytes/sec}}_{\text{Total demand}} = 1{,}073{,}152 \text{ bits/sec}$$

but the available capacity of the link is C = 1 Mbps = 1,000,000 bits/sec. By the notion of fairness and given that all sources are equally "important," each source is entitled to C/n = ¼ of the total capacity = 250 Kbps. Some sources may not need this much and the surplus should be equally divided among the sources that need more than their fair share. This table shows the max-min fair allocation procedure.

Sources	Demands [bps]	Balances after 1st round	Allocation #2 [bps]	Balances after 2nd round	Allocation #3 (Final) [bps]	Final balances
Application 1	131,072 bps	+118,928 bps	131,072	0	131,072 bps	0
Application 2	409,600 bps	–159,600 bps	332,064	–77,536 bps	336,448 bps	–73,152 bps
Application 3	204,800 bps	+45,200 bps	204,800	0	204,800 bps	0
Application 4	327,680 bps	–77,680 bps	332,064	+4,384 bps	327,680 bps	0

After the first round in which each source receives ¼C, sources 1 and 3 have excess capacity, because they are entitled to more than what they need. The surplus of C' = 118,928 + 45,200 = 164,128 bps is equally distributed between the sources in deficit, that is sources 2 and 4. After the second round of allocations, source 4 has excess of C'' = 4,384 bps and this is allocated to the only remaining source in deficit, which is source 2. Finally, under the fair resource allocation, sources 1, 3, and 4 have fulfilled their needs, but source 2 remains short of 73.152 Kbps.

Thus far, we have assumed that all sources are equally entitled to the resources. Sometimes, we may want to assign some sources a greater share than others. In particular, we may want to associate ***weights*** $w_1, w_2, ..., w_n$ with sources 1, 2, …, n, to reflect their relative entitlements to the resource. We extend the concept of max-min fair share to include such weights by defining the *max-min weighted fair share allocation* as follows:

- Resources are allocated in order of increasing demand, normalized by the weight
- No source obtains a resource share larger than its demand
- Sources with unsatisfied demands obtain resource shares in proportion to their weights

The following example illustrates the procedure.

Example 5.2 Weighted Max-Min Fair Share

Consider the same scenario as in Example 5.1, but now assume that the sources are weighted as follows: $w_1 = 0.5$, $w_2 = 2$, $w_3 = 1.75$, and $w_4 = 0.75$. The first step is to normalize the weights so they are all integers, which yields: $w'_1 = 2$, $w'_2 = 8$, $w'_3 = 7$, and $w'_4 = 3$. A source i is entitled to $w'_i \cdot \frac{1}{\sum w'_j}$ of the total capacity, which yields 2/20, 8/20, 7/20, and 3/20, for the respective four sources. The following table shows the results of the *weighted* max-min fair allocation procedure.

Src	Demands	Allocation #1 [bps]	Balances after 1st round	Allocation #2 [bps]	Balances after 2nd round	Allocation #3 (Final) [bps]	Final balances
1	131,072 bps	100,000	−31,072	122,338	−8,734 bps	131,072 bps	0
2	409,600 bps	400,000	−9,600	489,354	+79,754 bps	409,600 bps	0
3	204,800 bps	350,000	+145,200	204,800	0	204,800 bps	0
4	327,680 bps	150,000	−177,680	183,508	−144,172 bps	254,528 bps	−73,152 bps

This time around, source 3 in the first round is allocated more than it needs, while all other sources are in deficit. The excess amount of C' = 145,200 bps is distributed as follows. Source 1 receives $\frac{2}{2+8+3} \cdot 145{,}200 = 22{,}338$ bps, source 2 receives $\frac{8}{2+8+3} \cdot 145{,}200 = 89{,}354$ bps, and source 4 receives $\frac{3}{2+8+3} \cdot 145{,}200 = 33{,}508$ bps. Note that in the denominators is always the sum of weights for the currently considered sources. After the second round of allocations, source 2 has excess of C'' = 79,754 bps and this is distributed among sources 1 and 4. Source 1 receives $\frac{2}{2+3} \cdot 79{,}754 = 31{,}902$, which along with 122,338 it already has yields more than it needs. The excess of C''' = 23,168 is given to source 4, which still remains short of 73.152 Kbps.

Min-max fair share (MMFS) defines the ideal fair distribution of a shared scarce resource. Given the resource capacity C and n customers, under MMFS a customer i is guaranteed to obtain at least $C_i = C/n$ of the resource. If some customers need less than what they are entitled to, then other customers can receive more than C/n. Under weighted MMFS (WMMFS), a customer i is guaranteed to obtain at least $C_i = \frac{w_i}{\sum_{j=1}^{n} w_j} C$ of the resource.

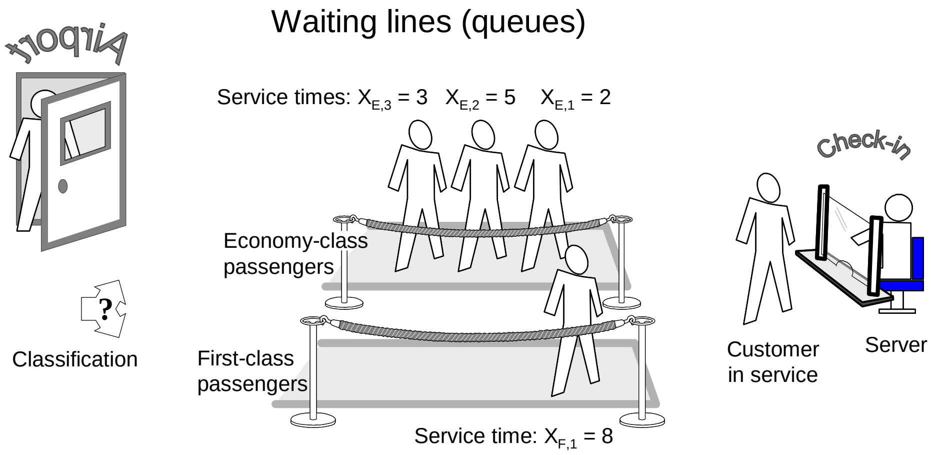
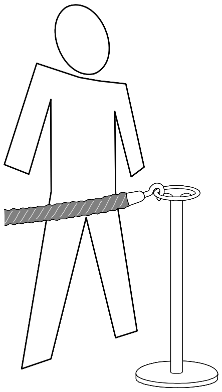


Figure 5-4: Dynamic fair-share problem: in what order should the currently waiting customers be called in service?

However, MMFS assumes that service times X_i are fixed and known ahead of actual service. It does not specify how to achieve this in a *dynamic system* where the demands for resource vary over time. To better understand the problem, consider the airport check-in scenario illustrated in Figure 5-4. Assume there is a single window (server) and both first-class and economy passengers are given the same weight. The question is, in which order the waiting customers should be called for service so that both queues obtain equitable access to the server resource? Based on the specified service times (Figure 5-4), the reader may have an intuition that, in order to maintain fairness on average, it is appropriate to call the first two economy-class passengers before the first-class passenger and finally the last economy-class passenger. The rest of this section reviews practical schemes that achieve just this and, therefore, guarantee (weighted) min-max fair share resource allocation when averaged over a long run.



Generalized Processor Sharing (GPS)

Min-max fair share cannot be directly applied in network scenarios because packets are transmitted as atomic units and they can be of different length, thus requiring different transmission times. In Example 5.1, packets from sources 1 and 2 are twice longer than those from source 4 and four times than from source 3. It is, therefore, difficult to keep track of whether each source is receiving its fair share of the server capacity. To arrive at a practical technique, we start by considering an imaginary technique called *Generalized Processor Sharing* (GPS).

GPS maintains different waiting lines for packets from different flows. Two restrictions apply:

- A packet cannot jump its waiting line, i.e., scheduling within individual queues is FCFS
- Service is *non-preemptive*, meaning that an arriving packet never bumps out a packet that is currently being transmitted (in service)

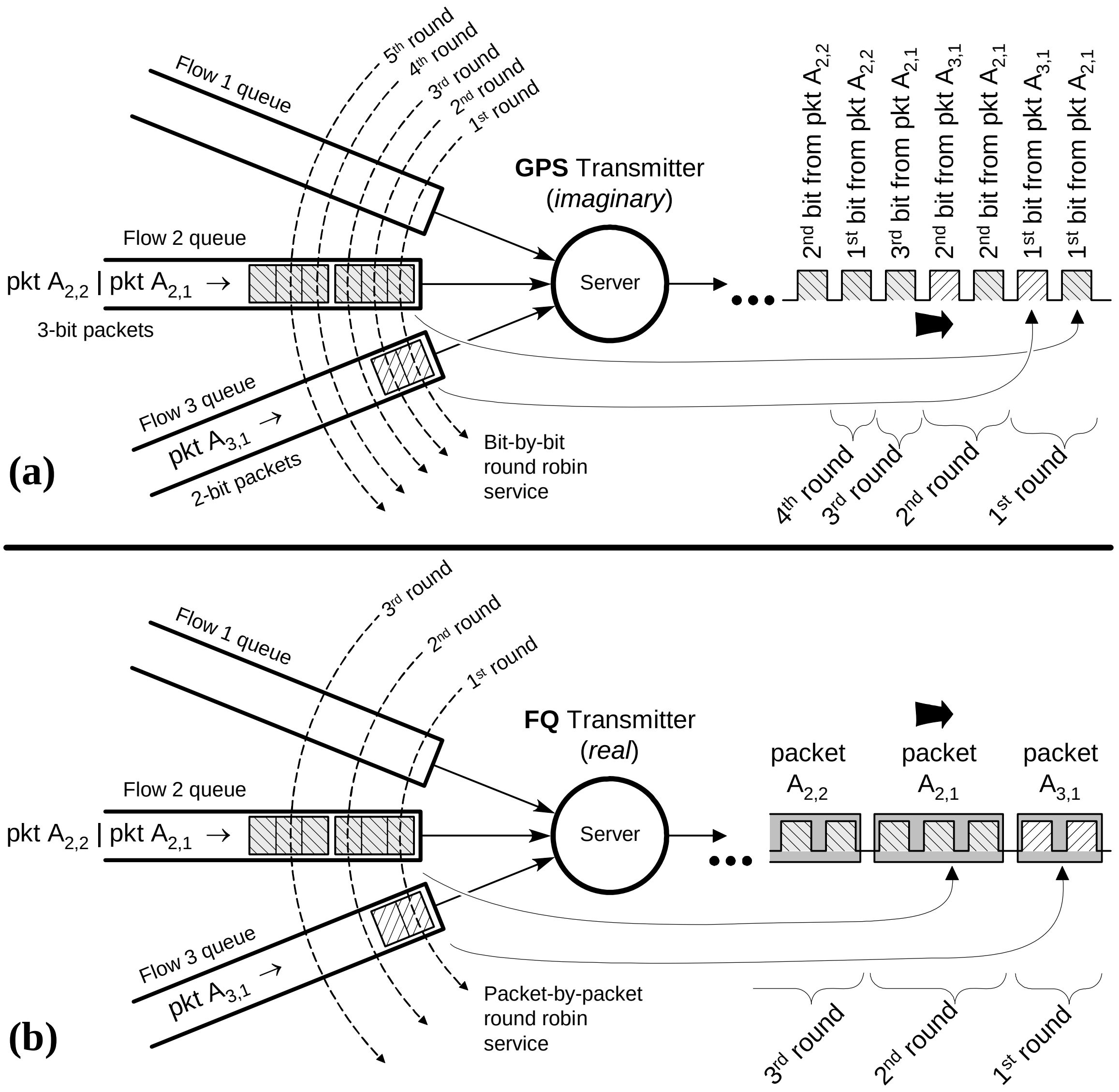


Figure 5-5: Imaginary bit-by-bit GPS (generalized processor sharing) (a) is used to derive the schedule for actual FQ (fair queuing) scheduling (b) used in real routers. According to GPS in (a), packet $A_{3,1}$ finishes first in second round and $A_{2,1}$ finishes second in third round. Therefore, actual FQ transmitter in (b) should transmit $A_{3,1}$ first, then $A_{2,1}$ second and so on.

The unit of service in GPS is one bit of a data packet. GPS works in a *bit-by-bit round robin* fashion, as illustrated in Figure 5-5(a). That is, the router transmits one bit from queue 1 (if there is any), then one bit from queue 2, and so on, for all queues that have packets ready for transmission (and skips without waiting those queues that do not have any packets). Let $A_{i,j}$ denote the time that j^{th} packet arrives from i^{th} flow at the server (transmitter). Let us, for the sake of illustration, consider the example with packets only 2 or 3 bits long. Packets from flows 1 and 2 are 3 bits long, and from flow 3 are 2 bits long. At time zero, packet $A_{2,1}$ arrives from flow 2 and packet $A_{3,1}$ arrives from flow 3. Both packets find no other packets in their respective waiting lines, so their transmission starts immediately, one bit per round. Note that one *round* of transmission now takes *two* time units, because two flows must be served per round. Because the

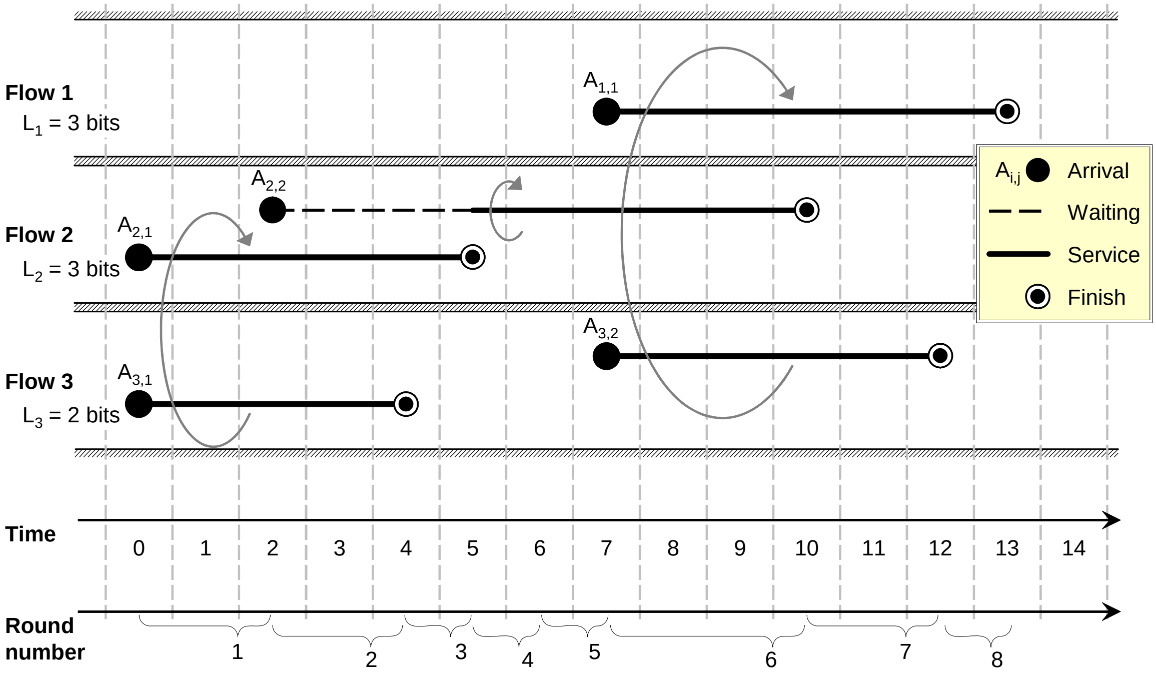


Figure 5-6. Example of bit-by-bit GPS. The output link service rate is *C* = 1 bit/s.

packet from flow 3 is shorter than that from flow 2 (and their transmission started simultaneously), the transmission of packet $A_{3,1}$ will be finished sooner. After this moment, one *round* of transmission now takes *one* time unit, because only one flow (flow 2) must be served per round.

The key idea of Fair Queuing (FQ) is to run imaginary GPS transmissions, as in Figure 5-5(a), determine the packet finish-round numbers under GPS, and then line up the packets for actual transmission under FQ in the ascending order of their finish numbers, as in Figure 5-5(b). In other words, an FQ transmitter makes its timetable of packet transmissions based on the finish-round numbers that it obtains by first running a GPS transmitter simulation. This process will be elaborated in the rest of this section.

The GPS example from in Figure 5-5(a) is continued in Figure 5-6. At time t = 2 there is one more arrival on flow 2: $A_{2,2}$. Because in flow 2 $A_{2,2}$ finds $A_{2,1}$ in front of it, it must wait until the transmission of $A_{2,1}$ is completed. At time t = 7 there are two arrivals: $A_{1,1}$ and $A_{3,2}$. The transmission of both packets starts immediately because currently they are the only packets in their flows. (The bits should be transmitted atomically, one bit from each flow per unit of time as shown in Figure 5-5(a), rather than continuously as shown in Figure 5-6, but because this is an abstraction anyway, we leave it as is.)

As seen, a k-bit packet takes always k rounds to transmit, but the actual time duration can vary, depending on the current number of active flows—the more flows are served in a round, the longer the round will take. (A flow is *active* if it has packets enqueued for transmission.) For example, in Figure 5-6 it takes 4 seconds to transmit the first packet of flow 3, and 5 s for the second packet of the same flow although both have equal length! The reason is that round #6 lasted 3 s because three concurrent flows had to be served. In summary, the number of rounds

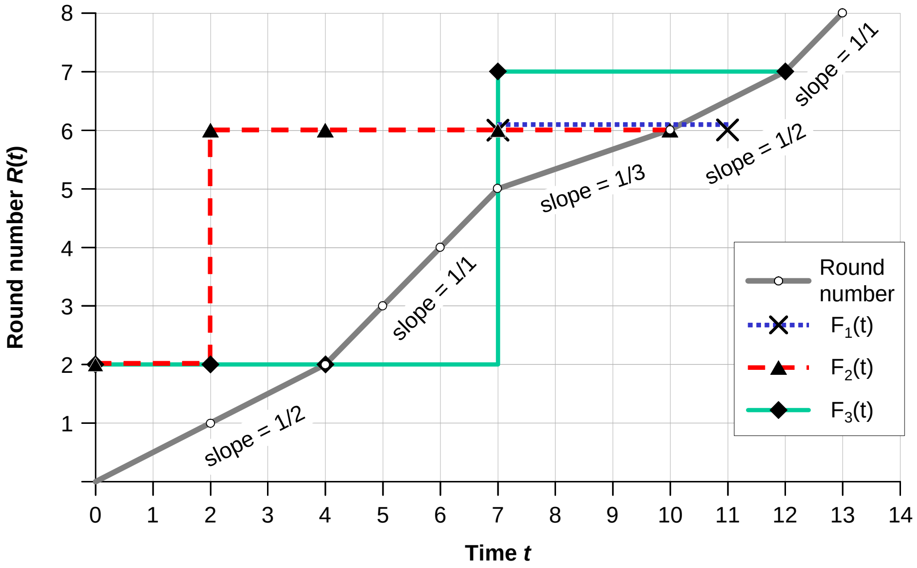


Figure 5-7. Piecewise linear relationship between round number and time for the example from Figure 5-6. Also shown are finish numbers $F_i(t)$ for the different flows under GPS.

(cycles of GPS) to service a packet does not depend on the number of active flows. However, the time it takes for service will shrink or expand depending on the number of active flows.

The piecewise linear relationship between time and round numbers is illustrated in Figure 5-7 for the example from Figure 5-6. The slope of each linear segment is computed as one round divided by the number of bits that need to be transmitted in that round. In other words, the slope is inversely proportional to the current number of active flows. Looking back at Figure 5-5(a), we see that in the beginning, the GPS transmitter needs to transmit two bits per round during the first two rounds (bits from flows 2 and 3). Hence, the slope of the round-number curve is 1/2. At time 4, transmission of the packet from flow 3 is finished and there remains a single active flow, so the slope rises to 1/1. Similarly, at time 7 the slope falls to 1/3 because now three concurrent flows need to be served. In general, the function $R(t)$ increases at a rate

$$\frac{dR(t)}{dt} = \frac{C}{n_{\text{active}}(t)} \tag{5.1}$$

where C is the transmission capacity of the output line. Obviously, if $n_{\text{active}}(t)$ is *constant* then $R(t) = t \cdot C / n_{\text{active}}$, but this need not be the case because packets in different flows arrive randomly. Our problem is to determine the round number $R(t)$ *dynamically*, because it may change any time a new packet arrives. In general, the round number is determined in a piecewise manner $R(t_i) = \sum_{j=1}^{i} \frac{(t_j - t_{j-1}) \cdot C}{n_{\text{active}}(t_j)} + \frac{t_1 \cdot C}{n_{\text{active}}(t_1)}$ as will be seen later in Example 5.3. Each time $R(t)$ reaches a new integer value marks an instant at which all the queues have been given an equal number of opportunities to transmit a bit (of course, an empty queue does not utilize its given opportunity).

GPS provides max-min fair resource allocation. Unfortunately, GPS cannot be implemented because it is not feasible to interleave the bits from different flows. A practical solution is the fair queuing mechanism that approximates this behavior on a packet-by-packet basis, which is presented next.

Fair Queuing

Similar to imaginary GPS, a real router using FQ maintains different waiting lines for packets belonging to different flows at each output port. FQ determines when a packet transmission would finish if it were being sent using GPS (bit-by-bit round robin) and then uses this finishing tag to rank order the packets for transmission.

The service round in which a packet $A_{i,j}$ would finish service under GPS is called the packet's **finish number**, denoted $F_{i,j}$. For example, in Figure 5-5(a) and Figure 5-6 packet $A_{3,1}$ has finish number $F_{3,1} = 2$, packet $A_{2,1}$ has finish number $F_{2,1} = 3$, and so on. It is important to recall that the finish number is, generally, different from the actual time at which the packet is served. For example, packet $A_{3,1}$ is serviced by $t = 4$, and $A_{2,1}$ is serviced by $t = 5$, because time is different from the round number (Figure 5-7).

Every time a new packet arrives, FQ scheduler assigns it a finish number. A key observation is that if we know the round number in which the new packet's service will start, then the packet's finish number depends only on the packet's size. In turn, when the service will start under GPS depends only on whether there are previous packets from this flow in front of the new packet. (Packets in other queues are irrelevant for the starting round of this packet.) The key idea of FQ is that the scheduler lines up the packets from different queues for service based on their ascending finish numbers that are obtained by running an imaginary GPS simulation.

Let $L_{i,j}$ denote the size (in bits) of packet $A_{i,j}$. Under bit-by-bit GPS it takes $L_{i,j}$ rounds of service to transmit this packet. Let $F_{i,j}$ denote the time when the transmitter finishes transmitting j^{th} packet from i^{th} flow. Suppose that a packet arrives at time t_a on a server which previously cycled through $R(t_a)$ rounds. Under GPS, the packet would have to wait for service *only if* there are currently packets from *this* flow either under service or enqueued for service or both—packets from other flows would not affect the start of service for this packet. Therefore, the start round number for servicing packet $A_{i,j}$ is the *highest* of these two

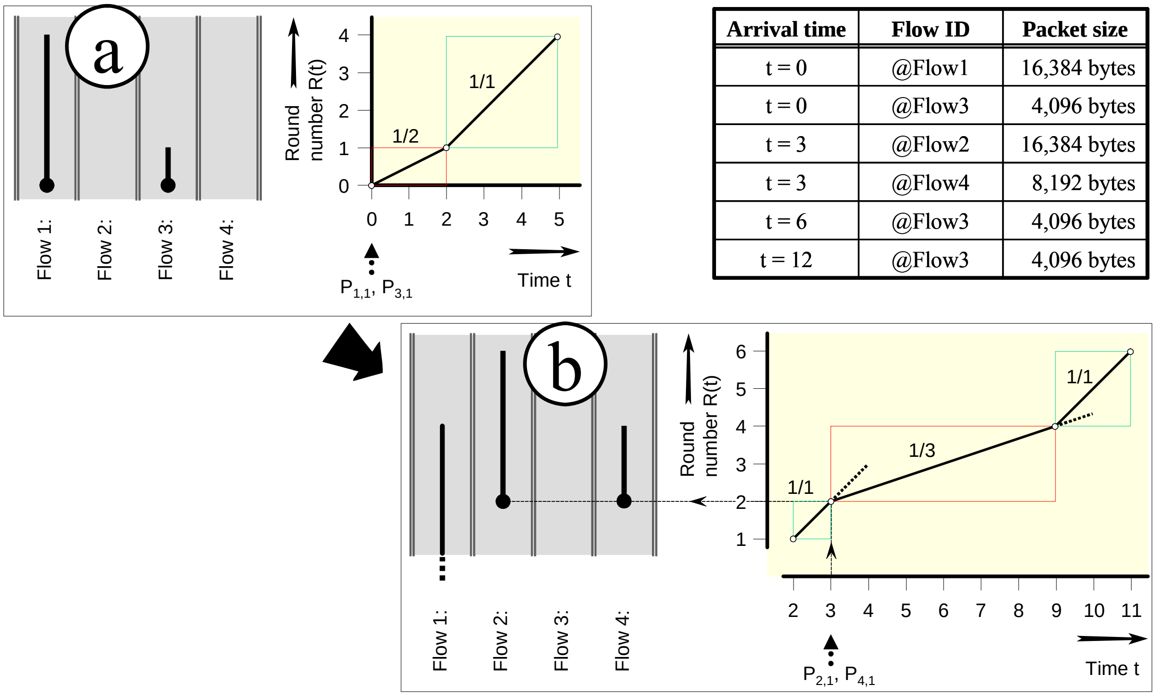
- The current round $R(t_a)$ at the packet's arrival time t_a
- The finishing round of the last packet, if any, from the *same* flow

or in short, the start round number of the packet $A_{i,j}$ is $\max\{F_{i,j-1}, R(t_a)\}$. The finish number of this packet is calculated by adding its transmission cycles under GPS as

$$F_{i,j} = \max\{F_{i,j-1}, R(t_a)\} + L_{i,j} \tag{5.2}$$

Once assigned, the finish number remains constant and does *not* depend on future packet arrivals and departures. As new packets arrive, packet's *finish number* will never change—what changes with new arrivals is that packet's *transmission time* may expand or shrink. FQ scheduler performs the following procedure every time a new packet arrives:

1. Calculate the finish number for the newly arrived packet using Eq. (5.2)
2. For all the packets *currently waiting* for service (in any queue), sort them in the ascending order of their finish numbers



Arrival time	Flow ID	Packet size
t = 0	@Flow1	16,384 bytes
t = 0	@Flow3	4,096 bytes
t = 3	@Flow2	16,384 bytes
t = 3	@Flow4	8,192 bytes
t = 6	@Flow3	4,096 bytes
t = 12	@Flow3	4,096 bytes

Figure 5-8: Determining the round numbers under bit-by-bit GPS for Example 5.3. (a) Initial round numbers as determined at the arrival of packets $P_{1,1}$ and $P_{3,1}$. (b) Round numbers recomputed at the arrival of packets $P_{2,1}$ and $P_{4,1}$. (Continued in Figure 5-9.)

3. When the packet currently in transmission, if any, is finished, call the packet with the *smallest finish number* in service

Note that the sorting in step 2 does not include the packet currently being transmitted, if any, because FQ uses non-preemptive scheduling. Also, it is possible that a new packet may be scheduled ahead of a packet already waiting in a different line because the former is shorter than the latter and its finish number happens to be smaller. The fact that FQ uses non-preemptive scheduling makes it an approximation of the bit-by-bit round robin GPS, rather than an exact simulation.

For example, Figure 5-7 also shows the curves for the finish numbers $F_i(t)$ under GPS of the three flows from Figure 5-6. At time 0, packets $A_{2,1}$ and $A_{3,1}$ arrive simultaneously, but because $F_{3,1}$ is smaller than $F_{2,1}$, packet $A_{3,1}$ goes into service first.

Example 5.3 Packet-by-Packet Fair Queuing

Consider the system from Figure 5-3 and Example 5.1 and assume for the sake of illustration that the time is quantized to the units of the transmission time of the smallest packets. The smallest packets are 512 bytes from flow 3 and on a 1 Mbps link it takes 4.096 ms to transmit such a packet. For the sake of illustration, assume that a packet arrives on flows 1 and 3 each at time zero, then a packet arrives on flows 2 and 4 each at time 3, and packets arrive on flow 3 at times 6 and 12. Show the corresponding packet-by-packet FQ scheduling.

The first step is to determine the round numbers for the arriving packets, given their arrival times. The process is illustrated in Figure 5-8. The round numbers are shown also in the units of the smallest

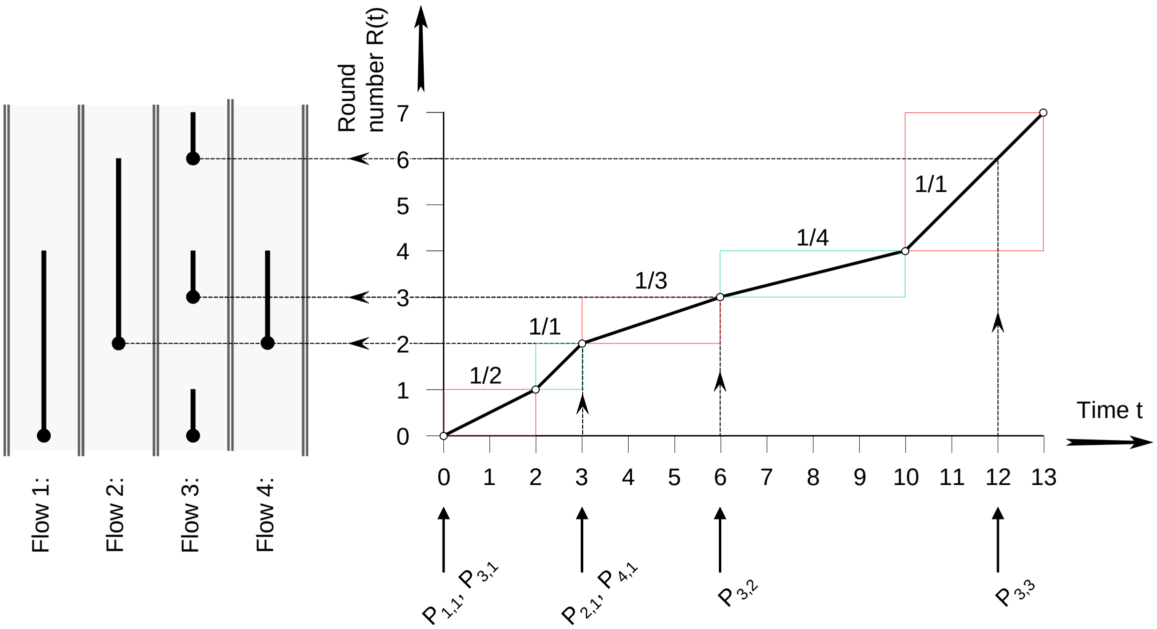


Figure 5-9: Determining the round numbers under bit-by-bit GPS for Example 5.3, completed from Figure 5-8.

packet's number of bits, so these numbers must be multiplied by 512×8=4096 to obtain the actual round number. Bit-by-bit GPS would transmit bits from two packets ($A_{1,1}$ and $A_{3,1}$) in the first round, so the round takes two time units and the slope is 1/2 (as seen in Figure 5-8(a)). In the second round, only one packet is being transmitted ($A_{1,1}$), the round duration is one time unit and the slope is 1/1. The GPS server completes two rounds by time 3, $R(3) = 2$, at which point two new packets arrive ($A_{2,1}$ and $A_{4,1}$). The next arrival is at time 6 on flow 3 (the actual time is $t_1 = 6\times4.096 = 24.576$ ms) and the round number is determined as

$$R(t_3) = \frac{(t_3 - t_2)\cdot C}{n_{\text{active}}(t_3)} + R(t_2) = \frac{(0.024576 - 0.012288)\times 1000000}{3} + 8192 = 4094 + 8192 = 12288$$

which in the units of multiples of smallest-packet-transmission-time is $R(6) = 12288/4096 = 3$. The left side of each diagram in Figure 5-8 also shows how the packet arrival times are mapped into the round numbers. Figure 5-9 summarizes the process of determining the round numbers for all arriving packets.

The actual order of transmissions under packet-by-packet FQ is shown in Figure 5-10. At time 0 the finish numbers are: $F_{1,1} = 4$ and $F_{3,1} = 1$, so packet $A_{3,1}$ is transmitted first and packet $A_{1,1}$ goes second. At time 3 the finish numbers for the newly arrived packets are: $F_{2,1} = \max\{0, R(3)\} + L_{2,1} = 2 + 4 = 6$ and $F_{4,1} = \max\{0, R(3)\} + L_{4,1} = 2 + 2 = 4$, so $F_{4,1} < F_{2,1}$. The ongoing transmission of packet $A_{1,1}$ is *not* preempted and will be completed at time 5, at which point packet $A_{4,1}$ will enter the service. At time 6 the finish number for packet $A_{3,2}$ is $F_{3,2} = \max\{0, R(6)\} + L_{3,2} = 3 + 1 = 4$. The current finish numbers are $F_{3,2} < F_{2,1}$ so $A_{3,2}$ enters the service at time 7, followed by $A_{2,1}$ which enters the service at time 8. Finally, at time 12 the finish number for the new packet $A_{3,3}$ is $F_{3,3} = \max\{0, R(12)\} + L_{3,3} = 6 + 1 = 7$ and it is transmitted at 12.

In summary, the order of arrivals is $\{A_{1,1}, A_{3,1}\}$, $\{A_{2,1}, A_{4,1}\}$, $A_{3,2}$, $A_{3,3}$ where simultaneously arriving packets are delimited by curly braces. The order of transmissions under packet-by-packet FQ is: $A_{3,1}$, $A_{1,1}$, $A_{4,1}$, $A_{3,2}$, $A_{2,1}$, $A_{3,3}$.

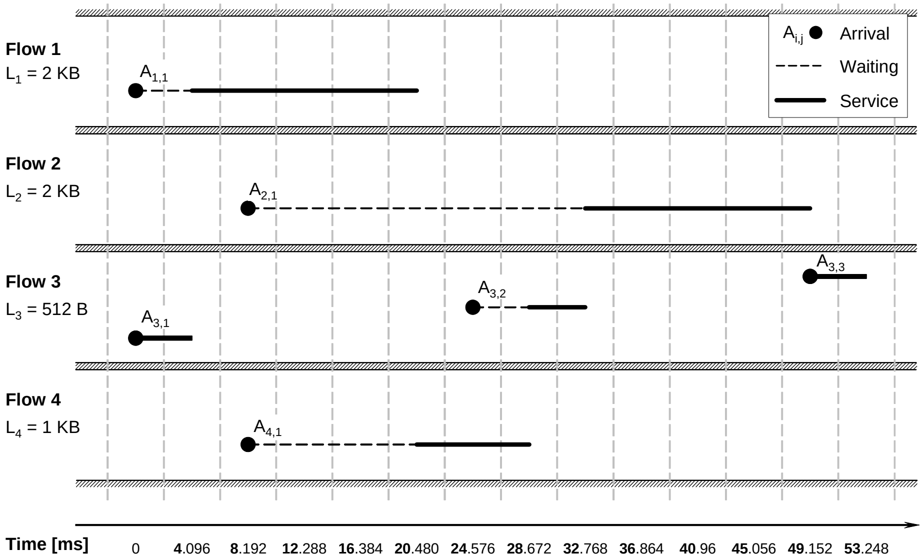


Figure 5-10: Time diagram of packet-by-packet FQ for Example 5.3.

There is a problem with the above algorithm of fair queuing which the reader may have noticed besides that computing the finish numbers is no fun at all! At the time of a packet's arrival we know only the current time, not the current round number. As suggested above, one could try using the round number slope, Eq. (5.1), to compute the current round number from the current time, but the problem with this approach is that the round number slope is not necessarily constant. An FQ scheduler computes the current round number on every packet arrival, to assign the finish number to the new packet. Because the computation is fairly complex, this poses a major problem with implementing fair queuing in high-speed networks. Some techniques for overcoming this problem have been proposed, and the interested reader should consult [Keshav 1997].

5.1.3 Weighted Fair Queuing

Now assume that weights $w_1, w_2, ..., w_n$ are associated with sources (flows) 1, 2, …, n, to reflect their relative entitlements to transmission bandwidth. As before, a queue is maintained for each source flow. Under weighted min-max fair share, flow i is guaranteed to obtain at least $C_i = \frac{w_i}{\sum w_j} C$ of the total bandwidth C. The bit-by-bit approximation of weighted fair queuing (WFQ) would operate by allotting each queue a different number of bits per round. The number of bits per round allotted to a queue should be proportional to its weight, so the queue with twice higher weight should receive two times more bits/round.

Packet-by-packet WFQ can be generalized from bit-by-bit WFQ as follows. For a packet of length $L_{i,j}$ (in bits) that arrives at t_a the finish number under WFQ is computed as

$$F_{i,j} = \max\left(F_{i,j-1},\ R(t_a)\right) + \frac{L_{i,j}}{w_i} \tag{5.3}$$

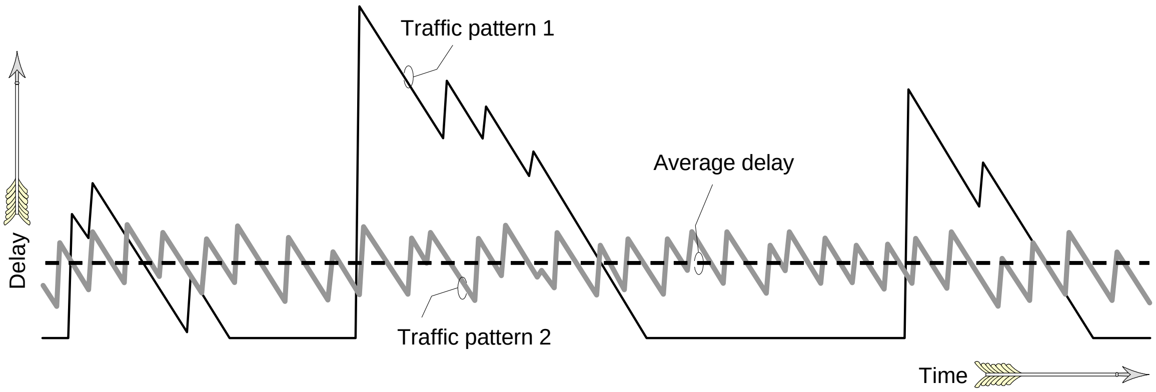


Figure 5-11: Different traffic patterns yield the same average delay.

From the second term in the formula, we see that if a packet arrives on each of the flows *i* and *k* and $w_i = 2 \cdot w_k$, then the finish number for a packet of flow *i* is calculated assuming a bit-by-bit depletion rate that is twice that of a packet from flow *k*.

All queues are set to an equal maximum size, so the flows with the highest traffic load will suffer packet discards more often, allowing lower-traffic ones a fair share of capacity. Hence, there is no advantage of being greedy. A greedy flow finds that its queues become long, because its packets in excess of fair share linger for extended periods of time in the queue. The result is increased delays and/or lost packets, whereas other flows are unaffected by this behavior. In many cases delayed packets can be considered lost because delay sensitive applications ignore late packets.

The problem is created not only for the greedy source, but lost packets represent wasting network resources upstream the point at which they are delayed or lost. Therefore, they should not be allowed into the network at the first place. This is a task for policing.

5.2 Policing

So far, we saw how to distribute fairly the transmission bandwidth or other network resources using WFQ scheduler. However, this does not guarantee delay bounds and low losses to traffic flows. A packet-by-packet FQ scheduler guarantees a fair distribution of the resource, which results in a certain average delay per flow. However, even an acceptable average delay may have great variability for individual packets. This point is illustrated in Figure 5-11. Multimedia applications are particularly sensitive to delay variability, known as “jitter” (Chapter 3).

One idea is to regulate the number of packets that a particular flow can pass through the router per unit of time by using the “leaky bucket” abstraction (Figure 5-12). Imagine that we install a turnstile (ticket barrier) inside the router to regulate the entry of packets into the service. To pass the turnstile (regulator), each packet must drop a token into the slot and proceed. Tokens are dispensed from a “leaky bucket” that can hold up to *b* tokens. If the bucket is currently empty, the

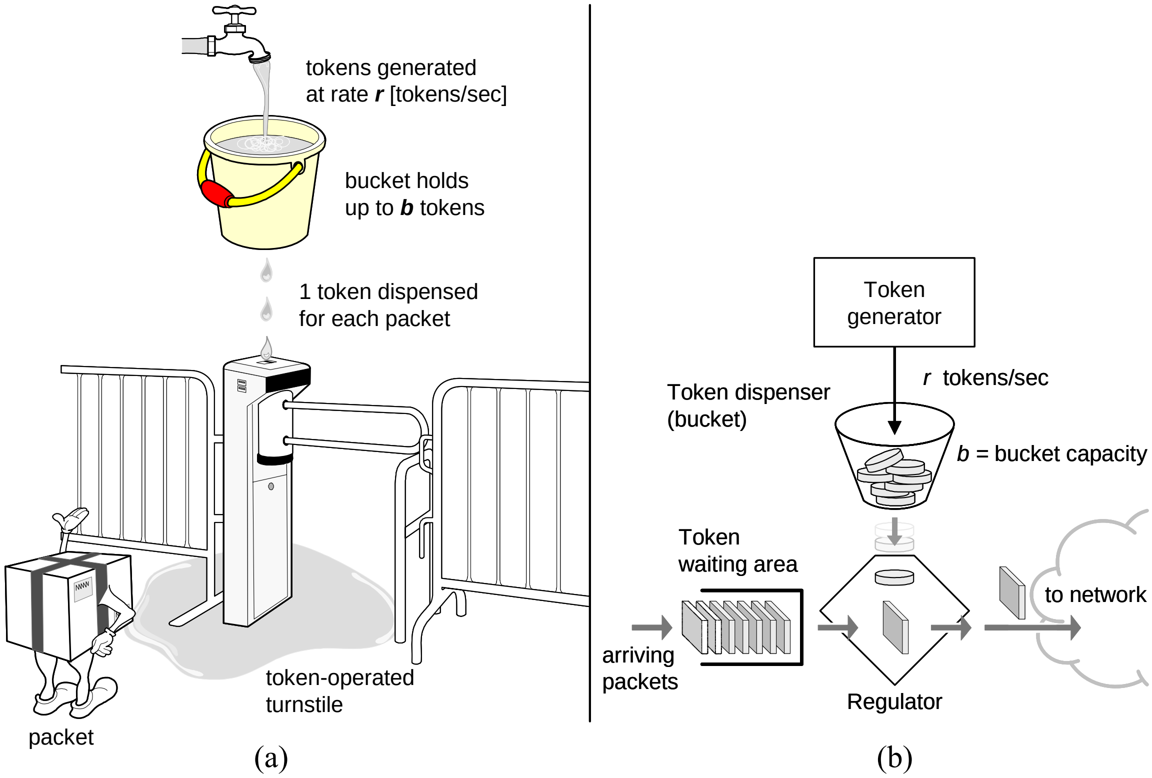


Figure 5-12: Leaky bucket mechanism: (a) metaphor and (b) abstract representation.

packets must wait for a token. If a packet arrives at a fully occupied waiting area, the packet is dropped.[19]

Each flow has a quota that is characterized by simple traffic descriptors (Section 3.2.1):

Peak rate: this parameter constrains the number of packets that a flow can send over a very short time interval.

Average rate: this parameter specifies the average number of packets that a particular flow is allowed to send over a relatively long time window. As discussed in Section 3.2.1, a key issue here is to decide the window length over which the average rate will be regulated.

Burst size: this parameter constrains the total number of packets (the "burst" of packets) that can be sent by a particular flow into the network over a short interval of time.

When a packet arrives at a router, it withdraws one token from the bucket before it is allowed to proceed. If the bucket is empty, the packet must wait for a token to appear.

When the packets are all the same size, this algorithm can be used as described. However, when variable-sized packets are being used, it is often better to allow a fixed number of bytes per token, rather than just one packet.

[19] There are many variations of the leaky bucket algorithm and different books introduce this abstraction in different way. In some variations, there are no tokens and the packets themselves arrive to the bucket and drain through a hole in the bucket. Because this is an abstraction anyway, I present it here in a way that I feel is the most intuitive. This does not affect the results of the algorithm.

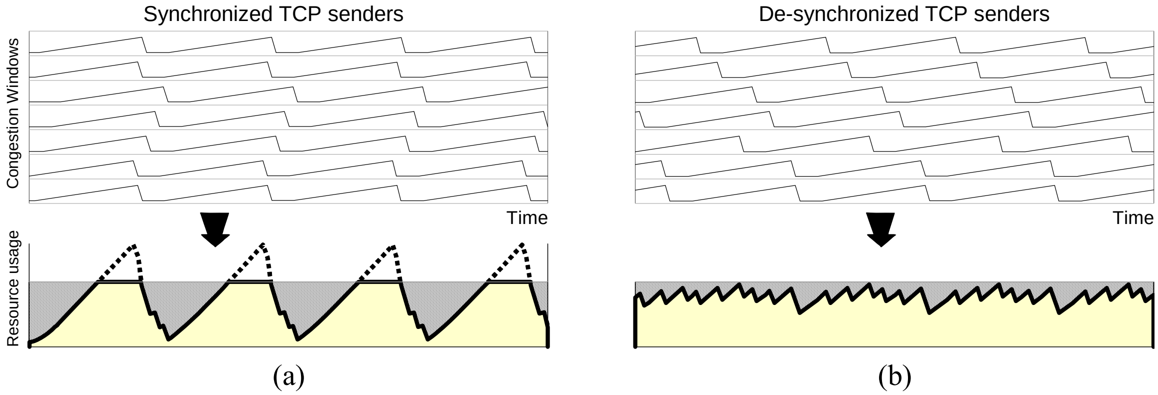


Figure 5-13: (a) Under drop-tail policy for router queue management, all TCP flows will have packets dropped at about the same time, leading to sender synchronization. (b) Active queue management results in sender de-synchronization and more even usage of network resources.

5.3 Active Queue Management

Packet-dropping strategies deal with the case when there is not enough memory to buffer an incoming packet. When the router runs out of memory space, the simplest policy is to drop the arriving packet, known as *drop-tail* policy.

A problem with drop-tail policy is that it may lead to **synchronization of senders** and *oscillatory behavior*. If a router waits until its buffer overflows, then all connections will start losing packets simultaneously. In turn, all of them will back off in synchrony and start building up their congestion windows (Figure 5-13(a)). The result is an oscillatory behavior in the network: first all flows build up, then all are throttled, next all build up, and so on. This behavior is undesirable because, on average, network resources are poorly utilized.

Active Queue Management (AQM) algorithms employ more sophisticated approaches. Routers with AQM detect congestion *before* the queue overflows and notify the source that the congestion is about to happen. This strategy allows the senders to *desynchronize* and back-off/ramp-up at different times (Figure 5-13(b)). Such behavior results in better average utilization of network bandwidth.

One of the most widely studied and implemented AQM algorithms is the *Random Early Detection* (RED) algorithm that uses *implicit feedback* by dropping packets. A recent approach, called *Explicit Congestion Notification* (ECN) uses *explicit feedback* by marking packets instead of dropping them.

5.3.1 Random Early Detection (RED)

Problems related to this section: Problem 5.13 → Problem 5.14

A router that implements RED uses two threshold values to mark positions in the queue: $Threshold_{min}$ and $Threshold_{max}$. A simplified description of RED operation follows. When a new packet arrives, its disposition is decided by these three rules:

1. If the queue currently contains fewer than $Threshold_{min}$ packets, the new packet is enqueued.
2. If the queue contains between $Threshold_{min}$ and $Threshold_{max}$ packets, the new packet is considered for enqueuing or dropping by generating a random number and evaluating its value.
3. If the queue currently contains more than $Threshold_{max}$ packets, the new packet is dropped.

where $0 \le Threshold_{min} < Threshold_{max} \le$ BufferSize, and the value 0 represents the head of the queue. Therefore, instead of waiting until the queue overflows, RED starts randomly dropping packets as congestion increases. The process is somewhat more complex, as described next.

We know that TCP often sends segments in bursts, depending on the congestion window size (Section 2.2). A burst represents a spike in traffic intensity that may last only shortly while most of the time traffic is low-intensity. In other words, the queue may be most of the time empty, and a temporary spike does not represent congestion. For a temporary spike, by the time the sender learns about the congestion and reacts, the spike would have passed, and we would be unnecessarily throttling the sender. Therefore, we should like avoid dropping packets from an isolated burst when queue length is greater than $Threshold_{min}$. (Of course, packets are always dropped if the queue capacity is exceeded.)

To better capture the notion of congestion and accommodate for bursty traffic, the router does *not* consider the *instantaneous* length of the queue. Instead, the router considers the *average* length of the queue when applying the three rules described above (Figure 5-14(a)). The average queue length is computed continuously using Exponential Weighted Moving Average (EWMA). This is the same method used by TCP for RTT estimation (Section 2.1.3) and by jitter buffer (Section 3.3.1). That is, at any time *t* when a new packet arrives and tries to join this queue, we compute

$$AverageQLen(t) = (1-\gamma) \cdot AverageQLen(t-1) + \gamma \cdot MeasuredQLen(t) \tag{5.4}$$

where γ denotes a value between 0 and 1. If γ is small, the average stays close to long-term trend and does not fluctuate for short bursts. This parameter should be determined empirically, and a recommended value is $\gamma = 0.002$.

When the average queue length is between $Threshold_{min}$ and $Threshold_{max}$, RED drops an arriving packet with an increasing probability as the average queue length increases (Figure 5-14(b)). As Figure 5-14(b) shows, for values $AverageQLen < Threshold_{min}$, the probability of a packet drop is zero, then grows linearly from zero to a maximum value $P_{\max}$, and finally jumps to 1 for $AverageQLen > Threshold_{max}$. For a given *AverageQLen* such that $Thr_{min} \le AvgQLen \le Thr_{max}$, we calculate the probability of a packet drop as

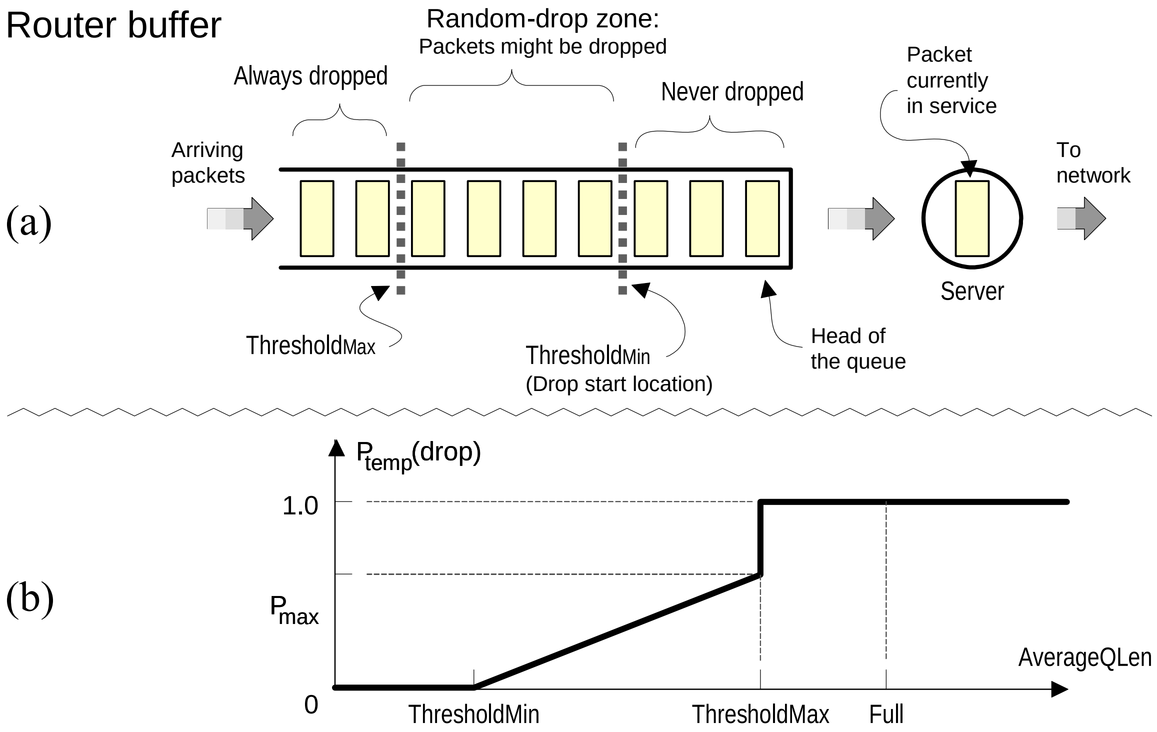


Figure 5-14: (a) RED thresholds on a FCFS (or, FIFO) queue. (b) Packet drop probability function for RED.

$$P_{\text{temp}}(AverageQLen) = P_{\max} \times \frac{(AverageQLen - Threshold_{\min})}{(Threshold_{\max} - Threshold_{\min})} \tag{5.5}$$

Research has suggested that RED works best when the probability function transitions smoothly at $Threshold_{max}$, i.e., for $P_{\max} = 1$. In addition to the average queue length, the drop probability also depends on the time elapsed since the last packet was dropped. Instead of actual time, the algorithm uses a proxy in terms of the number of newly arriving packets (variable: *count*) that have been queued (not dropped) while the *AverageQLen* value has been between the two thresholds ($Thr_{min} \leq AvgQLen < Thr_{max}$). Therefore, given an *AverageQLen* the actual probability of the arrived packet being dropped is computed as

$$P(AverageQLen) = \frac{P_{\text{temp}}(AverageQLen)}{1 \; - \; count \times P_{\text{temp}}(AverageQLen)} \tag{5.6}$$

We can observe that *P* increases as *count* increases. This behavior helps make packet drops more evenly distributed and avoid bias against bursty traffic. Here is a summary of the RED algorithm:

Listing 5-1: Summary of the RED algorithm.

```
Set $count = -1$
For **each** newly arrived packet, do:
    1. Calculate $AvgQLen$ using equation (5.4)
    2. If $Thr_{min} \leq AvgQLen < Thr_{max}$
            a) increment $count \leftarrow count + 1$
            b) calculate the drop probability for this packet using equation (5.6)
            c) decide if to drop the packet (Y/N) and drop it if decided YES
            d) if in step c) the packet is dropped, reset $count \leftarrow 0$
    3. Else if $Thr_{max} \leq AvgQLen$
            a) drop the packet and reset $count \leftarrow 0$
    4. Otherwise, reset $count \leftarrow -1$
End of the for loop
```

RED is intended to work primarily with TCP sources. RED is a queue management technique that attempts to provide equitable access to an FCFS system. The source that transmits at the highest rate will suffer from higher packet-dropping rate than others will. As a result, this source will reduce its transmission rate more, which yields more uniform transmission rates across the sources and more equitable access to the router-buffer resource. Under RED, TCP connections will experience randomly dropped packets (rather than synchronously when the buffer becomes full). Therefore, TCP senders will back off at different times (Figure 5-13(b)). This behavior avoids the *global synchronization effect* of all connections and maintains high throughput in the routers. Both analysis and simulations shows that RED works as intended, providing high utilization of network resources. It handles congestion, avoids the synchronization that results from drop-tail policy, and allows short bursts without dropping packets unnecessarily. It is also important that RED can control the queue length irrespective of endpoint sender cooperation. The IETF now recommends that routers implement RED.

5.3.2 Explicit Congestion Notification (ECN)

Problems related to this section: Problem 5.16

Explicit Congestion Notification (ECN) allows end-to-end notification of network congestion without dropping packets. It requires support by the underlying network (i.e., routers). ECN is an optional feature that is only used when both endpoints of a connection support it and have it activated.

We have seen that Random Early Detection (RED) mechanism drops packets when it senses potential congestion (Section 5.3.1). Unlike RED, instead of dropping a packet, an ECN-aware router may set a mark in the packet's IP header in order to signal that congestion is about to happen. The receiver of the packet is the first to learn about the potential congestion and echoes the congestion indication to the sender. The receiver uses the next acknowledgement packet to inform the sender about the impending congestion, which must react as if a packet was dropped (as if its RTO timer fired or it received ≥3 duplicate ACKs), i.e., by reducing its congestion window (Section 2.2).

Figure 5-15

ECN uses two bits in the IP header of packets from ECN-capable transports to allow routers to record congestion (Figure 5-16(a)), and uses two bits in the TCP header to allow the TCP sender and receiver to communicate (Figure 5-16(b)). The two bits in the IP header are taken from the 8-bit Differentiated Services field (Figure 1-39) and are called the *Congestion Experienced* (CE) codepoint. These are bits 6 and 7 (rightmost) of the Differentiated Services field, and a router can

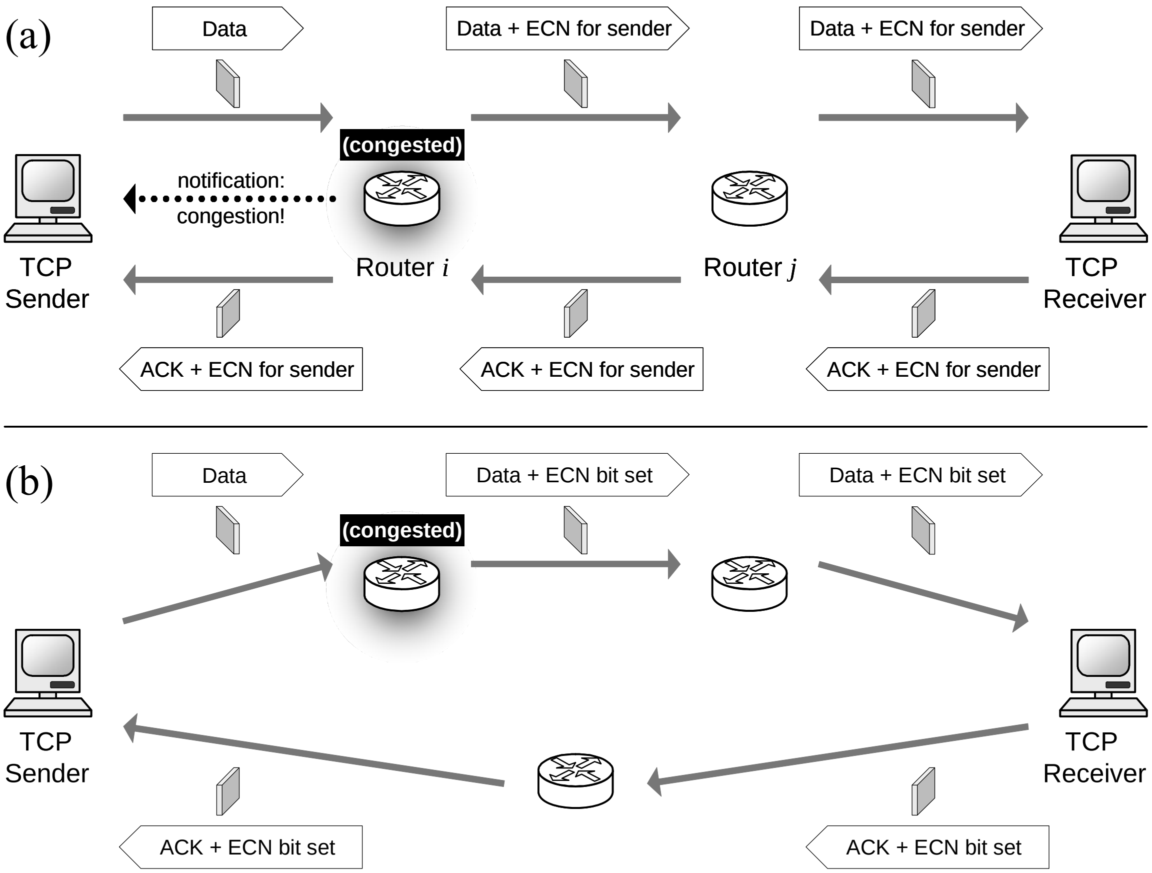


Figure 5-15: Explicit Congestion Notification (ECN)

choose to set either bit to indicate congestion. The reason for using two bits is to increase the robustness of the mechanism. The four different codepoints are as follows:

00: Transport not ECN-capable — Not-ECT

10: ECN capable transport — ECT(0)

01: ECN capable transport — ECT(1)

11: Congestion encountered — CE

The two bits in the TCP header are taken from the 6-bit unused field (Figure 2-3).

The use of the CE codepoint with ECN allows the receiver(s) to receive the packet, avoiding the potential for excessive delays due to retransmissions after packet losses.

The ECN-Capable Transport (ECT) codepoints "10" and "01" are set by the data sender to indicate that the end-points of the transport protocol are ECN-capable; we call them ECT(0) and ECT(1) respectively. Routers treat the ECT(0) and ECT(1) codepoints as equivalent. Senders are free to use either the ECT(0) or the ECT(1) codepoint to indicate ECT, on a packet-by-packet basis.

The not-ECT codepoint "00" indicates a packet that is not using ECN. The CE codepoint "11" is set by a router to indicate congestion to the end nodes. Routers that have a packet arriving at a full queue drop the packet, just as they do in the absence of ECN.

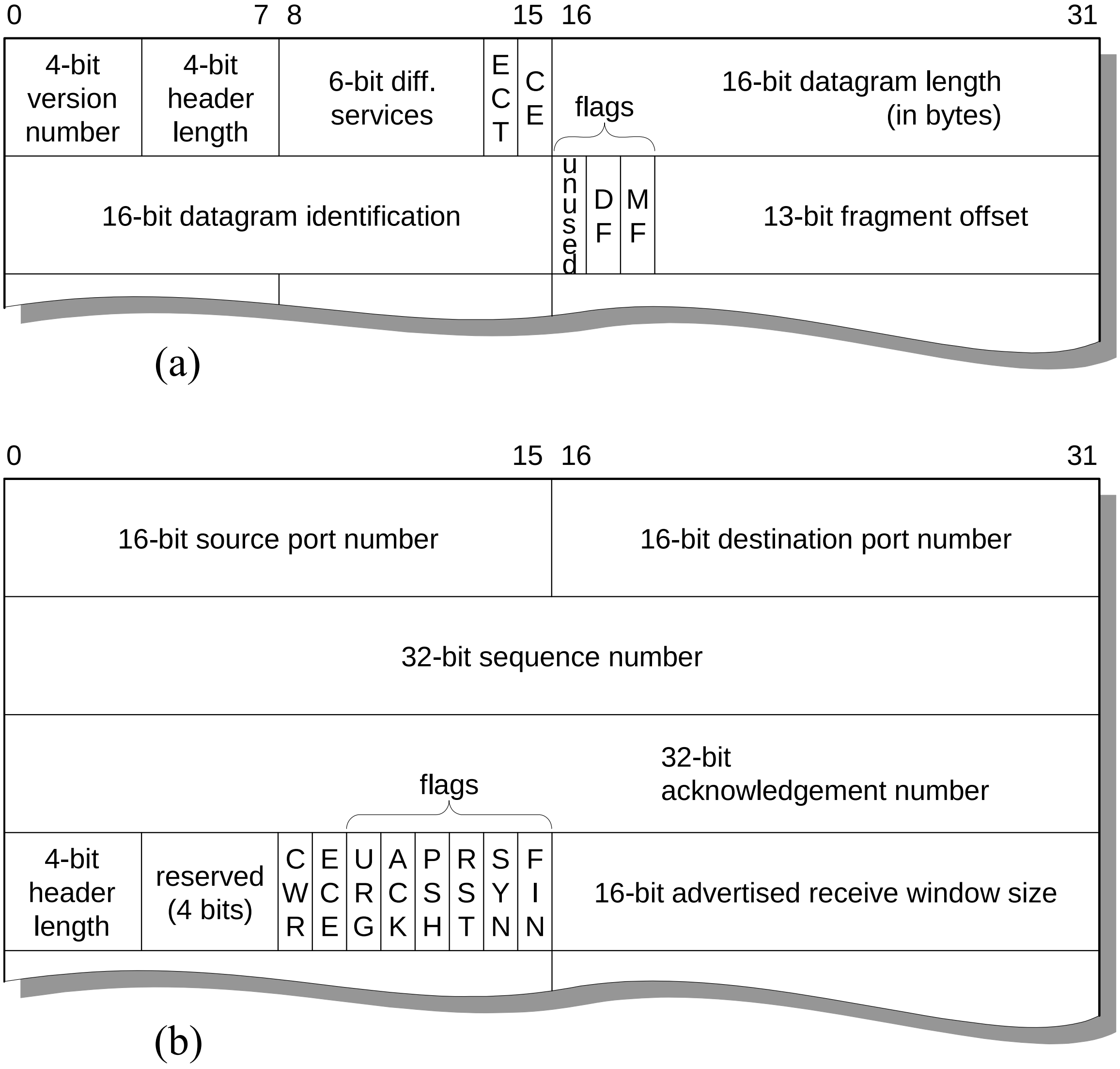
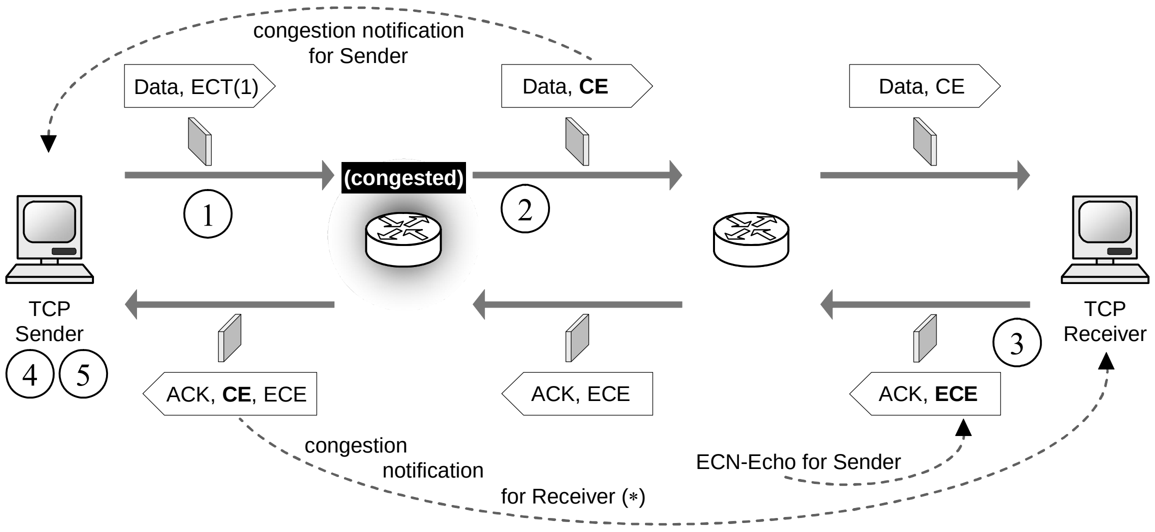


Figure 5-16: ECN-based modifications: (a) IP header modification to add ECT and CE flags in the first 32-bit word. (b) TCP header modification to add CRW and ECE flags.

For a router, the CE codepoint of an ECN-Capable packet should only be set if the router would otherwise have dropped the packet as an indication of congestion to the end nodes. ECN uses the same mechanism as RED (Section 5.3.1) to detect an impending congestion: it monitors the average queue size. The same rules described in Listing 5-1 are applied to detect a congestion onset. However, instead of dropping packets, ECN just marks them to indicate a congestion onset.

A router might drop a non-ECN-Capable packet but set the CE codepoint in an ECN-Capable packet, for equivalent levels of congestion. When a packet is received by a router that already has the CE codepoint set, the router leaves the CE codepoint unchanged and transmits the packet is as usual. When severe congestion has occurred and the router's queue is full, then the router has no choice but to drop some packet when a new packet arrives. It is anticipated that such packet losses will become relatively infrequent when a majority of end-systems become ECN-Capable and participate in TCP or other compatible congestion control mechanisms.



1 An ECT codepoint is set in packets transmitted by the sender to indicate that ECN is supported by the transport entities for these packets.

2 An ECN-capable router detects impending congestion and detects that an ECT codepoint is set in the packet it is about to drop. Instead of dropping the packet, the router chooses to set the CE codepoint in the IP header and forwards the packet.

3 The receiver receives the packet with the CE codepoint set, and sets the ECN-Echo flag in its next TCP ACK sent to the sender.

4 The sender receives the TCP ACK with ECN-Echo set, and reacts to the congestion as if a packet had been dropped.

5 The sender sets the CWR flag in the TCP header of the next packet sent to the receiver to acknowledge its receipt of and reaction to the ECN-Echo flag.

Figure 5-17: Illustration of the typical Explicit Congestion Notification (ECN) operation.

Figure 5-17 illustrates the typical operation of ECN. The TCP sender transmits *only new data* packets with an ECT codepoint set in the IP header. In other words, retransmitted data packets and “pure” ACK packets carry a Not-ECT codepoint in their IP header.

A *single* packet with the CE codepoint set in an IP packet causes the transport layer to respond, in terms of congestion control, as it would to a packet drop. If the sender receives an ECN-Echo (ECE) ACK packet (that is, an ACK packet with the ECN-Echo flag set in the TCP header), then the sender knows that congestion was encountered in the network on the path from the sender to the receiver. The indication of congestion should be treated just as a congestion loss in non-ECN-Capable TCP. That is, the TCP source reduces the congestion window size and the slow start threshold as described in Chapter 2. The TCP sender *should not* increase the congestion window in response to the receipt of an ECN-Echo ACK packet. The TCP receiver uses the Congestion Window Reduced (CWR) flag received from the TCP sender to determine when to stop setting the ECN-Echo flag. After a TCP receiver sends an ACK packet with the ECN-Echo bit set, this receiver will continue to set the ECN-Echo flag in all the ACK packets it sends (whether they acknowledge CE data packets or non-CE data packets) until it receives a packet with the CWR flag set. After receiving a CWR packet, the receiver ceases setting the ECN-Echo flag in the subsequent ACKs. If in the future another CE packet is received by the data receiver, the receiver would again send ACK packets with the ECN-Echo flag set.

When an ECN-Capable TCP sender reduces its congestion window for any reason (because of a retransmit timeout, a Fast Retransmit, or in response to an ECN Notification), the TCP sender sets the CWR flag *only in the first new* data packet sent after the window reduction. If that data packet is dropped in the network, then the TCP sender may receive another ACK with the ECN-Echo flag set or detect loss by regular means (Chapter 2). In any case, the sender will have to reduce the congestion window again and retransmit the dropped packet. The CWR bit in the TCP header *should not* be set on retransmitted packets. Again, the sender should set the CWR bit only in the first *new* data packet that it transmits after it reduced its congestion window.

TCP should not react to congestion indications more than once every window of data (or more loosely, more than once every round-trip time). That is, the TCP sender's congestion window should be reduced only once in response to a series of dropped and/or CE packets from a single window of data. In addition, the TCP source should not decrease the slow-start threshold if it has been decreased within the last round trip time. However, if any retransmitted packets are dropped, then this is interpreted by the source TCP as a new instance of congestion.

Transport protocols such as TCP do not necessarily detect all packet drops, such as the drop of a “pure” ACK packet with zero payload; for example, TCP does not reduce the arrival rate of subsequent ACK packets in response to an earlier dropped ACK packet. Therefore, “pure” ACK packets *must not* indicate ECN-Capability, i.e., they must use codepoint “00.” Because this indicates a non-ECT-capable source, the router cannot set congestion notifications on the ACK path to notify the TCP receiver side about the congestion.

5.4 Multiprotocol Label Switching (MPLS)

Problems related to this section: Problem 5.18 → Problem 5.23

Multiprotocol Label Switching (MPLS) is essentially a mechanism for creating and using special paths, known as “tunnels,” in IP networks. We know that IP forwards data packets on a packet-by-packet basis, where each packet is forwarded independently of any other packet, possibly over different routes even if they are heading to the same destination. This approach is said to be “connectionless packet forwarding.” MPLS allows forwarding packets on a flow-by-flow basis, so that all packets belonging to a given traffic flow are forwarded in the same manner and along the same network path (or, tunnel). In this sense, MPLS supports connection-oriented packet forwarding and the fixed forwarding paths (tunnels) represent “virtual circuits” in the network.

MPLS relies on IP addresses and IP routing protocols to set up the paths/tunnels. An MPLS-enabled router is known as a **Label Switching Router (LSR)**. A set of LSRs where each LSR is reachable from any other LSR via some other LSRs in the same set is called an **MPLS domain**. In other words, an MPLS domain is formed by a contiguous network of LSRs. A label switching router forwards packets by examining a short, fixed-length MPLS label (Figure 5-18). The label

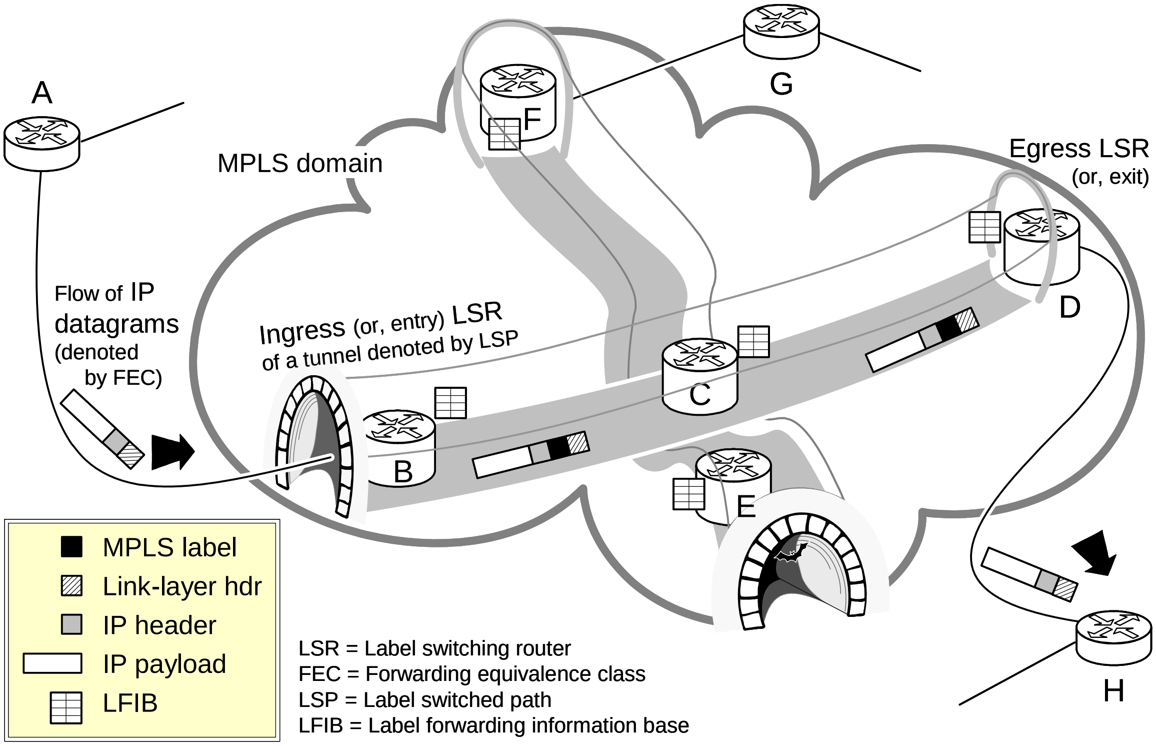
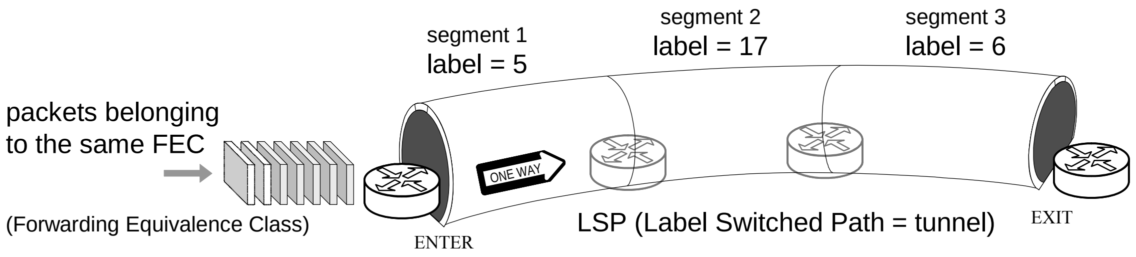

A
MPLS domain
F
G
Egress LSR
(or, exit)
Flow of IP datagrams (denoted by FEC)
Ingress (or, entry) LSR
of a tunnel denoted by LSP
D
C
B
E
H
MPLS label
Link-layer hdr
IP header
IP payload
LFIB
LSR = Label switching router
FEC = Forwarding equivalence class
LSP = Label switched path
LFIB = Label forwarding information base

Figure 5-18: MPLS operation. MPLS label is inserted at the ingress router and removed at the egress router. The label identifies the MPLS tunnel to which this packet belongs.

represents a given traffic flow, and all the packets belonging to this flow should be forwarded along the path/route/tunnel associated with this label. The label is inserted in the packet at its *ingress router* (*B* in Figure 5-18), which is an entry router to the MPLS domain, based on a service agreement with the packet sender. A traffic flow is called a **Forwarding Equivalence Class (FEC)**, also a Functional Equivalence Class, and this is a group of IP packets that are forwarded in the same manner (i.e., along the same path, with the same forwarding treatment). In other words, An FEC is a set of packet flows with common forwarding-path requirements across the MPLS domain.

A sequence of routers on a path along which a given FEC flow is forwarded forms a *tunnel*, which is known as a **Label Switched Path (LSP)**. For example, one such path is formed by routers *B*, *C*, and *D* in Figure 5-18. Each LSP tunnel is *unidirectional* (one-way), starting with an ingress LSR (*B*), going through intermediate LSRs (*C*), if any, and ending with an *egress LSR* (*D*), which is an exit router from the MPLS domain. If data needs to travel in the opposite direction as well, which is usually true, then a separate one-way tunnel must be built in the opposite direction (using a different FEC). To summarize, an FEC is uniquely associated with an LSP (but an LSP may be associated with several FECs). Each pair of adjacent routers agrees on a label independently of other router pairs along an LSP. Therefore, each label has *local scope* and different segments of an LSP tunnel may be represented by different (or same) MPLS labels. If there is a tunnel *B*→*C*→*D* and an opposite tunnel *D*→*C*→*B*, then tunnel segments *B*→*C* and *C*→*B* are different and use different labels. Why this is so, will be explained later.



Why MPLS:

- Uses switching instead of routing. MPLS switching is based on small, fixed length labels, that can directly index an array of output ports representing the switching table.

- Supports IP Traffic Engineering (TE): the ability to specify routes based on resource constraints, rather than on distance (shortest path) only (see Section 5.4.3). MPLS adds the ability to forward packets over arbitrary non-shortest paths, using constraint-based routing.

- Supports Virtual Private Networks (VPNs, see Section 5.4.4): Controllable tunneling mechanism emulates high-speed "tunnels" between IP-only domains.

- Route protection and restoration

A key reason for initial MPLS development was the promise of ultra-fast packet forwarding: Longest prefix match is (was) computationally expensive (Section 4.1.3); Label matching was seen as much less computationally expensive. However, with the emergence of gigabit IP routers capable of IP forwarding as fast as any MPLS-capable router performs label switching, the speed advantage has diminished (although not disappeared, because label switching still can be implemented in a much simpler hardware). Currently, MPLS key strengths are seen in Traffic Engineering and Virtual Private Networks—capabilities critical to network service providers who need to better manage resources around their backbones or need to offer VPN services.

Note that MPLS is a set of protocols (specified by IETF), rather than a single protocol. The key protocols are Label Distribution Protocol and Link Management Protocol (described in Section 5.4.2). Several existing IP-family protocols are also adapted to work with MPLS.

5.4.1 MPLS Architecture and Operation

MPLS is located between the link-layer and network-layer protocols (Figure 5-19), so it is referred to as a layer 2.5 protocol (in the OSI reference architecture, where Link is layer 2 and Network is layer 3; see Section 1.1.4). MPLS can run over different link-layer technologies, such as Ethernet or PPP (Section 1.5). The protocol-identifier field of the link-layer header should identify the payload as an MPLS frame. For example, unique PPP code points (carried in the Protocol field, Figure 1-63) identify the PPP frame's contents as an MPLS frame. The value of the PPP Protocol field for MPLS unicast is hexadecimal 0x0281. A similar encapsulation scheme is used when transmitting over Ethernet, where the payload is identified as an MPLS frame with unique Ether-Types (Figure 1-66(a)) or LLC frame's DSAP addresses (Figure 1-66(b)). The value of Ether-Type for MPLS unicast is hexadecimal 0x8847.

Forwarding Equivalence Classes (FECs) designate different classes of service or service priorities. Each MPLS-capable router (label switching router, or LSR) keeps a list of labels that

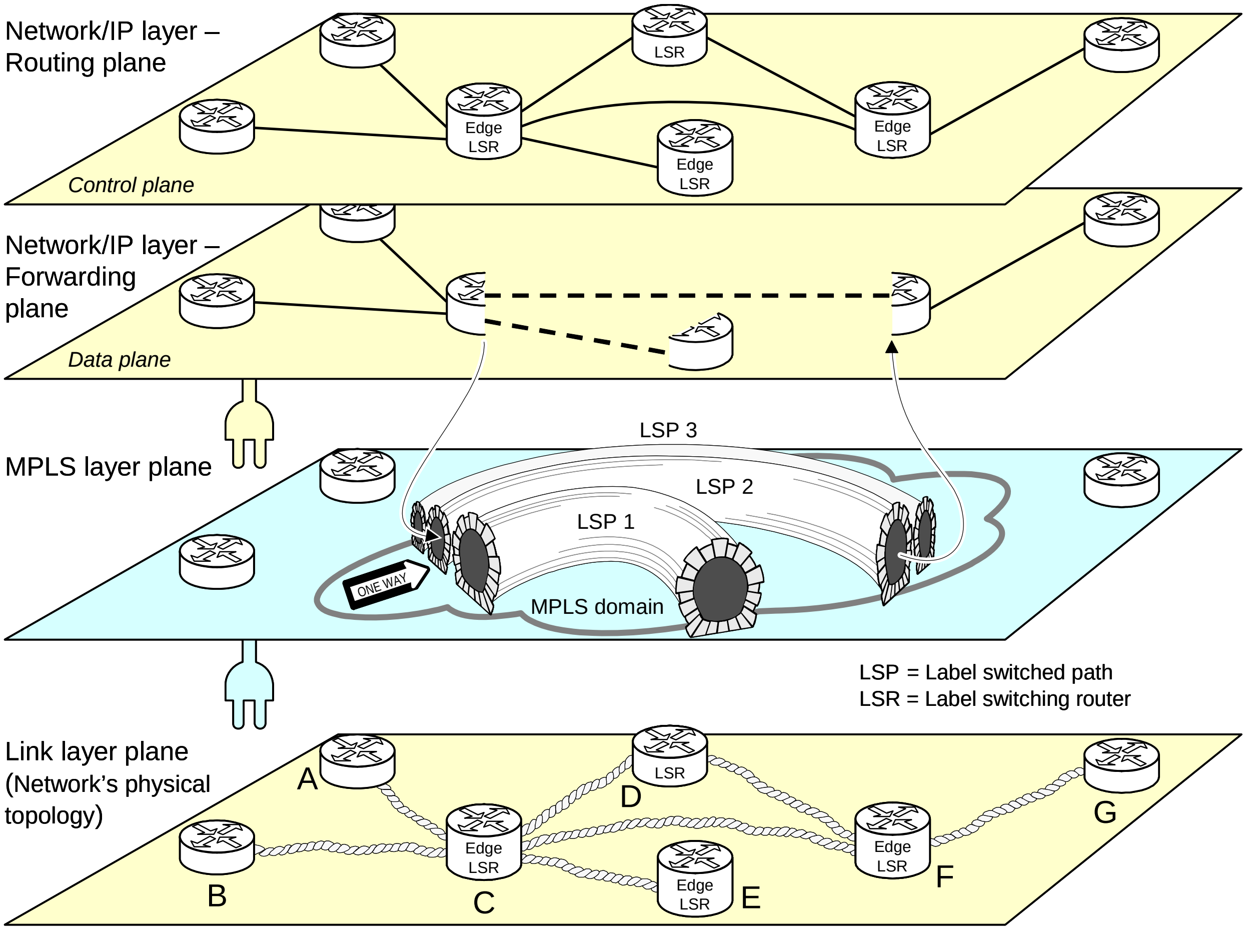


Figure 5-19: Protocol layering of an MPLS network.

correspond to different FECs on each outgoing link. All packets belonging to the same FEC have the same MPLS label value. However, not all packets that have the same label value belong to the same FEC. This fact will become clear later, as we see that FEC is determined by the label value and experimental bits (Exp) of the MPLS header. By default, packet's FEC is determined by its destination IP address. Other classification parameters include source IP address, IP protocol type, Differentiated Services field of the IP header (Figure 1-39), and TCP/UDP port numbers. The packet's arrival port may also be considered a classification parameter. Multicast packets belonging to a particular multicast group also form a separate FEC.

An edge-LSR terminates and/or originates LSPs (label switched paths) and performs both label-based forwarding and conventional IP forwarding functions. The edge-LSR converts IP packets into MPLS packets, and MPLS packets into IP packets. On ingress to an MPLS domain, an LSR accepts unlabelled IP packets and creates an initial MPLS frame by pushing a "shim" header between link-layer and network-layer headers (Figure 5-20). A special table in the ingress LSR, known as **Label Forwarding Information Base (LFIB)**, matches the FEC to the label. LFIB is an MPLS equivalent for the forwarding table of the IP protocol. On egress, the edge LSR terminates an LSP by popping the top MPLS stack entry from the packet, and forwarding the resulting packet based on rules indicated by the popped label (e.g., that the payload represents an IPv4 packet and should be processed according to IP forwarding rules).

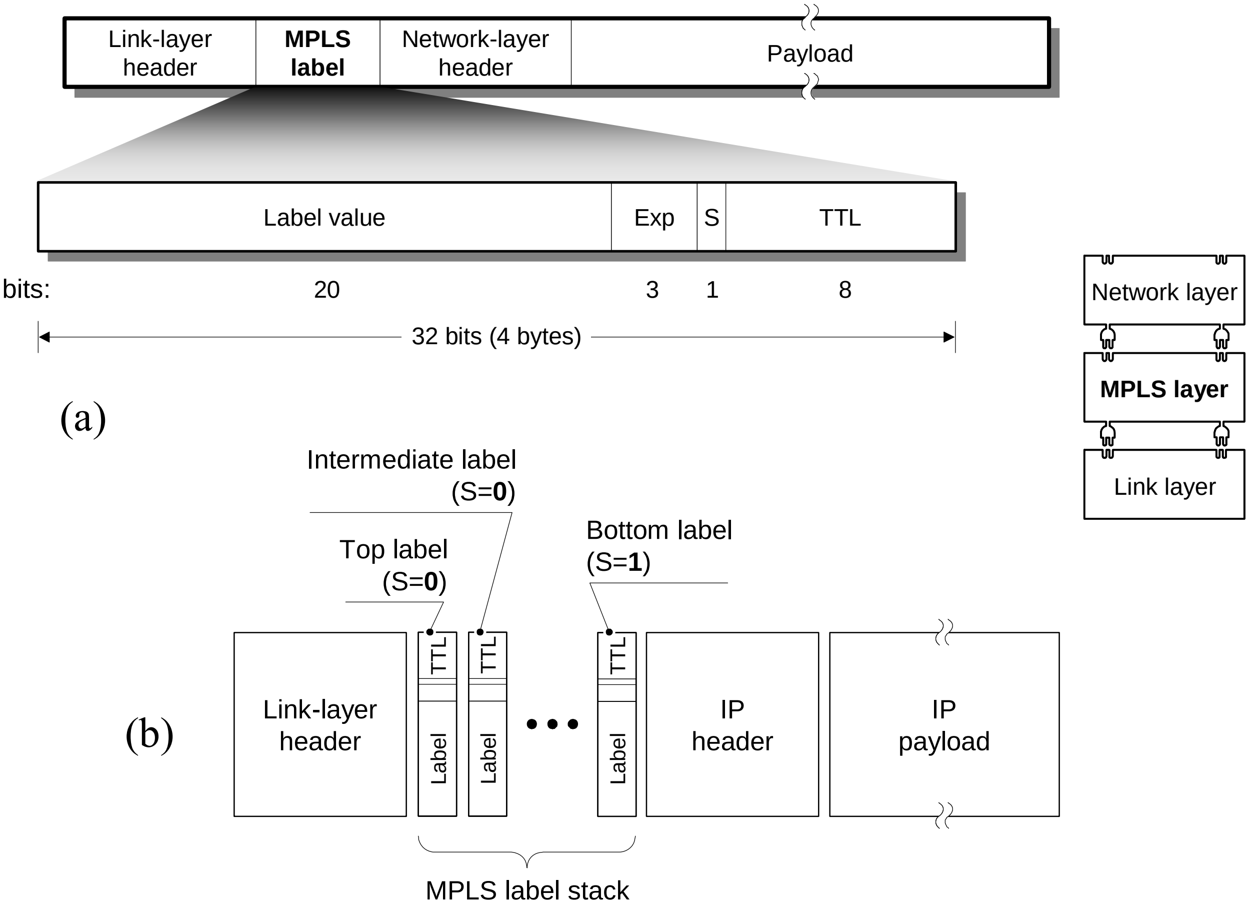


Figure 5-20: (a) MPLS label ("shim" header) format and its relative placement in IP packets. (b) The placement of the label stack; note that the S bit is set only for the bottommost label.

Edge LSRs provide the interface between external networks and the internal label-switched paths, and core/intermediate LSRs provide transit service in the middle of the network. An intermediate LSR examines incoming MPLS packets, looks up and follows the packet's label instructions, and then forwards the packet according to the instructions. In general, the LSR performs a label swapping function.

Paths or routes are established between the edge LSRs via intermediate LSRs. These paths are called **Label Switched Paths (LSPs)**. The LSPs (or, tunnels) are designed for their traffic characteristics. The traffic-handling capability of each path is calculated. These characteristics can include peak traffic load, inter-packet variation, and dropped packet percentage calculation.

MPLS Labels

Figure 5-20(a) shows the MPLS label for frame-based packets (such as Ethernet or PPP). MPLS label is also known as "shim" header. The meaning of the fields is as follows:

Label value: A number representing a forwarding equivalence class (FEC) on a given outgoing link. The label has a local scope limited to a network link connecting two adjacent LSRs and has

Visit http://en.wikipedia.org/wiki/Optical_switch for information about optical switches no meaning otherwise. Given 20 bits length, a link may support up to one million (2^{20} = 1,048,576) distinct labels.

Exp: experimental bits identify the class of service (or QoS). We will see in Section 5.4.5 how EXP bits are used to support Differentiated Services (DiffServ) in MPLS networks.

Bottom-of-stack bit (S): value "1" indicates that this label header is the bottom label in the stack of MPLS labels; otherwise, it is set to zero. The *stack* is a collection of labels that represent a hierarchy of nested tunnels created over a particular outgoing link. The stack can have unlimited depth, although it is rare to see a stack of four or more labels.

TTL: time-to-live counter that has the same function as the TTL found in the IP header (Figure 1-39), which is to prevent packets from being stuck in a routing loop. The TTL counter is decreased by 1 at each hop, and if the value reaches 0, the packet is discarded. Special processing rules are used to support IP TTL semantics, as described below.

The label value at each hop is a local key representing the next-hop and QoS requirements for packets belonging to each FEC. In conventional routing, a packet is assigned to an FEC at each hop (i.e., forwarding table look-up, Section 4.1.3). Conversely, in MPLS it is only done once at the ingress of the MPLS domain. At the ingress LSR, a packet is classified and assigned an FEC/label. Packet forwarding in the MPLS domain is performed by swapping the label.

The label stack entries appear after the link-layer header and before the network-layer header (Figure 5-20(b)). The label that was last pushed on the stack (newest) is called the *top label*, and it is closest to the link-layer header. The label that was first pushed on the stack (oldest) is called the *bottom label*, and it is closest to the network-layer header. The edge LSR that pops up the bottommost label is left with a regular IP packet, which it passes up to the network layer and the regular IP forwarding is used to move the packet onwards.

Label Bindings, LIB, and LFIB

The ordinary IP control plane builds and maintains the routing table, also known as RIB (Routing Information Base). Routing table is just built here, but forwarding decisions are made in the forwarding or data plane (top of Figure 5-19). Forwarding decisions are made based on the forwarding table, also known as FIB (Forwarding Information Base), which is derived from the routing table. In case of MPLS, the equivalent data structures are LIB (label information base) and LFIB (label forwarding information base). The prefixes-to-label bindings are built and stored in the LIB, control plane, which is then used to create the LFIB data or forwarding plane. The lookups are actually done in the LFIB, not the LIB (same as for IP, in the FIB and not the RIB). Their relationship is illustrated in Figure 5-21.

Table 5-1: Comparison of data structures maintained in IP routers versus MPLS LSRs.

IP Routers		**MPLS Switches (Label Switching Routers, or LSRs)**	
IP routing table (Routing Information Base = RIB)	IP forwarding table (or, Forwarding Information Base = FIB)	LIB (Label Information Base = LIB)	MPLS switching table (Label Forwarding Information Base = LFIB)
Built by the routing protocols, in the IP control plane	Derived from the routing table, in the IP data plane.	Built by a label distribution protocol, in the MPLS control plane.	Derived from the LIB in the MPLS data plane.
Contains shortest path routes from this to any other router		Contains only labels (i.e., LSP tunnels, no routes) from this LSR to others in the same MPLS domain.	Contains bindings between labels and LSPs.
Exchange route-advertising packets.	An IP packet is always looked up in forwarding table (not in routing table!)		

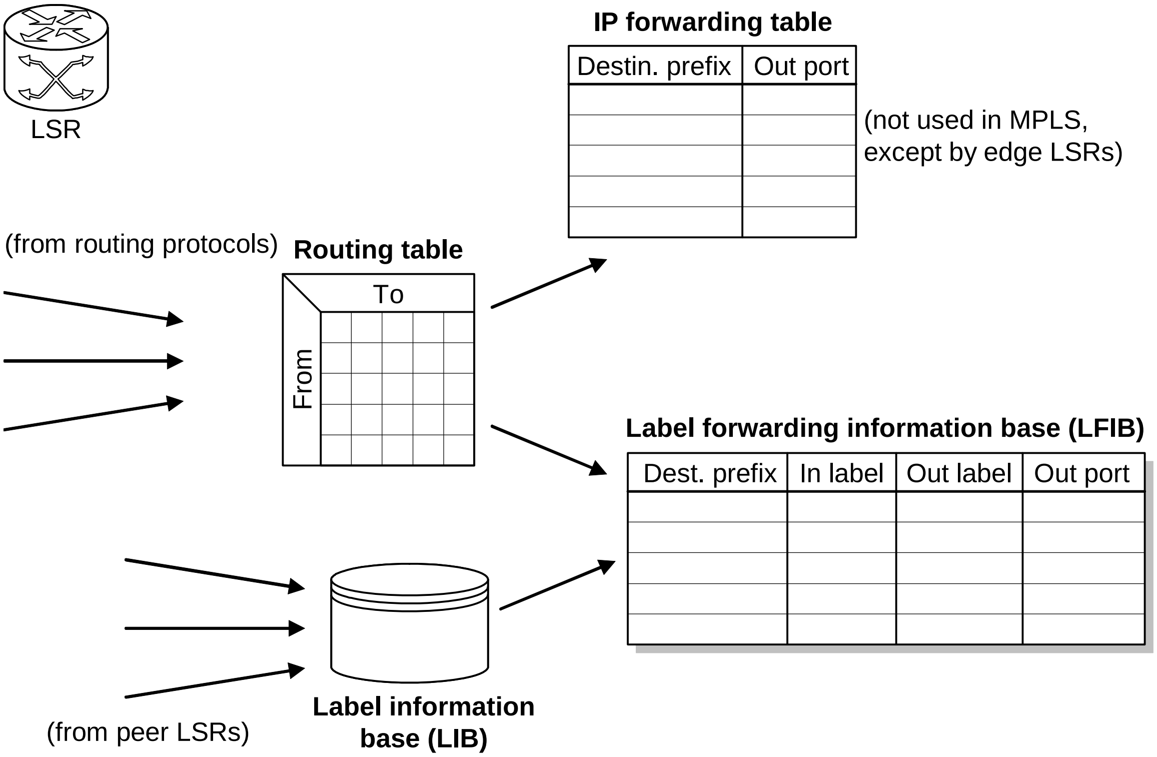


Figure 5-21: Relationship of LIB (label information base) and LFIB (label forwarding information base).

Label Forwarding Information Base (LFIB) is a data structure and way of managing forwarding in which destinations and incoming labels are associated with outgoing interfaces/ports and labels. The LFIB resides in the data plane and contains a local-label-to-next-hop label mapping along with the outgoing port, which is used to forward labeled packets.

Table 5-1 summarizes the data structures maintained in IP routers versus MPLS LSRs. However, the process is somewhat more complex for edge LSRs. On the ingress LSR, the lookup is performed against the combined IP forwarding table and LFIB, as described in the next section. In the core (intermediary LSRs), the lookup is performed only against the LFIB. On the egress LSR, the lookup is performed against the IP forwarding table if there was only a single label in the stack and this label was popped by the penultimate hop; otherwise, the LFIB is looked up.

As mentioned, the table to lookup into is determined by the link-layer header's Ether-Type or PPP Protocol field. The protocol identifier in the link-layer header tells the router what type of packet is coming in and therefore which table to look in:

- 0x0800 → IPv4: Lookup in the IP forwarding table
- 0x8847 → MPLS Unicast: Lookup in the MPLS LFIB (label forwarding information base)
- 0x8848 → MPLS Multicast: Lookup in the MPLS LFIB (label forwarding information base)

Next, we consider how MPLS routers (LSRs) build and utilize label-forwarding tables.

Forwarding Labeled Packets

Initially, all routers start with empty routing and forwarding tables and label-bindings. We assume that regular IP routing protocols run first and build regular IP routing tables, or RIBs (routing information bases). MPLS builds label-binding tables based on regular IP routing tables, using label distribution protocols (described in later Section 5.4.2). Label bindings can also be configured manually, particularly for the purposes of Traffic Engineering, but this is a tedious and error-prone task, so even here it is preferred to use label distribution protocols combined with constraint-based routing protocols. To illustrate how MPLS-capable routers (LSRs) forward labeled packets, let us assume the simplest scenario where label bindings are derived from the hop-by-hop information in IP routing tables.

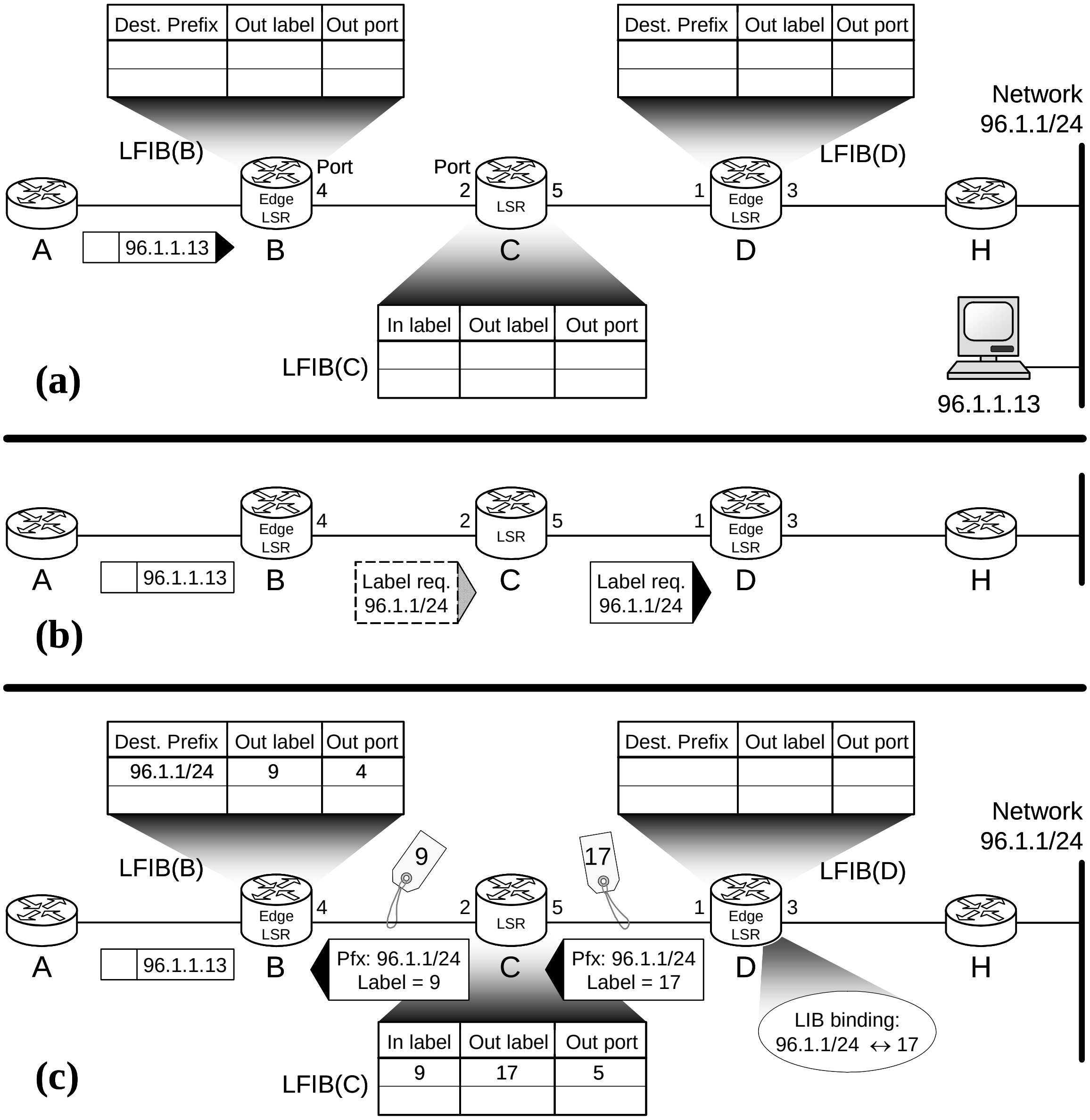
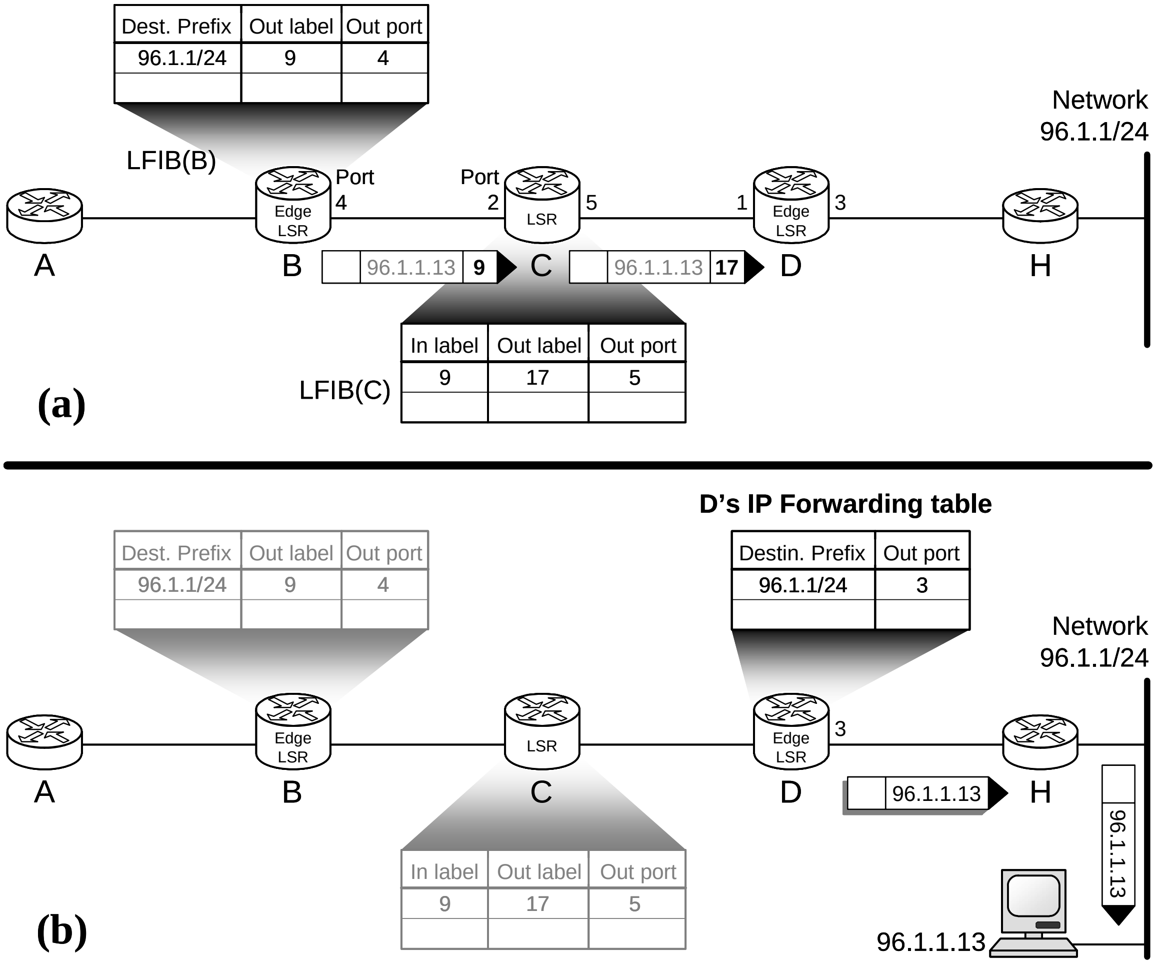


Figure 5-22: LSP path setup. (a) Packet arrives from router *A* to *B* towards a host in *H*'s network (note that all LFIB tables are empty). (b) LSR *B* sends label request towards the destination while the data packet is held waiting until the tunnel is set up. (c) LSR *D* is the edge router, so it replies with label 17, then LSR *C* selects its own label as 9 and replies to *B*.

Consider the example in Figure 5-22, which illustrates one of the tunnels (*B*→*C*→*D*) from Figure 5-18. Here, edge LSR *B* receives a packet with destination IP address `96.1.1.13`. LSR *B* has the corresponding network prefix `96.1.1/24` in its routing table, but does not have the label binding in its LFIB (label forwarding information base). To obtain a label for this prefix, *B* uses a label distribution protocol and sends a message downstream (to *C* and on to *D*) requesting their label for prefix `96.1.1/24`. When edge LSR *D* receives the request, it knows that itself is the egress LSR of the new tunnel (LSP, label switched path), because it knows that router *H* is the next hop on the path to `96.1.1/24` and that it is not label-switching capable (i.e., not an LSR).



LFIB(B):

Dest. Prefix	Out label	Out port
96.1.1/24	9	4

LFIB(C):

In label	Out label	Out port
9	17	5

D's IP Forwarding table

Destin. Prefix	Out port
96.1.1/24	3

Figure 5-23: Forwarding labeled packets (*after the tunnel is set up in* Figure 5-22). (a) Within MPLS domain, a data packet is forwarded based on its MPLS label. (b) Once the packet exits MPLS domain, it is forwarded based on its destination IP address.

Therefore, *D* selects a label value that it is not currently using for any other LSP and sends a response message back to *C* using the label distribution protocol. In our example (Figure 5-22(c)), *D* selected the label value 17 and stored the label 17 binding for prefix `96.1.1/24` in its LIB (label information base), *not* LFIB (label forwarding information base)! *D*'s LFIB remains empty because LSR *D* does not use LFIB to forward packets from this tunnel. *D* is the egress of the tunnel and to forward packets towards *H* (which is not an LSR and is not MPLS capable), *D* will use conventional IP forwarding.

When *C* receives the label response from *D*, in general it will need to assign a different label for the next segment of the same LSP tunnel, because it may be already using label 17 for another LSP. Remember, routers are at crossroads of many different paths and these paths are established at unpredictable times. In our example (Figure 5-22), *C* selects the label value 9 for the upstream segment of LSP. Because *C* is an intermediate LSR, it does not store prefixes in its LFIB; *C* might even not be IP-capable. Rather, *C* needs just the incoming and outgoing labels. The incoming label value is 9 (will be received in MPLS packets from *B*) and the outgoing label value is 17 (will be sent in MPLS packets to *D*). In other words, the intermediate LSR *C* performs *label swapping*. Unlike intermediate LSRs, edge LSRs do not perform label swapping. Each edge LSR

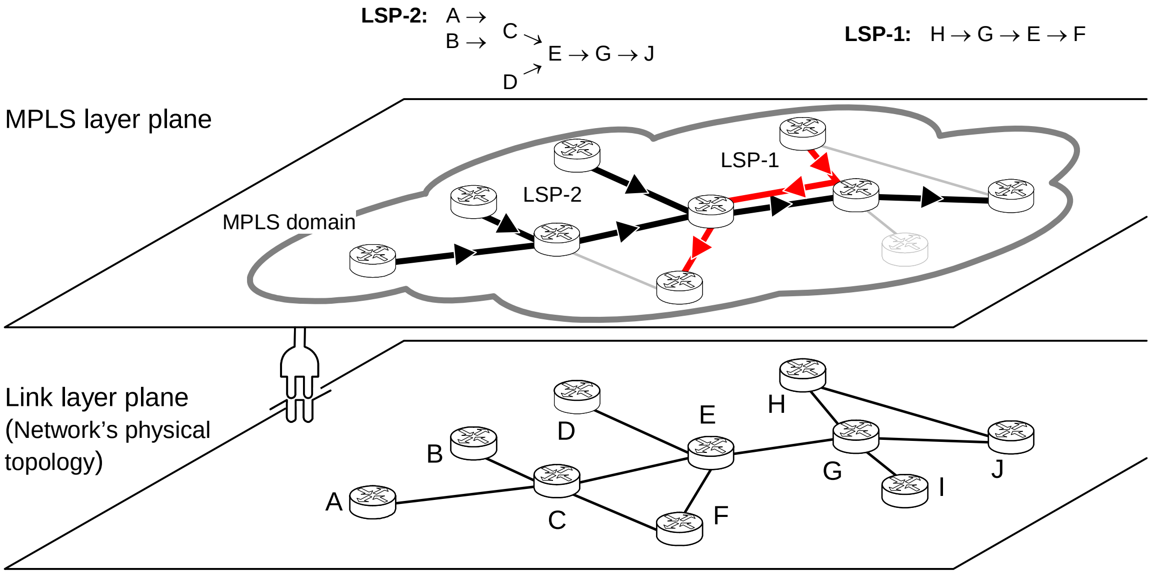


Figure 5-24: LSP topologies (or Point-of-Presence/PoP designs). LSP-1: Unique ingress and egress LSR. LSP-2: Multiple ingress LSRs, unique egress LSR.

must understand both IP and MPLS, and its LFIB (label forwarding information base) may have different format. Note also that some LSRs may play both roles: edge and intermediate, for different LSP tunnels.

Continuing with the example in Figure 5-22, the data packet has been sitting in LSR *B* while the LSP setup process took place. Once liable bindings become available, *B* will forward all data packets from this flow using label switching. An example for the first packet is shown in Figure 5-23. LSR *B*, as the ingress router of this LSP, inserts an MPLS label with value 9 (which it obtained from *C*), and sends this packet on output port 4 towards *C* (Figure 5-23(a)). When *C* receives the packet, it performs *label swapping*: For the given input label 9, *C* looks up its LFIB and finds that the outgoing label is 17 (which it received from *D*), swaps the packet's label to 17 and sends it on output port 5 towards *D*. When *D* receives the packet, it looks up the incoming label (17) and recognizes that itself is the egress LSR for this LSP. Therefore, *D* strips off the MPLS label and forwards the packet towards the next hop *H* using conventional IP forwarding (Figure 5-23(b)).

Topology of LSPs

The design of PoPs (Points-of-Presence) for all backbone IP networks, including MPLS networks, is constrained by the choice of access link type(s) to be supported for the customers of the network and the choice of core link type(s) to be used in the backbone network. Based on PoP designs, LSP (Label Switched Path) trees can be classified as these topology types (Figure 5-24):

- **Unique ingress and egress LSR:** In this case, a point-to-point path through the MPLS domain is set up. An example is LSP-1 in Figure 5-24, from the ingress LSR *H*, through the intermediate LSRs *G* and *E* towards the egress LSR *F*.

- **Multiple ingress LSRs, unique egress LSR:** In this case, LSP forms a multipoint-to-point tree topology. This happens when traffic assigned to a single FEC arises from different sources.

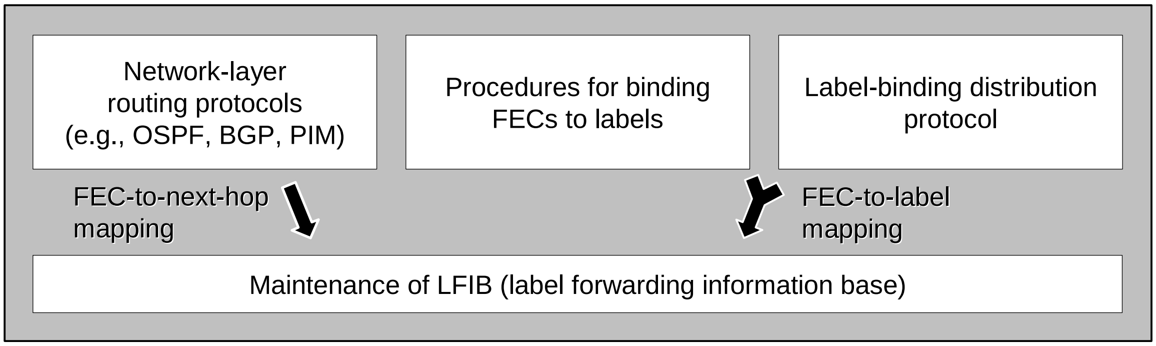

Network-layer routing protocols (e.g., OSPF, BGP, PIM)
Procedures for binding FECs to labels
Label-binding distribution protocol
FEC-to-next-hop mapping
FEC-to-label mapping
Maintenance of LFIB (label forwarding information base)

Figure 5-25: Components of the control plane of an LSR perform LFIB construction.

An example is LSP-2 in Figure 5-24, where traffic assigned to a single FEC enters at three different ingress LSRs: *A*, *B*, and *D*. The branches from *A* and *B* join at LSR *C*, then this branch joins with *D* at *E*, and the final two hops through *G* to the egress LSR *J* are shared.

- **Multicast:** In this case, multicast traffic is carried over the MPLS domain from a single ingress LSR to multiple egress LSRs. The multicast LSP is determined by the multicast tree constructed by the multicast routing protocol (Section 3.3.2).

In principle, an ISP backbone network could configure a separate LSP to carry each class of traffic (FEC) between each pair of edge LSRs. A more practical solution is to merge LSPs of the same traffic class to obtain multipoint-to-point flows that are rooted at an egress LSR. An example is LSP-2 in Figure 5-24. The LSRs serving each of these flows would be configured to provide the desired levels of performance to each traffic class.

5.4.2 Label Distribution Protocols

Setup of LSPs (Label Switched Paths) is done by a process of *label distribution*. Label distribution may be based on information obtained from conventional *hop-by-hop routing* protocols, or it may use *explicit routing* over non-shortest paths (Section 5.4.3). Label distribution protocol dynamically establishes an LSP tree between all the edge LSRs for each identifiable FEC. Requirements for a label distribution protocol include per-hop traffic differentiation capabilities, the ability to route traffic over non-shortest paths, and the ability to dynamically signal (or provision) QoS and path information across a network of routers or switches. There are many similarities between conventional routing protocols and label distribution protocols for MPLS. A key difference is the MPLS capability for explicit non-shortest-path routing.

The control plane of an LSR performs the following functions (Figure 5-25):

1. Create bindings between FECs and labels.
2. Inform the adjacent LSRs of the bindings it created (using a label distribution protocol).
3. Use information received from the adjacent LSRs to construct and maintain the forwarding table (LFIB) used by the MPLS label switching.

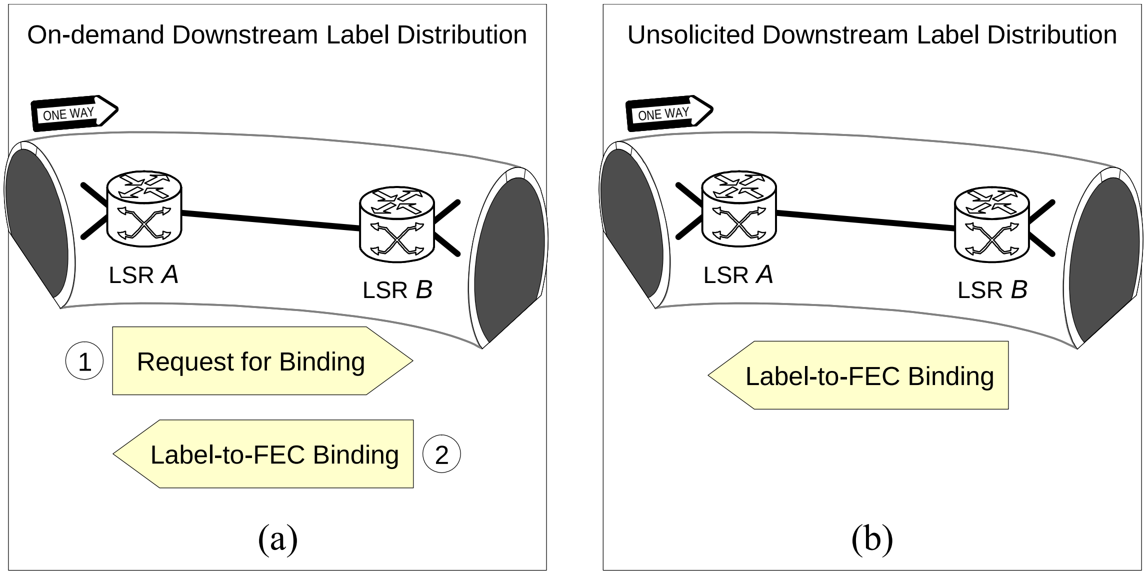


Figure 5-26: Methods of MPLS downstream label distribution: (a) on demand; (b) unsolicited.

Label Distribution

In general, label bindings between two LSRs can be distributed by either a downstream LSR or an upstream LSR. MPLS architecture requires *downstream label distribution*: label-bindings must be distributed in the direction from a downstream LSR to an upstream LSR. There are two methods for downstream label distribution (Figure 5-26):

- **On-demand Downstream Label Distribution:** In this case, a downstream LSR distributes a label binding in response to an explicit request from an upstream LSR (Figure 5-26(a)). An upstream LSR *A* recognizes a downstream LSR *B* as its next-hop for an FEC and sends a request to LSR *B* for a binding between the FEC and a label. If LSR *B* recognizes the FEC and has a next hop for it, LSR *B* creates a binding and replies to LSR *A*. Both LSRs then have a common understanding, represented by the shared MPLS label value. This process is also illustrated in the example in Figure 5-22.

- **Unsolicited Downstream Label Distribution:** In this case, a downstream LSR distributes a label binding in response to an explicit request from an upstream LSR (Figure 5-26(b)). A downstream LSR *B* discovers a "next hop" for a particular FEC, generates a label for this FEC, and communicates the binding to an upstream LSR *A*. LSR *A* inserts the binding into its LIB (label information base) and checks if it need to update the corresponding entry in its LFIB (label forwarding information base). If LSR *B* is the next hop for the FEC, LSR *A* can use that label as an outgoing label, knowing that its meaning is understood by *B*.

Each FEC is specified as a set of one or more FEC elements. Each FEC element identifies a set of packets that may be mapped to the corresponding LSP tunnel. When an LSP is shared by multiple FEC elements, the shared LSP is terminated at (or before) the node where the FEC elements can no longer share the same path. Currently defined types of FEC elements are:

1. *Address Prefix*. This element is an IP address prefix of any length from 0 to a full IP address, inclusive.
2. *Host Address*. This element is a full host IP address.

New element types may be added as needed.

Distribution Control

Independent LSP Control

Each LSR makes independent decision on when to generate labels and communicate them to upstream peers

Unsolicited: Communicate label-to-FEC binding to peers once next-hop has been recognized

LSP is formed as incoming and outgoing labels are spliced together

Characteristics:

Labels can be exchanged with less delay

Does not depend on availability of egress node

Granularity may not be consistent across the nodes at the start

May require separate loop detection/mitigation method

Ordered LSP Control

Label-to-FEC binding is communicated to peers if:

- LSR is the egress LSR to a particular FEC
- label binding has been received from an upstream LSR

LSP formation ‘flows’ from egress to ingress

Characteristics:

Requires more delay before packets can be forwarded along the LSP

Depends on availability of egress node

Mechanism for consistent granularity and freedom from loops

Used for explicit routing and multicast

Both methods of downstream label distribution (Figure 5-26) are supported in the MPLS standard and can be fully interoperable.

LDP: Label Distribution Protocol

Label Distribution Protocol (LDP) provides LSR discovery mechanisms to enable LSR peers to find each other and establish communication. The LDP protocol is defined in RFC-5036. It defines four types of messages:

- DISCOVERY: used to find neighboring LSRs. Each LSR announces and maintains its presence in a network. LSRs indicate their presence by sending Hello messages periodically. Hello messages are transmitted as UDP packets to the LDP port at the group multicast address for all routers on the subnet.

- SESSION ADJACENCY: used to initialize, keep alive, and shutdown LDP sessions. If two LSRs have discovered each other by means of the LDP Hello messages, they then establish sessions and become *LDP peers*. For session establishment, routers use LDP initialization procedure over TCP transport (unlike UDP-based discovery). After the initialization procedure is completed, the two routers are LDP peers and can exchange Advertisement messages (described next).

- LABEL ADVERTISEMENT: used for label-binding advertisements, request, withdrawal, and release. This is the main purpose of the LDP protocol. Advertisement messages are used to maintain label mappings for FECs (Figure 5-25). In general, an LSR requests a label mapping from an LDP peer when it needs one (Figure 5-26(a)), and advertises a label mapping to an LDP peer when it wants that peer to use the advertised label (Figure 5-26(b)).

- NOTIFICATION: used to distribute advisory information and to signal error information.

LDP depends on a routing protocol, such as OSPF (Section 9.2.2), to initially establish reachability between the LSRs. The LDP protocol runs over TCP to ensure reliable delivery of messages, except for discovery of LSR peers (Hello messages), which runs over UDP and IP multicast. It is designed to be extensible, using messages specified as **TLV**s (**type, value, length**) encoded objects.

The IP routing protocol can also be used to define the route for LSP tunnels (hop-by-hop routing). Alternatively, traffic-engineering considerations can determine the explicit route of the LSP (Section 5.4.3). Once a route is determined for an LSP, LDP is used to set up the LSP and assign the labels. Because each LSP is unidirectional, label assignment propagates back from the egress LSR to the originating point (ingress LSR), either on-demand or unsolicited (Figure 5-26).

RSVP-TE

RFC-3209

Resource Reservation Protocol with Traffic Engineering ("RSVP-TE") is a more complex protocol, with more overhead than LDP, but which also includes support for traffic-engineering via network resource reservations (Section 5.4.3).

Explicit Routing

The exchange of PATH and RESV messages between any two LSRs establishes a label association with specific forwarding requirements. The concatenation of these label associations creates the desired edge-to-edge LSP.

5.4.3 Traffic Engineering

Problems related to this section: Problem 5.21

Service providers and enterprise operators face the challenge of providing acceptable service levels, or QoS, to their customers and users while simultaneously running an efficient and reliable network. Conventional IP routing aims to find and follow the shortest path between a packet's current location and its destination. This can lead to "hot spots" in the network—routers and links that are on the intersection of shortest paths to many destinations are subject to high traffic load. As the average load on a router rises, packets experience increased loss rates, latency, and jitter. Two solutions exist (and may be deployed in parallel): upgrading the networking equipment (routers and links), or using the existing equipment more efficiently by distributing ("load balancing") the packet forwarding across alternate (potentially non-shortest-path) routes. The latter solution is called Traffic Engineering (TE).

Traffic Engineering (TE) refers to the process of setting up the paths for data traffic in order to facilitate efficient and reliable network operations while simultaneously optimizing network resource utilization and traffic performance. The goal of TE is to compute path from one given LSR to another such that the path does not violate any constraints (bandwidth and administrative requirements) and is optimal with respect to some scalar metric. Note that before MPLS TE can be deployed, the traffic load pattern of the network (i.e., bandwidth requirements) must be known.

Once the LSP path is computed, TE is responsible for establishing and maintaining forwarding state along such a path. The path bandwidth is removed from the "available bandwidth pool" and future LSPs may be denied if there is insufficient bandwidth. Existing LSPs may be "preempted" for new higher priority LSPs. When the required bandwidth is not available, no packets will be dropped but instead we are just giving high-priority traffic access to shorter paths. Lower priority traffic will be routed via some other path, with a higher latency.

Traffic Engineering may set up static *explicit* routes (i.e., tunnels) or *dynamic* routes (tunnels). In the explicit path option, the administrator specifies the IP addresses of the routers on the LSP path. In the dynamic path option, the path is dynamically calculated (the head router figures out the best path). Traffic engineering tunnels are calculated at the LSP head (ingress LSR in Figure 5-18) based on a fit between required and available resources. The fit is calculated by *Constraint-Based Routing* (CBR).

Constraint-based Routing and CSPF

One type of constraint would be the ability to find a route (path) that has certain performance characteristics, such as minimum available bandwidth. In this case, the constraint imposed on the routing algorithm is that the computed path must have at least the specified amount of available bandwidth on all links along the path. Different paths (defined by source-destination endpoints) may have different demands for the minimum available bandwidth.

Another type of constraint would be administrative. For example, a network administrator may want to exclude certain traffic from traversing certain links in the network, where such links would be identified by a link attribute. In this case, the constraint imposed on the routing algorithm is that the computed path must not traverse through any of the specified links. On the other hand, the network administrator may want to require certain traffic to traverse only the specified links. Similar to performance constraints, different paths may have different administrative constraints.

Constraint-based routing (CBR) cannot be supported by conventional IP routing protocols. The key reason is that constraint-based routing requires route (path) calculation at the source router.

This requirement is because different sources may have different constraints for a path to the same destination, and the constraints associated with a particular source router are known only to this router, but not to any other router in the network. Unlike this, in conventional IP routing, a route is computed in a distributed manner by every router in the network.

Constrained Shortest Path First (CSPF) is an enhanced version of the *link state routing* algorithm (Section 1.4.2), also known as *shortest-path first* (SPF). There are two major differences between regular SPF, done by routing protocols such as OSPF (Section 9.2.2), and CSPF, run by MPLS Traffic Engineering. First, the path computation is not designed to find the best route to all routers in the network but rather to the tunnel endpoint. When computing paths for LSP tunnels, instead of trying to calculate the shortest path to all nodes, CSPF algorithm stops as soon as the tunnel endpoint is in the Confirmed Set, also known as the PATH list (see Table 1-2 in Section 1.4.2). Second, there is now more than one metric for each link. CSPF computes paths by taking into account the constraints. In addition to a link *Cost*, CSPF also considers the link's *Bandwidth*, *Link attributes*, and *Administrative weight*. CSPF attempts to satisfy the requirements for a new LSP tunnel while minimizing congestion by balancing the network load.

An example, shown in Figure 5-27(a), is similar to Figure 1-44, but each link is characterized by its available bandwidth (in addition to its cost). Assume that Router *A* needs to build an LSP tunnel to Router *D* with a bandwidth of 90 Mbps. Recall from Section 1.4.2 that without considering the link bandwidths, Router *A*'s best path to Router *D* is $A \rightarrow C \rightarrow B \rightarrow D$, with the path length of 3. The computation is shown in Table 1-2, where this path is shown as (*D*, 3, *C*) in Step 6, with the notation as (*destination-node, path-length, next-hop*). In addition to this triplet used in regular SPF, CSPF needs to hold *minimum-path-bandwidth*, *link-attributes*, and *administrative-weight*. Therefore, instead of the triplet, the path properties are characterized by a sextuplet. For simplicity, our example uses only the available bandwidth and not link attributes or administrative weight, so a sextuplet is simplified to a quadruplet.

Table 5-2 shows the CSPF computation steps for building a tunnel from *A* to *D* in Figure 5-27(a). Recall that when a node is moved to the Confirmed set, its immediate neighbors are added to the Tentative set. A neighbor will *not* be added to the Tentative set the if it is already in the Tentative or Confirmed set with a lower cost or if it does not meet all configured constraints for the desired tunnel, such as the required bandwidth. If the node just added to the Tentative set is already in this set, but with a higher cost or lower minimum bandwidth, we replace the inferior path with this new better path. CSPF stops if the Tentative set is empty or, unlike regular SPF, if the tunnel endpoint *D* is in the Confirmed set. The fact that in our example the Tentative set happens to finish empty is an artifact of our example network topology: *D* happened to be the last node we encountered in the CSPF computation. The CSPF algorithm would have stopped even if it had reached the best path to *D* and there were still more nodes left in the Tentative set.

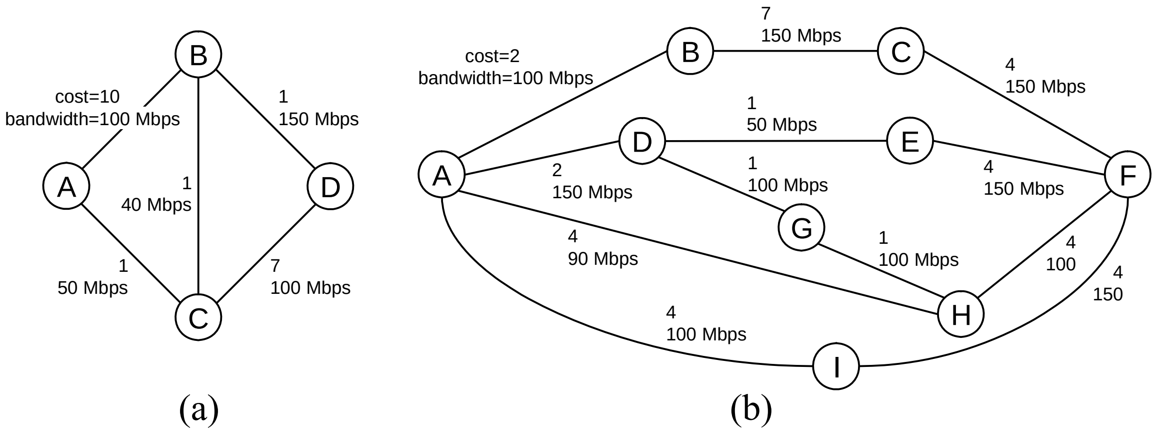


Figure 5-27: Example MPLS domains running the CSPF algorithm. See text for details.

Actual path calculation in CSPF is more complex than this example implies. CSPF has to keep track of all the nodes in the path, not just the next hop. In addition, it must consider not only the minimum path bandwidth but also link attributes and administrative weights.

Given that CSPF searches for *one path to one destination*, rather than all the shortest paths to all possible destinations, it needs a strategy to resolve the ties. **Tiebreakers in CSPF** differentiate the paths with the same cost from one another. The tiebreakers in CSPF are as follows, in order:

1. Choose the path with the largest minimum available bandwidth
2. If there is still a tie, choose the path with the lowest hop count (the number of routers in the path)
3. If there is still a tie, choose at random one of the tied paths

With these tiebreakers, the path sextuplet (simplified as quadruplet in Table 5-2) becomes a nonet (nine items in the object), with the three additional items: minimum path bandwidth, the lowest IP

Table 5-2: Steps for building a routing table using CSPF at node *A* in Figure 5-27(a). Each node is represented with a quadruplet (*Destination node ID, Path length, Next hop, Minimum bandwidth*). Compare to Table 1-2.

Step	Confirmed path set	Tentative set	Comments
0	(*A*, 0, –, N/A)	∅	Initially, *A* is the only member of Confirmed(*A*), with a distance of 0, a next hop of self, and the bandwidth set to Not/Available.
1	(*A*, 0, –, N/A)	(*B*, 10, *B*, 100)	Put *A*'s neighbors in the Tentative(*A*) set. Note that (*C*, 1, *C*, 50) was not added to Tentative because it does not meet the minimum bandwidth requirement.
2	(*A*, 0, –, N/A), (*B*, 10, *B*, 100)	(*D*, 11, *B*, 100)	Move *B* to the Confirmed set, and put *B*'s neighbors in the Tentative set. Again, (*C*, 11, *B*, 40) was not added to Tentative because it does not meet the minimum bandwidth requirement.
3	(*A*, 0, –, N/A), (*B*, 10, *B*, 100), **(*D*, 11, *B*, 100)**	∅	Move *D* to the Confirmed set. At this point, the tunnel endpoint (*D*) is in Confirmed, so we are done. END.

routing metric to a path, and the path's lowest hop count.

Consider the example in Figure 5-27(b), and assume that the administrator wishes to build a tunnel from router *A* to router *F* with a bandwidth of 80 Mbps. The following table lists all the possible paths from router *A* to router *F* and their attributes:

Path name	**Routers in path**	**Path length**	**Minimum bandwidth on path (in Mbps)**
P1	A→B→C→F	2+7+4= 13	Min{100, 150, 150} = 100
P2	A→D→E→F	2+1+4= 7	Min{150, 50, 150} = 50
P3	A→D→G→H→F	2+1+1+4= 8	Min{150, 100, 100, 100} = 100
P4	A→H→F	4+4= 8	Min{90, 100} = 90
P5	A→H→G→D→E→F	4+1+1+1+4= 11	Min{150, 100, 100, 50, 150} = 50
P6	A→I→F	4+4= 8	Min{100, 150} = 100

Here is the decision process that illustrates the application of tiebreakers to select the best path:

- Eliminate the paths with minimum bandwidth <80 Mbps (~~P2 and P5~~) because their minimum bandwidth is lower than the minimum required bandwidth of 80 Mbps;
  Remaining paths: P1, P3, P4, and P6
- Select the path(s) with the shortest length and eliminate all greater length paths (~~P1~~);
  Remaining paths: P3, P4, and P6 — all have the length equal to 8
- Path ~~P4~~ is not used because its minimum bandwidth is lower than the minimum bandwidths of paths P3 and P6
- Path ~~P3~~ is not used because it has a hop count of 5, and the other remaining path (P6) has a hop count of 3

Finally, LSR *A* selects path P6 to build a tunnel to LSR *F*. The actual CSPF computation must follow the steps in Table 5-2 and apply the tiebreakers where needed. Note that the bandwidth and other LSP attributes configured on the TE tunnel allow the setup of an LSP only if LSP path options satisfy the constraints. If a path cannot be found that satisfies the configured path options, then the tunnel is not set up. (Routers usually allow the administrator to set the path bandwidth to zero effectively removing the bandwidth constraint imposed by the constraint-based routing calculation.)

Link Coloring (Affinities / Admin-Groups) is an additional constraint in the CSPF algorithm. Link Coloring allows for 32 unique "color" markings that can be placed on a link. Multiple color markings can be applied on a link. Link colors are advertised as a link attribute, periodically flooded out just like Bandwidth information. The operator can use these markings in any way they wish. Link Coloring is useful for specifying:

• Geographic / Political boundaries—prefer to keep traffic routing within a specific country/region/continent

• Cost-Out/Maintenance Activities—can instruct all LSPs to immediately move off a path

• Prevent traffic from traversing specific links/paths—not have "core-to-core" LSPs traverse edge routers or metro networks.

Establishing and Maintaining Traffic Engineering Tunnels

As already mentioned, traffic engineering tunnels are calculated at the LSP head router. The computation of explicit routes (known as **Explicit Route Objects** or **ERO**s) is done using a

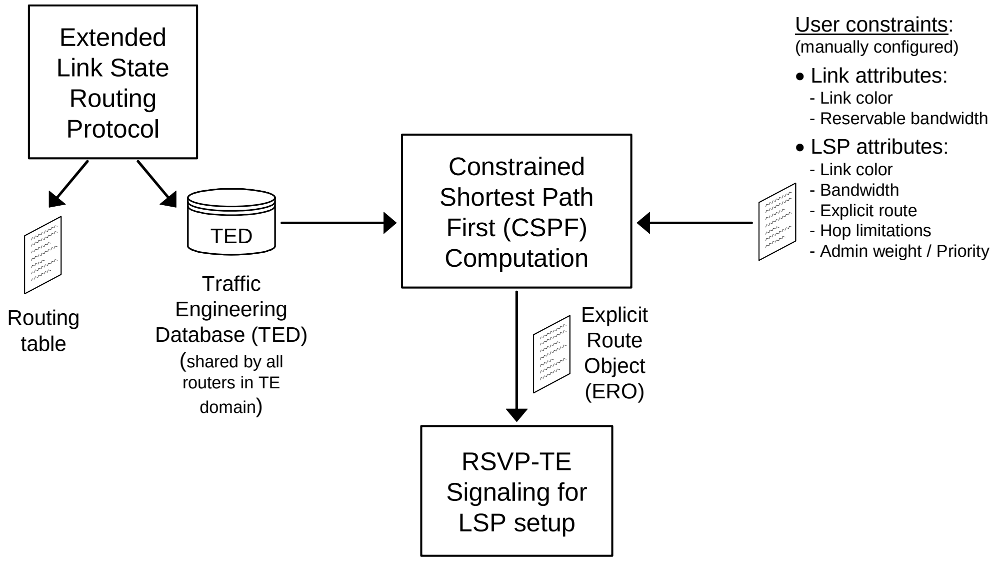

Extended Link State Routing Protocol
Routing table
TED
Traffic Engineering Database (TED) (shared by all routers in TE domain)
Constrained Shortest Path First (CSPF) Computation
User constraints:
(manually configured)
• Link attributes:
- Link color
- Reservable bandwidth
• LSP attributes:
- Link color
- Bandwidth
- Explicit route
- Hop limitations
- Admin weight / Priority
Explicit Route Object (ERO)
RSVP-TE Signaling for LSP setup

Figure 5-28: Traffic Engineering computing explicit routes at the LSP head-end router.

distributed **Traffic Engineering Database** (**TED**), as shown in Figure 5-28. The TED database contains all IP addresses, links and current bandwidth reservation states. The head-end router will begin the process of signaling the Constrained Shortest Path. LSP path establishment is based on RSVP-TE messages (Section 5.4.2).

Tunnel reoptimization occurs when a device with traffic engineering tunnels periodically examines tunnels with established LSPs to learn if better LSPs are available. If a better LSP seems to be available, the device attempts signaling to build the better LSP. If the signaling is successful, the device replaces the older LSP with the new, better LSP.

An important issue is how to determine the required bandwidth of a particular LSP. After all, IP networks are dynamic and packet switched, so bandwidth use can change in an instant and be unpredictable. There are two approaches:

• *Offline calculation*, which occurs outside of the router, typically based on some traffic modeling, and often using a third party script or tool. This is how MPLS was first implemented, and is still commonly used today by most large networks and early MPLS adopters. However, most offline calculations are based expectations of stable traffic patterns. Unusual traffic spikes can cause congestion or inefficient bandwidth use.

• *Automatic bandwidth adjustment* feature of MPLS TE tunnels allows network providers to set up tunnels with arbitrary initial bandwidth values, and then dynamically and automatically adjust the bandwidth based on traffic patterns without traffic disruption. TE autobandwidth adjustment runs directly on the router and it can dynamically respond to changing traffic conditions much more rapidly, with less overhead than offline calculation. If autobandwidth adjustment is configured for a tunnel, traffic engineering automatically adjusts the bandwidth allocation for the TE tunnel based on its measured usage of the bandwidth of the tunnel. TE autobandwidth periodically samples the average traffic output rate and adjusts the allocated bandwidth for the tunnel to be the largest sample for the tunnel since the last adjustment. The frequency with which tunnel bandwidth is adjusted and the allowable range of adjustments is configurable on a per-tunnel basis. In addition, the sampling interval and the interval over which to average tunnel traffic to obtain the average output rate is user-configurable on a per-tunnel basis.

5.4.4 MPLS Virtual Private Networks

Virtual private networks (VPNs) provide relative or absolute protection for a given traffic flow from other traffic on any particular network segment. VPNs are also used to support tiered services for traffic flows. In general, a VPN provides wide area connectivity to an organization located in multiple sites. MPLS can provide connectivity among VPN sites through LSPs that are dedicated to the given VPN. The LSPs can be used to exchange routing information between the various VPN sites, transparently to other users of the MPLS network. This behavior gives the appearance of a dedicated wide-area network.

We already classified MPLS routers as "edge LSRs" and "core LSRs." These roles are alternatively expressed in VPN terminology as "PE," which stands for "Provider Edge Router," and "P," which stands for "Provider Router." A *Provider Router* "P" is a core (or backbone) LSR that is doing label switching only. A pure P-router can operate without any customer or Internet routes at all. This mode of operation is common in large service provider networks.

A *Provider Edge Router* "PE" is a customer-facing router on the border of an MPLS domain, which imposes MPLS labels on or removes from IP packets. A PE-router typically has various edge features for terminating multiple services: regular IP-based Internet on one end and virtual private networks (VPNs) on the other end. Additionally, we define a *Customer Edge Router* "CE" as the customer device a PE-router talks to.

OSI Layer-2 VPNs, OSI Layer-3 VPNs

OSI Layer-2 VPNs implement virtual Layer-2 links. For example, **VPLS (Virtual Private LAN Service)** creates an Ethernet multipoint switching service over MPLS. It is used to link a large number of customer endpoints in a common broadcast domain. It avoids the need to provision a full mesh of Layer-2 links. A VPLS emulates the basic functions of a Layer-2 switch, such as unknown unicast flooding; MAC address learning; and broadcasts. VPLS is typically load-balancing friendly since the Ethernet headers are examined and used, unlike Layer-2 pseudowires where they are passed transparently.

It is possible to build VPNs using a pure IP solution. Although gigabit IP routers are capable of IP forwarding as fast as any MPLS-capable router performs label switching, MPLS VPNs are significantly more efficient than IP VPNs.

Networks build virtual routing domains (VRFs) on their edge routers. Customers are placed within a VRF, and exchange routes with the provider router in a protected routing-instance, usually BGP or IGP. Layer-3 VPNs are signaled via BGP within the provider network.

Layer-3 VPNs can support complex topologies and interconnect many sites. Usually load-balancing is hash friendly (has exposed IP headers). However, Layer-3 VPNs can add a significant load to the service provider infrastructure because the PE-router must absorb the customer's routing table, consuming RIB and FIB capacity. Layer-3 VPNs are typically seen in mostly enterprise environments.

5.4.5 MPLS and Quality of Service

Route Protection and Restoration

A **fast-reroute** feature enables an MPLS Traffic Engineering tunnel to use a *backup tunnel* in the event of a link failure, if a backup tunnel exists. Fast Reroute achieves this capability by pre-calculating backup paths for potential link or node failures.

In a normal IP network, the best path calculation happens on-demand when a link failure is detected. It can take several seconds to recalculate the best paths and push those changes to the router hardware, particularly on a busy router. As we know from Section 1.4, transient routing loops may also occur, as every router in the network learns about the topology change.

With MPLS Fast Reroute, the next best path calculation happens *before* the failure actually occurs. The backup paths are pre-programmed into the router FIB awaiting activation, which can happen in milliseconds following failure detection. Because the entire path is set within the LSP, routing loops cannot occur during convergence, even if the path is briefly suboptimal.

MPLS Fast Reroute involves end-to-end protection, and fast node- and link reroute. There are different types of failures that can be protected against:

• Link Protection / Next-Hop Backup, where a bypass LSP is created for every possible link failure.

• Node Protection / Next-Next-Hop Backup, where a bypass LSP is created for every possible node (router) failure

There are two approaches to provide LSP protection:

(1) *One-to-One Protection / Detour*: Backup LSP established in advance, resources dedicated, data simultaneously sent on both primary and backup. Switchover performed only by egress LSR: An individual backup path is fully signaled through RSVP for every LSP, at every point where protection is provided (i.e., every LSR node). The label depth remains at one, but this can involve a huge number of reservations, and can cause significant overhead. This approach is fastest, but also most resource intensive.

(2) 1:1 : Same as 1+1 with the difference that data is not sent on the backup

Requires failure notification to the ingress LSR to start transmitting on backup

Notification may be send to egress also

Resources in the backup may be used by other traffic

Low priority traffic (e.g., plain IP traffic), shared by other backup paths.

Integrating MPLS and QoS using DiffServ

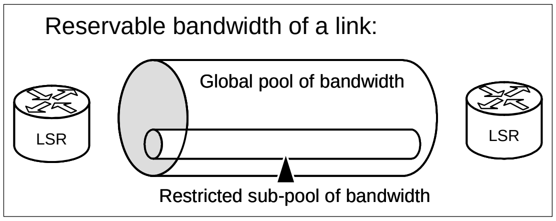
Traffic engineering (Section 5.4.3) does not differentiate among traffic types. To carry voice and data traffic on the same network, it may be necessary to account separately for the amount of voice traffic being transferred over the network, to provide the necessarily stricter QoS guarantees.

There are two architectures for supporting QoS in today's networks: Integrated Services (IntServ) and Differentiated Services (DiffServ), see Sections 3.3.5 and 3.3.4. Most attention in integrating QoS into MPLS has been on using the DiffServ approach. In DiffServ model, packets entering a DiffServ-enabled network are grouped into a small number of classes. Each class has a color or mark associated with it (using the DiffServ Code Point DSCP bits in IP header, Figure 1-39).

In a MPLS+DiffServ Traffic Engineering architecture (MPLS/DS-TE), packets marked with DiffServ Code Point will enter the MPLS network and per hop behavior is enforced by every LSR along the path. Because LSRs do not have any knowledge of the IP header, per hop behavior needs to be achieved by looking at different information.

Two general approaches are used to mark MPLS traffic for QoS handling within an MPLS network. In the first method, the DiffServ coloring information is mapped in the EXP (experimental) field of the MPLS shim header (Figure 5-20). This field allows up to eight different QoS marking versus 64 for DSCP. The packet scheduling (PHB) at each hop (in the MPLS domain) is done based on the EXP. Label Switched Paths that use this approach are called E-LSPs, where QoS information is inferred from the EXP bits.

Alternatively, the label associated with each MPLS packets carries the portion of the DiffServ marking that specifies how a packet should be queued. The dropping precedence portion of the DiffServ marking is carried in the EXP bits. The ingress LSR examines the DSCP in the IP header and selects an LSP that has been provisioned for that QoS level. At the egress as the label is removed, the packet is sent to the next IP hop with its original DSCP. LSPs using this approach are called L-LSP where QoS information is inferred in part from the MPLS label.


Reservable bandwidth of a link:
LSR
Global pool of bandwidth
LSR
Restricted sub-pool of bandwidth

Diff-Serv-aware Traffic Engineering extends MPLS TE to enable constraint-based routing of "guaranteed" traffic, which satisfies a more restrictive bandwidth constraint than that satisfied by CBR for regular traffic.. Specifically, the bandwidth reservable on each link for constraint-based routing purposes can now be managed through two bandwidth pools: a *global pool* and a *sub-pool*. A (restrictive) sub-pool dedicates a portion of the link bandwidth to the high-priority traffics. (The sub-pool is a portion of the global pool.) This ability to satisfy a more restrictive bandwidth constraint translates into the capability to achieve higher QoS (in terms of delay, jitter or loss) for the traffic using the sub-pool.

The concept of "Class-Type" defined by IETF corresponds to the concept of bandwidth pool implemented by MPLS/DS-TE. And because DS-TE supports two bandwidth pools (global pool and sub-pool), DS-TE should be seen as supporting two Class-Types (Class-Type 0 and Class-Type 1). Each LSR uses one pool—the sub-pool—for tunnels that carry traffic requiring strict bandwidth guarantees or delay guarantees, use the other pool—the global pool—for tunnels that carry traffic requiring only Differentiated service/Best effort. When a new LSP tunnel is created, it is by default a "global pool" tunnel, unless explicitly designated as a "sub-pool" tunnel. Within the MPLS core, every LSR along the LSP path must assure that the traffic sent in the "sub-pool" LSP is placed in a "high-priority/low latency" queue at the outbound interface of each LSR.

5.5 Summary and Bibliographical Notes

Section 5.1: Queue Scheduling

If a server (router, switch, etc.) is handling multiple flows, there is a danger that aggressive flows will grab too much of its capacity and starve all the other flows. Simple processing of packets in the order of their arrival is not appropriate in such cases, if it is desired to provide equitable access to transmission bandwidth. Scheduling algorithms have been devised to address such issues. The best known is *fair queuing* (FQ) algorithm, originally proposed by [Nagle, 1987], which has many known variations. A simple approach is to form separate waiting lines (queues) for different flows and have the server scan the queues *round robin*, taking the first packet (head-of-the-line) from each queue (unless a queue is empty). In this way, with *n* hosts competing for a given transmission line, each host gets to send one out of every *n* packets. Aggressive behavior does not pay off, because sending more packets will not improve this fraction.

A problem with the simple round robin is that it gives more bandwidth to hosts that send large packets at the expense of the hosts that send small packets. *Packet-by-packet FQ* tackles this problem by transmitting packets from different flows so that the packet completion times approximate those of a bit-by-bit fair queuing system. Every time a packet arrives, its completion time under bit-by-bit FQ is computed as its finish number. The next packet to be transmitted is the one with the smallest finish number among all the packets waiting in the queues.

If it is desirable to assign different importance to different flows, e.g., to ensure that voice packets receive priority treatment, then *packet-by-packet weighted fair queuing* (WFQ) is used. WFQ plays a central role in QoS architectures and it is implemented in today's router products [Cisco, 1999; Cisco, 2006]. Organizations that manage their own intranets can employ WFQ-capable routers to provide QoS to their internal flows.

[Keshav, 1997] provides a comprehensive review of scheduling disciplines in data networks.

[Bhatti & Crowcroft, 2000] has a brief review of various packet scheduling algorithms

[Elhanany *et al*., 2001] reviews hardware techniques of packet forwarding through a router or switch

Packet scheduling disciplines are also discussed in [Cisco, 1995]

One of the earliest publications mentioning the leaky bucket algorithm is [Turner, 1986].

Section 5.3: Active Queue Management

Random Early Detection (RED) keeps the overall throughput high while maintaining a small average queue length, and tolerates transient congestion. When the average queue has exceeded a certain threshold, RED routers drop packets at random so that TCP connections back off at different times. This avoids the global synchronization effect of all connections. RED was proposed by Floyd and Jacobson [1993]. Sally Floyd maintains a list of papers on RED here: http://www.icir.org/floyd/red.html. Christiansen, *et al.*, [2001] also provides an overview of various versions of RED and additional references. Srikant [2004] presents an in-depth account on RED techniques and their analysis.

Clark and Fang [1998] proposed an extension of RED to provide different levels of drop precedence for two classes of traffic. Their algorithm is called RED with IN/OUT or RIO for short. A device, located on the sourcing traffic side of a network boundary, serves a "policy meter." Packets are classified as being inside (IN) or outside (OUT), depending on whether they conform to the service allocation profile of a given sender/user. RIO uses twin RED algorithms for dropping packets, one for INs and one for OUTs. RIO chooses different parameters of RED algorithms for IN and OUT packets, which may be lined up in the same or different queues. When congestion sets in, RIO is able to preferentially drop OUT packets.

Explicit Congestion Notification (ECN) is described in RFC-3168. As expected, ECN reduces the number of packets dropped by a TCP connection, which, in turn, reduces latency and especially jitter, because packet retransmissions are avoided [RFC-2884]. This outcome is most dramatic when the TCP connection sends occasional isolated segments, which is common for interactive connections (such as remote logins) and transactional protocols (such as HTTP requests, the conversational phase of SMTP, or SQL requests. Such a sender will receive ECN notification, which it ignores because it sends only occasional isolated segments, but it benefits from the fact that its segment was not dropped. The reason for this effect is that the sender can detect a loss of an isolated segment only by an RTO timeout (which is relatively long), because there are no subsequent segments to generate duplicate ACKs. Effects of ECN on bulk transports are less clear because subsequent segments will soon generate duplicate ACKs and recent TCP versions use fast recovery to resend dropped segments in a timely manner (Section 2.2).

Section 5.4: Multiprotocol Label Switching (MPLS)

Multiprotocol Label Switching (MPLS) provides the ability to forward packets over arbitrary non-shortest paths, and emulate high-speed "tunnels" between IP-only (non-label-switched) domains. When packets enter the MPLS domain, labels are imposed on the packets, and the label (not the IP header) determines the next hop. Labels are removed at the egress of the MPLS domain.

MPLS offers a capability not available to conventionally routed solutions: the forwarding packets over arbitrary, non-shortest paths, which is particularly useful for managing network resources, known as "traffic engineering." MPLS technology is key to scalable virtual private networks (VPNs) and end-to-end quality of service (QoS), enabling efficient utilization of existing networks to meet future growth and rapid fault correction of link and node failure.

Label Distribution Protocol (LDP) is defined in RFC-3036 and is used to provide mechanisms for MPLS routers to process and route labeled traffic across an MPLS network.

Davie and Rekhter [2000] offer a very readable account of MPLS fundamentals, which, although dated, is still relevant to study because it explains well the basic concepts. A relatively recent and comprehensive review of MPLS is available in [De Ghein, 2007].

Traffic engineering (TE) is essential for service provider and Internet service provider (ISP) backbones. Such backbones must support a high use of transmission capacity, and the networks must be very resilient, so that they can withstand link or node failures. MPLS traffic engineering provides an integrated approach to traffic engineering. With MPLS, traffic engineering capabilities are integrated into OSI Layer 3, which optimizes the routing of IP traffic, given the constraints imposed by backbone capacity and topology. MPLS TE employs constraint-based routing, in which the path for a traffic flow is the shortest path that meets the resource requirements (constraints) of the traffic flow. In MPLS TE, the traffic flow has bandwidth requirements, media requirements, a priority versus other flows, and so on.

The Constrained Shortest Path First (CSPF) is analyzed in [Ziegelmann, 2007]. A more practical description of CSPF with applications to traffic engineering is available in [Osborne & Simha, 2002].

Virtual path capability and the capacity to engineer precise traffic in a core network have driven MPLS towards becoming a standard within service provider core networks.

Problems

Problem 5.1

Problem 5.2

Eight hosts, labeled *A*, *B*, *C*, *D*, *E*, *F*, *G*, and *H*, share a transmission link the capacity of which is 85. Their respective bandwidth demands are 14, 7, 3, 3, 25, 8, 10, and 18, and their weights are 3, 4, 1, 0.4, 5, 0.6, 2, and 1. Calculate the max-min weighted fair share allocation for these hosts. Show your work neatly, step by step.

Problem 5.3

Problem 5.4

Consider a packet-by-packet FQ scheduler that discerns three different classes of packets (forms three queues). Suppose a 1-Kbyte packet of class 2 arrives upon the following situation. The current round number equals 85000. There is a packet of class 3 currently in service and its finish number is 106496. There are also two packets of class 1 waiting in queue 1 and their finish numbers are $F_{1,1}$ = 98304 and $F_{1,2}$ = 114688.

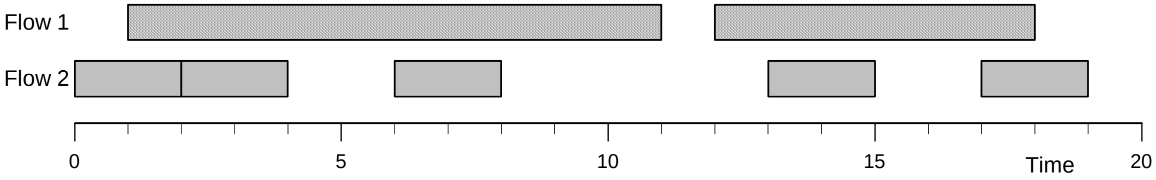
Determine the finish number of the packet that just arrived. For all the packets under consideration, write down the order of transmissions under packet-by-packet FQ. Show the process.

Problem 5.5

Consider the following scenario for a packet-by-packet FQ scheduler and transmission rate equal 1 bit per unit of time. At time t=0 a packet of $L_{1,1}$=100 bits arrives on flow 1 and a packet of $L_{3,1}$=60 bits arrives on flow 3. The subsequent arrivals are as follows: $L_{1,2}$=120 and $L_{3,2}$=190 at t=100; $L_{2,1}$=50 at t=200; $L_{4,1}$=30 at t=250; $L_{1,3}$=160 and $L_{4,2}$=30 at t=300, $L_{4,3}$=50 at 350, $L_{2,2}$=150 and $L_{3,3}$=100 at t=400; $L_{1,4}$=140 at t=460; $L_{3,4}$=60 and $L_{4,4}$=50 at t=500; $L_{3,5}$=200 at t=560; $L_{2,3}$=120 at t=600; $L_{1,5}$=700 at t=700; $L_{2,4}$=50 at t=800; and $L_{2,5}$=60 at t=850. For every time new packets arrive, write down the sorted finish numbers. What is the actual order of transmissions under packet-by-packet FQ?

Problem 5.6

Consider the timetable of packet arrivals on two flows as shown in the figure below. All packets are to be transmitted on the same outgoing port. The arrival time for each packet is exactly aligned on the time axis and the packet "length" symbolizes the packet's transmission time.



Solve the following:

(a) Calculate the timetable of packet departures under FCFS and FQ scheduling disciplines.

(b) Calculate the waiting delays that will be experienced by each packet under different scheduling disciplines, as well as the average delays for each flow.

(c) Draw the curve representing the packet delay that would be experienced by a hypothetical packet that arrived at any time on each flow under different scheduling disciplines (similar to the diagram in the book in Figure 5-11).

Problem 5.7

Suppose a router has four input flows and one output link with the transmission rate of 1 byte/second. The router receives packets as listed in the table below. Assume the time starts at 0 and the "arrival time" is the time the packet arrives at the router. Write down the order and times at which the packets will be transmitted under:

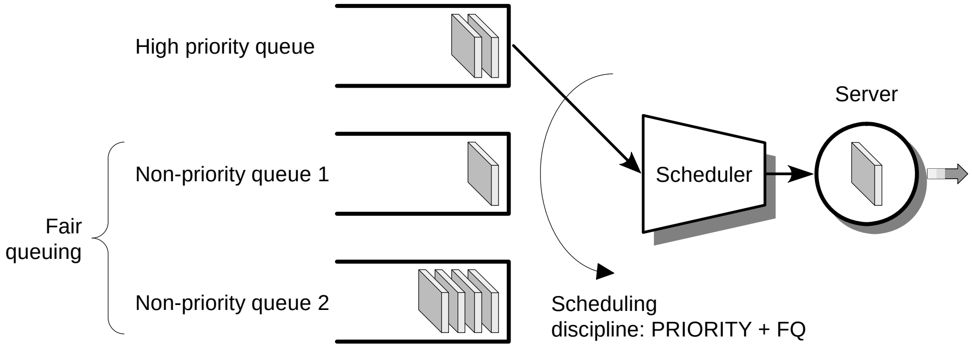
(a) Packet-by-packet fair queuing (FQ)

(b) Packet-by-packet weighted fair queuing (WFQ), where flows 2 and 4 are entitled to twice the link capacity of flow 3, and flow 1 is entitled to twice the capacity of flow 2

Packet #	Arrival time [sec]	Packet size [bytes]	Flow ID	Departure order/ time under FQ	Departure order/ time under WFQ
1	0	100	1		
2	0	60	3		
3	100	120	1		
4	100	190	3		
5	200	50	2		
6	250	30	4		
7	300	30	4		
8	300	60	1		
9	650	50	3		
10	650	30	4		
11	710	60	1		
12	710	30	4		

Problem 5.8

[*Priority + Fair Queuing*] Consider a scheduler with three queues: one high priority queue and two non-priority queues that should share the resource that remains after the priority queue is served in a fair manner. The priority packets are scheduled to go first (lined up in their order of arrival), regardless of whether there are packets in non-priority queues. The priority packets are scheduled in a *non-preemptive* manner, which means that any packet currently being serviced from a non-priority queue is allowed to finish.



Modify the formula for calculating the packet finish number.

Assume that the first several packets arrive at following times:

Priority queue: (arrival time $A_{1,1}$ = 5, packet length $L_{1,1}$ = 2); ($A_{1,2}$ = 8, $L_{1,2}$ = 2);

First non-priority queue: ($A_{2,1}$ = 0, $L_{2,1}$ = 6); ($A_{2,2}$ = 7, $L_{2,2}$ = 1);

Second non-priority queue: ($A_{3,1}$ = 1, $L_{3,1}$ = 2); ($A_{3,2}$ = 7, $L_{3,2}$ = 1);

Show the order in which these packets will leave the server and the departure times.

Problem 5.9

Problem 5.10

Problem 5.11

Problem 5.12

Problem 5.13

Active Queue Management: RED.

A router has a buffer size of 10 packets. Assume that all packets are of the same size. The router employs RED, with parameters set as Thr_{min} = 0, Thr_{max} = 7. Initially the buffer contains 9 packets queued, AvgQLen(0) = 6.75, count = 0, and P_{max} = 0.1. Packets are departing the router at the rate of 1 packet per unit of time.

Suppose that a new packet will arrive during each of the subsequent five time units. Solve the following:

(a) Calculate the average queue length and the probability of a packet being dropped for the first five time units.

(b) Determine if any packet will be dropped and if yes, when.

For the sake of simplicity, assume that in Listing 5-1 the decision in step 2.c) is made by simple thresholding: only if $P(AvgQLen) \geq 0.15$ then drop the packet.

Problem 5.14

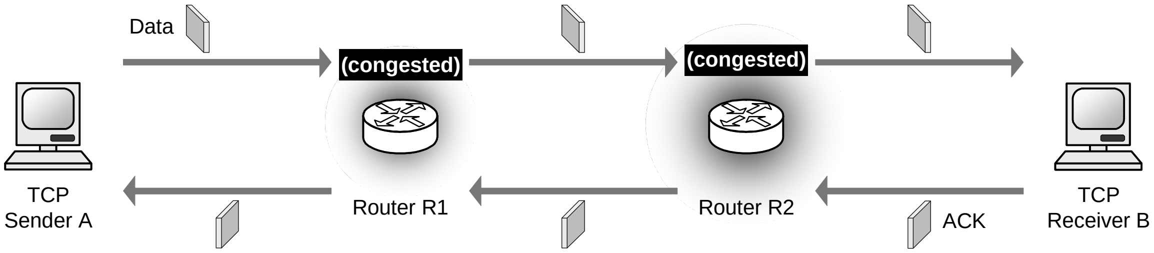
Consider the Random Early Detection (RED) algorithm (Section 5.3.1). One may wonder why in Figure 5-14 is $Threshold_{max} < BufferSize$ instead of being equal to *BufferSize*? If $Threshold_{max}$ is less than *BufferSize*, then it appears that part of the buffer will never be filled because as soon as $Threshold_{max}$ is reached, packets will be dropped. Explain this design choice.

Hint: Carefully examine equation (5.6) and Listing 5-1.

Problem 5.15

Problem 5.16

Consider the following scenario for Explicit Congestion Notification (ECN). The TCP sender *A* is currently in the slow start state and does not have unacknowledged packets. It has just sent a burst of 4 packets and while forwarding these packets, both routers became congested. Assume that router *R*1 marked with the CE codepoint "11" all of *A*'s packets starting with the second packet of the burst. (In other words, only the first packet from *A* was let pass unmarked.) However, router *R*2 experienced the buffer overflow and *dropped* only the second packet from *A* (i.e., *A*'s first marked packet). *R*2 leaves the ECN field unchanged for all other packets from *A*.



Assume that there is no packet reordering in the network, i.e., all packets are forwarded in-order, and no other packet is lost except for the second packet from *A*. All ACKs carry no payload and are sent without delay (i.e., no cumulative ACKs are used). Also assume that none of the nodes in the network is misbehaving. Finally, after the first burst, congestion on both routers subsides and no future packets are marked.

Solve the following:

(a) Determine the subsequent sequence of packets from TCP sender *A* until it discovers the packet loss and retransmits the lost packet. Explain how will TCP sender *A* discover the packet loss.

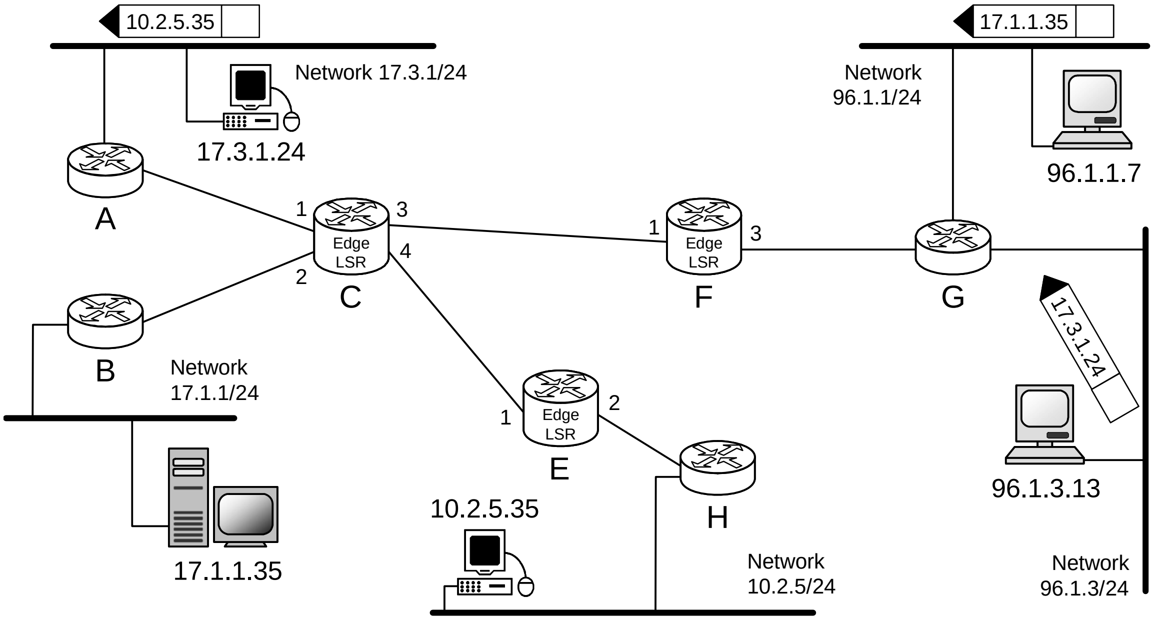
(b) For all *A*'s packets, starting with the burst of 4 packets until and including the packet with the Congestion Window Reduced (CWR) flag set, show the values of all ECN-specific flags in their IP headers and TCP headers. Similarly, for all ACK packets from the receiver *B*, show the values of all ECN-specific flags in their IP headers and TCP headers.

Indicate if and how the sender will reduce its congestion parameters, but do not specify the numeric values.

Problem 5.17

Problem 5.18

Consider the network in Figure 5-19 with the hosts attached as shown in the figure below. (As in Figure 5-19, routers *C*, *E*, and *F* are MPLS-capable.) Assume that the network starts in the initial state, where all IP routing tables are already built, but LFIBs (label forwarding information bases) are empty. Now assume that three hosts start sending data, first host `96.1.1.7` sends a packet to host `17.1.1.35`, then host `17.3.1.24` sends a packet to `10.2.5.35`, and finally `96.1.3.13` sends a packet to `17.3.1.24`. Assume that all LSPs (label switched paths) will be built based on the shortest paths found by the IP routing protocols and that the FECs (forwarding equivalence classes) will be determined only based on the destination IP addresses.



(a) Show step-by-step how LSPs will be built and what will be the entries of the LFIBs for MPLS-capable routers.

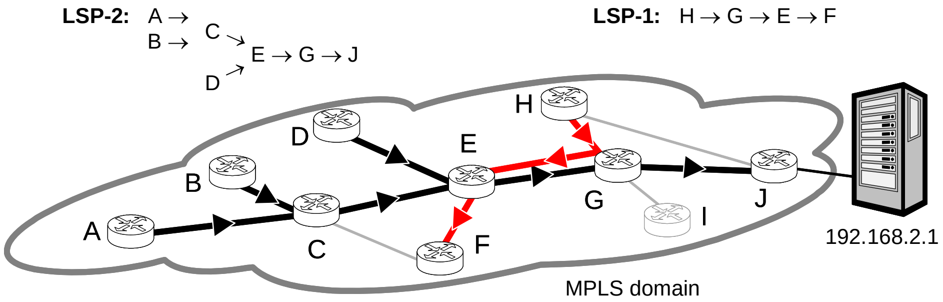
(b) For every instance of packet forwarding, indicate whether an LFIB or an ordinary IP forwarding table will be used to forward the packet. In case of LFIB-based forwarding, show the packet's MPLS label value.

(c) What is the minimum number of FECs and what is the minimum number of LSPs that needs to be set up?

Problem 5.19

Consider a scenario where customer company computers connect from different locations to a remote server (which has the IP address 192.168.2.1). The client hosts are connected to an ISP at different ingress points of an MPLS domain and all clients receive the same level of service.

Assume that the MPLS domain topology is as shown in Figure 5-24 (reproduced below), our customer forms LSP-2 and another customer forms LSP-1.



Solve the following:

(a) Write down a list of plausible MPLS label values (preferably use small numbers) for all links that belong to LSPs in the figure.

(b) Describe any constraints on the label values in (a) that you can think of.

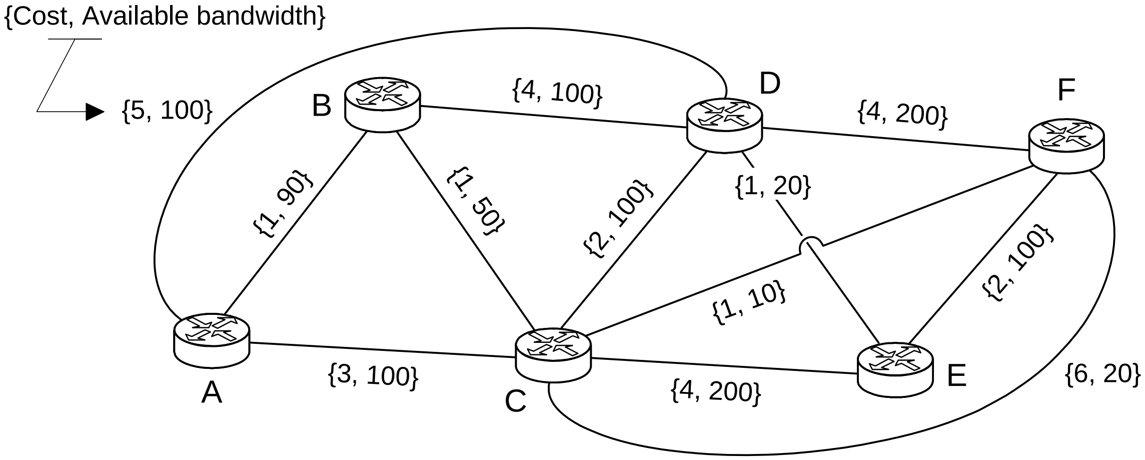
(c) Based on the label values from part (a), write down the LFIB tables of all LSRs in the figure. (Label the port numbers that you will need for the LFIBs and show your labeling on the figure.)

(d) Assuming a packet is sent to the server from a host connected to router *B*, write down the content of the packet's label stack on each link along the path from *B* to *J*.

Problem 5.20

Problem 5.21

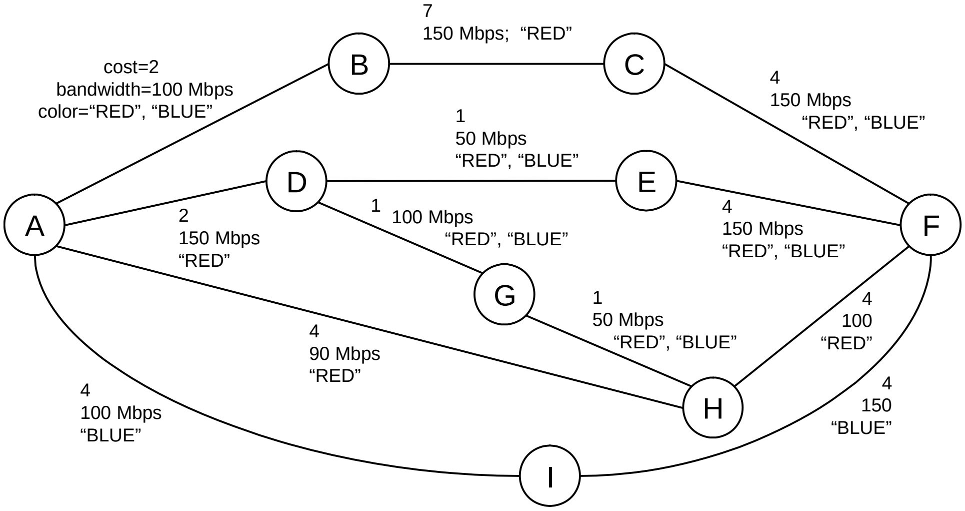
Consider the MPLS domain shown in the figure below (the units of the available bandwidth are Mbps). All LSRs are using Constrained Shortest Path First (CSPF) routing. Assume that you want to build a tunnel from LSR *A* to LSR *F* with a bandwidth of 20 Mbps. Show step-by-step how CSPF will select a path that best fits this description. Always list all possible alternative paths that will be considered and explain which one will be selected and why.



Problem 5.22

Consider the MPLS domain shown in Figure 5-27(b), that is slightly modified as shown in the figure below. (Note that the reservable bandwidth of link $G \rightarrow H$ is 50 Mbps.) Assume that you want to build an LSP tunnel from LSR A to LSR F using Constrained Shortest Path First (CSPF) routing. In addition to minimum reserved path bandwidth of 80 Mbps, the LSP path must satisfy the *link color* constraint. The traffic for which the tunnel is built is colored "RED," which is achieved by setting a corresponding binary pattern in the Differentiated Services field of the IP header of each packet belonging to this flow. Packets are allowed to travel only over the links that include their color. For example, "RED" packets can only travel over "RED" links.

Show step-by-step how CSPF will select a path that best fits this description. Always list the alternative paths that will be considered and explain which one will be selected.



Problem 5.23

Chapter 6
Wireless Networks

Contents

This chapter reviews wireless networks. The focus is on the network and link layers, and very little is mentioned about the physical layer of wireless networks. In addition, there is a little mention of infrastructure-based wireless networks and the focus is on infrastructure-less wireless networks.

6.1 Mesh Networks

In a multihop wireless ad hoc network, mobile nodes cooperate to form a network without the help of any infrastructure such as access points or base stations. The mobile nodes, instead, forward packets for each other, allowing nodes beyond direct wireless transmission range of each other to communicate over possibly multihop routes through a number of forwarding peer mobile nodes. The mobility of the nodes and the fundamentally limited capacity of the wireless channel, together with wireless transmission effects such as attenuation, multipath propagation, and interference, combine to create significant challenges for network protocols operating in an ad hoc network.

Figure 6-1

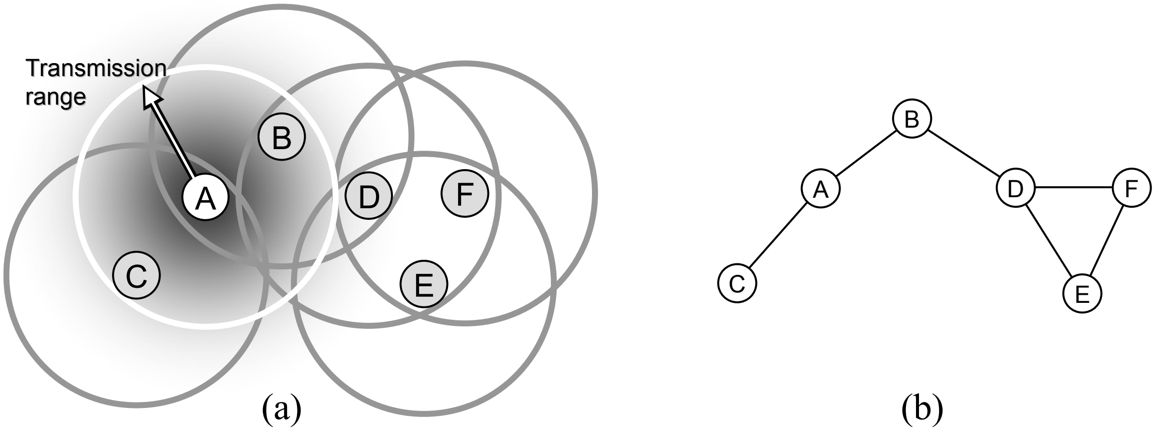


Figure 6-1: Example wireless mesh network: (a) Physical wireless links; (b) Network topology.

Figure 6-2

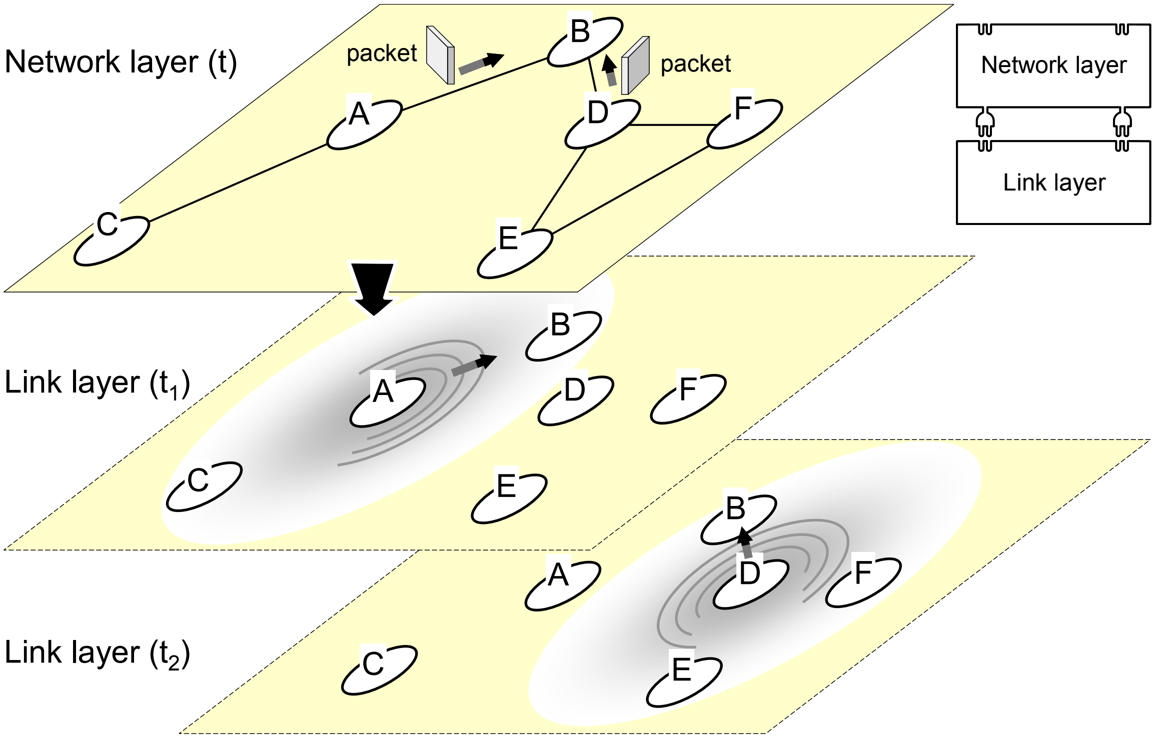


Figure 6-2: In network layer notation (top plane), a node can simultaneously receive two packets. In reality, link layer transmissions (bottom two planes, at different times) must contend for channel access and transmit by taking turns.

6.2 Routing Protocols for Mesh Networks

In wired networks with fixed infrastructure, a communication endpoint device, known as "host," does not normally participate in routing protocols. This role is reserved for intermediary computing "nodes" that relay packets from a source host to a destination host. On the other hand, in wireless mesh networks it is common that computing nodes assume both "host" and "node" roles—all nodes may be communication endpoints and all nodes may relay packets for other nodes. Therefore, in this chapter I use the terms "host" and "node" interchangeably.

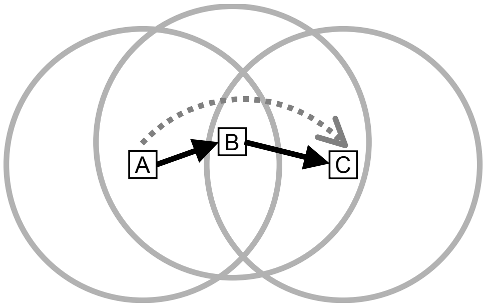


Figure 6-3: Example mobile ad-hoc network: *A* communicates with *C* via *B*.

Although there have been dozens of new routing protocols proposed for MANETs, the majority of these protocols actually rely on fundamental techniques that have been studied rigorously in the wired environment. However, each protocol typically employs a new heuristic to improve or optimize a legacy protocol for the purposes of routing in the mobile wireless environment. In fact, there are a few mechanisms that have received recent interest primarily because of their possible application to MANETs. There are two main classes of routing protocols:

- Proactive
  - Continuously update reachability information in the network
  - When a route is needed, it is immediately available
  - DSDV by Perkins and Bhagwat (SIGCOMM 94)
  - Destination Sequenced Distance vector
- Reactive
  - Routing discovery is initiated only when needed
  - Route maintenance is needed to provide information about invalid routes
  - DSR by Johnson and Maltz
  - AODV by Perkins and Royer
- Hybrid
  - Zone routing protocol (ZRP)

Centralized vs. localized solution:

Nodes in *centralized* solution need to know full network information to make decision; mobility or changes in activity status (power control) cause huge communication overhead to maintain the network information.

Nodes in *localized* algorithm require only local knowledge (direct neighbors, 2-hop neighbors) to make decisions. Majority of published solutions are centralized, compared with other centralized solutions.

Next, a brief survey of various mechanisms is given.

6.2.1 Dynamic Source Routing (DSR) Protocol

Problems related to this section: Problem 6.2 → ??

Source routing means that the sender must know in advance the complete sequence of hops to be used as the route to the destination. DSR is an *on-demand* (or *reactive*) ad hoc network routing protocol, i.e., it is activated only when the need arises rather than operating continuously in background by sending periodic route updates. DSR divides the routing problem in two parts: *Route Discovery* and *Route Maintenance*, both of which operate entirely on-demand. In Route Discovery, a node actively searches through the network to find a route to an intended destination node. While using a route to send packets to the destination, the sending node runs the Route

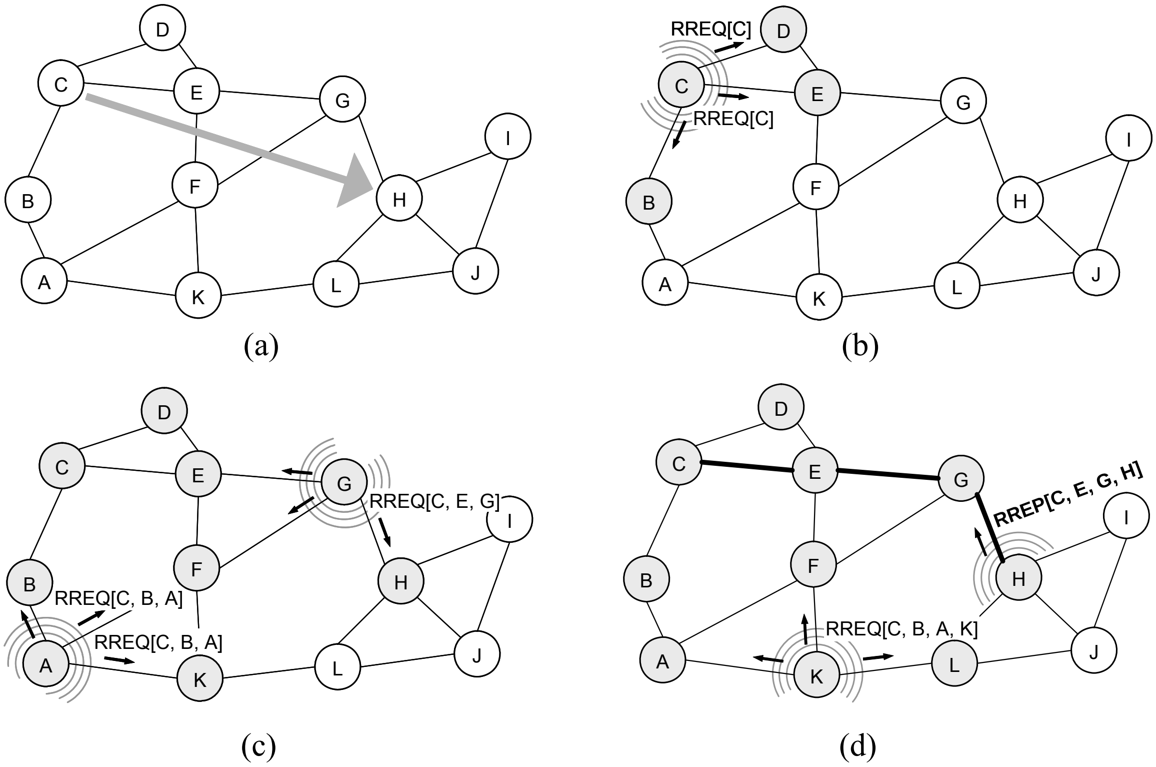


Figure 6-4: Route discovery in DSR: node *C* seeks to communicate to node *H*. Gray shaded nodes already received RREQ. The path in bold in (c) indicates the route selected by *H* for RREP. See text for details. (Note: the step where *B* and *E* broadcast RREQ is not shown.)

Maintenance process by which it determines if the route has broken, for example because two nodes along the route have moved out of wireless transmission range of each other.

An example is illustrated in Figure 6-4, where host *C* needs to establish a communication session with host *H*. A node that has a packet to send to a destination (*C* in our example) searches its Route Cache for a route to that destination. If no cached route is found, node *C* initiates Route Discovery by broadcasting a ROUTE REQUEST (RREQ) packet containing the destination node address (known as the *target* of the Route Discovery), a list (initially empty) of nodes traversed by this RREQ, and a *request identifier* from this source node. The request identifier, the address of this source node (known as the *initiator* of the Route Discovery), and the destination address together uniquely identify this Route Discovery attempt.

A node receiving a ROUTE REQUEST checks to see if it has previously forwarded a RREQ from this Discovery by examining the IP Source Address, destination address, and request identifier. For example, in Figure 6-4(b), nodes *B*, *E*, and *D* are the first to receive RREQ and they re-broadcast it to their neighbors. If the recipient of RREQ has recently seen this identifier, or if its own address is already present in the list in RREQ of nodes traversed by this RREQ, the node silently drops the packet. Otherwise, it appends its address to the node list and re-broadcasts the REQUEST. When a RREQ reaches the destination node, *H* in our example, this node returns a ROUTE REPLY (RREP) to the initiator of the ROUTE REQUEST. If an intermediary node receives a RREQ for a destination for which it caches the route in its Route Cache, it can send RREP back to the source without further propagating RREQ. The RREP contains a copy of the node list from the RREQ, and can be delivered to the initiator node by reversing the node list, by using a route back to the initiator from its own Route Cache, or "piggybacking" the RREP on a new ROUTE REQUEST targeting the original initiator. This path is indicated with bold lines in Figure 6-4(d). When the initiator of the request (node *C*) receives the ROUTE REPLY, it adds the newly acquired route to its Route Cache for future use.

In Route Maintenance mode, an intermediary node forwarding a packet for a source attempts to verify that the packet successfully reached the next hop in the route. A node can make this confirmation using a hop-to-hop acknowledgement at the link layer (such as is provided in IEEE 802.11 protocol), a passive acknowledgement (i.e., listen for that node sending packet to its next hop), or by explicitly requesting network- or higher-layer acknowledgement. Transmitting node can also solicit ACK from next-hop node. A packet is possibly retransmitted if it is sent over an unreliable MAC, although it should not be retransmitted if retransmission has already been attempted at the MAC layer. If a packet is not acknowledged, the forwarding node assumes that the next-hop destination is unreachable over this link, and sends a ROUTE ERROR to the source of the packet, indicating the broken link. A node receiving a ROUTE ERROR removes that link from its Route Cache.

In the basic version of DSR, every packet carries the entire route in the header of the packet, but some recent enhancements to DSR use implicit source routing to avoid this overhead. Instead, after the first packet containing a full source route has been sent along the route to the destination, subsequent packets need only contain a flow identifier to represent the route, and nodes along the route maintain flow state to remember the next hop to be used along this route based on the address of the sender and the flow identifier; one flow identifier can designate the default flow for this source and destination, in which case even the flow identifier is not represented in a packet.

A number of optimizations to the basic DSR protocol have been proposed [Perkins 2001, Chapter 5]. One example of such an optimization is *packet salvaging*. When a node forwarding a packet fails to receive acknowledgement from the next-hop destination, as described above, in addition to sending a ROUTE ERROR back to the source of the packet, the node may attempt to use an alternate route to the destination, if it knows of one. Specifically, the node searches its Route Cache for a route to the destination; if it finds one, then it salvages the packet by replacing the existing source route for the packet with the new route from its Route Cache. To prevent the possibility of infinite looping of a packet, each source route includes a *salvage count*, indicating how many times the packet has been salvaged in this way. Packets with salvage count larger than some predetermined value cannot be salvaged again.

In summary, DSR is able to adapt quickly to dynamic network topology but it has large overhead in data packets. The protocol does not assume bidirectional links.

6.2.2 Ad Hoc On-Demand Distance-Vector (AODV) Protocol

Problems related to this section: ?? → ??

DSR includes source routes in packet headers and large headers can degrade performance, particularly when data contents of a packet are small. AODV attempts to improve on DSR by maintaining *routing tables* at the nodes, so that data packets do *not* have to contain routes. AODV

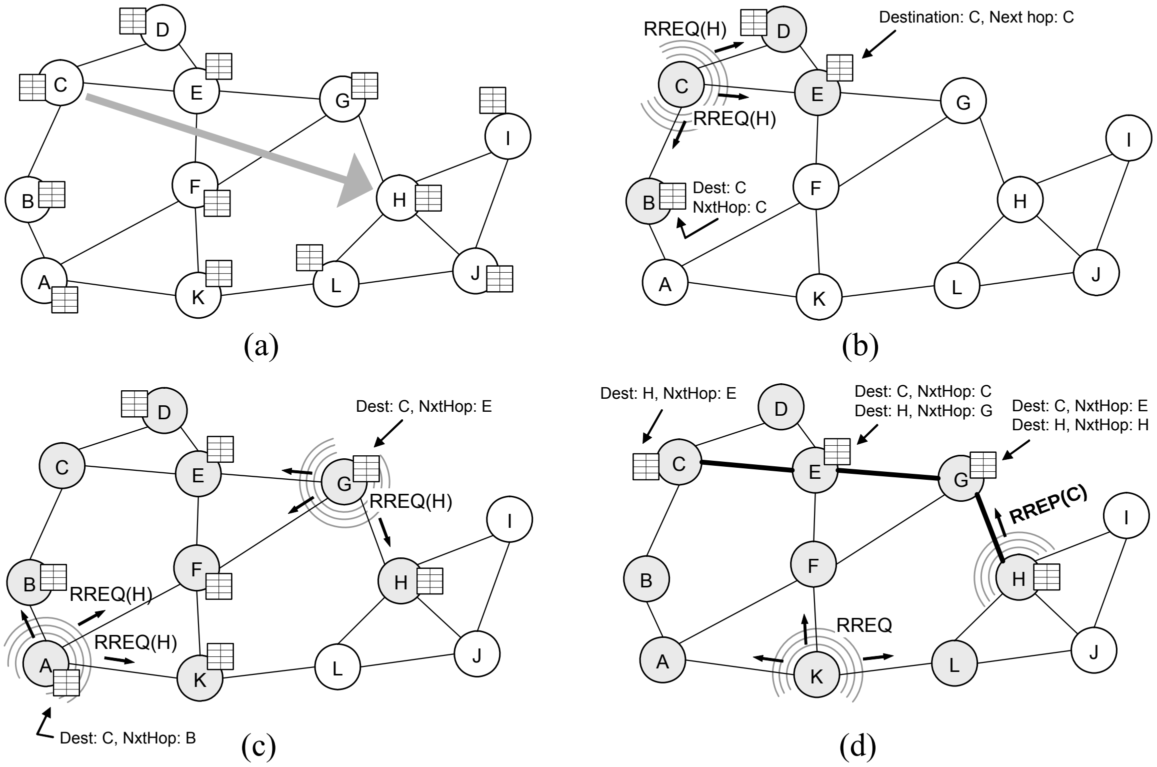


Figure 6-5: Route discovery in AODV (Compare to DSR in Figure 6-4.)

retains the desirable feature of DSR that routes are maintained only between nodes which need to communicate.

Route Request packets are forwarded in a manner similar to DSR (Figure 6-5). When a node re-broadcasts a Route Request, it sets up a reverse path pointing towards the source. AODV assumes symmetric (bidirectional) links. When the intended destination receives a RREQ, it replies by sending a Route Reply. RREP travels along the reverse path set-up when RREQ is forwarded.

An intermediate node (not the destination) may also send a RREP, provided that it knows a more recent path than the one previously known to sender S. To determine whether the path known to an intermediate node is more recent, destination sequence numbers are used. The likelihood that an intermediate node will send a RREP when using AODV is not as high as in DSR. A new RREQ by node S for a destination is assigned a higher destination sequence number. An intermediate node, which knows a route but with a smaller sequence number, *cannot send* RREP.

A routing table entry maintaining a reverse path is purged after a timeout interval. Timeout should be long enough to allow RREP to come back. A routing table entry maintaining a forward path is purged if not used for an active_route_timeout interval. If no is data being sent using a particular routing table entry, that entry will be deleted from the routing table (even if the route may actually still be valid).

In summary, routes in AODV need not be included in the headers of data packets (unlike DSR, where every data packet carries the source route). Nodes maintain routing tables containing entries only for routes that are in active use. At most one next-hop per destination is maintained at each node, whereas DSR may maintain several routes for a single destination. Lastly, unused routes expire even if topology does not change.

6.3 More Wireless Link-Layer Protocols

Section 1.5.3 described Wi-Fi (IEEE 802.11). This section describes more wireless link-layer protocols and technologies.

6.3.1 IEEE 802.11n (MIMO Wi-Fi)

Problems related to this section: ?? → ??

IEEE 802.11n builds on previous 802.11 standards (Section 1.5.3) by adding mechanisms to improve network throughput. 802.11n operates in the 2.4-GHz and 5-GHz frequency bands. A key improvement is in the radio communication technology, but 802.11n is much more than just a new radio for 802.11. In addition to providing higher bit rates (as was done in 802.11a, b, and g), 802.11n significantly changed the frame format of 802.11. Specifically, 802.11n added *multiple-input multiple-output* (MIMO, pronounced *my-moh*) and 40-MHz channels to the physical layer (PHY), and *frame aggregation* to the MAC layer. It achieves a significant increase in the maximum raw data rate over the two previous standards (802.11a and 802.11g), from 54 Mbps to 600 Mbps, improves reliability, and increases transmission distance. At 300 feet, 802.11g performance drops to 1 Mbps; on the other hand, at the same distance 802.11n networks operate at up to 70 Mbps, which is 70 times faster than 802.11g.

IEEE 802.11n-capable devices are also referred as *High Throughput (HT) devices*. An HT device declares that it is an HT device by transmitting the *HT Capabilities element*. The device also uses the HT Capabilities element to advertise which optional capabilities of the 802.11n standard it implements. The HT Capabilities element is carried as part of some control frames that wireless devices exchange during the connection setup or in periodical announcements. It is present in these frames: Beacon, Association Request, Association Response, Reassociation Request, Reassociation Response, Probe Request and Probe Response frames.

IEEE 802.11n standard modifies the frame formats used by 802.11n devices from those of "legacy" 802.11 devices. When 802.11n devices are operating in pure high-throughput mode, this is known as "greenfield mode," because it lacks any constraints imposed by prior technologies. This mode achieves the highest effective throughput offered by the 802.11n standard. To avoid rendering the network useless due to massive interference and collisions, the standard describes some mechanisms for backward compatibility with existing 802.11a/b/g deployments. These mechanisms are reviewed at the end of this section.

Physical (PHY) Layer Enhancements

A key to the 802.11n speed increase is the use of multiple antennas to send and receive more than one communication signals simultaneously, thus multiplying total performance of the Wi-Fi signal. This is similar to having two FM radios tuned to the same channel at the same time—the signal becomes louder and clearer. As for a receiver side analogy, people hear better with both ears than if one is shut. **Multiple-input multiple-output (MIMO)** is a technology that uses multiple antennas to resolve coherently more information than possible using a single antenna. Each 802.11n device has two radios: for transmitter and receiver. Although previous 802.11 technologies commonly use one transmit and two receive antennas, MIMO uses multiple independent transmit and receive antennas. This is reflected in the two, three, or even more antennas found on some 802.11n access points or routers (Figure 6-6). The network client cards on 802.11n mobile devices also have multiple antennas, although these are not that prominently visible. Each antenna can establish a separate (but simultaneous) connection with the corresponding antenna on the other device.

MIMO technology takes advantage of what is normally the enemy of wireless networks: *multipath propagation*. Multipath is the way radio frequency (RF) signals bounce off walls, ceilings, and other surfaces and then arrive with different amounts of delay at the receiver. MIMO is able to process and recombine these scattered and otherwise useless signals using sophisticated signal-processing algorithms.

A MIMO transmitter divides a higher-rate data stream into multiple lower-rate streams. (802.11n MIMO uses up to four streams.) Each of the unique lower-rate streams is then transmitted on the same spectral channel, but through a different transmit antenna via a separate spatial path to a corresponding receiver. The multiple transmitters and antennas use a process called *transmit beamforming* (TxBF) to focus the output power in the direction of the receivers. TxBF steers an outgoing signal stream toward the intended receiver by concentrating the transmitted radio energy in the appropriate direction. This increases signal strength and data rates. On the receiving end, multiple receivers and antennas reverse the process using *receive combining*.

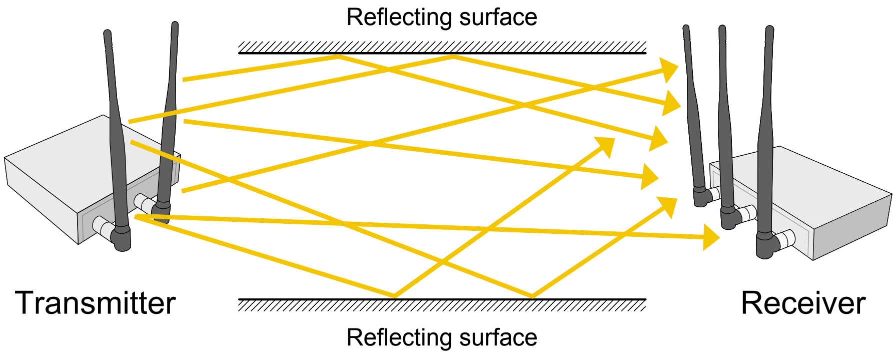


Figure 6-6: MIMO wireless devices with two transmitter and three receiver antennas. Notice how multiple radio beams are reflected from objects in the environment to arrive at the receiver antennas.

The receiving end is where the most of the computation takes place. Each receiver receives a separate data stream and, using sophisticated signal processing, recombines the data into the original data stream. This technique is called Spatial Division Multiplexing (SDM). MIMO SDM can significantly increase data throughput as the number of resolved spatial data streams is increased. Spatial multiplexing combines multiple beams of data at the receiving end, theoretically multiplying throughput—but also multiplying the chances of interference. This is why the transmitter and the receiver must cooperate to mitigate interference by sending radio energy only in the intended direction. The transmitter needs feedback information from the receiver about the received signal so that the transmitter can tune each signal it sends. This feedback is available only from 802.11n devices, not from 802.11a, b, or g devices. This feedback is not immediate and is only valid for a short time. Any physical movement by the transmitter, receiver, or elements in the environment will quickly invalidate the parameters used for beamforming. The wavelength for a 2.4-GHz radio is only 120mm, and only 55mm for 5-GHz radio. Therefore, a normal walking pace of 1 meter per second will rapidly move the receiver out of the spot where the transmitter's beamforming efforts are most effective. In addition, transmit beamforming is useful only when transmitting to a single receiver. It is not possible to optimize the phase of the transmitted signals when sending broadcast or multicast transmissions.

The advantage of this approach is that it can achieve great throughput on a single, standard, 20-MHz channel while maintaining backward compatibility with legacy 801.11b/g devices. The net impact is that the overall signal strength (that is, *link budget*) is improved by as much as 5 dBi (dB isotropic). Although that may not sound significant, the improved link budget allows signals to travel farther, or, alternatively, maintains a higher data rate at a given distance as compared with a traditional 802.11g single transmitter/receiver product.

Each spatial stream requires a separate antenna at both the transmitter and the receiver. 802.11n defines many "$M \times N$" antenna configurations, ranging from "1 × 1" to "4 × 4." This refers to the number of transmit (M) and receive (N) antennas—for example, an access point with two transmit and three receive antennas is a "2 × 3" MIMO device. In addition, MIMO technology requires a separate radio frequency chain and analog-to-digital converter for each MIMO antenna. This translates to higher implementation costs compared to non-MIMO systems.

Channel bonding. In addition to MIMO, the physical layer of 802.11n can use double-wide channels that occupy 40 MHz of bandwidth. Legacy 802.11a, b, and g devices use 20-MHz-wide channels to transmit data. 802.11n can bond two 20-MHz channels that are adjacent in the frequency domain into one that is 40 MHz wide. That doubling of bandwidth results in a theoretical doubling of information-carrying capacity (data transmission rate). Up to four data-streams can be sent simultaneously using 20MHz or 40MHz channels. A theoretical maximum data rate of 600 Mbps can be achieved using four double-width channels (40 MHz). Although the initial intention of the 40MHz channel was for the 5 GHz band, because of the additional new spectrum, 40MHz channels are permitted in the 2.4 GHz band. Due to the limited spectrum and overlapping channels, 40MHz operation in 2.4 GHz requires special attention.

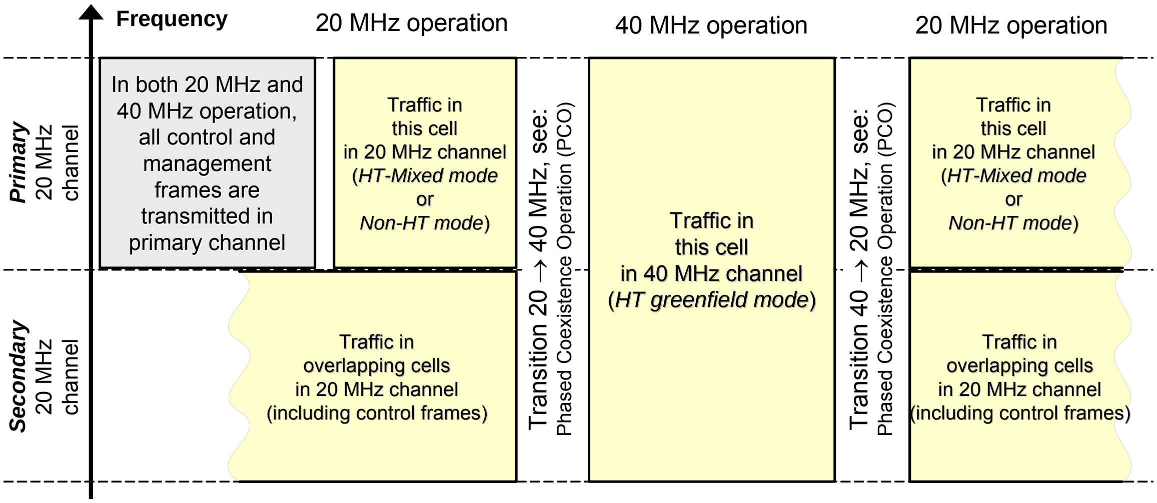

Frequency
20 MHz operation
40 MHz operation
20 MHz operation
Primary 20 MHz channel
Secondary 20 MHz channel
In both 20 MHz and 40 MHz operation, all control and management frames are transmitted in primary channel
Traffic in this cell in 20 MHz channel (HT-Mixed mode or Non-HT mode)
Traffic in overlapping cells in 20 MHz channel (including control frames)
Transition 20 → 40 MHz, see: Phased Coexistence Operation (PCO)
Traffic in this cell in 40 MHz channel (HT greenfield mode)
Transition 40 → 20 MHz, see: Phased Coexistence Operation (PCO)
Traffic in this cell in 20 MHz channel (HT-Mixed mode or Non-HT mode)
Traffic in overlapping cells in 20 MHz channel (including control frames)

Figure 6-7: 802.11n channel bonding and 20/40 MHz operation. (Phased Coexistence Operation (PCO) is described later, in Figure 6-22.)

A 40-MHz channel is created by bonding two contiguous 20-MHz channels: a “primary” or “control” 20-MHz channel and a “secondary” or “extension” 20-MHz channel (Figure 6-7). **Primary channel** is the common channel of operation for all stations (including HT and non-HT) that are members of the BSS (Basic Service Set, defined in Section 1.5.3). To preserve interoperability with legacy clients, 802.11n access point transmits all control and management frames in the primary channel. All 20-MHz clients (whether HT or legacy non-HT) only associate to the primary channel, because the beacon frame is only transmitted on the primary channel. All transmissions to and from clients must be on the primary 20 MHz channel. Hence, all 40-MHz operation in 802.11n is termed “20/40 MHz.” **Secondary channel** is a 20-MHz channel associated with a primary channel; it may be located in the frequency spectrum below or above the primary channel. It is used only by HT stations for creating a 40-MHz channel. A station is not required to react to control frames received on its secondary channel, even if it is capable of decoding those frames. The secondary channel of one BSS may be used by an overlapping BSS as its primary channel. If an access point detects an overlapping BSS whose primary channel is the access point’s secondary channel, it switches to 20-MHz operation and may subsequently move to a different channel or pair of channels.

Phased Coexistence Operation (PCO) is an option in which an 802.11n access point alternates between using 20-MHz and 40-MHz channels. Before operating in the 40-MHz mode, the access point explicitly reserves both adjacent 20-MHz channels. This mechanism is described later in this section.

Based on the bandwidth used by devices in an 802.11n network, the operational modes can be classified as follows:

- *Legacy (non-HT) mode*. The operation is similar to IEEE 802.11a/b/g. This mode uses the primary 20-MHz channel for transmission.
- *Duplicate legacy mode*. In this mode, the devices use a 40-MHz channel bandwidth, but the same data are transmitted in the primary and secondary halves of the 40-MHz

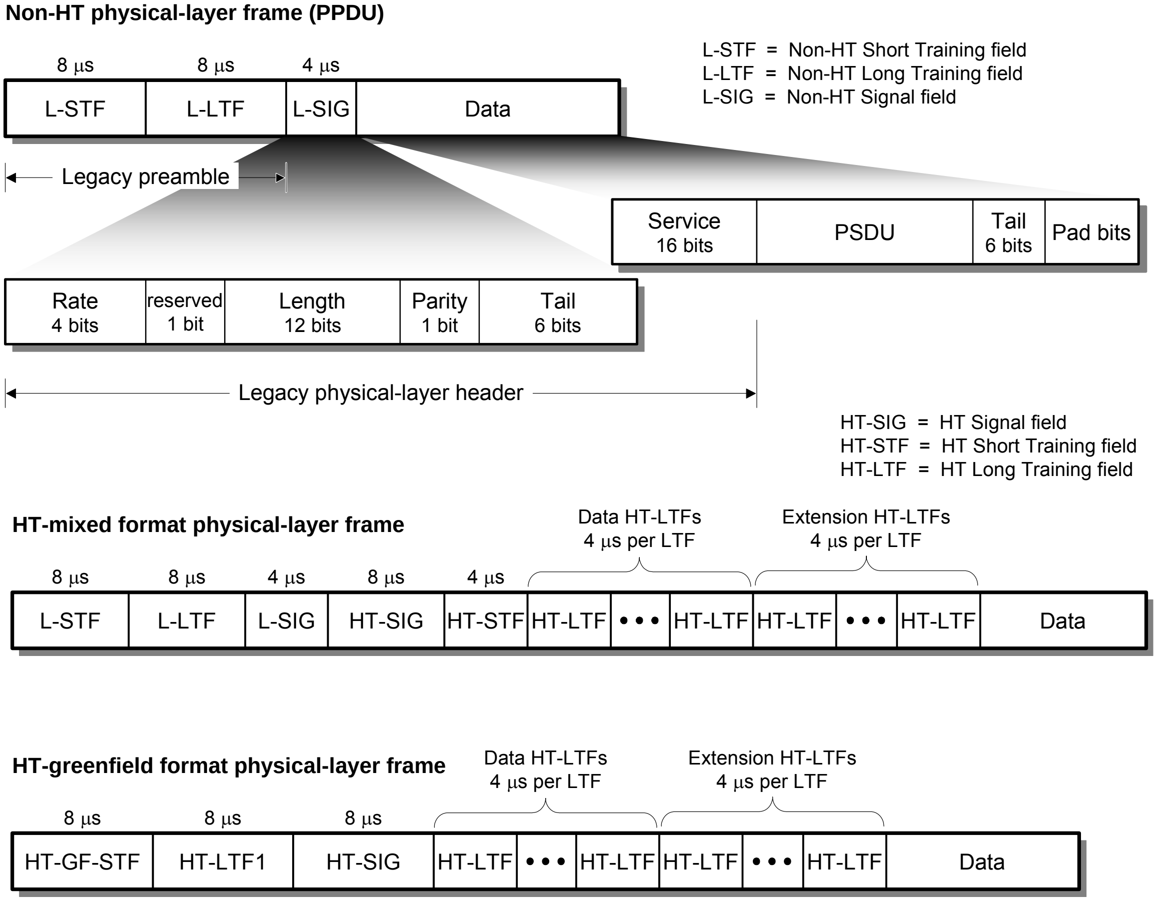

Non-HT physical-layer frame (PPDU)
8 µs
8 µs
4 µs
L-STF
L-LTF
L-SIG
Data
L-STF = Non-HT Short Training field
L-LTF = Non-HT Long Training field
L-SIG = Non-HT Signal field
Legacy preamble
Service 16 bits
PSDU
Tail 6 bits
Pad bits
Rate 4 bits
reserved 1 bit
Length 12 bits
Parity 1 bit
Tail 6 bits
Legacy physical-layer header
HT-SIG = HT Signal field
HT-STF = HT Short Training field
HT-LTF = HT Long Training field
HT-mixed format physical-layer frame
Data HT-LTFs 4 µs per LTF
Extension HT-LTFs 4 µs per LTF
8 µs
8 µs
4 µs
8 µs
4 µs
L-STF
L-LTF
L-SIG
HT-SIG
HT-STF
HT-LTF
• • •
HT-LTF
HT-LTF
• • •
HT-LTF
Data
HT-greenfield format physical-layer frame
Data HT-LTFs 4 µs per LTF
Extension HT-LTFs 4 µs per LTF
8 µs
8 µs
8 µs
HT-GF-STF
HT-LTF1
HT-SIG
HT-LTF
• • •
HT-LTF
HT-LTF
• • •
HT-LTF
Data

Figure 6-8: 802.11n physical-layer frame formats. Compare to Figure 1-78(b).

channel. This feature allows the station to send a control frame simultaneously on both 20-MHz channels, which improves efficiency. Examples are given later in this section.

- *High-throughput (HT) mode*. HT mode is available for both 20- and 40MHz channels. In this mode, supporting one and two spatial streams is mandatory. A maximum of four spatial streams is supported.
- *HT duplicate mode*. This mode uses the modulation and coding scheme (MCS) #32 that provides the lowest transmission rate in a 40-MHz channel (6 Mbps), as well as longer transmission range.

Figure 6-8 shows the physical-layer frame formats supported by 802.11n: the legacy format and new formats. Two new formats, called *HT formats*, are defined for the PLCP (PHY Layer Convergence Protocol): *HT-mixed format* and *HT-greenfield format*. There is also an MCS-32 frame format used for the *HT duplicate mode*. In addition to the HT formats, there is a non-HT duplicate format, used in the *duplicate legacy mode*, which duplicates a 20-MHz non-HT (legacy) frame in two 20-MHz halves of a 40-MHz channel.

The legacy **Non-HT frame format** (top row in Figure 6-8) is the 802.11a/g frame format and can be decoded by legacy stations. (Note that this “legacy” 802.11a/g frame format is different from 802.11b legacy format, shown in Figure 1-78(b). Both are “legacy” in the sense that they predate 802.11n.) The preamble uses short and long training symbols. This allows legacy receivers to

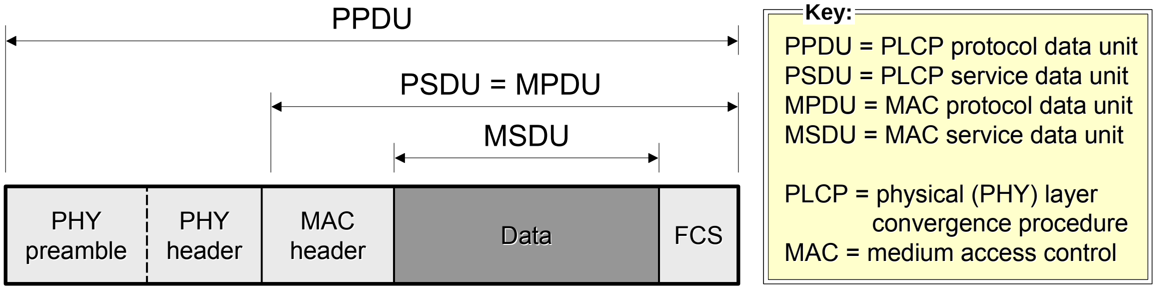

PPDU
PSDU = MPDU
MSDU
PHY preamble
PHY header
MAC header
Data
FCS
Key:
PPDU = PLCP protocol data unit
PSDU = PLCP service data unit
MPDU = MAC protocol data unit
MSDU = MAC service data unit
PLCP = physical (PHY) layer convergence procedure
MAC = medium access control

Figure 6-9: Terminology review for the 802.11 frame structure. Compare to Figure 1-78.

detect the transmission, acquire the carrier frequency, and synchronize timing. The physical-layer header contains the legacy Signal field (L-SIG) which indicates the transmission data rate (Rate subfield, in Mbps) and the payload length of the physical-layer frame (Length subfield, in bytes in the range 1–4095), which is a MAC-layer frame.

The **HT-mixed format** (middle row in Figure 6-8) starts with a preamble compatible with the legacy 802.11a/g. The legacy Short Training Field (L-STF), the legacy Long Training Field (L-LTF) and the legacy Signal field (L-SIG) can be decoded by legacy 802.11a/g devices. The rest of the HT-Mixed frame has a new format, and cannot be decoded by legacy 802.11a/g devices.

The HT preambles are defined in HT-mixed format and in HT-greenfield format to carry the information required to operate in a system with multiple transmit and multiple receive antennas. The HT-SIG field contains the information about the modulation scheme used, channel, bandwidth, length of payload, coding details, number of HT training sequences (HT-LTFs), and tail bits for the encoder. The number of HT-LTFs is decided by the antenna configuration and use of space-time block codes. HT training sequences are used by the receiver for estimating various parameters of the wireless MIMO channel.

The **HT-greenfield format** (bottom row in Figure 6-8) is completely new, without any legacy-compatible part. The preamble transmission time is reduced as compared to the mixed format. Support for the HT Greenfield format is optional and the HT devices can transmit using both 20-MHz and 40-MHz channels.

When an 802.11n access point is configured to operate in Mixed Mode (for example, 802.11b/g/n mode), the access point sends and receives frames based on the type of a client device. By default, the access point always selects the optimum rate for communicating with the client based on wireless channel conditions.

MAC Layer Enhancement: Frame Aggregation

First, the reader may find it useful to review Figure 6-9 for the terminology that will be used in the rest of this section. Figure 6-10 shows the 802.11n MAC frame format. Compared to the legacy 802.11 (Figure 1-78(a)), the change comprises the insertion of the High Throughput (HT) Control field and the change in the length of the frame body. The maximum length of the frame body is 7955 bytes (or, octets) and the overall 802.11n frame length is 8 Kbytes.

Every frame transmitted by an 802.11 device has a significant amount of fixed overhead, including physical layer header, MAC header, interframe spaces, and acknowledgment of transmitted frames (Figure 6-12(a)). (The reader should also check Figure 2-27 and the discussion

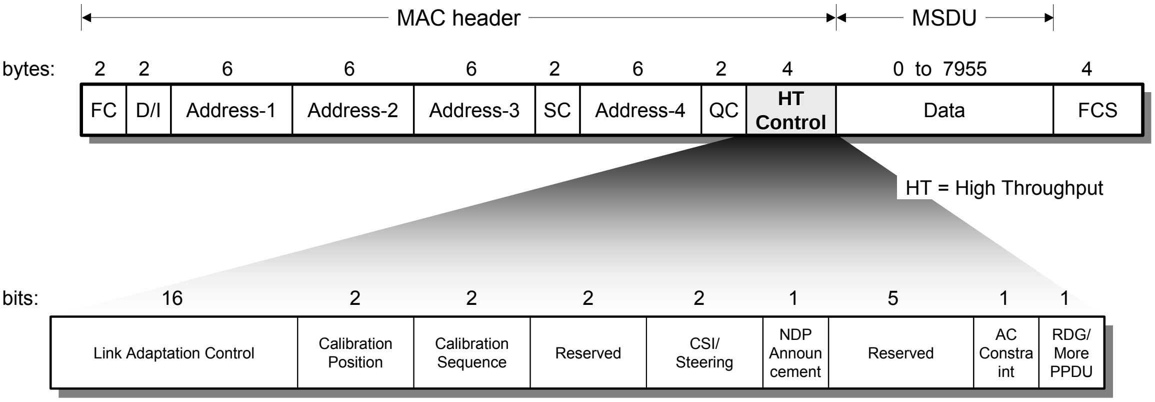


Figure 6-10: 802.11n link-layer frame format. Compare to Figure 1-78(a).

in Section 2.5) At the highest of data rates, this overhead alone can be longer than the entire data frame. In addition, contention for the channel and collisions also reduce the maximum effective throughput of 802.11. 802.11n addresses these issues by making changes in the MAC layer to improve on the inefficiencies imposed by this fixed overhead and by contention losses.

To reduce the link-layer overhead, 802.11n employs the mechanism known as *packet aggregation*, which is the process of joining multiple packets together into a single transmission unit, in order to reduce the overhead associated with each transmission. It is equivalent to a group of people riding a bus, rather than each individually riding a personal automobile (Figure 6-11). Generally, packet aggregation is useful in situations where each transmission unit may have significant overhead (preambles, headers, CRC, etc.) or where the expected packet size is small compared to the maximum amount of information that can be transmitted. Because at link layer packets are called frames, the mechanism is correspondingly called “frame aggregation.”

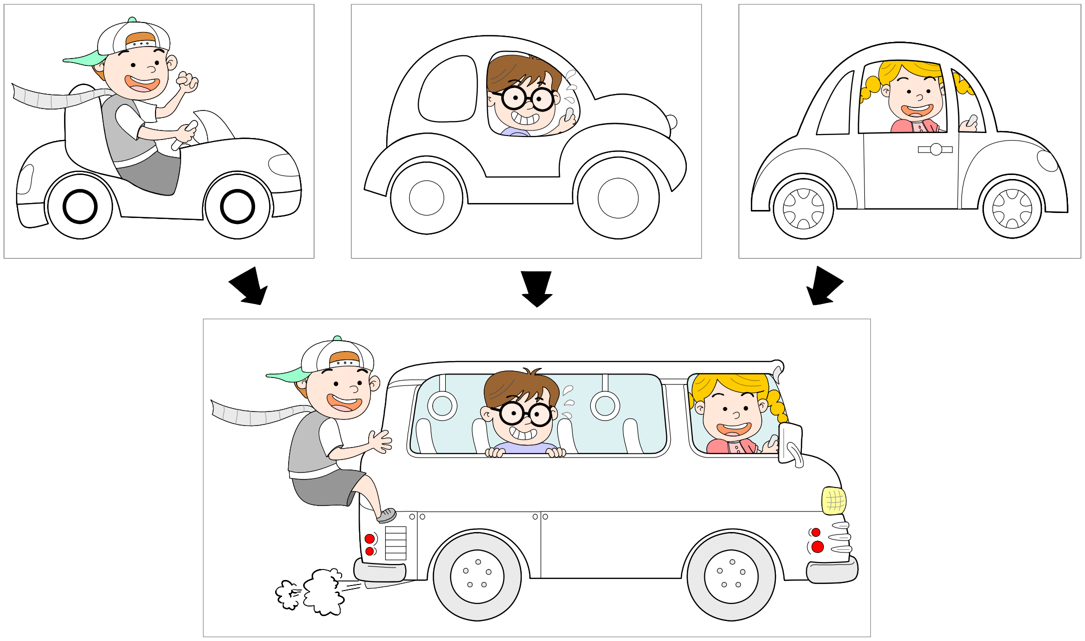


Figure 6-11: Packet aggregation analogy.

Frame aggregation is essentially putting the payloads of two or more frames together into a single transmission. Frame aggregation is a feature of the IEEE 802.11e and 802.11n standards that increases throughput by sending two or more data frames in a single transmission (Figure 6-12(b)). Because control information needs to be specified only once per frame, the ratio of payload data to the total volume of data is higher, allowing higher throughput. In addition, the reduced number of frame transmissions significantly reduces the waiting time during the CSMA/CA backoff procedure as well as the number of potential collisions. The maximum frame size is also increased in 802.11n, to accommodate these large, aggregated frames. The maximum frame size is increased from 4 KB to 64 KB. (64 KB frame size is achieved by sending multiple 8 KB frames in a burst, as explained later.)

There are several limitations of frame aggregation. First, all the frames that are aggregated into a transmission must be sent to the same destination; that is, all the frames in the aggregated frame must be addressed to the same mobile client or access point. Second, all the frames to be aggregated have to be ready for transmission at the same time, potentially delaying some frames while waiting for additional frames, in order to attempt to send a larger aggregate frame. Third, the maximum frame size that can be successfully sent is affected by a factor called *channel coherence time*. The time for frame transmission must be shorter than the channel coherence time. Channel coherence time depends on how quickly the transmitter, receiver, and other objects in the environment are moving. When the things are moving faster, the channel data rate is reduced, and therefore the allowed maximum frame size becomes smaller.

Although frame aggregation can increase the throughput at the MAC layer under ideal channel conditions, a larger aggregated frame will cause each station to wait longer before its next chance for channel access. Thus, there is a tradeoff between throughput and delay (or, latency) for frame aggregation at the MAC layer (as throughput increases, latency increase as well). Furthermore,

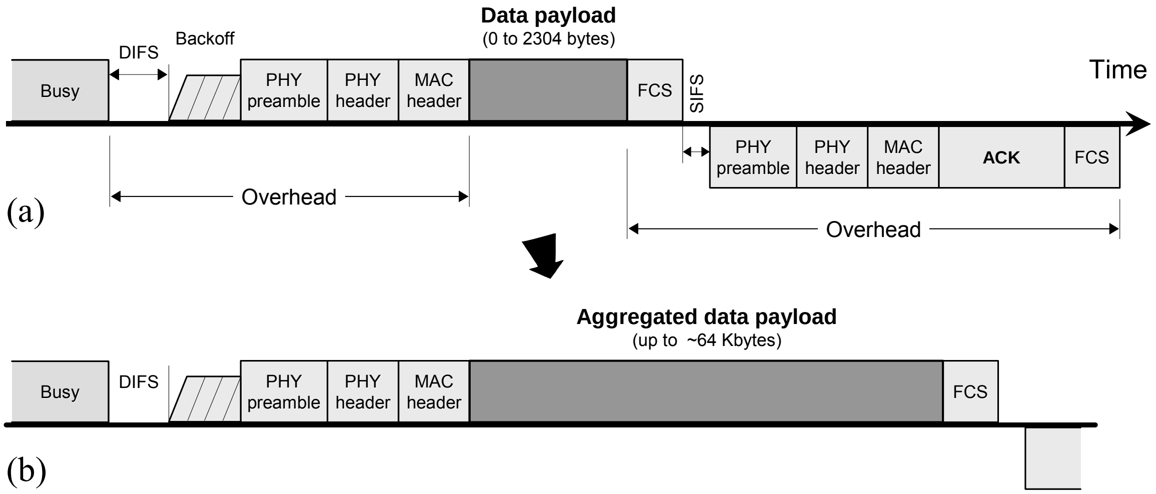


Figure 6-12: (a) Link-layer overhead in legacy IEEE 802.11. (b) 802.11n frame aggregation.

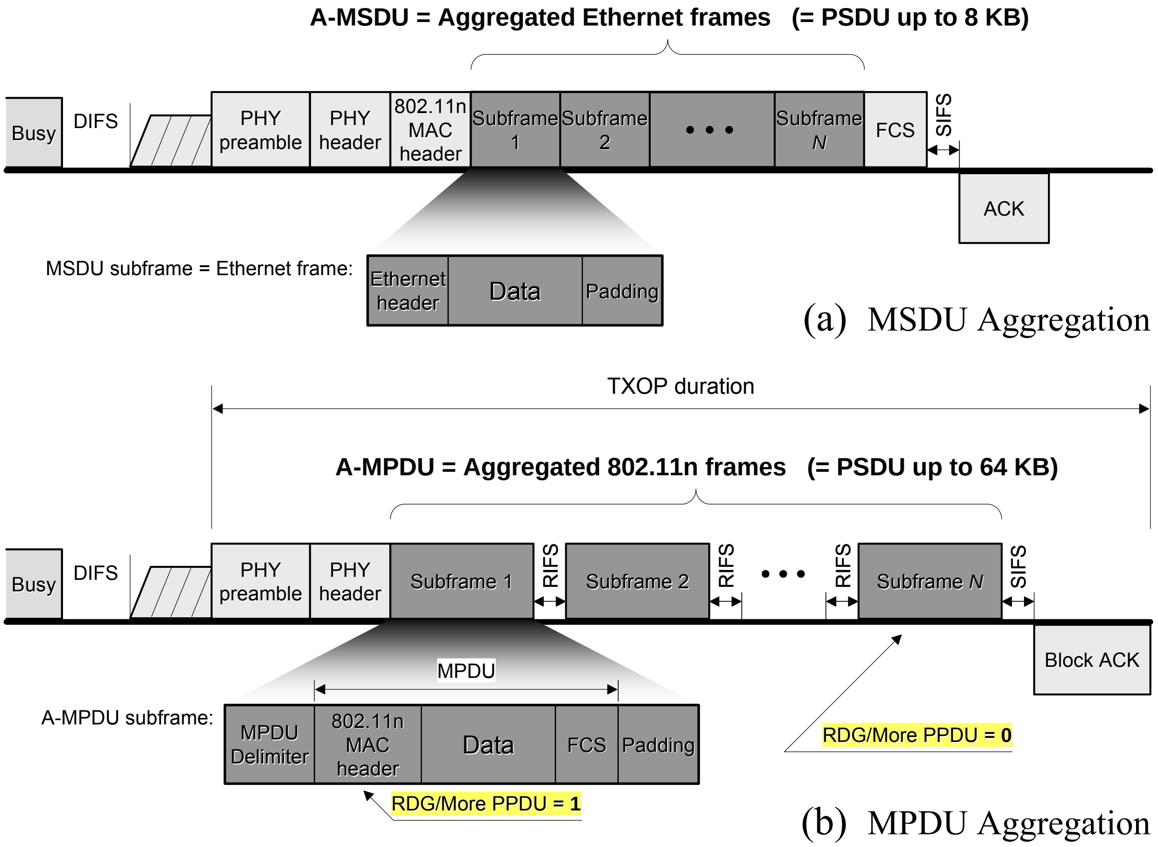
under error-prone channels, corrupting a large aggregated frame may waste a long period of channel time and lead to a lower MAC efficiency.

The ability to send multiple frames without entering the backoff procedure and re-contending for the channel first appeared in the IEEE 802.11e MAC. This mechanism reduces the contention and backoff overhead and thus enhances the efficiency of channel utilization. The notion of **transmit opportunity (TXOP)** is used to specify duration of channel occupation. During TXOP period of time, the station that won channel access can transmit multiple consecutive data frames without re-contending for the channel. If the station determines that it allocated too long TXOP and currently does not have more data to transmit, it may explicitly signal an early completion of its TXOP. This action, known as **truncation of TXOP**, prevents waste by allowing other stations to use the channel. Until the NAV has expired, even if the transmitting station has no data to send and the channel is sensed as idle, other stations do not access the medium for the remaining TXOP. The TXOP holder performs truncation of TXOP by transmitting a **CF-End** (Contention-Free-End) frame, if the remaining TXOP duration is long enough to transmit this frame. CF-End frame indicates that the medium is available. Stations that receive the CF-End frame reset their NAV and can start contending for the medium without further delay.

The frame aggregation can be performed within different sub-layers of the link layer. The 802.11n standard defines two types of frame aggregation: *MAC Service Data Unit (MSDU)* aggregation and *MAC Protocol Data Unit (MPDU)* aggregation. Both aggregation methods group several data frames into one large frame and reduce the overhead to only a single radio preamble for each frame transmission (Figure 6-12(b)). However, there are slight differences in the two aggregation methods that result in differences in the efficiency gained (MSDU aggregation is more efficient). These two methods are described here.

- **MAC Service Data Units (MSDUs) Aggregation**

MSDU aggregation exploits the fact that most mobile access points and most mobile client protocol stacks use Ethernet as their “native” frame format (Figure 1-66). It collects Ethernet frames to be transmitted to a single destination and wraps them in a single 802.11n frame. This is efficient because Ethernet headers are much shorter than 802.11 headers (compare Figure 1-66 and Figure 1-78). For this reason, MSDU aggregation is more efficient than MPDU aggregation.



(a) MSDU Aggregation

(b) MPDU Aggregation

Figure 6-13: 802.11n Frame aggregation methods: (a) MAC Service Data Unit aggregation (A-MSDU); (b) MAC Protocol Data Unit aggregation (A-MPDU).

MSDU aggregation allows several MAC-level service data units (MSDUs) to be concatenated into a single Aggregated MSDU (A-MSDU). Figure 6-13(a) shows the frame format for A-MSDU. In MSDU aggregation, the aggregated payload frames share not just the same physical (PHY) layer header, but also the same 802.11n MAC header. The resulting 802.11n frames can be up to 8 Kbytes in size.

When the source is a mobile device, the aggregated frame is sent to the access point, where the constituent Ethernet frames are forwarded to their ultimate destinations. When the source is an access point, all of the constituent frames in the aggregated frame must be destined to a single mobile client, because there is only a single destination in each mobile client.

With MSDU aggregation, the entire, aggregated frame is encrypted once using the security association of the destination of the outer 802.11n frame wrapper. A restriction of MSDU aggregation is that all of the constituent frames must be of the same quality-of-service (QoS) level. For example, it is not permitted to mix voice frames with best-effort frames.

If no acknowledgement is received, the whole 802.11n frame must be retransmitted. That is, an A-MSDU aggregate fails as a whole even if just one of the enclosed MSDUs contains bit errors.

- **MAC Protocol Data Units (MPDUs) Aggregation**

MPDU aggregation also collects Ethernet frames to be transmitted to a single receiver, but it converts them into 802.11n frames. Normally this is less efficient than MSDU aggregation, but it may be more efficient in environments with high error rates, because of a mechanism called block acknowledgement (described later). This mechanism allows each of the aggregated data frames to be individually acknowledged or retransmitted if affected by an error.

MPDU aggregation scheme enables aggregation of several MAC-level protocol data units (MPDUs) into a single PHY-layer protocol data unit (PPDU). Figure 6-13(b) shows the frame format for an Aggregated MPDU (A-MPDU). A-MPDU consists of a number of MPDU delimiters each followed by an MPDU. Except when it is the last A-MPDU subframe in an A-MPDU, padding bytes are appended to make each A-MPDU subframe a multiple of 4 bytes in length, which can facilitate subframe delineation at the receiver. A-MPDU allows bursting 802.11n frames up to 64 KB.

The purpose of the *MPDU delimiter* (4 bytes long) is to locate the MPDU subframes within the A-MPDU such that the structure of the A-MPDU can usually be recovered when one or more MPDU delimiters are received with errors. Subframes are sent as a burst (not a single unbroken transmission). The subframes are separated on the air from one other by the *Reduced Inter-Frame Space* (RIFS) interval of 2 μs duration (compared to SIFS interval which is 16 μs).[18] Figure 6-13(b) also indicates that the sender uses the "RDG/More PPDU" bit of the HT Control field in the MAC frame (Figure 6-10) to inform the receiver whether there are more subframes in the current burst. If the "RDG/More PPDU" field is set to "1," there will be one or more subframes to follow the current subframe; otherwise, the bit value "0" indicates that this is the last subframe of the burst.

Subframes of an A-MPDUs burst can be acknowledged individually with a single Block-Acknowledgement (described in the next subsection). The MPDU structure can be recovered even if one or more MPDU delimiters are received with errors. Unlike A-MSDU where the whole aggregate needs to be retransmitted, only unacknowledged MPDU subframes need to be retransmitted.

Summary of the characteristics for the two frame aggregation methods:

• MSDU aggregation is more efficient than MPDU aggregation, because the Ethernet header is much shorter than the 802.11 header.

• MPDU structure can be recovered even if one or more MPDU subframes are received with errors; conversely, an MSDU aggregate fails as a whole—even if just one of the enclosed MSDUs contains bit errors the whole A-MSDU must be retransmitted.

• A-MPDU is performed in the software whereas A-MSDU is performed in the hardware.

[18] RIFS is a means of reducing overhead and thereby increasing network efficiency. A transmitter can use RIFS after a transmission when it does not expect to receive immediately any frames, which is the case here. Note that RIFS intervals can only be used within a Greenfield HT network, with HT devices only and no legacy devices.

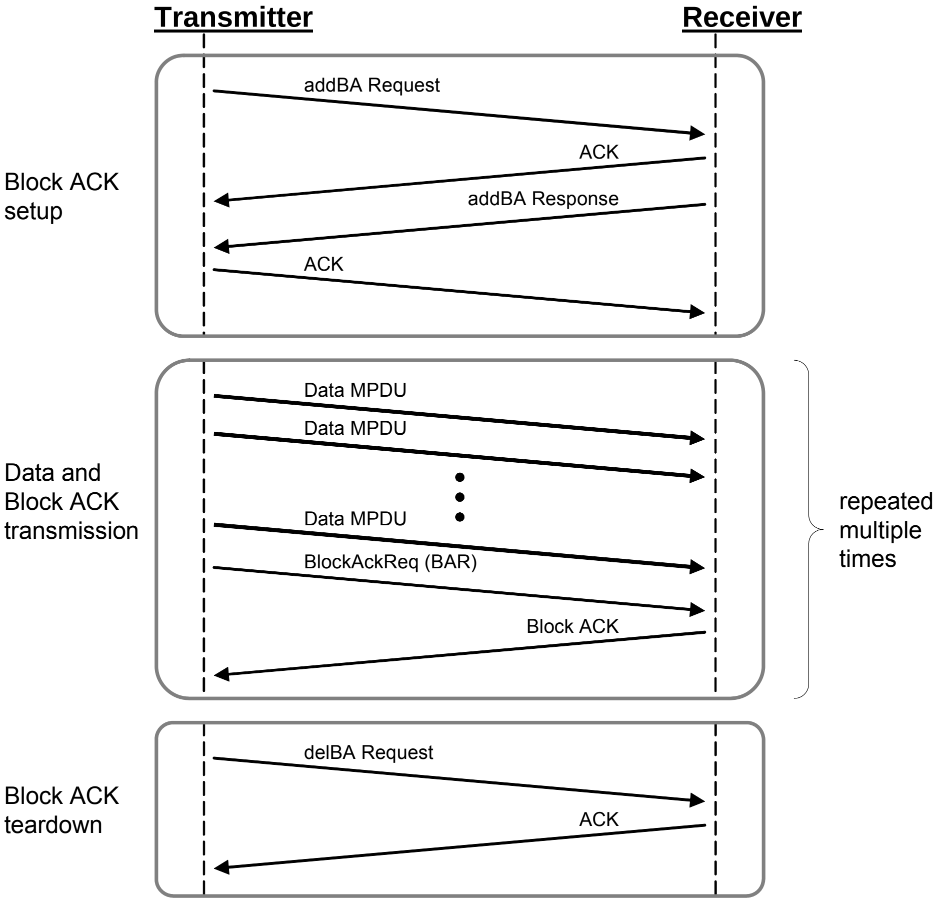


Figure 6-14: Initiation, use, and termination of 802.11n block acknowledgements.

MAC Layer Enhancement: Block Acknowledgement

Rather than sending an individual acknowledgement following each data frame, 802.11n introduces the technique of confirming a burst of up to 64 frames with a single **block acknowledgement** (Block ACK or BACK) frame. The Block ACK mechanism significantly reduces overhead due to bursts of small frames. Block acknowledgment was initially defined in IEEE 802.11e as an optional scheme to improve the MAC efficiency. The 802.11n standard made the Block ACK mechanism mandatory to support by all the HT devices. The Block ACK contains a bitmap to acknowledge selectively individual frames of a burst. This feature is comparable to selective acknowledgements of TCP, known as TCP SACK (Chapter 2).

Figure 6-14 shows how the Block ACK capability is activated, used, and deactivated by sending action frames. *Action frames* are used to request a station to take action on behalf of another. To initiate a Block ACK session, the transmitter sends an Add-Block-Acknowledgment request (addBA, also written as ADDBA). The addBA request indicates a starting frame sequence number and a window size of frame sequence numbers that the receiver should expect as part of the transmission. The receiver can choose to accept or reject the request and informs the transmitter by an addBA response frame. If the receiver rejects the addBA request, the session will continue with the legacy sequential transmit/acknowledgment exchanges. If the receiver

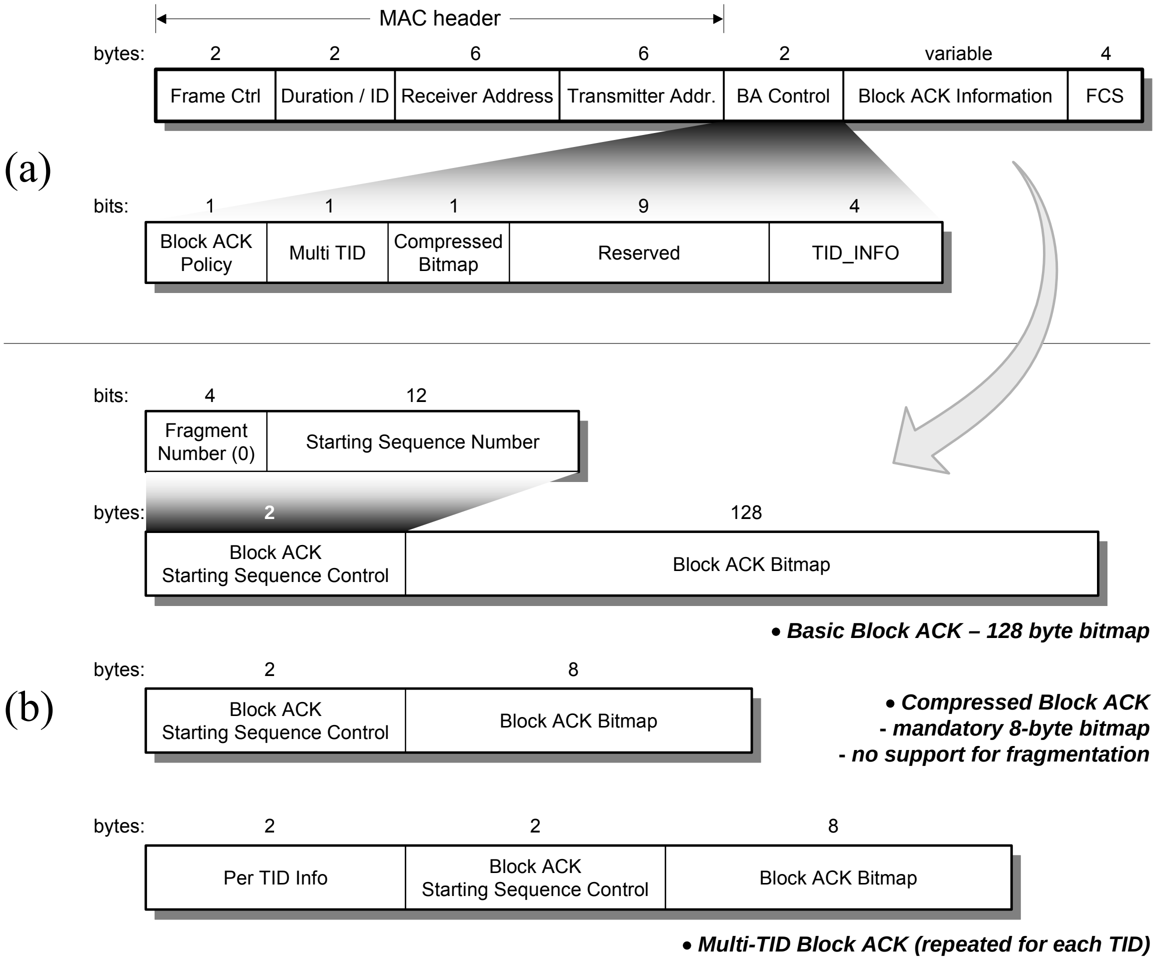


Figure 6-15: (a) 802.11n block acknowledgement frame format. (b) Format of the Block ACK Information field for the three variants of the Block ACK frame.

accepts the addBA request, the transmitter can send multiple frames without waiting for ACK frames. The receiver silently accepts frames that have sequence numbers within the current window. Only after the transmitter solicits a Block ACK by sending a *Block ACK Request* (BlockAckReq or BAR), the receiver responds by a Block ACK response frame indicating the sequence numbers successfully received. Frames that are received outside of the current window are dropped. This cycle may be repeated many times. Finally, when the transmitter does not need Block ACKs any longer, it terminates the BA session by sending a Delete-Block-Acknowledgment request (delBA or DELBA).

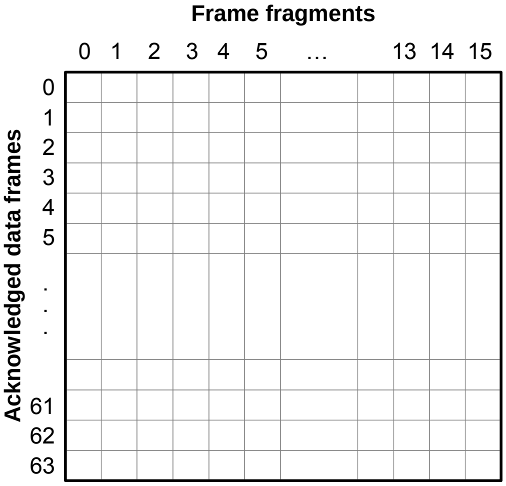
The Block ACK carries ACKs for individual frames as bitmaps. The exact format depends on the encoding. Figure 6-15(a) shows the format of Block ACK frames. The subfields of the Block ACK Control field are as follows:

- The Block ACK Policy bit specifies whether the sender requires acknowledgement immediately following BlockAckReq (bit value “0”), or acknowledgement can be delayed (bit value “1”).

- The values of the Multi-TID and Compressed Bitmap fields determine which of three possible Block ACK frame variants is represented (Figure 6-16(a)). The Block ACK frame variants are shown in Figure 6-15(b).

Multi-TID	Compressed Bitmap	Block ACK frame variant
0	0	Basic Block ACK
0	1	Compressed Block ACK
1	0	reserved
1	1	Multi-TID Block ACK

(a)



(b)

Figure 6-16: (a) 802.11n Block ACK frame variant encoding. (b) Block ACK Bitmap subfield (128 bytes long = 64×16 bits) of a Basic Block ACK frame variant. Each bit represents the received status (success/failure) of a frame fragment.

- The meaning of the TID_INFO subfield of the BA Control field depends on the Block ACK frame variant type. For the first two variants (Basic Block ACK and Compressed Block ACK), the TID_INFO subfield of the BA Control field contains the TID for which a Block ACK frame is requested. The *traffic identifier (TID)* is assigned by upper-layer protocols to inform the MAC protocol about the type of data that it is asked to transmit. This is important for MAC protocols that support quality of service (QoS), such as 802.11e and 802.11n. Therefore, the first two BA variants are capable of acknowledging only traffic of a single identifier type. A Block ACK frame could be extended to include the bitmaps for multiple TIDs. This extended Block ACK frame variant is called Multi-TID Block ACK (MTBA). More details are provided later.

Figure 6-15(b) shows the structure of the three variants of the Block ACK frame:

- **Basic Block ACK variant.** The Basic Block ACK variant is inherited from the IEEE 802.11e standard. The BA Information field within the Basic Block ACK frame contains the Block ACK Starting Sequence Control subfield and the Block ACK Bitmap, as shown in the top row of Figure 6-15(b). The Starting Sequence Number subfield (12-bit unsigned integer) of the Block ACK Starting Sequence Control field contains the sequence number of the first data frame (MPDU) that this Block ACK is acknowledging. This is the same number as in the previously received BlockAckReq frame to which this Block ACK is responding. When the transmitter receives a Block ACK, based on this number it knows to which BlockAckReq it corresponds. The Fragment Number subfield is always set to 0.

Before describing the BA Bitmap structure, it is necessary to mention the fragmentation mechanism in 802.11. The process of partitioning a MAC-level frame (MPDU) prior to transmission into smaller MAC-level frames is called **fragmentation**. Fragmentation creates smaller frames to increase reliability, by increasing the probability of successful transmission in cases where channel characteristics limit reception reliability for longer frames. Recall IP packet fragmentation (Section 1.4.1), which is done for different reasons, and where the fragments are reassembled only at the final destination. Conversely, defragmentation in 802.11 is accomplished at each immediate receiver. In the 802.11e and 802.11n standards, each MAC frame can be partitioned into up to 16 fragments.

The 128-byte long Block ACK Bitmap subfield represents the received status of up to 64 frames. In other words, the bitmap size is 64×16 bits (Figure 6-16(b)). That is, because each MAC-level frame can be partitioned into up to 16 fragments, 16 bits (2 bytes) are allocated to acknowledge each frame. Each bit of this bitmap represents the received status (success/failure) of a frame fragment. Two bytes are equally allocated even if the frame is not actually fragmented or is partitioned into less than 16 fragments. Suppose a frame has 11 fragments; then 11 bits are used, and remaining 5 bits are not used. Even so, this frame will consume 16 bits in the bitmap. If the frame is not fragmented, only one bit is used. Obviously, in cases with no fragmentation it is not efficient to acknowledge each frame using 2 bytes when all is needed is one bit. The overhead problem occurs also when the number of frames acknowledged by a Block ACK is small, because the bitmap size is fixed to 128 bytes. Thus, using two bytes per acknowledged frame in the bitmap results in an excessive overhead for Block ACK frames.

To overcome the potential overhead problem, 802.11n defines a modified Block ACK frame, called Compressed Block ACK.

- **Compressed Block ACK variant.** This Block ACK frame variant uses a reduced bitmap of 8 bytes, as shown in the middle row of Figure 6-15(b). Fragmentation is not allowed when the compressed Block ACK is used. Accordingly, a compressed Block ACK can acknowledge up to 64 non-fragmented frames. The bitmap size is reduced from 1024 (64×16) bits to 64 (64×1) bits.

The BA Information field within the Compressed Block ACK frame comprises the Block ACK Starting Sequence Control field and the Block ACK bitmap. The Starting Sequence Number subfield of the Block ACK Starting Sequence Control field is the sequence number of the first MSDU or A-MSDU for which this Block ACK is sent. The Fragment Number subfield of the Block ACK Starting Sequence Control field is set to 0.

The 8-byte Block ACK Bitmap within the Compressed Block ACK frame indicates the received status of up to 64 MSDUs and A-MSDUs. Each bit that is set to 1 acknowledges the successful reception of a single MSDU or A-MSDU in the order of sequence number, with the first bit of the bitmap corresponding to the MSDU or A-MSDU with the sequence number that matches the Starting Sequence Number field value.

Figure 6-17 shows an example using Compressed Block ACK frames. Here we assume that the transmitter sends aggregate A-MPDUs with 32 subframes. Bitmap bit position n is set to 1 to acknowledge the receipt of a frame with the sequence number equal to (Starting Sequence Control + n). Bitmap bit position n is set to 0 if a frame with the sequence number (Starting Sequence Control + n) has not been received. For unused fragment numbers of an aggregate frame, the corresponding bits in the bitmap are set to 0. For example, the Block ACK bitmap of the first Block ACK in Figure 6-17 contains [`7F FF FF FF 00 00 00 00`]. The first byte corresponds to the first 8 frames, but read right to left (that is why `7F` instead of `F7`). This means that, relative to the Starting Sequence Number 146, the first four frames and sixth to eight frames are successfully received. The fifth frame is lost (sequence number 150). The second byte corresponds to the second 8 frames, also read right to left, and so on. The last 32 bits are all zero because the A-MPDU contained 32 subframes. In the second transmission, the transmitter resends frame #150 and additional 32 frames (starting with the sequence number 179 up to #211).

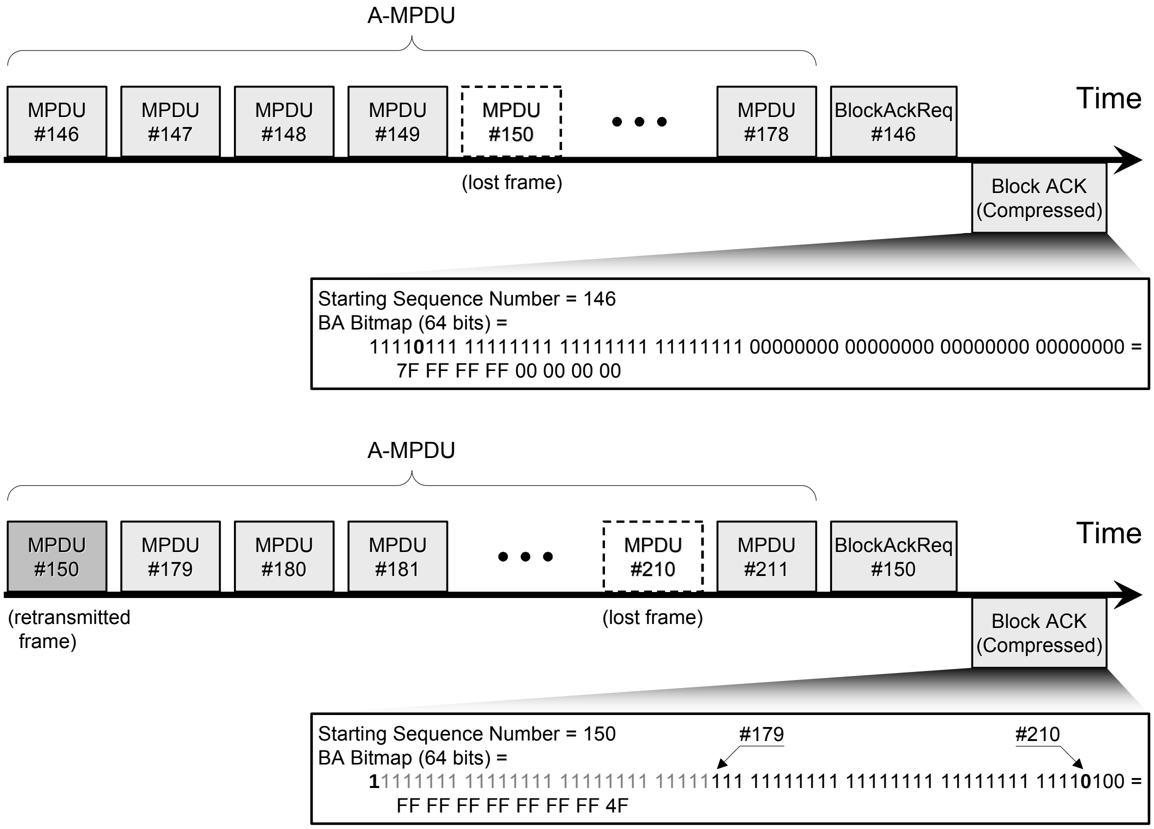


Figure 6-17: 802.11n Block ACK example using the Compressed Block ACK frame variant.

As seen, if a frame is not acknowledged, the sequence numbers can keep moving forward while the sending station keeps retrying that frame. However, when the span between the sequence number of the next frame to be sent and the retry frame becomes 64, the sending unit has to decide what to do. It can stop aggregating while it keeps retrying the old frame, or it can simply drop that frame.

- **Multi-TID Block ACK variant.** The TID_INFO subfield of the BA Control field of the Multi-TID Block ACK frame contains the number of traffic identifiers (TIDs), less one, for which information is reported in the BA Information field. For example, a value of 2 in the TID_INFO field means that information for 3 TIDs is present.

The BA Information field within the Multi-TID Block ACK frame contains one or more instances of the Per TID Info, Block ACK Starting Sequence Control field and the Block ACK Bitmap, as shown in the bottom row of Figure 6-15(b).

The Starting Sequence Number subfield of the Block ACK Starting Sequence Control field is the sequence number of the first MSDU or A-MSDU for which this Block ACK is sent. The first instance of the Per TID Info, Block ACK Starting Sequence Control and Block ACK Bitmap fields that is transmitted corresponds to the lowest TID value, with subsequent instances following ordered by increasing values of the Per TID Info field. The 8-byte Block ACK bitmap within the Multi-TID Block ACK frame functions the same way as for the Compressed Block ACK frame variant.

MAC Layer Enhancement: Reverse Direction (RD) Protocol

The 802.11n also specifies a *bidirectional data transfer* method, known as Reverse Direction (RD) protocol. Conventional transmit opportunity (TXOP) operation described above and already present in IEEE 802.11e allows efficient *unidirectional* transfer of data: the station holding the TXOP can transmit multiple consecutive data frames without reentering backoff procedure. The 802.11n RD protocol provides more efficient *bidirectional* transfer of data between two 802.11 devices during a TXOP by eliminating the need for either device to contend for the channel. This is achieved by piggybacking of data from the receiver on acknowledgements (ACK frame).

Reverse direction mechanism is useful in network services with bidirectional traffic, such as VoIP and online gaming. It allows the transmission of feedback information from the receiver and may enhance the performance of TCP, which requires bidirectional transmission (TCP data segments in one direction and TCP ACK segments in the other). (See Section 2.5 for more discussion.) The conventional TXOP operation only helps the forward direction transmission but not the reverse direction transmission. For application with bidirectional traffic, their performance degrades due to contention for the TXOP transmit opportunities. Reverse direction mechanism allows the holder of TXOP to allocate the unused TXOP time to its receivers to enhance the channel utilization and performance of reverse direction traffic flows.

Before the RD protocol, each unidirectional data transfer required the initiating station to contend for the channel. If RTS/CTS is used, the legacy transmission sequence of RTS (Request To Send) - CTS (Clear To Send) - DATA (Data frame) - ACK (Acknowledgement) allows the sender to transmit only a single data frame in forward direction (Figure 6-18(a)). In the bidirectional data transfer method (i.e., with the RD protocol), once the transmitting station has obtained a TXOP, it may essentially grant permission to the other station to send information back during its TXOP.

Reverse data transmission requires that two roles be defined: RD initiator and RD responder. *RD initiator* is the station that holds the TXOP and has the right to send Reverse Direction Grant (RDG) to the *RD responder*. The RD initiator sends its permission to the RD responder using a Reverse Direction Grant (RDG) in the "RDG/More PPDU" bit of the HT Control field in the MAC frame (Figure 6-10). The RD initiator grants permission to the RD responder by setting this bit to "1." When the RD responder receives the data frame with "RDG/More PPDU" bit set to "1," it decides whether it will send to the RD initiator more frames immediately following the one just received. It first sends an acknowledgement fro the received frame in which the "RDG/More PPDU" bit is set to "1" if one or more data frames will follow the acknowledgement, or with the bit set to "0" otherwise. For the bidirectional data transfer, the reverse DATAr frame can contain a Block ACK to acknowledge the previous DATAf frame. The transmission sequence will then become RTS-CTS-DATAf-DATAr-ACK (Figure 6-18(b)).

If the "RDG/More PPDU" bit in the acknowledgement frame is set to "1," the RD initiator will wait for the transmission from the RD responder, which will start with SIFS or *Reduced Inter-Frame Space* (RIFS) interframe time after the RDG acknowledgement is sent. A transmitter can use RIFS after a transmission when it does not expect to receive immediately any frames, which is the case here. If there is still data to be sent from the RD responder, it can set "RDG/More PPDU" bit in the data frame header to "1" to notify the initiator. The RD initiator still has the right to accept or reject the request. To allocate the extended TXOP needed for additional reverse

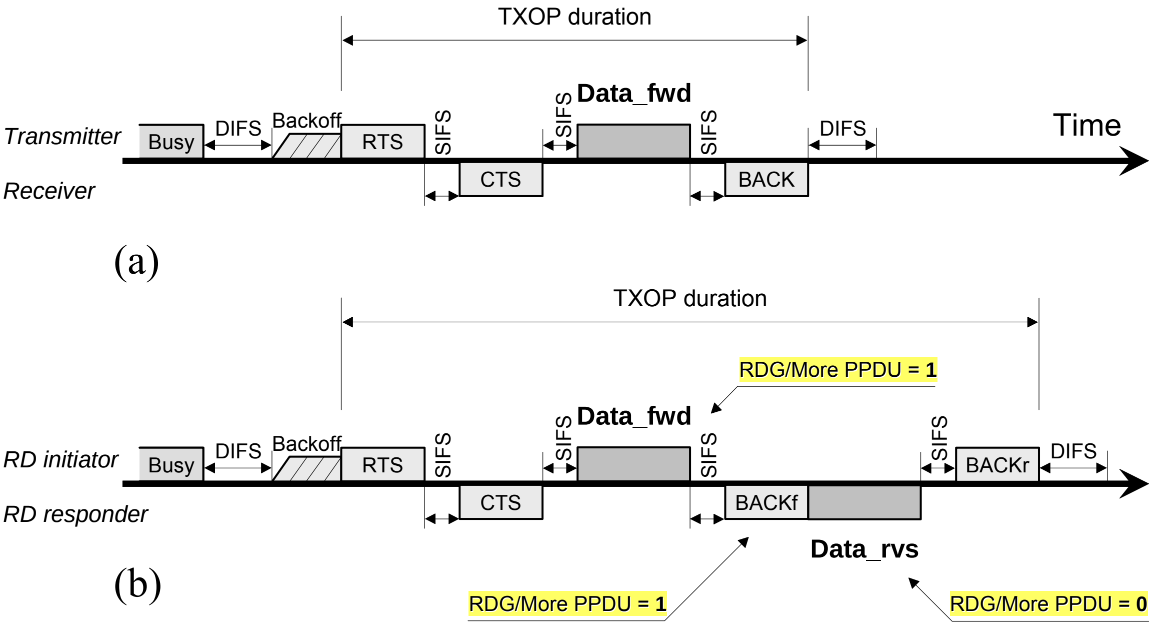


Figure 6-18: 802.11n Reverse Direction (RD) protocol. (a) Unidirectional RTS/CTS access scheme. (b) Bidirectional RTS/CTS access scheme. RD initiator invites RD responder to send reverse traffic by setting the RPG/MorePPDU flag to "1." RD responder sends zero or more frames and sets RPG/MorePPDU to "0" in the last frame of the "RD response burst."

frames, the initiator will set to "1" the "RDG/More PPDU" bit in the acknowledgement frame or the next data frame. To reject the new RDG request, the initiator sets "RDG/More PPDU" to "0."

Backward Compatibility

802.11n devices transmit a signal that cannot be decoded by devices built to an earlier standard. To avoid rendering the network useless due to massive interference and collisions, 802.11n devices must provide backwards compatibility. Compatibility with legacy 802.11 devices has been a critical issue continuously faced throughout the evolution of 802.11. For example, 802.11g provides a mechanism for operation with 802.11b devices. Similarly, 802.11n has a number of mechanisms to provide backward compatibility with 802.11 a, b, and g devices. These mechanisms ensure that the legacy stations are aware of 802.11n transmissions in the same area and do not interfere with them. The cost for achieving this protection is the throughput degradation for 802.11n.

Because the 802.11 MAC layer operation is based on carrier sense multiple access (CSMA/CA) protocol, it is essential for the station that won access to the channel to inform other stations how long it will transmit, to avoid being interrupted. The mechanism of announcing the duration of the transmission is called **protection mechanism**, and different options have emerged in the evolution of 802.11 wireless LANs. Before transmitting at a high data rate, the station must attempt to update the *network allocation vector* (NAV) of non-HT stations that do not support the high data rate, so that they can defer their transmission. (See Section 1.5.3 for the description of NAV.) The duration information has to be transmitted at physical data-rates that are decodable by the legacy stations (the pure 802.11n transmission is not).

Three different *operating modes* are defined for 802.11n devices (actually, four, but one is a kind of sub-mode and omitted here for simplicity). The legacy **Non-HT operating mode** sends data frames in the old 802.11a/g format (shown in the top row of Figure 6-8) so that legacy stations can understand them. However, only 802.11a and g stations understand Non-HT mode format—802.11b stations predate 802.11a/g and do not understand it. Non-HT mode is used by 802.11n devices only to communicate with legacy 802.11 devices, rather than with other 8021.11n devices. It cannot be used with 40-MHz channels (Figure 6-7). At the transmitter, only one transmitting antenna is used in Non-HT mode. Receive diversity is exploited in this mode. An 802.11n device using Non-HT delivers no better performance than 802.11a/g. This mode gives essentially no performance advantage over legacy networks, but offers full compatibility.

The legacy operating mode is a Non-HT (High Throughput) mode, whereas the Mixed and Greenfield modes are HT modes. In **Mixed operating mode**, frames are transmitted with a preamble compatible with the legacy 802.11a/g (middle row in Figure 6-8). The legacy Short Training Field (L-STF), the legacy Long Training Field (L-LTF) and the legacy Signal field (L-SIG) can be decoded by legacy 802.11a/g devices. The rest of the HT-Mixed frame has a new format, and cannot be decoded by legacy 802.11a/g devices.

In **Greenfield operating mode**, high throughput frames are transmitted without any legacy-compatible part (bottom row in Figure 6-8). In this mode, there is no provision to allow a legacy device to understand the frame transmission. Receivers enabled in this mode should be able to decode frames from the legacy mode, mixed mode, and the Greenfield mode transmitters. The preamble is not compatible with legacy 802.11a/g devices and only 802.11n devices can communicate when using the Greenfield format. Support for the Greenfield format is optional and the HT devices can transmit using both 20-MHz and 40-MHz channels.

When a Greenfield device is transmitting, the legacy systems may detect the transmission, and therefore avoid collision, by sensing the presence of a radio signal, using the carrier-sensing mechanism in the physical layer. However, legacy devices cannot decode any part of an HT Greenfield frame. Therefore, they cannot set their NAV and defer the transmission properly. They must rely on continuous physical-layer carrier sensing to detect the busy/idle states of the medium. In the worst case, HT Greenfield transmissions will appear as noise bursts to the legacy devices (and vice versa).

The HT Mixed mode is mandatory to support and transmissions can occur in both 20-MHz and 40-MHz channels. Support for the HT Greenfield mode is optional; again, transmissions can occur in both 20-MHz and 40-MHz channels (Figure 6-7). Support for Non-HT Legacy mode is mandatory for 802.11n devices, and transmissions can occur only in 20-MHz channels.

An 802.11n access point (AP) starts in the Greenfield mode, assuming that all stations in the BSS (Basic Service Set) will be 802.11n capable. If the access point detects a legacy (non-HT) 802.11a/b/g device (at the time when it associates to the access point or from transmissions in an overlapping network), the access point switches to the mixed mode. 802.11n stations are communicating mutually using the mixed mode, and with legacy stations using the non-HT mode. When non-HT stations leave the BSS, the access point, after a preset time, will switch back from the Mixed mode to the Greenfield mode. The same is true of when the access point ceases to hear nearby non-HT stations; it will switch back to the Greenfield mode.

The following protection mechanisms (described later) are defined for 802.11n to work with legacy stations:

- Transmit control frames such as RTS/CTS or CTS-to-self using a legacy data rate, before the HT transmissions. For control frame transmissions, use 20-MHz non-HT frames or 40-MHz non-HT duplicate frames (Figure 6-7).
- L-SIG TXOP protection
- Transmit the first frame of a transmit opportunity (TXOP) period using the non-HT frame that requires a response frame (acknowledgement), which is also sent as a non-HT frame or non-HT duplicate frame. After this initial exchange, the remaining TXOP frames can be transmitted using HT-Greenfield format and can be separated by RIFS (Reduced Inter Frame Spacing).
- Using the HT-Mixed frame format, transmit a frame that requires a response. The remaining TXOP may contain HT-Greenfield frames and/or RIFS sequences.

The first two protection schemes are extension of the protection mechanisms that have been introduced in the migration from 802.11b to 802.11g. Use of control frames such as RTS/CTS or CTS-to-self is a *legacy compatibility mode*. L-SIG TXOP protection is a *mixed compatibility mode* (uses HT-mixed frame format) and is optional to implement. The last two schemes are applicable only in the presence of TXOP, which is a feature that might be enabled only for certain services, such as voice and video transmission.

In an 802.11n HT coverage cell that operates in 20/40-MHz channels, there may be legacy 802.11 devices (operating in the primary 20-MHz channel) along with the 40-MHz HT devices. Furthermore, there may be an overlapping cell with legacy 802.11 devices operating in the secondary channel of this cell. A protection mechanism must take into account both cases and provide protection for the 40-MHz HT devices against interference from either source (i.e., legacy devices inside or outside this cell). Next, we review those protection mechanisms.

- **Control Frames Protection: RTS/CTS Exchange, CTS-to-self, and Dual-CTS**

We have already seen in Section 1.5.3 how RTS/CTS (Request-to-Send/Clear-to-Send) exchange is used for protection of the current transmission. The RTS/CTS frames let nearby 802.11 devices—including those in different but physically overlapping networks—set their network allocation vector (NAV) and defer their transmission. This mechanism is called "virtual carrier sensing" because it operates at the MAC layer, unlike physical-layer carrier sensing. Transmission of RTS/CTS frames helps avoid hidden station problem irrespective of transmission rate and, hence, reduces the collision probability.

802.11g introduced another NAV-setting protection mechanism (also adopted in 802.11n), called **CTS-to-self mechanism**. CTS-to-self allows a device to transmit a short CTS frame, addressed to itself, that includes the NAV duration information for the neighboring legacy devices, which will protect the high-rate transmission that will follow. The advantage of the CTS-to-self NAV distribution mechanism is lower network overhead cost than with the RTS/CTS NAV distribution mechanism—instead of transmitting two frames separated by a SIFS interval, only one frame is transmitted. However, CTS-to-self is less robust against hidden nodes and collisions than RTS/CTS. Stations employing NAV distribution should choose a mechanism that is appropriate

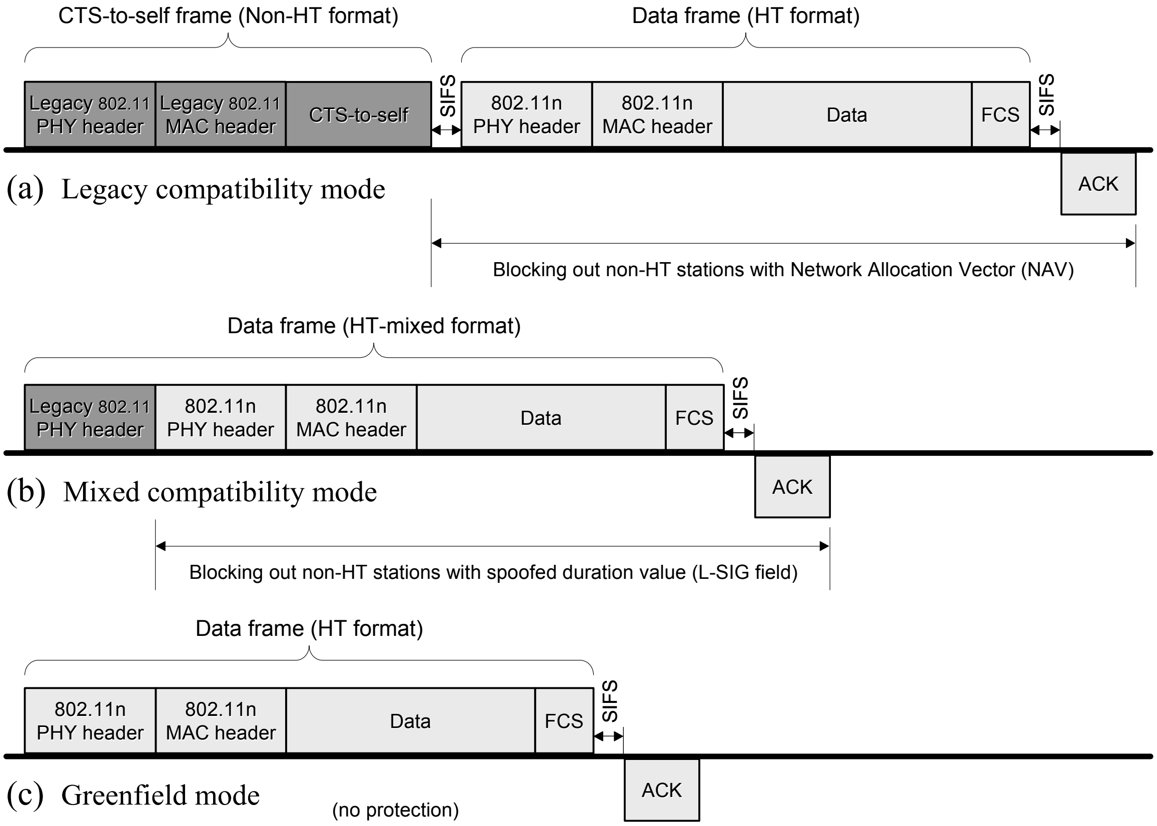


Figure 6-19: 802.11n backwards compatibility modes. (a) Using control frames for NAV distribution. (b) L-SIG TXOP protection. (c) Greenfield mode offers no protection.

for the given network conditions. If errors occur when employing the CTS-to-self mechanism, stations should switch to the more robust RTS/CTS mechanism.

HT protection requires 802.11n devices to announce their intent to transmit by sending legacy-format control frames prior to HT data transmission (Figure 6-19(a)). The CTS-to-self frame must be transmitted using one of the legacy data rates that a legacy device will be able to receive and decode. Transmission rate of the control frames depends on the type of legacy device that is associated in the BSS. If both 802.11b and 802.11g devices are associated, then 802.11b rates (known as Clause 15 rates) are used to transmit protection frames because 802.11g stations can decode such frames.

In **Dual-CTS protection mode**, the RTS receiver transmits two CTS frames, one in Non-HT mode and another in HT mode, so the subsequent data frame is protected by a legacy CTS and an HT CTS. The dual-CTS feature can be enabled or disabled by setting the Dual CTS Protection subfield in beacon frames. Dual-CTS protection has two benefits. First, using the legacy RTS/CTS or legacy CTS-to-self frames to reset NAV timers prevents interference with any nearby 802.11a/b/g cells. Second, it resolves the hidden node problem within the 802.11n cell.

Figure 6-20 shows an example network with an 802.11n access point (AP) and two mobile stations, one 802.11n (station *A*) and the other legacy 802.11g (station *B*). When traffic is generated by station *A*, it first sends an RTS to the AP. The AP responds with two CTS frames, one in HT and the other in legacy format. Station *A* is then free to transmit the data frame, while

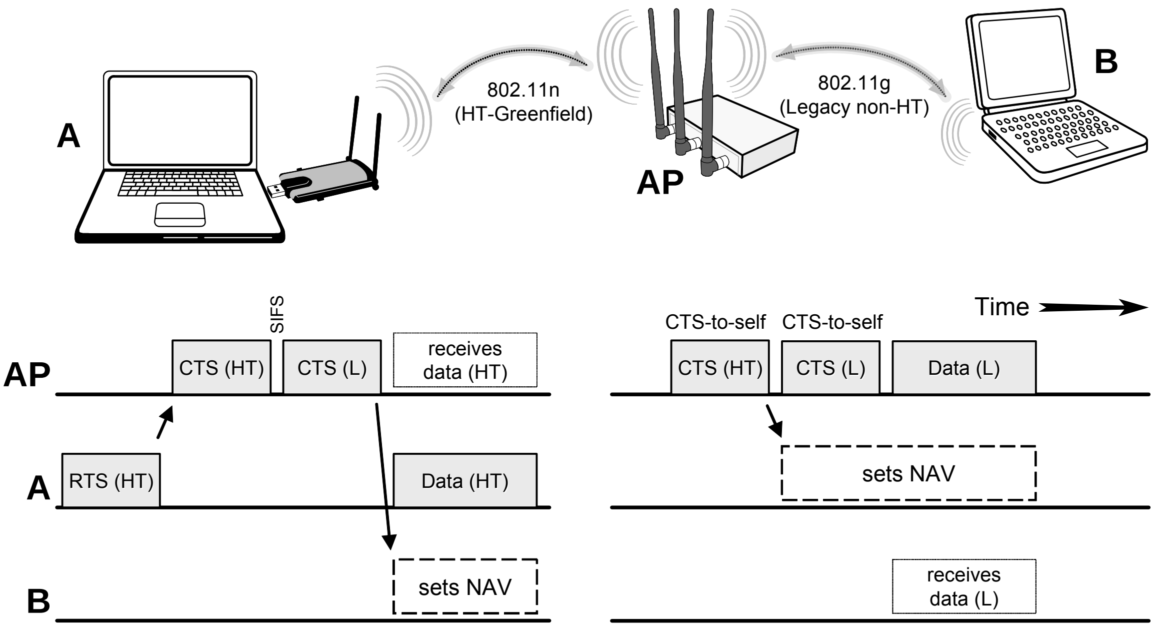


Figure 6-20: Example of 802.11n Dual-CTS protection (CTS-to-self).

other stations in the same and neighboring networks (e.g., station *B*) set their NAV correctly so they do not transmit over the authorized frame, interfering with it. Later, when the AP has traffic to send to station *B*, it uses dual CTS-to-self frames to perform the same function (Figure 6-20).

Dual-CTS makes the 802.11n network a good neighbor to overlapping or adjacent legacy 802.11 networks. It also solves the hidden-station problem where different clients in a cell may not be able to hear each other's transmissions, although, by definition they all can hear the AP and its CTS frames. However, the use of control frames further reduces the data throughput of the network. Although RTS/CTS frames are short (20 and 14 bytes, respectively), it takes more time to transmit them at the legacy rate of 6 Mbps than it takes to transmit 500 bytes of data at 600 Mbps. Therefore, HT protection significantly reduces an 802.11n W-LAN's overall throughput.

- **L-SIG TXOP Protection**

In *Legacy Signal field Transmit Opportunity* (L-SIG TXOP) protection mechanism, protection is achieved by transmitting the frame-duration information in a legacy-formatted physical header, and then transmitting the data at an 802.11n high rate (Figure 6-19(b)). Each frame is sent in an HT-mixed frame format. A legacy device that receives and successfully decodes an HT-mixed frame defers its own transmission based on the duration information present in the legacy Signal (L-SIG) field (see Figure 6-8). Such legacy clients remain silent for the duration of the forthcoming transmission. Following the legacy physical header, the 802.11n device sends the remaining part of the frame using 802.11n HT rates and its multiple spatial streams. L-SIG TXOP protection is also known as *PHY layer spoofing*.

The Rate and Length subfields of the L-SIG field (Figure 6-8) determine the duration of how long non-HT stations should defer their transmission:

L-SIG Duration = (legacy Length / legacy Rate)

This value should be equal to the duration of the remaining HT part of the HT-mixed format frame. The Rate parameter should be set to the value 6 Mbps. Non-HT stations are not able to

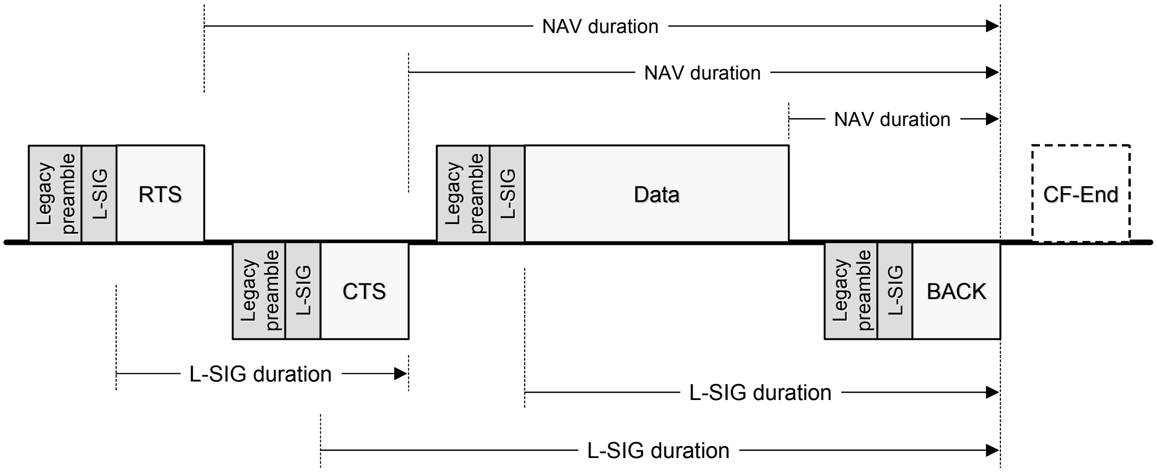


Figure 6-21: 802.11n L-SIG TXOP protection: Example of L-SIG duration setting.

receive any frame that starts throughout the L-SIG duration. Therefore, no frame may be transmitted to a non-HT station during an L-SIG protected TXOP.

Figure 6-21 illustrates an example of how L-SIG Durations are set when using L-SIG TXOP Protection. In this example, an L-SIG TXOP protected sequence starts with an RTS/CTS initial handshake, which provides additional protection from hidden stations. Any initial frame exchange may be used that is valid for the start of a TXOP. The term "L-SIG TXOP protected sequence" includes these initial frames and any subsequent frames transmitted within the protected L-SIG duration.

The TXOP holder should transmit a CF-End frame starting a SIFS period after the L-SIG TXOP protected period (Figure 6-21). Because legacy stations are unable to distinguish a Mixed-mode acknowledgement frame from other Mixed-mode frames, they may mistakenly infer that ACK frame is lost. As a result, they would wait unnecessarily until the EIFS time elapses (see Figure 1-82(b)), which leads to potential unfairness or a "capture effect." CF-End enables other stations to avoid such undesirable effects. Note that this is not an instance of TXOP truncation (described earlier), because here the CF-End frame is not transmitted to reset the NAV.

All HT-mixed mode frames contain the L-SIG field, so is not necessary to send special control frames to announce the medium reservation duration explicitly. An 802.11n station must indicate whether it supports L-SIG TXOP Protection in its L-SIG TXOP Protection Support capability field in Association-Request and Probe-Response frames. The mixed mode can be used in a 40-MHz channel, but to make it compatible with legacy clients, all broadcast and non-aggregated control frames are sent on a legacy 20-MHz channel as defined in 802.11a/b/g, to be interoperable with those clients (Figure 6-7). And, of course, all transmissions to and from legacy clients must be within a legacy 20-MHz channel. L-SIG TXOP protection mechanism is not applicable when 802.11b stations are present, because the Signal field (L-SIG) is encoded in 802.11g frame format that 802.11b devices do not understand. The double physical-layer header (legacy plus 802.11n headers) adds overhead, reducing the throughput. However, it makes possible for 802.11n stations to take advantage of HT features for the remaining part of the frame transmission.

- **Phased Coexistence Operation (PCO)**

Another mechanism for coexistence between 802.11n HT cells and nearby legacy 802.11a/b/g cells is known as *Phased Coexistence Operation (PCO)*. This is an optional mode of operation that divides time into slices and alternates between 20-MHz and 40-MHz transmissions. The HT access point designates time slices for 20-MHz transmissions in both primary and secondary 20-MHz channels, and designates time slices for 40-MHz transmissions. This operation is depicted in Figure 6-7 and now we describe the mechanism for transitioning between the phases. The algorithm for deciding when to switch the phase is beyond the scope of the 802.11n standard.

The phased coexistence operation (PCO) of 802.11n is illustrated in Figure 6-22, where an 802.11n coverage cell (BSS-1) is overlapping a legacy 802.11 cell (BSS-2). Stations *A* and *B* are associated with BSS-1 and station *C* is associated with BSS-2, but it can hear stations in BSS-1 and interfere with their transmissions. Only station *A* is capable of transmitting and receiving frames in the 40-MHz channel. As explained earlier (Figure 6-7), a 40-MHz channel is formed by bonding two contiguous 20-MHz channels, one designated as *primary channel* and the other as *secondary channel*. In this example, BSS-2 happens to operate in what BSS-1 considers its own secondary channel, i.e., the secondary channel of BSS-1 is the primary channel for BSS-2. In 20-MHz phase, all stations contend for medium access in their respective primary channels. When the 802.11n access point wishes to use a 40-MHz channel, it needs to reserve explicitly both adjacent 20-MHz channels. The access point is coordinating the phased operation of the associated stations with 20-MHz and 40-MHz bandwidth usage.

The bottom part of Figure 6-22 shows the MAC-protocol timing diagram for reservation and usage of the 40-MHz channel. Transitions back and forth between 20-MHz and 40-MHz channels start with the Beacon frame or Set-PCO-Phase frame. The 802.11n access point (AP) accomplishes the reservation by setting the NAV timers of all stations with appropriate control frames transmitted on the respective channels. The access point uses CTS-to-self frames to set the NAV timers. As Figure 6-22 depicts, the AP transmits both CTS-to-self frames simultaneously using the *duplicate legacy mode* (described earlier in this section). Although control frames are transmitted only on the primary channel, the secondary channel of BSS-1 is the primary channel of BSS-2, so station *C* will receive the second CTS-to-self and react to it. This feature improves efficiency (but note that it could not be exploited in Figure 6-20). When the NAV timer is set, the station is blocked from transmission until the NAV timer expires. However, as seen in Figure 6-22 station *A* will also set its own NAV, which means that station *A* too will be blocked. This is why the AP will transmit a CF-End frame in the HT-Greenfield format on the 40-MHz channel, so that only station *A* will decode it and start contending for access to the 40-MHz channel. Recall that CF-End truncates TXOP and clears the NAV timers of the clients that receive this frame.

To end the 40-MHz phase, the HT access point first sends a Set-PCO-Phase frame so station *A* knows the 40-MHz phase is over. Next, to release the 40-MHz channel, the AP uses two CF-End frames sent simultaneously on both 20-MHz channels using the duplicate legacy mode. This will truncate the remaining TXOP for the legacy clients (stations *B* and *C*). Thereafter, all stations may again contend for medium access on their respective 20-MHz primary channels.

Reservation of the 40-MHz channel may not happen smoothly, because traffic in BSS-2 may continue for a long time after the access point transmitted the Beacon frame or Set-PCO-Phase frame (see the initial part of the timing diagram in Figure 6-22). If the secondary channel continues to be busy after the phase transition started, the stations in BSS-1 are not allowed to transmit on the primary 20-MHz channel because their NAV timers are set. If waiting for

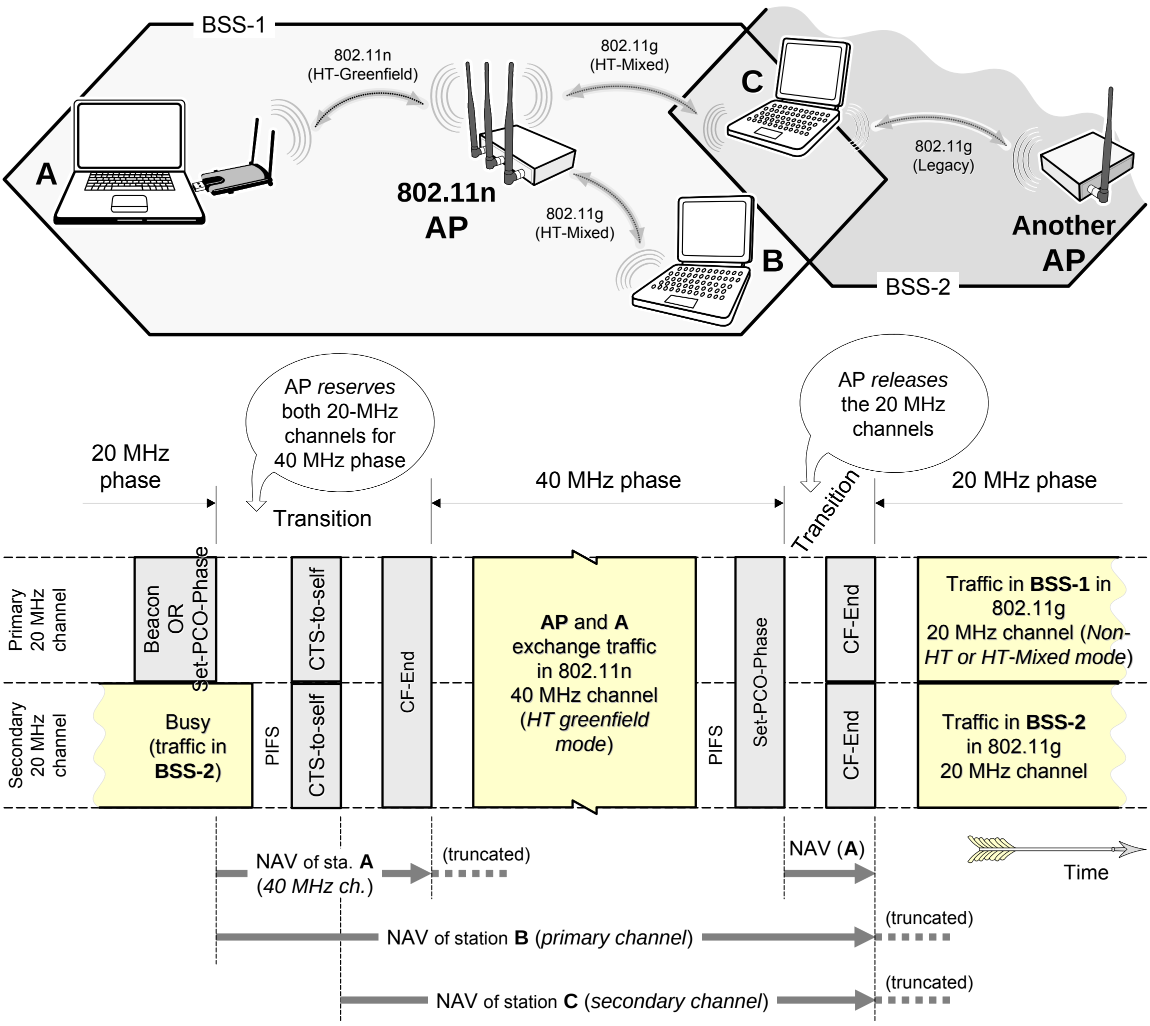

BSS-1
802.11n
(HT-Greenfield)
802.11g
(HT-Mixed)
C
A
802.11n
AP
802.11g
(HT-Mixed)
B
802.11g
(Legacy)
Another
AP
BSS-2
AP *reserves* both 20-MHz channels for 40 MHz phase
AP *releases* the 20 MHz channels
20 MHz phase
Transition
40 MHz phase
Transition
20 MHz phase
Primary 20 MHz channel
Secondary 20 MHz channel
Beacon OR Set-PCO-Phase
Busy (traffic in **BSS-2**)
PIFS
CTS-to-self
CTS-to-self
CF-End
AP and A exchange traffic in 802.11n 40 MHz channel (*HT greenfield mode*)
PIFS
Set-PCO-Phase
CF-End
CF-End
Traffic in **BSS-1** in 802.11g 20 MHz channel (*Non-HT or HT-Mixed mode*)
Traffic in **BSS-2** in 802.11g 20 MHz channel
NAV of sta. **A** (*40 MHz ch.*)
(truncated)
NAV (**A**)
Time
NAV of station **B** (*primary channel*)
(truncated)
NAV of station **C** (*secondary channel*)
(truncated)

Figure 6-22: Example timing diagram for the (optional) 802.11n phased coexistence operation (PCO).

reservation of the secondary 20-MHz channel exceeds a given threshold, the access point may decide to terminate the transition process and go back to the 20-MHz phase.

Phased coexistence operation (PCO) makes an 802.11n access point more tolerant of nearby legacy APs operating on overlapping channels and might improve throughput in some situations. However, once again, this option reduces throughput due to transmission of many control frames (CTS-to-self and CF-End). In addition, switching back and forth between channels could potentially increase delay jitter for data frames, and therefore PCO mode would not be recommended for real-time multimedia traffic.

The cost of backwards compatibility features is additional overhead on every 802.11n transmission. This reduces the benefits of all the 802.11n improvements, resulting in significantly lower effective throughput by 802.11n devices in mixed environments. The HT-Mixed format will most likely be the most commonly used frame format because it supports both HT and legacy 802.11a/g devices. The HT-Mixed format is also considered mandatory and transmissions can occur in both 20-MHz and 40-MHz channels. It can be expected that protection mechanisms will be in use in the 2.4-GHz band (802.11b and 802.11g) until nearly every legacy device has disappeared. This is because there are too few channels available in that band to effectively overlay pure 802.11n wireless LANs in the same areas as legacy 2.4-GHz W-LANs. Given the larger number of channels available in the 5-GHz band in many countries, it is possible that two completely separate W-LANs could be operating in the same area in the 5-GHz band, with 802.11a operating on one set of channels and 802.11n operating on a different, nonintersecting set of channels. This would allow 802.11n to operate in pure high-throughput mode (HT-Greenfield mode), achieving the highest effective throughput offered by the 802.11n standard.

6.3.2 WiMAX (IEEE 802.16)

The IEEE 802.16 standard is also known as WiMAX, which stands for Worldwide Interoperability for Microwave Access. WiMAX, as approved in 2001, addressed frequencies from 10 to 66 GHz, where extensive spectrum is available worldwide but at which the short wavelengths introduce significant deployment challenges. A new effort will extend the air interface support to lower frequencies in the 2–11 GHz band, including both licensed and license-exempt spectra. Compared to the higher frequencies, such spectra offer the opportunity to reach many more customers less expensively, although at generally lower data rates. This suggests that such services will be oriented toward individual homes or small to medium-sized enterprises.

Medium Access Control (MAC) Protocol

The IEEE 802.16 MAC protocol was designed for point-to-multipoint broadband wireless access applications. It addresses the need for very high bit rates, both uplink (to the Base Station) and downlink (from the BS). Access and bandwidth allocation algorithms must accommodate hundreds of terminals per channel, with terminals that may be shared by multiple end users.

6.3.3 ZigBee (IEEE 802.15.4)

Problems related to this section: ?? → ??

ZigBee is a specification for a suite of high level communication protocols using small, low-power digital radios based on the IEEE 802.15.4-2003 standard for wireless personal area networks (WPANs), such as wireless headphones connecting with cell phones via short-range radio. The technology defined by the ZigBee specification is intended to be simpler and less expensive than other WPANs, such as Bluetooth (Section 6.3.4). ZigBee is targeted at radio frequency (RF) applications that require a low data rate, long battery life, and secure networking.

6.3.4 Bluetooth

6.3.5 RFID: Radio-Frequency Identification

Problems related to this section: Problem 6.7 → ??

6.4 Wi-Fi Quality of Service

There is anecdotal evidence of W-LAN spectrum congestion; Unlicensed systems need to scale to manage user "QoS." Density of wireless devices will continue to increase; ~10x with home gadgets; ~100x with sensors/pervasive computing

Decentralized scheduling

We assume that each message carries the data of a single data type. The messages at the producer are ordered in priority-based queues. The priority of a data type is equal to its current utility, $U(T_i \mid S_j)$. Figure 6-23 shows the architecture at the producer node. Scheduler works in a round-robin manner, but may have different strategies for sending the queued messages, called queuing discipline. It may send all high priority messages first, or it may assign higher probability of sending to the high-priority messages, but the low-priority messages still get non-zero probability of being sent.

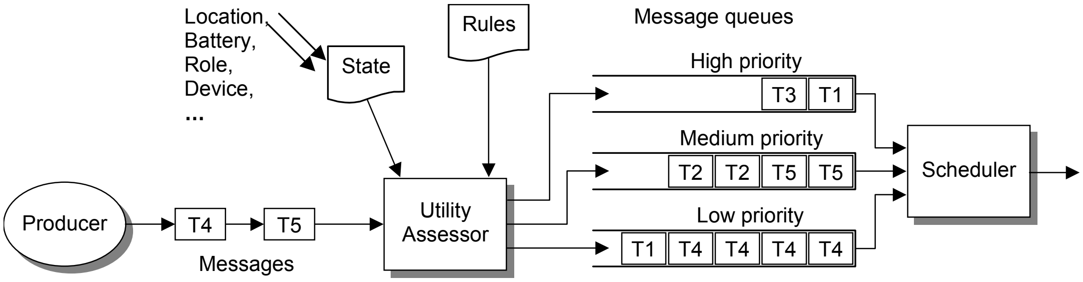


Figure 6-23: Priority-based scheduling of the messages generated by a producer. Messages are labeled by data types of the data they carry (T_1, …, T_5).

It is not clear whose rules for assigning the utilities should be used at producers: producer's or consumer's. If only the consumer's preferences are taken into the account, this resembles to the filtering approach for controlling the incoming information, e.g., blocking unsolicited email messages. One of the drawbacks of filtering is that does not balance the interests of senders and recipients: filtering is recipient-centric and ignores the legitimate interests of the sender [**Error! Reference source not found.**]. This needs to be investigated.

6.5 Summary and Bibliographical Notes

A collection of articles on mobile ad hoc networks, particularly the routing protocol aspect, is available in [Perkins, 2001]. [Murthy & Manoj, 2004] provide a comprehensive overview of ad hoc networks.

The major enhancement in IEEE 802.11e MAC protocol is providing Quality-of-Service (QoS), which is lacking in the legacy IEEE 802.11 MAC protocol. In IEEE 802.11e, enhanced distributed channel access (EDCA) is introduced to enhance legacy IEEE 802.11 DCF operation. EDCA is a contention-based channel access mechanism. QoS support is provided with different access categories (ACs). Four ACs are used in EDCA, each with an independent backoff mechanism and contention parameters. The parameters of ACs are set differently to provide differentiated QoS priorities for ACs.

The Block ACK has a potential to be more efficient than the regular ACK policy. However, the Basic Block ACK frame (defined in IEEE 802.11e and adopted in 802.11n) includes a Block ACK Bitmap of 128 bytes, and the efficiency of the Block ACK might be seriously compromised when the number of frames acknowledged by a Block ACK is small or the frames are not fragmented.

802.11n

The objective of IEEE 802.11n standard is to increase the throughput beyond 100 Mbps as well as extending the effective range from previous 802.11a/b/g standards. Use of Multiple Input Multiple Output (MIMO) technology along with OFDM (MIMO-OFDM) and doubling the channel bandwidth from 20-MHz to 40-MHz helps increase the physical (PHY) rate up to 600 Mbps. The data rates supported in an 802.11n network range from 6.5 Mbps to 600 Mbps. Support for 2 spatial streams is mandatory at the access point and up to 4 spatial streams can be used. The PHY layer enhancements are not sufficient to achieve the desired MAC throughput of more than 100 Mbps due to rate-independent overheads. To overcome this limitation, frame aggregation at the MAC layer is used in 802.11n to improve the efficiency. In the aggregate MSDU (A-MSDU) scheme, multiple MSDUs form a MPDU i.e., an 802.11n MAC-level frame (A-MSDU) consists of multiple subframes (MSDUs), either from different sources or destinations. The aggregate MPDU (A-MPDU) scheme can be used to aggregate multiple MPDUs into a single PSDU. Both aggregation schemes have their pros and cons along with associated implementation aspects. However, most product manufacturers support both features.

802.11b and 802.11g devices operate in the 2.4 GHz band and the 5 GHz band is used by 802.11a devices. The 802.11n-based devices can operate in both bands and hence backward compatibility with the respective legacy devices in the bands are an important feature of the standard. Most of the benefits of 802.11n will only be realized when 802.11n-capable clients are used with similar infrastructure, and even a few legacy (802.11a/b/g) clients in the cell will drastically reduce overall performance compared to a uniform 802.11n network. For quite a long time, 802.11n will need to operate in the presence of legacy 802.11a, b, and g devices. This mixed-mode operation will continue until all the devices in an area have been upgraded or replaced with 802.11n devices. However, sometimes protection mechanism is needed even in an 802.11n-only network as the devices can have different capabilities. Hence, protection schemes are not only used for coexistence with legacy devices but also for interoperability with various different operating modes of 802.11n devices. Each protection mechanism has a different impact on the performance and 802.11n devices will operate more slowly when working with legacy Wi-Fi devices.

802.11n and HT technology is so complex that an entire book dedicated to the topic would probably not be able to cover fully every aspect of HT. Section 6.3.1 highlights only some of the key features of 802.11n. MIMO technology is only briefly reviewed. 802.11n link adaptation and receiver feedback information is not reviewed at all. Other issues that were not covered include security, power management, and quality-of-service (QoS). The reader should consult other sources for these topics.

Xiao and Rosdahl [2002; 2003] have shown that control overhead is a major reason for inefficient MAC. The overhead is large either when the data rate is high or when the frame size is small. Throughput in 802.11 has an upper bound even the data rate goes to infinity. [Xiao, 2005] …

Thornycroft [2009] offers a readable introduction to 802.11n. Perahia and Stacey [2008] provide an in-depth review of 802.11n. The reader should consult the IEEE 802.11n standard for a more technical walk-through of the newly introduced enhancements.

Wang and Wei [2009] investigated the performance of the IEEE 802.11n MAC layer enhancements: frame aggregation, block acknowledgement, and reverse direction (RD) protocol. They conclude that VoIP performance is effectively improved with 802.11n MAC enhancements.

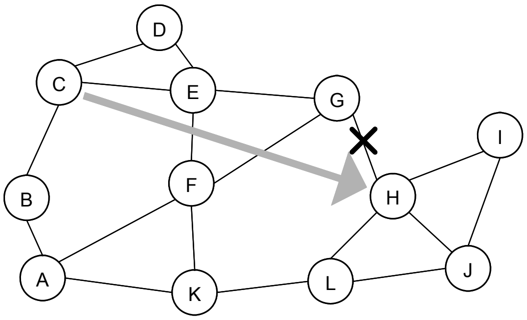
Problems

Problem 6.1

Problem 6.2

DSR Routing in Ad-hoc Networks.

Consider the ad-hoc network shown in Figure 6-4 (reproduced below). Assume that the network is using DSR routing protocol and host *C* is sending data packets along the path *C*-*E*-*G*-*H* (discovered as described in Section 6.2.1). The nodes are not using route caching, so they cannot do packet salvaging.



Suppose that suddenly link *G*-*H* is broken when a data packet is being forwarded. Starting from the moment when *G* decided that the link is broken after unsuccessful retransmissions of the data packet, solve the following:

(a) List all the packets that will be propagated backwards towards *C*.

(b) List all the packets that will be propagated after *C* initiates a new route discovery, until *C* learns about a new route to *H*.

Always show the path information in each packet header.

Problem 6.3

Problem 6.4

Problem 6.5

Problem 6.6

Problem 6.7

Consider a scenario with ten Gen-2 RFID tags within the coverage of the RFID reader. Solve the following:

(a) What is the optimum value for the parameter Q for inventorying the tags in our scenario? Is this the minimum value of Q that would allow inventorying all the tags?

(b) Draw the timeline of message exchanges between the reader and the tags. Assume that the reader starts with $Q = 2$. The timeline should show *any* plausible scenario of how the reader might succeed with inventorying all the tags. Write down the exact rules which you used to verify that your scenario is plausible.

Problem 6.8

Using the same scenario as in Problem 6.7 and assuming $Q = 2$, what is the probability that one of the ten tags will respond with no collision in the first slot of an inventorying cycle? How would you find the probability of no collision in the second slot?

Chapter 7
Network Security

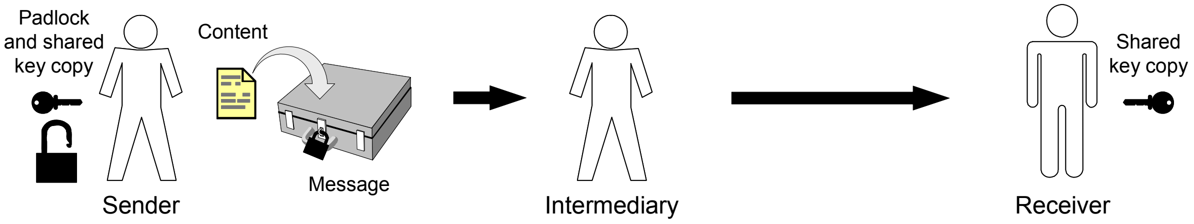
Contents

7.1 Elements of Cryptography

Information security is a nonfunctional property of the system, it is an *emergent* property. Owing to different types of information use, there are two main security disciplines. *Communication security* is concerned with protecting information when it is being transported between different systems. *Computer security* is related to protecting information within a single system, where it can be stored, accessed, and processed. Although both disciplines must work in accord to successfully protect information, information transport faces greater challenges and so communication security has received greater attention. Accordingly, this review is mainly concerned with communication security. Notice that both communication- and computer security must be complemented with physical (building) security as well as personnel security. Security should be thought of as a chain that is as strong as its weakest link.

The main objectives of information security are:

- *Confidentiality*: ensuring that information is not disclosed or revealed to unauthorized persons



Sender needs:
- receive securely a copy of the shared key
- positively identify the message receiver

Threats posed by intruder/adversary:
- forge the key and view the content
- damage/substitute the padlock
- damage/destroy the message
- observe characteristics of messages (statistical and/or metric properties)

Receiver needs:
- receive securely a shared key copy
- positively identify the message sender
- detect any tampering with messages

Figure 7-1: Communication security problem: Sender needs to transport a confidential document to Receiver over an untrusted intermediary.

- *Integrity*: ensuring consistency of data, in particular, preventing unauthorized creation, modification, or destruction of data
- *Availability*: ensuring that legitimate users are not unduly denied access to resources, including information resources, computing resources, and communication resources
- *Authorized use*: ensuring that resources are not used by unauthorized persons or in unauthorized ways

To achieve these objectives, we institute various *safeguards*, such as concealing (*encryption*) confidential information so that its meaning is hidden from spying eyes; and *key management* which involves secure distribution and handling of the "keys" used for encryption. Usually, the complexity of one is inversely proportional to that of the other—we can afford relatively simple encryption algorithm with a sophisticated key management method.

Figure 7-1 illustrates the problem of transmitting a confidential message by analogy with transporting a physical document via untrusted carrier. The figure also lists the security needs of the communicating parties and the potential threats posed by intruders. The sender secures the briefcase, and then sends it on. The receiver must use a correct key in order to unlock the briefcase and read the document. Analogously, a sending computer encrypts the original data using an *encryption algorithm* to make it unintelligible to any intruder. The data in the original form is known as ***plaintext*** or ***cleartext***. The encrypted message is known as ***ciphertext***. Without a corresponding "decoder," the transmitted information (ciphertext) would remain scrambled and be unusable. The receiving computer must regenerate the original plaintext from the ciphertext with the correct *decryption algorithm* in order to read it. This pair of data transformations, encryption and decryption, together forms a ***cryptosystem***.

There are two basic types of cryptosystems: (*i*) *symmetric* systems, where both parties use the *same* (secret) key in encryption and decryption transformations; and, (*ii*) *public-key* systems, also known as *asymmetric* systems, where the parties use two related keys, one of which is secret and the other one can be publicly disclosed. I first review the logistics of how the two types of cryptosystems work, while leaving the details of encryption algorithms for the next section.

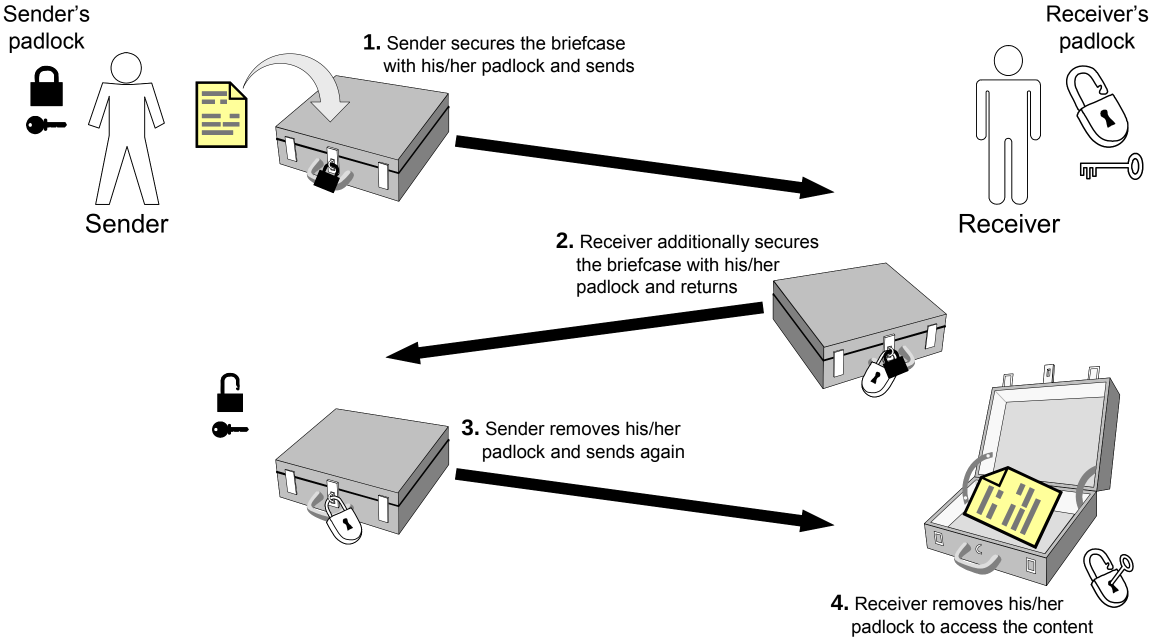

Sender's padlock
Sender
1. Sender secures the briefcase with his/her padlock and sends
Receiver's padlock
Receiver
2. Receiver additionally secures the briefcase with his/her padlock and returns
3. Sender removes his/her padlock and sends again
4. Receiver removes his/her padlock to access the content

Figure 7-2: Secure transmission via untrustworthy carrier. Note that both sender and receiver keep their own keys with them all the time—the keys are never exchanged.

7.1.1 Symmetric and Public-Key Cryptosystems

In symmetric cryptosystems, both parties use the *same* key in encryption and decryption transformations. The key must remain secret and this, of course, implies trust between the two parties. This is how cryptography traditionally works and prior to the late 1970s, these were the only algorithms available.

The system works as illustrated in Figure 7-1. In order to ensure the secrecy of the shared key, the parties need to meet prior to communication. In this case, the fact that only the parties involved know the secret key implicitly *identifies* one to the other.

Using encryption involves two basic steps: *encoding* a message, and *decoding* it again. More formally, a code takes a message M and produces a coded form $f(M)$. Decoding the message requires an inverse function f^{-1}, such that $f^{-1}\big(f(M)\big) = M$. For most codes, anyone who knows how to perform the first step also knows how to perform the second, and it would be unthinkable to release to the adversary the method whereby a message can be turned into code. Merely by "undoing" the encoding procedures, the adversary would be able to break all subsequent coded messages.

In the 1970s Ralph Merkle, Whitfield Diffie, and Martin Hellman realized that this need not be so. The weasel word above is "merely." Suppose that the encoding procedure is very hard to undo. Then it does no harm to release its details. This led them to the idea of a *trapdoor function*. We call f a trapdoor function if f is easy to compute, but f^{-1} is very hard, indeed impossible for practical purposes. A trapdoor function in this sense is not a very practical code, because the legitimate user finds it just as hard to decode the message as the adversary does. The final twist is

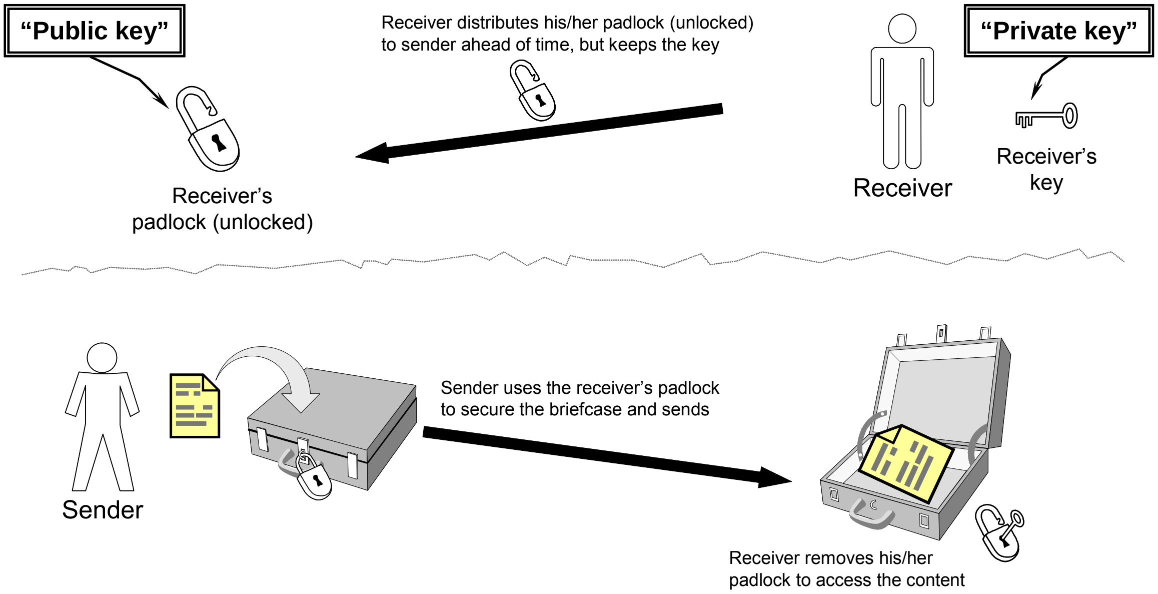

"Public key"
Receiver distributes his/her padlock (unlocked) to sender ahead of time, but keeps the key
"Private key"
Receiver's padlock (unlocked)
Receiver
Receiver's key
Sender
Sender uses the receiver's padlock to secure the briefcase and sends
Receiver removes his/her padlock to access the content

Figure 7-3: Public-key cryptosystem simplifies the procedure from Figure 7-2.

to define *f* in such a way that a single extra piece of information makes the computation of f^{-1} easy. This is the only bit of information that must be kept secret.

This alternative approach is known as *public-key cryptosystems*. To understand how it works, it is helpful to examine the analogy illustrated in Figure 7-2. The process has three steps. In the first step, the sender secures the briefcase with his or her padlock and sends. Second, upon receiving the briefcase, the receiver secures it additionally with their own padlock and returns. Notice that the briefcase is now doubly secured. Finally, the sender removes his padlock and re-sends. Hence, sender manages to send a confidential document to the receiver without needing the receiver's key or surrendering his or her own key.

There is an inefficiency of sending the briefcase back and forth, which can be avoided as illustrated in Figure 7-3. Here we can skip steps 1 and 2 if the receiver distributed his/her padlock (unlocked, of course!) ahead of time. When the sender needs to send a document, i.e., message, he/she simply uses the receiver's padlock to secure the briefcase and sends. Notice that, once the briefcase is secured, nobody else but receiver can open it, not even the sender. Next I describe how these concepts can be implemented for electronic messages.

7.1.2 Cryptographic Algorithms

Encryption has three aims: keeping data confidential; authenticating who sends data; and, ensuring data has not been tampered with. All cryptographic algorithms involve substituting one thing for another, which means taking a piece of plaintext and then computing and substituting the corresponding ciphertext to create the encrypted message.

Symmetric Cryptography

The Advanced Encryption Standard has a fixed block size of 128 bits and a key size of 128, 192, and 256 bits.

Public-Key Cryptography

As stated above, f is a trapdoor function if f is easy to compute, but f^{-1} is very hard or impossible for practical purposes. An example of such difficulty is factorizing a given number n into prime numbers. An encryption algorithm that depends on this was invented by Ron Rivest, Adi Shamir, and Leonard Adelman (RSA system) in 1978. Another example algorithm, designed by Taher El Gamal in 1984, depends on the difficulty of the discrete logarithm problem.

In the RSA system, the *receiver* does as follows:

1. Randomly select two large prime numbers p and q, which always must be kept secret.
2. Select an integer number E, known as the *public exponent*, such that $(p - 1)$ and E have no common divisors, and $(q - 1)$ and E have no common divisors.
3. Determine the product $n = p \cdot q$, known as *public modulus*.
4. Determine the *private exponent*, D, such that $(E \cdot D - 1)$ is exactly divisible by both $(p - 1)$ and $(q - 1)$. In other words, given E, we choose D such that the integer remainder when $E \cdot D$ is divided by $(p - 1) \cdot (q - 1)$ is 1.
5. Release publicly the **public key**, which is the pair of numbers n and E, $K^{+} = (n, E)$. Keep secret the **private key**, $K^{-} = (n, D)$.

Because a digital message is a sequence of digits, break it into blocks which, when considered as numbers, are each less than n. Then it is enough to encode block by block.

Encryption works so that the sender substitutes each plaintext block B by the ciphertext $C = B^{E}$ % n, where % symbolizes the modulus operation. (The *modulus* of two integer numbers x and y, denoted as x % y, is the integer remainder when x is divided by y.)

Then the encrypted message C (ciphertext) is transmitted. To decode, the receiver uses the *decoding key D*, to compute $B = C^{D}$ % n, that is, to obtain the plaintext B from the ciphertext C.

Example 7.1 RSA cryptographic system

As a simple example of RSA, suppose the receiver chooses $p = 5$ and $q = 7$. Obviously, these are too small numbers to provide any security, but they make the presentation manageable. Next, the receiver chooses $E = 5$, because 5 and $(5 - 1) \cdot (7 - 1)$ have no common factors. Also, $n = p \cdot q = 35$. Finally, the receiver chooses $D = 29$, because $\frac{E \cdot D - 1}{(p-1) \cdot (q-1)} = \frac{5 \cdot 29 - 1}{4 \cdot 6} = \frac{144}{24} = 6$, i.e., they are exactly divisible. The receiver's public key is $K^{+} = (n, E) = (35, 5)$, which is made public. The private key $K^{-} = (n, D) = (35, 29)$ is kept secret.

Now, suppose that the sender wants to send the plaintext "hello world." The following table shows the encoding of "hello." I assign to letters a numeric representation, so that a=1, b=2, …, y=25, and z=26, and I assume that block B is one letter long. In an actual implementation, letters are represented as binary numbers, and the blocks B are not necessarily aligned with letters, so that plaintext "l" will not always be represented as ciphertext "12."

Plaintext letter	Plaintext numeric representation	B^E	Ciphertext B^E % n
h	8	$8^5 = 32768$	8^5 % 35 = 8
e	5	$5^5 = 3125$	5^5 % 35 = 10
l	12	$12^5 = 248832$	12^5 % 35 = 17
l	12	248832	17
o	15	$15^5 = 759375$	15^5 % 35 = 15

The sender transmits this ciphertext to the receiver: 8, 10, 17, 17, 15. Upon the receipt, the receiver decrypts the message as shown in the following table.

Ciphertext	C^D	$B = C^D$ % n	Plaintext letter
8	$8^{29} = 154742504910672534362390528$	8^{29} % 35 = 8	h
10	100000000000000000000000000000	5	e
17	481968572106750915091411825223071697	12	l
17	481968572106750915091411825223071697	12	l
15	12783403948858939111232757568359375	15	o

We can see that even this toy example produces some extremely large numbers.

The point is that while the adversary knows n and E, he or she does not know p and q, so they cannot work out $(p - 1)\cdot(q - 1)$ and thereby find D. The designer of the code, on the other hand, knows p and q because those are what he started from. So does any legitimate receiver: the designer will have told them. The security of this system depends on exactly one thing: the notoriously difficulty of factorizing a given number n into primes. For example, given $n = 2^{67} - 1$ it took three years working on Sundays for F. N. Cole to find by hand in 1903 p and q for $n = p\cdot q = 193707721 \times 761838257287$. It would be waste of time (and often combinatorially self-defeating) for the program to grind through all possible options.

Encryption strength is measured by the length of its "key," which is expressed in bits. The larger the key, the greater the strength of the encryption. Using 112 computers, a graduate student decrypted one 40-bit encrypted message in a little over 7 days. In contrast, data encrypted with a 128-bit key is 309,485,009,821,345,068,724,781,056 times stronger than data encrypted with a 40-bit key. RSA Laboratories recommends that the product of p and q be on the order of 1024 bits long for corporate use and 768 bits for use with less valuable information.

7.1.3 Authentication

7.2 Authentication Protocols

7.3 Network-Layer Security

7.4 Firewalls

7.5 Network Attacks and Defenses

7.6 Summary and Bibliographical Notes

Problems

Problem 7.1

Problem 7.2

Problem 7.3

Recall the authentication protocol (Figure 7-___Error!___) and suppose that while Sender is authenticating itself to Receiver, it is also required that at the same time Receiver must authenticate itself to Sender (known as “mutual authentication” or “two-way authentication”). Give a scenario in which Adversary (pretending to be Sender) can now authenticate itself to Receiver as Sender. Note that the authentication protocol steps of both parties can be arbitrarily interleaved. Pay particular attention to a potential misuse of a nonce.

Problem 7.4

Chapter 8
Network Monitoring

Contents

8.1 Introduction

See: http://www.antd.nist.gov/

Wireless link of a mobile user does not provide guarantees. Unlike wired case, where the link parameters are relatively stable, stability cannot be guaranteed for a wireless link. Thus, even if lower-level protocol layers are programmed to perform as best possible, the application needs to know the link quality. The "knowledge" in the wired case is provided through quality guarantees, whereas here link quality knowledge is necessary to adapt the behavior.

Adaptation to the dynamics of the wireless link bandwidth is a frequently used approach to enhance the performance of applications and protocols in wireless communication environments [Katz 1994]. Also, for resource reservation in such environments, it is crucial to have the knowledge of the dynamics of the wireless link bandwidth to perform the admission control.

End-to-end methodologies for detecting the state of the network generally share the same fundamental principle. They use some kind of probe packets in which the sender transmits probe packets of a certain pattern through the network to the receiver. Along the path, the probe packets experience different amounts of delay as incurred by the network. The changing pattern (e.g., the interval gap) in which the probe packets arrive at the receiver is caused by the characteristics of the network. By observing the incoming packets, the receiver can infer the state of the network.

8.2 Available Bandwidth Estimation

In general, an accurate available bandwidth estimation is essential in monitoring if the different flows are living up to the required Quality-of-Service (QoS). For instance, streaming applications could adapt their sending rate to improve the QoS depending on a real-time knowledge of the end-to-end available bandwidth. Within a mobile-communications core network, the available bandwidth could also be used as an input to take decisions concerning issues such as load control, admission control, handover and routing. However, the scale of the different systems, the different traffic characteristics and the diversity of network technologies make this characterization of the end-to-end available bandwidth a challenging task.

One possible way to implement available bandwidth estimation would be to deploy special software or hardware on each router of the network. However, the cost in time and money of new equipment, maintenance of new nodes and software development makes this impractical. Moreover, this wide-scale deployment of specialized routers, which are continuously reporting bandwidth properties, might overwhelm the network. Another limitation is the lack of control over hosts and routers across autonomous domains.

An alternative is to run software on end hosts, which is usually called **active probing**. In this approach, the available bandwidth is inferred rather than directly measured. Ideally, a probing scheme should provide an accurate estimate as quickly as possible, while keeping the increased load on the network to the necessary minimum. There are several obstacles for measuring the available bandwidth by active probing. First, the available bandwidth is a time-varying metric. Second, the available bandwidth exhibits variability depending on the observing time-scale. Third, in the current networks increasingly intelligent devices are being used for traffic prioritization.

A **narrow link** (bottleneck) is a communication link with a small upper limit on the bandwidth. Conversely, a **tight link** (overused) is a communication link with a small available bandwidth but the overall bandwidth may be relatively large.

8.2.1 Packet-Pair Technique

A *packet pair* consists of two packets, usually of the same size, that are sent back-to-back via a network path. Unlike the techniques mentioned in the previous section, packet-pair probing directly gives a value for the capacity of the *narrow link*, with no additional information about the capacities of other links on the path. They assume FIFO queuing model in network routers and probe packets could be ICMP (Internet Control Message Protocol) packets. Packet-pair techniques for bandwidth measurement are based on measuring the transmission delays that packets suffer on their way from the source to the destination. The idea is to use inter-packet time to estimate the characteristics of the bottleneck link. If two packets travel together so that they are queued as a pair at the bottleneck link with no packet intervening between them, then their inter-

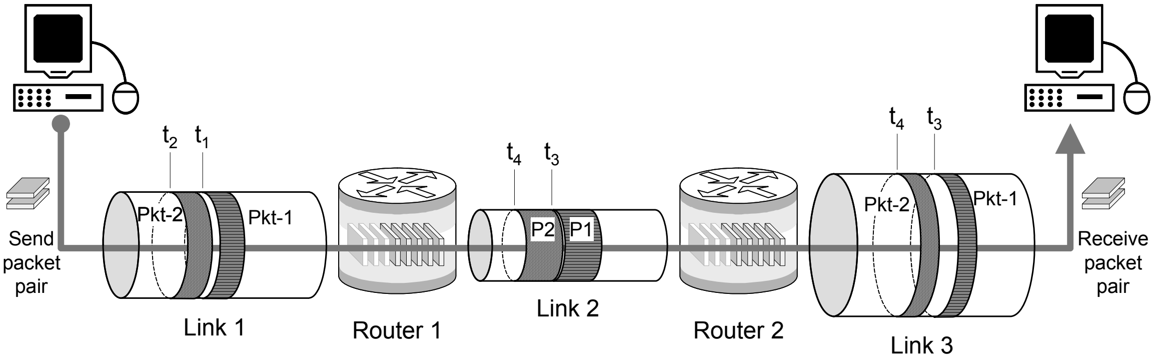


Figure 8-1: Packet-pair technique for bandwidth measurement. The packet spacing on the bottleneck link (Link 2) will be preserved on downstream links (Link 3).

packet spacing is proportional to the time needed to transmit the second packet of the pair on the bottleneck link (**Error! Reference source not found.**).

Recall that packet transmission delay is computed using Eq. (1.2) in Section 1.3 as $t_x = L/R$, where L is the packet length and R is the link transmission rate. Le us assume that the receiver measures the difference of arrival times for a packet pair as $\Delta = t_4 - t_3$. Then, the transmission rate of the bottleneck link (i.e., link bandwidth), b, can be computed as:

$$b = L/\Delta \tag{6.1}$$

Note that the inter-packet spacing at the source must be sufficiently small so that:

$$t_2 - t_1 \leq L/b = t_4 - t_3 \tag{6.2}$$

The *packet-pairs* technique is useful for measuring the bottleneck capacity, i.e., the minimum capacity along the path. The main advantage of this method is that it performs the measurement of inter-arrival times between packets only at the end host. This fact avoids the problem of asymmetric routing, ICMP dependency and link-layer effects of RTT-based Capacity Estimation methods. On the other hand, this technique is very sensitive, not only to the probing packet size and user time resolution, but also to the cross-traffic.

PathMon [Kiwior, et al., 2004]

8.3 Dynamic Adaptation

Holistic QoS, system level

Computation and communication delays

Combinatorial optimization

Adaptive Service Quality

It may be that only two options are offered to the customers by the server: to be or not to be processed. In other words, quality of servicing is offered either in the fullest or no servicing at all. But, it may be that the server offers different options for customers "in hurry." In this case, we can speak of different *qualities of service*—from no service whatsoever, through partial service, to full service. The spectrum of offers may be discrete or continuous. Also, servicing options may be *explicitly* known and advertised as such, so the customer simply chooses the option it can afford. The other option is that servicing options are *implicit*, in which case they could be specified by servicing time or cost, or in terms of complex circumstantial parameters. Generally, we can say that the customer specifies the *rules* of selecting the quality of service in a given rule specification language.

Associated with processing may be a cost of processing. Server is linked with a certain resource and this resource is limited. Server capacity *C* expresses the number of customers the server can serve per unit of time and it is limited by the resource availability.

Important aspects to consider:

- Rules for selecting QoS
- Pricing the cost of service
- Dealing with uneven/irregular customer arrivals
- Fairness of service
- Enforcing the policies/agreements/contracts
    - Admission control
    - Traffic shaping

8.3.1 Data Fidelity Reduction

Compression

Simplification

Abstraction

Conversion (different domains/modalities)

We consider the model shown in Figure 6-?? where there are multiple clients producing and/or consuming dynamic content. Some shared content may originate or be cached locally while other may originate from remote and change with time. The data that originates locally may need to be distributed to other clients. The clients have local computing resources and share some global resources, such as server(s) and network bandwidth, which support information exchange. Although producers and consumers are interrelated, it is useful to start with a simpler model where we consider them independently to better understand the issues before considering them jointly. We first consider individual clients as *data consumers* that need to visualize the content with the best possible quality and provide highest interactivity. We then consider clients as *data producers* that need to update the consumers by effectively and efficiently employing global resources.

We will develop a formal method for maximizing the utility of the shared content given the limited, diverse, and variable resources. Figure 9-1 illustrates example dimensions of data adaptation; other possible dimensions include modality (speech, text, image, etc.), security, reliability, etc. The user specifies the rules *R* for computing the utilities of different data types that may depend on contextual parameters. We define the *state* of the environment as a touple containing the status of different environmental variables. For example, it could be defined as: state = (time, location, battery energy, user's role, task, computer type). The location may include both the sender and the receiver location. Given a state S_j the utility of a data type T_i is determined by applying the user-specified rules: $U(T_i \mid S_j) = R(T_i \mid S_j)$. We also normalize the utilities because it is easier for users to specify relative utilities, so in a given state S_j the utilities of all data types are: $\sum_i U(T_i \mid S_j) = 1$.

Our approach is to vary the fidelity and timeliness of data to *maximize the sum of the utilities* of the data the user receives. Timeliness is controlled, for example, by the parameters such as update frequency, latency and jitter. Fidelity is controlled by parameters such as the detail and accuracy of data items and their structural relationships. Lower fidelity and/or timeliness correspond to a lower demand for resources. Our method uses nonlinear programming to select those values for

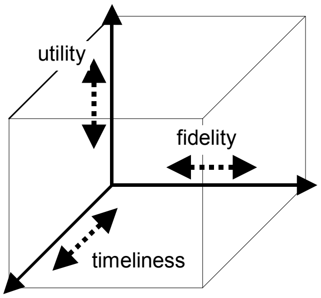


Figure 8-2: Dimensions of data adaptation.

fidelity and timeliness that maximize the total data utility, subject to the given resources. Note that the user can also require fixed values for fidelity and/or timeliness, and seek an optimal solution under such constraints.

8.3.2 Application Functionality Adaptation

8.3.3 Computing Fidelity Adaptation

Review CMU-Aura work

8.4 Summary and Bibliographical Notes

When deploying a real-time or multimedia application, you need to establish a thorough baseline of current network activity on all segments that will host the application. You need to understand the degree to which latency, jitter and packet loss affect your network before deploying a real-time or multimedia application. You must understand current network load and behavior, including any areas where latency is elevated or highly variable. In many networks, traffic loads may vary substantially over time. As loads increase, inconsistent packet delivery rates are probable. Thus, increasing loads form the foundation for excessive latency and jitter—which are two of the most prevalent inhibitors for consistent application performance. When collecting baseline metrics, remember that network behavior varies widely as various business activities occur. Be sure to create a baseline that reflects all major phases and facets of your network's activities.

V. Firou, J. Le Boudec, D. Towsley, and Z. Zhang, "Theories and Models for Internet Quality of Service," Proceedings of the IEEE, Special Issue on Internet Technology, August 2002.

Various forms of the packet-pair technique were studied by [Bolot, 1993], [Carter & Crovella, 1996], [Paxson, 1997a], and [Lai & Baker, 1999].

[Jain & Dovrolis, 2002] investigated how to deal with cross-traffic, using statistical methods

[Hu & Steenkiste, 2003], available bandwidth discovery: Initial Gap Increasing (IGI)
estimate both upper limit and background traffic and subtract both. Problem: presumes that the bottleneck link is also the tight link

PathMon [Kiwior, et al., 2004]

Problems

Chapter 9

Internet Protocols

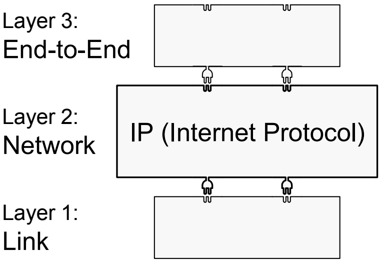
Contents

This chapter describes several important protocols that are used in the current Internet. I feel that these protocols are not critical for the rest of this text. However, the reader may feel otherwise, so I included them for completeness. Also, a student new to the field may wish to know about practical implementations of the concepts and algorithms described in the rest of this text.

Although this chapter is about the Internet protocols, the key Internet protocols are not reviewed here. Because of their great importance, IP and TCP are described early one, in Chapters 1 and 2, respectively.

9.1 Internet Protocol Version 6 (IPv6)

Internet Protocol version 4 (Sections 1.4.1 and 1.4.4) was first developed in the 1970s and the main document that defines IPv4 functionality (RFC-791) was published in 1981. Many things could not be envisioned at such early stage of Internet development, especially shortage of address space availability. Internet Protocol version 6 (IPv6) was developed primarily to



Visit http://en.wikipedia.org/wiki/Internet_reference_model for more details on the Internet reference model

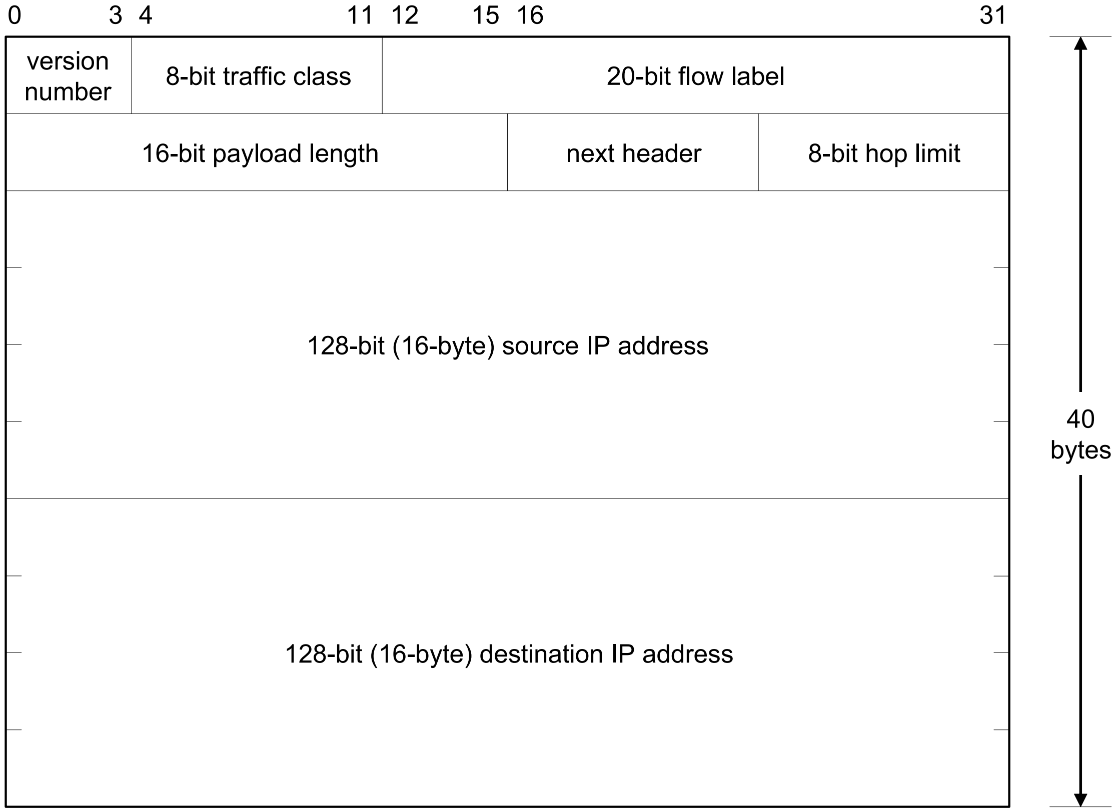


Figure 9-1: The header format of IPv6 datagrams.

address the rapidly shrinking supply of IPv4 addresses, but also to implement some novel features based on the experience with IPv4.

Figure 9-1 shows the format of IP version 6 datagram headers. It is much simpler than the IPv4 header (Figure 1-39), but also double the size of a default IPv4 header, mainly because of longer IP addresses in IPv6. The IPv6 header fields are as follows:

Version number: This field indicates version number, and the value of this field for IPv6 datagrams is 6.

Traffic class: The 8-bit Traffic Class field is available for use by originating nodes or forwarding routers to identify and distinguish between different classes or priorities of IPv6 packets. This filed is equivalent to IPv4 Differentiated Services field (Figure 1-39), and it is intended to provide various forms of "differentiated service" or DiffServ for IP packets (Section 3.3.4). Experiments are currently underway to determine what sorts of traffic classifications are most useful for IP packets.

Flow label: The 20-bit Flow Label field may be used by a source to label packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service. This feature is still experimental and evolves as more experience is gained with IntServ flow support in the Internet (Section 3.3.5). Hosts or routers that do not support the functions of the Flow Label field are required to set the field to zero when originating a packet; pass the field on unchanged when forwarding a packet; and ignore the field when receiving a packet.

Payload length: This field tells how many bytes follow the 40-byte header in the IP datagram. Unlike IPv4 Datagram Length, IPv6 Payload Length does not count the datagram header.

Next header: This 8-bit selector identifies the type of header immediately following the IPv6 header within the IPv6 datagram. Currently, there are six extension headers (optional), which may follow the main IPv6 header. If a header is the last IP header, this field identifies the type of the upper-layer protocol to pass the payload to at the destination, and uses the same values as the IPv4 "User Protocol" field (Figure 1-39). See more about extension headers in Section 9.1.2.

Hop limit: This 8-bit unsigned integer specifies how long a datagram is allowed to remain in the Internet, to catch packets that are stuck in routing loops. It is decremented by one by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero. It is equivalent to the "Time-to-Live" (TTL) field in IPv4 datagrams.

Source IP address: This address identifies the end host that originated the datagram.

Destination IP address: This is the IP address of the intended recipient of the packet (possibly not the ultimate recipient, if a *Routing header* is present, as described in the next section).

Note that there are none of the fields related to packet fragmentation from IPv4 (Figure 1-39). This is because IPv6 takes a different approach to fragmentation. To simplify the work of routers and speed up their performance, IPv6 assumes that router do not perform any fragmentation. IPv6 requires that every link in the Internet have a maximum transmission unit (MTU) of 1280 bytes or greater. On any link that cannot convey a 1280-byte packet in one piece, link-specific fragmentation and reassembly must be provided at a protocol layer below IPv6.

Links that have a configurable MTU (for example, PPP links Section 1.5.1) must be configured to have an MTU of at least 1280 bytes. It is strongly recommended that IPv6 nodes implement Path MTU Discovery (described in RFC-1981), in order to dynamically discover and take advantage of path MTUs greater than 1280 bytes. This rule makes fragmentation less likely to occur in the first place. In addition, when a host sends an IPv6 packet that is too large, instead of fragmenting it, the router that is unable to forward it drops the packet and sends back an error message. This message tells the originating host to break up all future packets to that destination.

9.1.1 IPv6 Addresses

The IPv6 address space is 128-bits (2^{128}) in size, which translates to the exact number of 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses. That seems like enough for all purposes that currently can be envisioned.

A new notation (*hexadecimal colon notation*) has been devised to writing 16-byte IPv6 addresses. A 128-bit address is divided into eight sections, each two bytes long. It is written as eight groups of four hexadecimal digits (total 32) with colons between the groups, like this:

2000:0000:0000:0000:0123:4567:89AB:CDEF

Some simplifications have been authorized for special cases. For example, if an address has a large number if consecutive zeros, the zero fields can be omitted and replaced with a double colon "::".The above example address can be written compactly as 2000::123:4567:89AB:CDEF. Note that only leading or intermediary zeroes can be abbreviated, but not the trailing zeroes. Also,

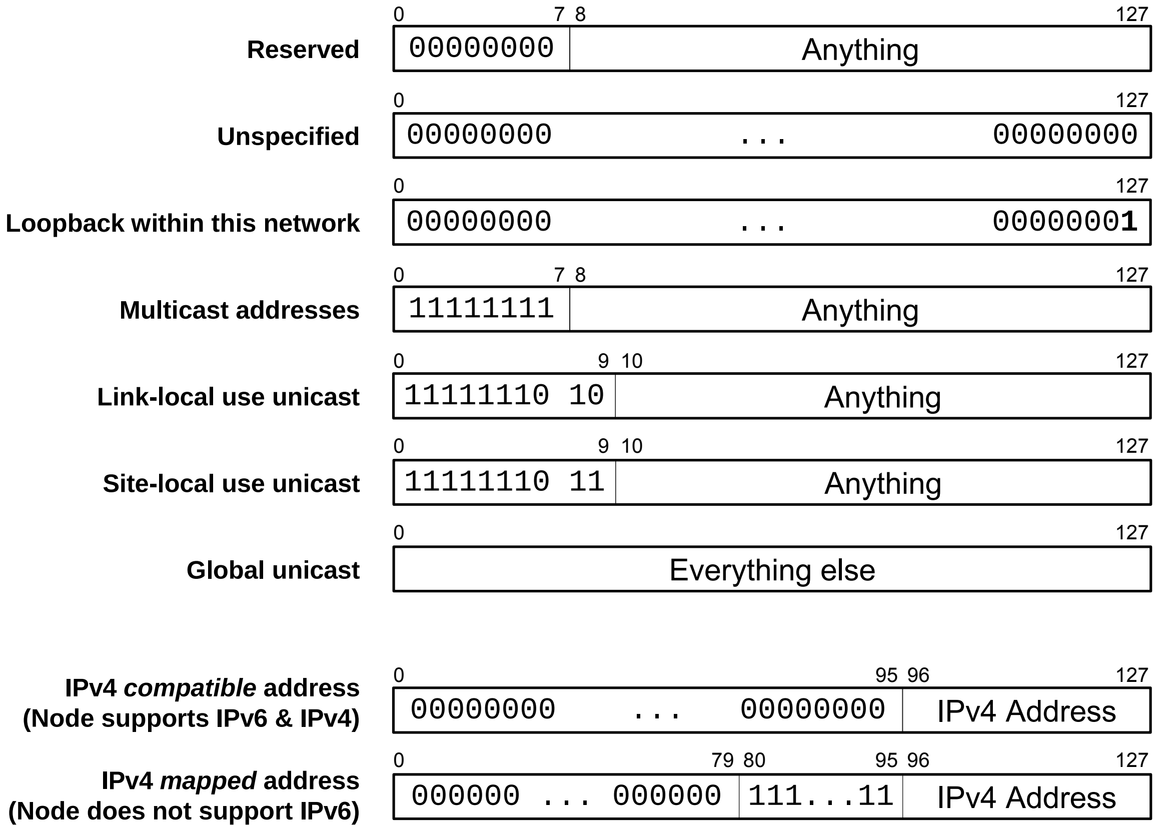


Figure 9-2: Selected address prefix assignments for IPv6, excerpted from RFC-2373.

this type of abbreviation is allowed only once per address; if there are two runs of zeroes, only one of the can be abbreviated.

As with IPv4 CIDR scheme, the notation *A*/*m* designates a subset of IPv6 addresses (subnetwork) where *A* is the prefix and the mask *m* specifies the number of bits that designate the subset, beginning from left to right. For example, the notation: 2000:0BA0:01A0::/48 implies that the part of the IPv6 address used to represent the subnetwork has 48 bits. Because each hexadecimal digit has 4 bits, the prefix representing the subnetwork is formed by 48/4 = 12 digits, that is: "2000:0BA0:01A0." The remaining (128 − 48)/4 = 20 digits would be used to represent the network interfaces inside the subnetwork.

Similar to CIDR in IPv4 (Section 1.4.4), IPv6 addresses are classless. However, the address space is hierarchically subdivided depending on the leading bits of an address. A variable number of leading bits specify the **type prefix** that defines the purpose of the IPv6 address. To avoid ambiguity, the prefix codes are designed such that no code is identical to the first part of any other code. The current assignment of prefixes is shown in Figure 9-2. Note that two special addresses ("unspecified address" and loopback address) are assigned out of the reserved 00000000-format prefix space. The address 0:0:0:0:0:0:0:0 is called the *unspecified address*. It indicates the absence of an address, and must never be assigned to any node as a destination address. However, it may be used by a host in the Source Address field of an IPv6 packet initially, when the host wants to learn its own address. The unicast address 0:0:0:0:0:0:0:1 is called the *loopback address*. It may be used by a node to send an IPv6 packet to itself. It may never be assigned to any physical interface. It may be thought of as being associated with a virtual interface (e.g., the loopback interface).

The IPv6 addresses with embedded IPv4 addresses are assigned out of the reserved 00000000-format prefix space. There are two types of IPv6 addresses that contain an embedded IPv4 address (see bottom of Figure 9-2). One type are the so-called *IPv4-compatible addresses*. These addresses are used by IPv6 routers and hosts that are directly connected to an IPv4 network and use the "tunneling" approach to send packets over the intermediary IPv4 nodes (Section 9.1.3). This address format consists of 96 bits of 0s followed by 32 bits of IPv4 address. Thus, and IPv4 address of 128.6.29.131 can be converted to an IPv4-compatible ::128.6.29.131. The other type is so called *IP-mapped addresses*. These addresses are used to indicate IPv4 nodes that do not support IPv6. The format of these addresses consist of 80 bits of 0s, followed by 16 bits of 1s, and then by 32 bits of IPv4 address. An example would be written as ::FFFF:128.6.29.131.

IPv6 allows three types of addresses:

- **Unicast:** An identifier for a single network interface. A packet sent to a unicast address should be delivered to the interface identified by that address.
- **Anycast:** A prefix identifier for a set of network interfaces. These typically belong to different nodes, with addresses having the same subnet prefix. A packet sent to an anycast address should be delivered to only one of the interfaces identified by that prefix. The interface selected by the routing protocol is the "nearest" one, according to the protocol's distance metrics. For example, all the routers of a backbone network provider could be assigned a single anycast address, which would then be used in the routing header. One expected use of anycasting is "fuzzy routing," which means sending a packet through "one router of network *X*." The anycast address will also be used to provide enhanced routing support to mobile hosts.
- **Multicast:** An identifier for a set of network interfaces, typically belonging to different nodes that may or may not share the same prefix. A packet sent to a multicast address should be delivered to all the interfaces identified by that address.

There are no broadcast addresses in IPv6; their function is taken over by multicast addresses.

It is anticipated that unicast addressing will be used for the vast majority of traffic under IPv6, just as is the case for older one, IPv4. It is for this reason that the largest of the assigned blocks of the IPv6 address space is dedicated to unicast addressing. RFC-2374 assigned the Format Prefix 2000::/3 (a "001" in the first three bits of the address) to unicast addresses. However, RFC-3587 invalidated this restriction. Although currently only 2000::/3 is being delegated by the IANA, implementations should not make any assumptions about 2000::/3 being special. In the future, the IANA might be directed to delegate currently unassigned portions of the IPv6 address space for the purpose of Global Unicast as well.

Figure 9-3(a) shows the general format for IPv6 global unicast addresses as defined in RFC-3587. The *Global Routing Prefix* is a (typically hierarchically-structured) value assigned to a site (a cluster of subnets or links), the *Subnet ID* is an identifier of a subnet within the site, and the *Interface ID* identifies the network interfaces on a link. The global routing prefix is designed to be structured hierarchically by the Regional Internet Registries (RIRs) and Internet Service

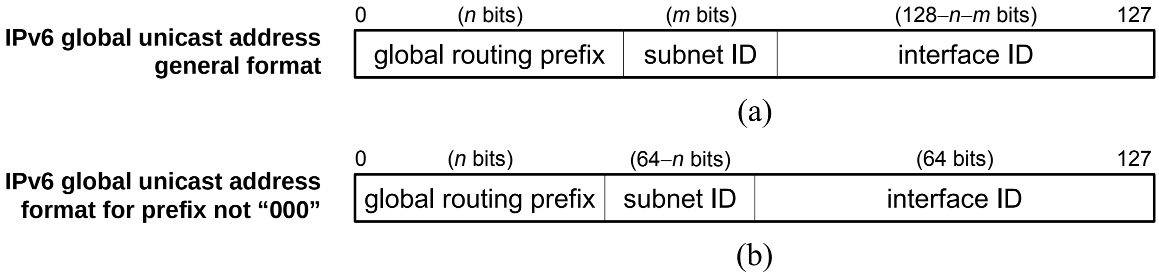


Figure 9-3: Format of IPv6 global unicast addresses.

Table 9-1: IPv6 extension headers.

Extension header	Description
Hop-by-hop options	Miscellaneous information for routers.
Destination options	Additional information for the destination node.
Routing	Loose list of routers to visit (similar to IPv4 source routing).
Fragmentation	Information about datagram fragmentation and reassembly.
Authentication	Verification of the sender's identity.
Encrypted security payload	Information about the encrypted contents.

Providers (ISPs). The subnet-ID field is designed to be structured hierarchically by site administrators.

RFC-3587 also requires that all unicast addresses, except those that start with binary value "000," have Interface IDs that are 64-bits long and to be constructed in Modified 64-bit Extended Unique Identifier (EUI-64) format. The format of global unicast address in this case is shown in Figure 9-3(b). This includes global unicast address under the 2000::/3 prefix (starting with binary value "001") that is currently being delegated by the IANA.

An IPv6 Address [RFC-4291] may be administratively assigned using DHCPv6 [RFC-3315] in a manner similar to the way IPv4 addresses are, but may also be autoconfigured, facilitating network management. Autoconfiguration procedures are defined in [RFC-4862] and [RFC-4941]. IPv6 neighbors identify each other's addresses using either Neighbor Discovery (ND) [RFC-4861] or SEcure Neighbor Discovery (SEND) [RFC-3971].

9.1.2 IPv6 Extension Headers

IPv6 header is relatively simple (compared to IPv4), because features that are rarely used or less desirable are removed. However, some of these features occasionally are still needed, so IPv6 has introduced the concept of an optional *extension header*. These headers can be supplied to provide extra information, and are placed between the IPv6 header and the upper-layer header in a packet. Extension headers allow the extension of the protocol if required by new technologies or applications. Six kinds of extension headers are defined at present (Table 9-1). Each one is optional, and if present, each is identified by the Next Header field of the preceding header. If more than one is present, they must appear directly after the main header, and preferably in the

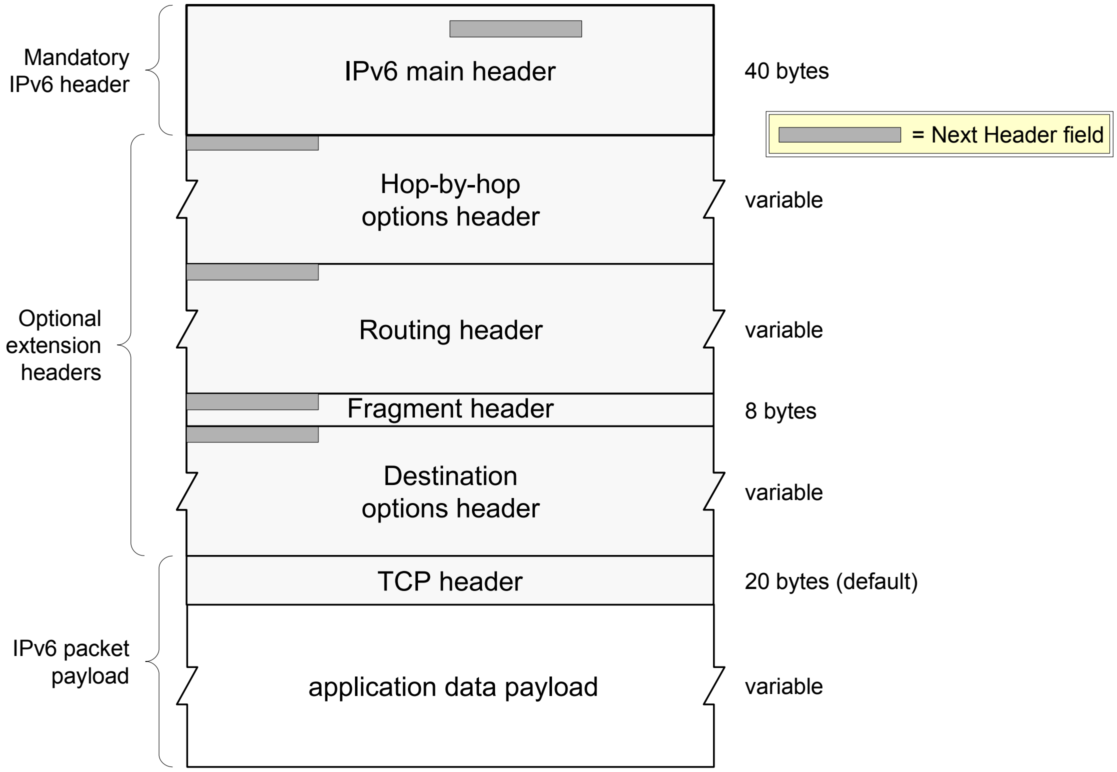


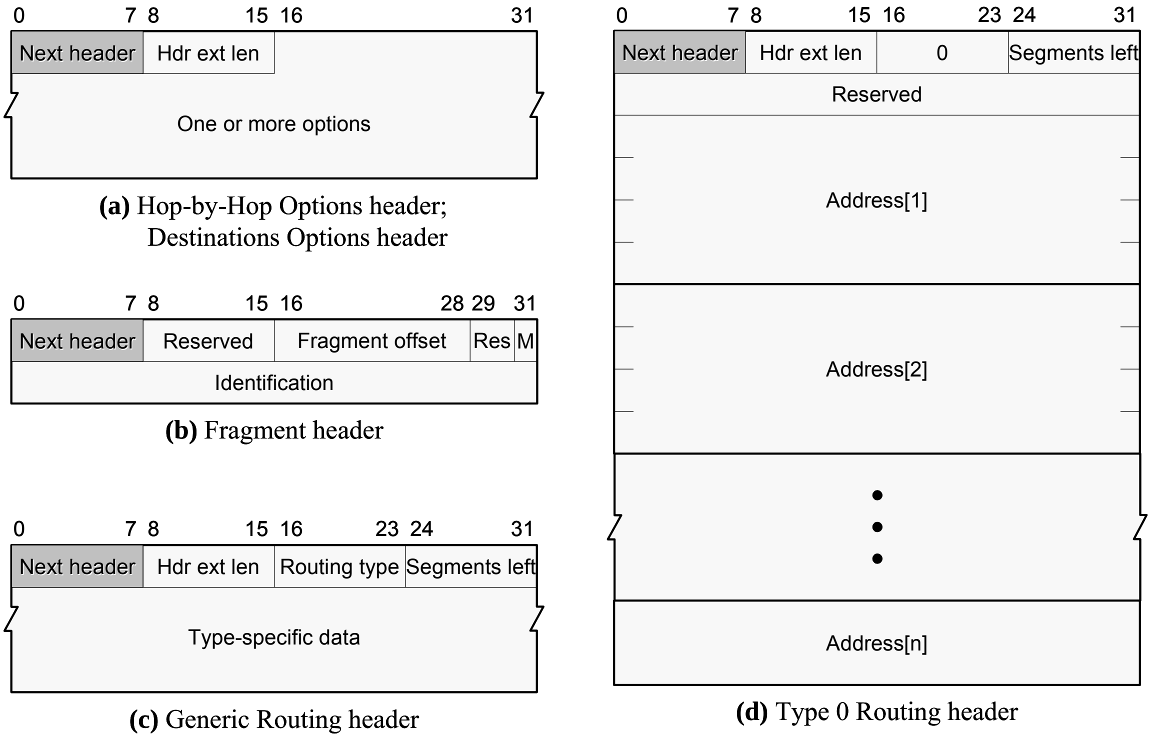
Figure 9-4: Example of IPv6 extension headers.

order shown in Table 9-1. After the extension headers follows the upper-layer header (e.g., TCP header), which is the header of the payload contained in this IPv6 datagram.

Figure 9-4 shows an example of an IPv6 datagram that includes an instance of each extension header, except those related to security. Note that the main IPv6 header and each extension header include a Next Header field. This field identifies the type of the immediately following header. If the next header is an extension header, then this field contains the type identifier of that header. Otherwise, this field contains the identifier of the upper-layer protocol to which the datagram will be delivered. In the latter case, the same values are used as for the IPv4 Protocol field. In the example in Figure 9-4, the upper-layer protocol is TCP and the payload carried by this IPv6 datagram is a TCP segment.

With one exception, extension headers are not examined or processed by any node along a packet's delivery path, until the packet reaches the node (or, in the case of multicast, each of the set of nodes) identified in the Destination Address field of the IPv6 header. There, regular demultiplexing on the Next Header field of the IPv6 header invokes the module to process the first extension header, or the upper-layer header if no extension header is present. The contents and semantics of each extension header determine whether or not to proceed to the next header. Therefore, extension headers must be processed strictly in the order they appear in the packet. A receiver must not, for example, scan through a packet looking for a particular kind of extension header and process that header prior to processing all preceding ones.

The exception referred to in the preceding paragraph is the Hop-by-Hop Options header, which carries information that must be examined and processed by every node along a packet's delivery path, including the source and destination nodes. The Hop-by-Hop Options header, when present,



(a) Hop-by-Hop Options header; Destinations Options header

(b) Fragment header

(c) Generic Routing header

(d) Type 0 Routing header

Figure 9-5: Format of IPv6 extension headers.

must immediately follow the main IPv6 header. Its presence is indicated by the value zero in the Next Header field of the main IPv6 header.

Hop-by-Hop Options Header

The Hop-by-Hop Options header carries optional information for the routers that will be visited by this IPv6 datagram. This header must be examined by every node along a packet's delivery path. The Hop-by-Hop Options header is identified by a Next Header value of 0 in the main IPv6 header, and has the following format (Figure 9-5(a)):

- **Next Header (8 bits):** Identifies the type of header immediately following the Hop-by-Hop Options header. Uses the same values as the IPv4 Protocol field.
- **Hdr Ext Len (8-bits):** Length of the Hop-by-Hop Options header in 64-bit units, not including the first 64 bits.
- **Options:** A variable-length field, of length containing one or more options, such that the complete Hop-by-Hop Options header is long an integer multiple of 64-bits. Each option is defined by three sub-fields: Option Type (8 bits), which identifies the option; Length (8 bits), which specifies the length of the Option Data field (in bytes); and Option Data, which is a variable-length specification of the option.

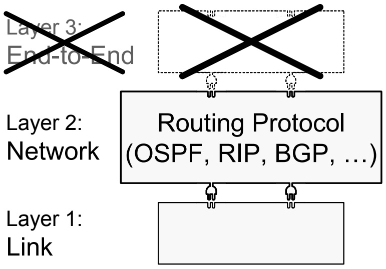
If, as a result of processing a header, a node is required to proceed to the next header but the Next Header value in the current header is unrecognized by the node, it should discard the packet and send an ICMP Parameter Problem message to the source of the packet, with an ICMP Code value of 1 ("unrecognized Next Header type encountered") and the ICMP Pointer field containing the offset of the unrecognized value within the original packet. The same action should be taken if a node encounters a Next Header value of zero in any header other than an IPv6 header.

9.1.3 Transitioning from IPv4 to IPv6

There are two approaches for gradually introducing IPv6 in the public Internet, which is based on IPv4:

- **Dual-stack approach:** IPv6 nodes also have a complete IPv4 implementation. Such a node, referred to as and IPv6/IPv4 node in RFC-4213, has the ability to send and receive both IPv4 and IPv6 datagrams.
- **Tunneling approach:** Any two IPv6 nodes that are connected via intermediary IPv4 routers (that are not IPv6-capable) create a "tunnel" between them. That is, the sending node takes the entire IPv6 datagram (header and payload included) and puts it in the data (payload) filed of an IPv4 datagram. This datagram is then addressed to the receiving IPv6 node and sent to the first intermediary node in the tunnel.

9.2 Routing Protocols



This section reviews several currently most popular Internet routing protocols.

9.2.1 Routing Information Protocol (RIP)

Routing Information Protocol (RIP) is a distance-vector routing protocol (Section 1.4.3), described in RFC-1058. Similar to OSPF (described next in Section 9.2.2) RIP is also used for routing within individual autonomous domains. Unlike OSPF, which scales to large intranets, RIP is useful for small subnets because of its simplicity of implementation and configuration, where its inadequacies are not prominent. RIP inadequacies include poor dealing with link failures and lack of support for multiple metrics. In addition, unlike OSPF, RIP for IP internetworks cannot be subdivided and no route summarization is done beyond the summarizing for all subnets of a network identifier. As a result, RIP networks are "flat."

The format of RIP version 2 route-advertisement packets is shown in Figure 9-6. The first four bytes of a RIP message contain the RIP header. The Command field is used to specify the purpose of this packet. For example, the possible values include: "1" which means a request for the receiver node to send all or part of its routing table; and "2" which symbolizes a response message containing all or part of the sender's routing table. This message may be sent in response to a request or poll, or it may be an update message generated by the sender.

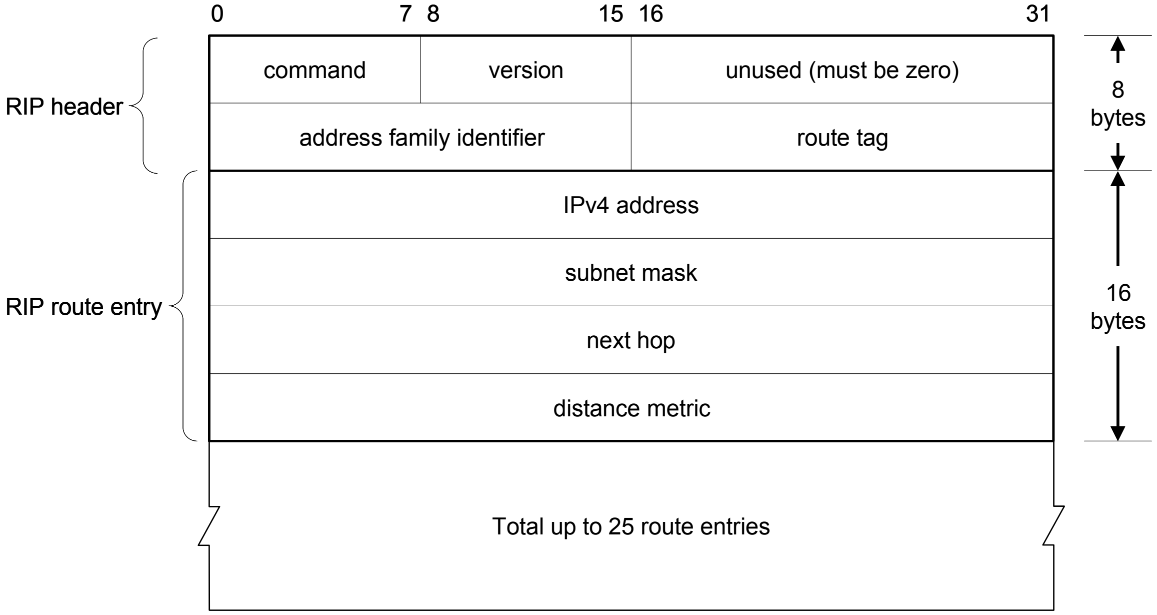


Figure 9-6: Routing Information Protocol (RIP) version 2 packet format for IPv4 addresses.

The Version field of the header specifies version number of the RIP protocol that the sender uses. The value should be “2” for RIP messages that use authentication or carry information in any of the newly defined fields (RIP version 2, defined in RFC-1723). The contents of the Unused field (two bytes) shall be ignored. The Address Family Identifier filed indicates what type of address is specified in the entries. The address family identifier for IPv4 is 4. The Route Tag field is an attribute assigned to a route that must be preserved and readvertised with a route. The intended use of the Route Tag is to provide a method of separating “internal” RIP routes (routes for networks within the RIP routing domain) from “external” RIP routes, which may have been imported from another routing protocol.

The remainder of a RIP message is composed of route entries. There may be between 1 and 25 route entries and each is 20 bytes long. The IP Address is the usual 4-byte IPv4 address. The Subnet Mask field contains the subnet mask that is applied to the IP address to yield the non-host portion of the address (recall Section 1.4.4). If this field is zero, then no subnet mask has been included for this entry.

The Next Hop field identifies the immediate next hop IP address to which packets to the destination (specified by the IP Address of this route entry) should be forwarded. Specifying a value of 0.0.0.0 in this field indicates that routing should be via the originator of this RIP advertisement packet. An address specified as a next hop must be directly reachable on the logical subnet over which the advertisement is made. The purpose of the Next Hop field is to eliminate packets being routed through extra hops in the system. This is particularly useful when RIP is not being run on all of the routers on a network, but some other routing protocols are used, as well. Note that Next Hop is an “advisory” field. That is, if the provided information is ignored, a possibly sub-optimal, but still valid, route may be taken. If the received Next Hop is not directly reachable, it should be treated as 0.0.0.0.

The Metric field of a route entry specifies the distance to the destination node identified by the IP Address of this entry. RIP takes the simplest approach to link cost metrics, a hop-count metric with all link costs being equal to 1. Valid distances are 1 through 15, with 16 defined as infinity. This limits RIP to running on relatively small networks, with no paths longer than 15 hops.

RIP, like most distance vector routing protocols, announces its routes in an unsynchronized and unacknowledged manner. Peer routers exchange distance vectors every 30 seconds, and a router is declared dead if a peer does not hear from it for 180 s, which is the hold-down timer period. RIP uses split horizon with poisoned reverse to tackle the counting-to-infinity problem.

Triggered updates allow a RIP router to announce changes in metric values almost immediately rather than waiting for the next periodic announcement. The trigger is a change to a metric in an entry in the routing table. For example, networks that become unavailable can be announced with a hop count of 16 through a triggered update. Note that the update is sent *almost immediately*, where a time interval to wait is typically specified on the router. If triggered updates were sent by all routers immediately, each triggered update could cause a cascade of broadcast traffic across the IP internetwork.

9.2.2 Open Shortest Path First (OSPF)

Open Shortest Path First (OSPF) is a link-state routing protocol (Section 1.4.2) that is currently the preferred protocol for interior routing—routing within individual *autonomous systems*, i.e., internetworks controlled by a single organization, also known as *intranets*. Autonomous systems (ASs) and inter-domain routing are described in Section 1.4.5.

The router broadcasts its link-state advertisements (LSAs) to *all* other routers in its autonomous system, not just to its neighboring routers. A router broadcasts link-state advertisements whenever there is a change in link status (e.g., outage or changed link cost). It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed. An OSPF cost advertised in LSAs is a unitless metric that indicates the degree of preference for using a link. The network administrator can configure the cost of individual links to represent delay, data rate, monetary cost, or other factors.

Each router gathers the received LSAs into a database called the *link state database* (LSDB). By synchronizing LSDBs between all neighboring routers, each router has each other router's LSA in its database. Therefore, every router has the same LSDB. From the LSDB, entries for the router's routing table are calculated using the algorithm described in Section 1.4.2 to determine the least-cost path, the path with the lowest accumulated cost, to each network in the AS internetwork.

OSPF allows introducing additional level of hierarchy, in addition to autonomous systems. Each AS that runs OSPF can be configured into **areas**, where each area behaves like an independent network. Different areas exchange information via routers that belong to several areas, known as **area-border routers**. Each OSPF area runs its own OSPF protocol, and each router in an area broadcasts its LSAs only to routers within its area and each router's LSDB includes only the state of this area's links. Exactly one OSPF area is configured to act as the **backbone area** that routes traffic between the other areas within the same AS. There must be at least one area-border router in each area, connecting the area to the backbone. Each area-border router maintains several LSDBs, one for each area to which it belongs.

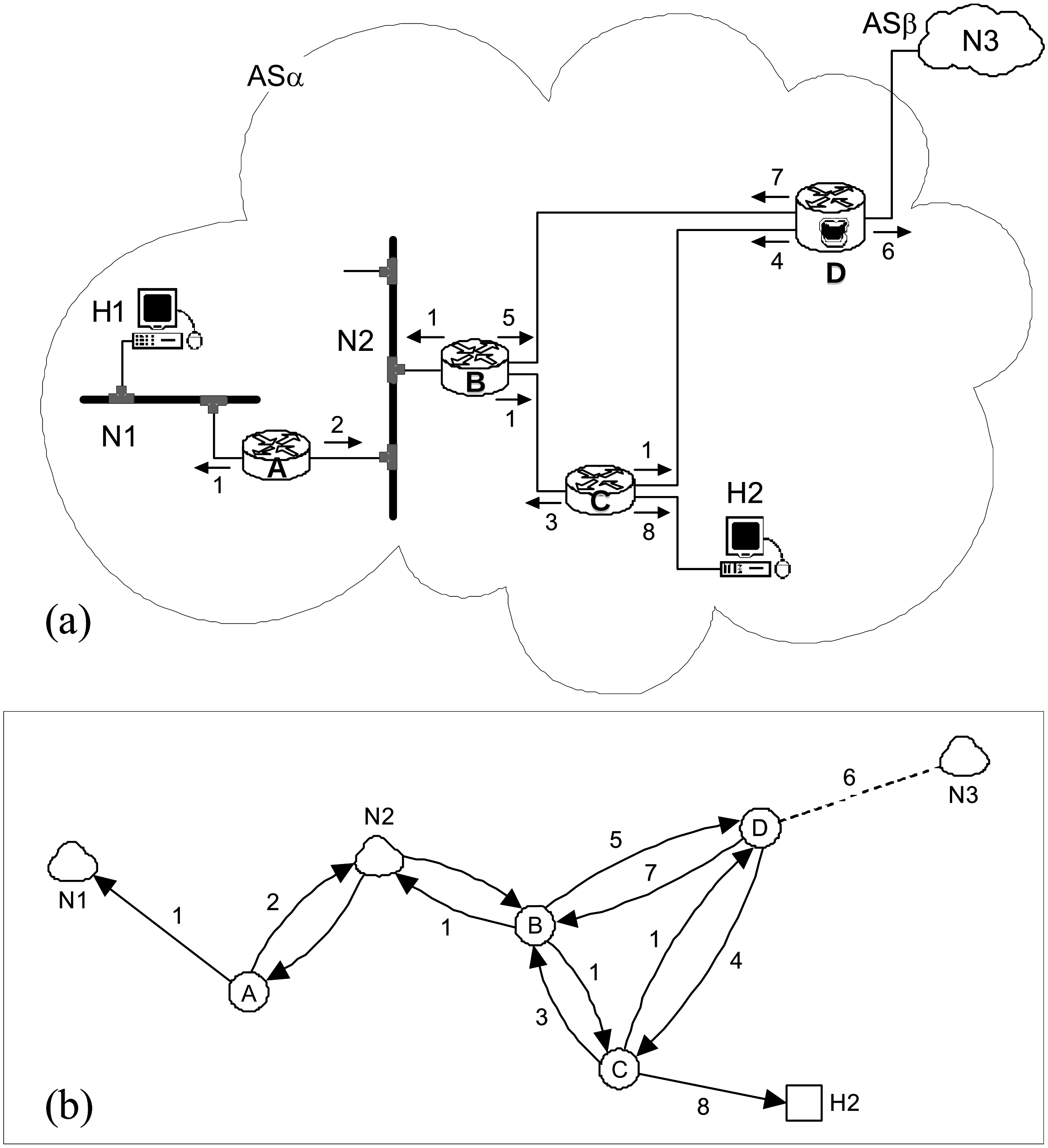


Figure 9-7: (a) Example of an autonomous system running OSPF. (b) Directed graph representation of the same AS. Notice that link costs may be asymmetric for both directions.

OSPF represents network topology as a *directed graph*. An example is shown in Figure 9-7, where autonomous system ASα is running OSPF as its interior gateway protocol. The vertices of the link-state database graph represent *routers* and *networks*. A graph edge connects two routers when they are attached via a physical point-to-point link. An edge connecting a router to a network indicates that the router has an interface on the network. Networks can be either *transit* or *stub* networks. **Transit network** is capable of carrying data traffic that is originated and destined externally to this network. A transit network is represented by a vertex having both incoming and outgoing edges. A stub network's vertex has only incoming edges. For example, in Figure 9-7(b), *N*2 is a transit network and *N*1 is a stub network. The mapping is as follows:

- Two routers connected by a point-to- point link are represented as two router vertices directly connected by a pair of edges, one in each direction. For example, in Figure 9-7 the cost of the edge from router *B* to *C* is "1" and from *C* to *B* is "3."
- When several routers are attached to a broadcast network (transit network), the graph shows all routers bidirectionally connected to the network vertex. For example, in Figure 9-7 network *N*2 has routers *A* and *B* attached and therefore its edges in Figure 9-7(b) are bidirectional.
- If a network has only one attached router (i.e., a stub network), the network appears on the end of a stub connection in the graph. See, for example, network *N*1 in Figure 9-7.
- Hosts attached directly to routers appear on the graph as stub networks. See, for example, network *H*2 in Figure 9-7. (Host *H*1 is not shown in the graph because it is not attached directly to a router.)
- If a router is connected to other autonomous systems (so called "speaker node," see Section 1.4.5), then the cost to each network in the other AS must be obtained from an exterior routing protocol, such as BGP (Section 9.2.3). Such a network is represented as a stub. For example, in Figure 9-7 router *D* is a speaker node and it is connected to network *N*3 in ASβ.

The cost of a link is associated with the output port of each router interface, so each end of the link may see the link cost differently. As already noted, the link cost is configurable by the network administrator. Arcs of the link-state database graph are labeled with the cost of the corresponding router output interface, as seen in Figure 9-7. Arcs without labels have a cost of zero. Arcs leading from transit networks to routers always have a cost of 0. See for example arcs from *N*2 to routers *A* and *B* in Figure 9-7(b).

All OSPF messages begin with the same header (Figure 9-8). Current (as of 2010) *Version* of the OSFP protocol is 2. There are five payload *Type*s of OSPF packets, as follows: 1 = Hello; 2 = Database Description; 3 = Link State Request; 4 = Link State Update; 5 = Link State Acknowledgment. The packet *Length* is given in bytes and includes the standard OSPF header. The *Address* identifies the source router of this packet. *Area ID* is a 32-bit number identifying the OSPF routing area to which the source router of this packet belongs. All OSPF packets are associated with a single area. Most travel a single hop only. Packets travelling over a virtual link are labeled with the backbone Area ID of `0.0.0.0`. The *Checksum* field represents the standard IP checksum of the entire contents of the packet, starting with the OSPF packet header but excluding the 64-bit Authentication field. This checksum is calculated as the 16-bit one's complement of the one's complement sum of all the 16-bit words in the packet, excepting the Authentication field. If the packet's length is not an integral number of 16-bit words, the packet is padded with a byte of zero before checksumming. *Authentication Type* identifies the authentication scheme to be used for the packet. Finally, *Authentication* is a 64-bit field for use by the authentication scheme.

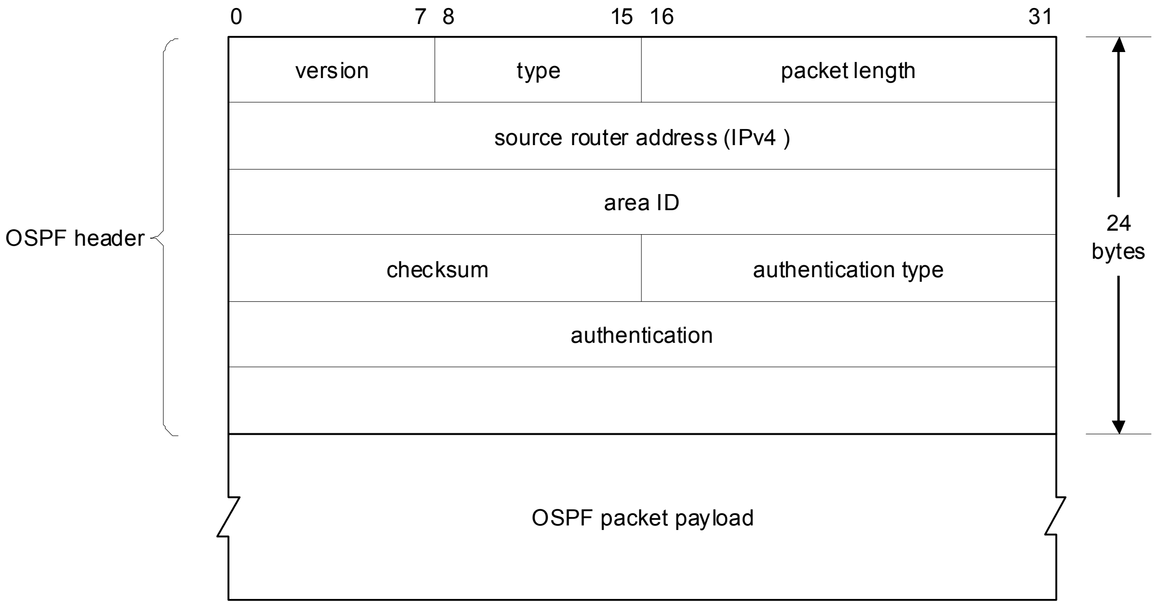


Figure 9-8: OSPF packet format for IPv4 addresses.

Link State Update packets are OSPF packet type 4. These packets implement the flooding of LSAs. Each Link State Update packet carries a collection of LSAs one hop further from their origin. Several LSAs may be included in a single packet. The payload format for type-4 packets is shown in Figure 9-9. There is one common LSA header for all LSA advertisement types, shown in the top part of Figure 9-9. The LSA advertisement type is specified in the *Type* field, see the top row in Figure 9-9.

As seen, these link-state advertisements (LSAa) are more complex than LSAs described in Section 1.4.2 for a basic version of link state routing. The complexity derives from the more complex link-state database graph representation for OSPF (Figure 9-7). For example, a router running OSPF may generate link-state advertisements that advertise one or more networks that are directly connected to this router. A router may also advertise a direct point-to- point link to another router.

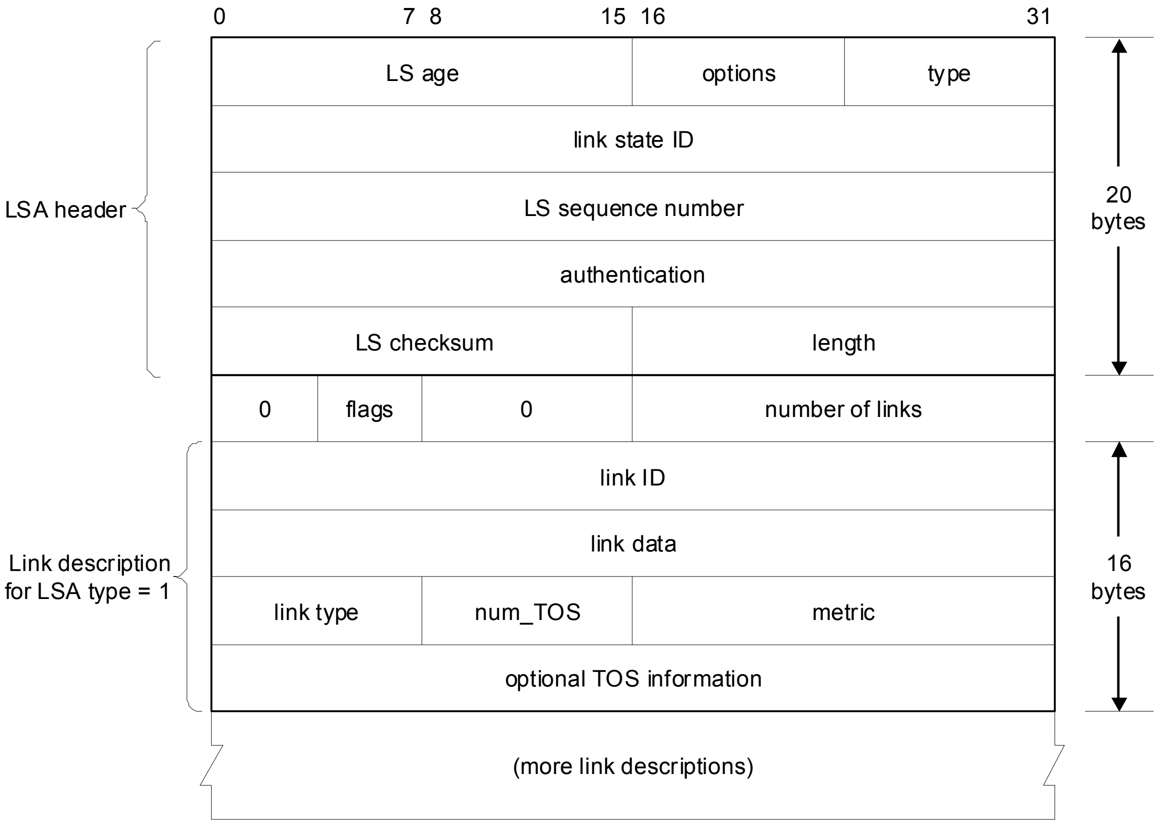


Figure 9-9: OSPF packet payload format for Type=4 packets (Link State Update packets). These packets carry one or more Link State Advertisements (LSAs). The bottom part shows the format of link description for Type 1 LSAs (specified in the LSA header in the top row).

OSPF has fast convergence rate. It can detect and propagate topology changes faster than a distance-vector routing protocol and it does not suffer from the counting-to-infinity problem (described in Section 1.4.3). OSPF-calculated routes are always loop-free. With OSPF, an autonomous system can be subdivided into contiguous groups of networks called areas. Routes within areas can be summarized to minimize route table entries. Areas can be configured with a default route summarizing all routes outside the AS or outside the area. As a result, OSPF can scale to large and very large internetworks.

9.2.3 Border Gateway Protocol (BGP)

Section 1.4.5 presented the key challenges that arise due to independent administrative entities that compete for profit to provide global Internet access. The key questions for an Autonomous System (AS) can be summarized as follows:

- What routing information to advertise to other ASs; how to process the routing information received from other ASs; and, what of the received information to readvertise?
- How can an AS achieve a consistent picture of the Internet viewed by all of its routers, so that for a given data packet each router would make the same forwarding decision (as if each had access to the routing tables of all the border routers within this AS)?

The inter-AS (or, external gateway routing protocol) routing protocol needs to decide whether to forward routing advertisement packets (import/export policies) and whether to disseminate reachability of neighboring ASs at the risk of having to carry their transit traffic unrecompensed.

Border Gateway Protocol (BGP) is an inter-Autonomous System routing protocol that addresses the above requirements. BGP is extremely complex and many issues about its operation are still not well understood. The main complexity of an external routing is not in the protocol for finding routes. Rather, the complexity lies in how border gateways (or, "BGP speakers") are configured to implement the business preferences, and in how external routes are learned from other ASs are disseminated internally within an AS. As will been seen later, there are two keys for this capability: (1) provider's *filtering policies* for processing and redistributing the received route advertisements, which are kept confidential from other ASes; and, (2) BGP *path attributes* that are included in route announcements and used when applying the local filtering policies.

BGP is a *path-vector routing protocol* (Section 1.4.5), where distance vectors are annotated not only with the entire path used to compute each distance, but also with path attributes that describe the advertised paths to destination prefixes. For example, the attributes include preference values assigned to an advertised path by the routers through which this advertisement passed. Unlike, distance-vector, path-vector routing converges quickly to correct paths and guarantees freedom from loops. However, there is an issue of large routing tables needed for path-vector routing. We will see later how BGP addresses this issue.

Routing Between and Within Autonomous Systems

In Section 1.4.5 we saw that the inter-AS routing protocol must exchange routing advertisements between different domains, as well as disseminate the received information within its own AS.

BGP routers use TCP (Chapter 2) on a well-known port (179) to communicate with each other, instead of layering the routing message directly over IP, as is done in other Internet routing protocols. Interior Gateway Protocols (IGPs), such as RIP (Section 9.2.1) and OSPF (Section 9.2.2) rely on periodic updates that carry the entire routing tables of the sending routers containing all active routes. Unlike this, BGP sends only **incremental updates** containing only the routing entries that have changed since the last update (or transmission of all active routes). TCP ensures reliable delivery and simplifies the error management in the routing protocol. However, routing updates are subject to TCP congestion control, which can lead to complicated network dynamics and performance problems. For example, routing updates might be delayed waiting for TCP sender to time out.

BGP neighbors, or peers, are established by manual configuration between routers to create a TCP session on port 179. Because TCP is a connection-oriented protocol with precisely identified endpoints, each BGP router maintains a separate TCP session with each other BGP router to which it is connected. There is typically one such BGP TCP connection for each link that directly connects two speaker routers in different ASs. There are also TCP connections between the speaker routers within the same AS (if there is more than one speaker within the AS), known as *internal peering*. Unlike BGP speakers in different ASs that are typically directly connected at the link layer, BGP speakers within the same AS are usually connected via non-speaker routers (i.e., at the network layer). For each TCP connection, the two routers at each end are called **BGP peers** and the TCP connection over which BGP messages are sent is called a **BGP session**. A BGP

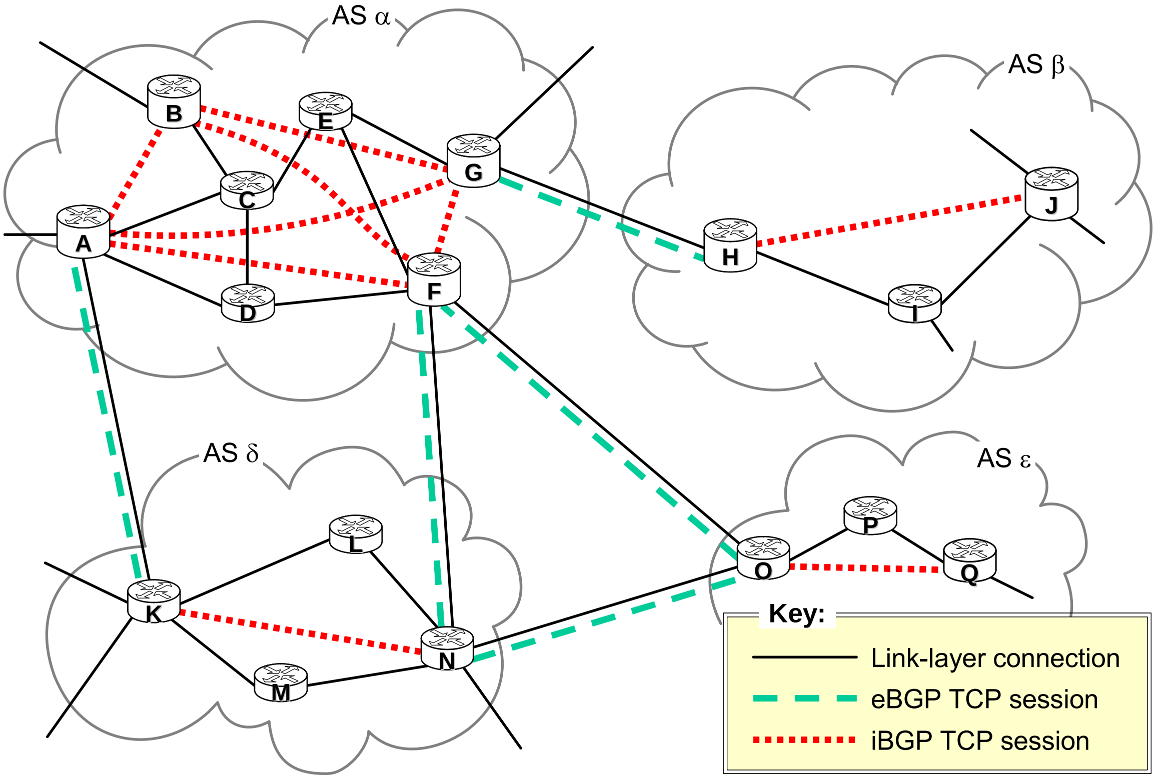


Figure 9-10: Example of eBGP and iBGP sessions.

session that spans two ASs is called an **external BGP (eBGP) session**; a BGP session between two routers within the same AS is called an **internal BGP (iBGP) session**. Recall from the discussion in Section 1.4.5 that the purpose of iBGP sessions is to ensure that network reachability information is consistent among the BGP speakers in the same AS. A BGP speaker can easily decide whether to open eBGP or iBGP session by comparing the Autonomous System Number (ASN) of the other router with its own. Figure 9-10 shows a detail from Figure 1-58 with eBGP and iBGP sessions.

As seen in Figure 9-10 for ASα, all iBGP peers must be fully connected to one another because each TCP session connects only one pair of endpoints. Full mesh connectivity (where everyone speaks to everyone directly) ensures that all the BGP routers in the same AS to exchange routing information and ensure that network reachability information is consistent among them. Given n BGP routers, the full mesh connectivity requires $n/2 \times (n - 1)$ iBGP sessions, which may be large for a large n. In addition, each BGP router will run at least one eBGP session (routers F, N, and O in Figure 9-10 run two each). Large number of sessions may degrade performance of routers, due either to high memory or processing requirements. Methods such as *confederations* (RFC-5065) and *route reflectors* (RFC-4456) help improve the scalability of iBGP sessions.

The BGP finite state machine (FSM) consists of six states (Figure 9-11): Idle, Connect, Active, OpenSent, OpenConfirm, and Established. To start participating in a *BGP session* with another router, a router first sets up a TCP connection on the BGP port 179 (states Connect and Active). If successful, the routers next exchange `OPEN` messages (states OpenSent and OpenConfirm). During the `OPEN` exchanges, BGP routers negotiate optional capabilities of the session, including multiprotocol extensions and various recovery modes.

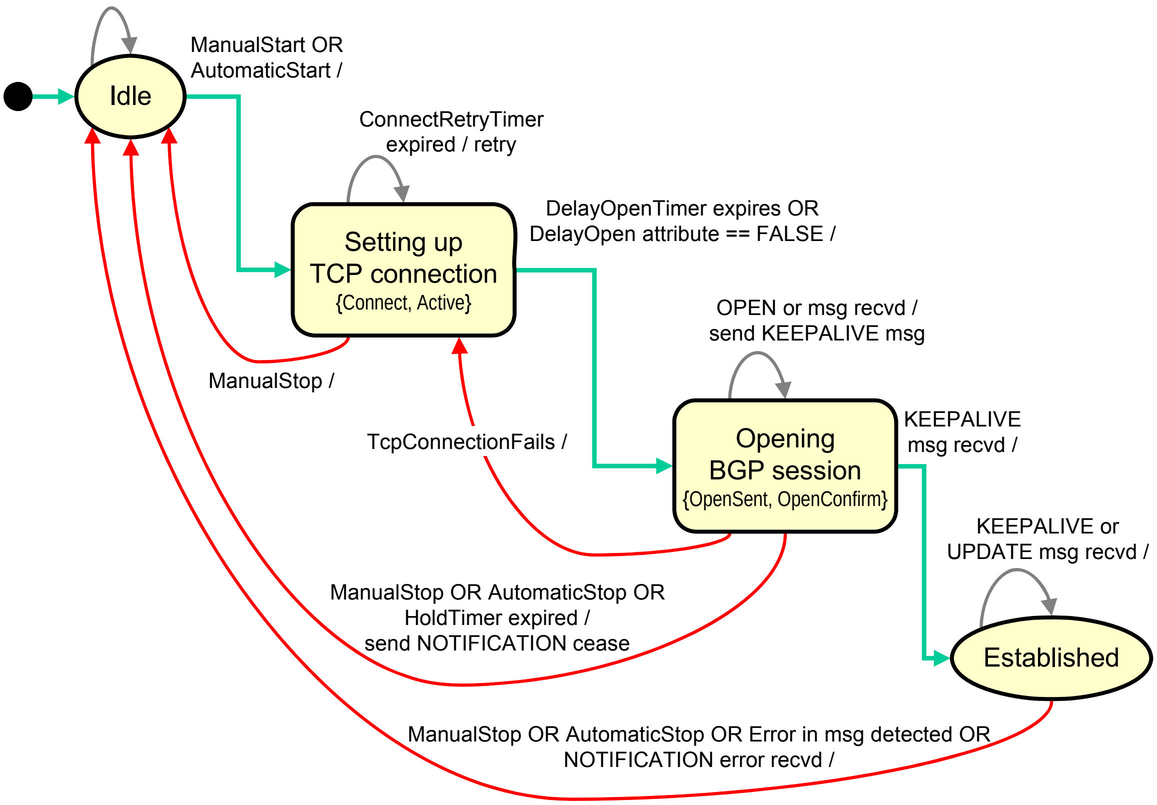
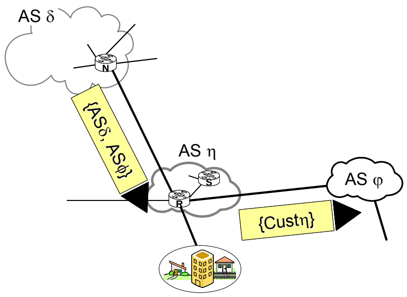


Figure 9-11: Finite state machine of the BGP4 protocol (highly simplified).

After the OPEN is completed and a BGP session is running, the BGP speakers transition to the Established state. They exchange UPDATE messages about destinations to which they offer connectivity (routing tables to all active routes). All subsequent UPDATE messages incrementally announce only routing entries that have changed since the last update. There are two kinds of updates: reachability *announcements*, which are changes to existing routes or new routes, and *withdrawals* of prefixes to which the speaker no longer offers connectivity (because of network failure or policy change). Both positive and negative reachability information can be carried in the same UPDATE message. In the protocol, the basic CIDR route description is called Network Layer Reachability Information (NLRI). NLRI includes the destination prefix, prefix length, path vector of the traversed autonomous systems and next hop in attributes, which can carry a wide range of additional information that affects the import policy of the receiving router.



The exchanged routing tables are not necessarily the exact copies of their actual routing tables, because each router first applies the logic rules that implement its **export policy**. If a BGP speaker has a choice of several different routes to a destination, it will choose the best one according to its own local policies, and then that will be the route it advertises. Of course, a BGP speaker is not obliged to advertise any route to a destination, even if it knows one. For example, in Figure 1-58 ASη refuses to provide transit service to its peers and does not readvertise the destination in ASϕ that it learned about from ASδ.

Each router integrates the received information to its routing table according to its **import policy** (or, acceptance policy). The rules defining the import policy are specified using attributes such as `LOCAL_PREF` and `WEIGHT`. These attributes are locally exchanged by routers in an AS (using iBGP), but are not disclosed to other ASs (using eBGP). A BGP speaker calculates the degree of preference for each external route based on the locally-configured policy, and includes the degree of preference when advertising a route to its internal peers. A receiving BGP speaker uses the degree of preference learned via `LOCAL_PREF` in its decision process and favors the route with the highest degree of preference. The rules for BGP route selection are summarized at the end of this section in Table 9-3.

BGP Messages

All BGP messages begin with a fixed-size header, 19-bytes long, that identifies the message type (Figure 9-12(a)). A description of the header fields is as follows:

Marker: This 16-byte field is included for backwards compatibility; it *must* be set to all ones, unless when used for security purposes.

Length: This 2-byte unsigned integer indicates the total length of the message, including the header, in bytes (or, octets). The value of the Length field *must* always be between 19 and 4096, and may be further constrained, depending on the message type.

Type: This 1-byte unsigned integer indicates the type code of the message. BGP defines four type codes: {1 = `OPEN`; 2 = `UPDATE`; 3 = `NOTIFICATION`; 4 = `KEEPALIVE`}.

The BGP message types are discussed next.

- **`OPEN` Messages (Figure 9-12(b))**

After a TCP connection is established, the first message sent by each side is an `OPEN` message (Figure 9-12(b)). If the `OPEN` message is acceptable, a `KEEPALIVE` message confirming the `OPEN` is sent back. A description of the message fields is as follows:

Version: Indicates the BGP protocol version number of the message; currently it is 4.

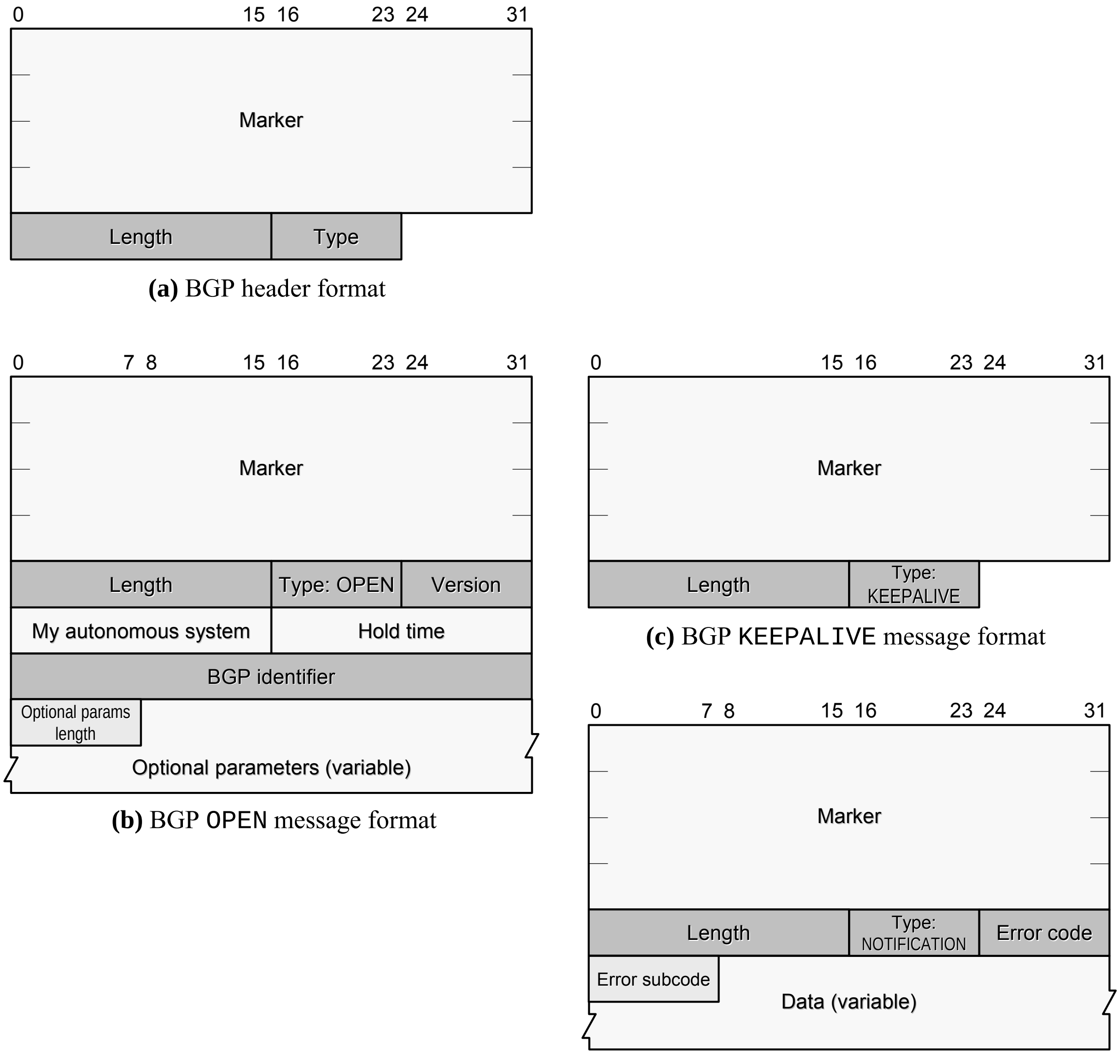
My Autonomous System: Indicates the Autonomous System number of the router that sent this message.

Hold Time: Indicates the proposed interval between the successive `KEEPALIVE` messages (in seconds). The receiving router calculates the value of the Hold Timer by using the smaller of its configured Hold Time and the Hold Time received in this `OPEN` message. A Hold Time value of zero indicates that `KEEPALIVE` messages will not be exchanged at all; otherwise, the minimum value is three seconds.

BGP Identifier: Identifies the sending BGP router. The value of the BGP Identifier is determined upon startup and is the same for every local interface and BGP peer.

Optional Parameters Length: Indicates the total length of the Optional Parameters field in bytes. If the value of this field is zero, no Optional Parameters are present.

Optional Parameters: Contains a list of optional parameters, in which each parameter is encoded in TLV format <*Type*, *Length*, *Value*>. Parameter Type is a 1-byte field that unambiguously identifies individual parameters. Parameter Length is a 1-byte field that contains



(a) BGP header format

(b) BGP `OPEN` message format

(c) BGP `KEEPALIVE` message format

(d) BGP `NOTIFICATION` message format

Figure 9-12: Format of BGP headers and messages, except for `UPDATE` (Figure 9-13).

the length of the Parameter Value field in bytes. Parameter Value is a variable length field that is interpreted according to the value of the Parameter Type field.

The minimum length of an `OPEN` message is 29 bytes (including the message header).

- **`KEEPALIVE` Messages (Figure 9-12(c))**

BGP does not use any TCP-based, keep-alive mechanism to determine if peers are reachable. Instead, `KEEPALIVE` messages are exchanged between peers at a rate that prevents the Hold Timer from expiring. A recommended time between successive `KEEPALIVE` messages is one-third of the Hold Time interval. `KEEPALIVE` messages must not be sent more frequently than one per second. If the negotiated Hold Time interval is zero, then periodic `KEEPALIVE` messages will not be sent.

A `KEEPALIVE` message (Figure 9-12(c)) consists of only the message header and has a total length of 19 bytes.

Table 9-2: BGP `NOTIFICATION` message error codes and subcodes.

Code	Description	Subcodes (if present)	
1	Message header error	1 – Connection not synchronized	3 – Bad message type
		2 – Bad message length	
2	`OPEN` message error	1 – Unsupported version number	4 – Unsupported optional parameter
		2 – Bad peer AS	5 – Deprecated
		3 – Bad BGP identifier	6 – Unacceptable Hold Time
3	`UPDATE` message error	1 – Malformed attribute list	7 – Deprecated
		2 – Unrecognized well-known attribute	
		3 – Missing well-known attribute	8 – Invalid `NEXT_HOP` attribute
		4 – Attribute flags error	9 – Optional attribute error
		5 – Attribute length error	10 – Invalid Network field
		6 – Invalid `ORIGIN` attribute	11 – Malformed `AS_PATH`
4	Hold timer expired		
5	Finite state machine error		
6	Cease		

- **`NOTIFICATION` Messages (Figure 9-12(d))**

A `NOTIFICATION` message is sent when an error condition is detected. The BGP connection is closed immediately after it is sent. In addition to the fixed-size BGP header, the NOTIFICATION message contains the following fields (Figure 9-12(d)): Error Code, Error Subcode, and Data of variable length. The Error Code indicates the type of error condition, while the Error Subcode provides more specific information about the nature of the reported error (Table 9-2). Each Error Code may have one or more Error Subcodes associated with it. If no appropriate Error Subcode is defined, then a zero (Unspecific) value is used for the Error Subcode field.

The variable-length Data field is used to diagnose the reason for the `NOTIFICATION`. The minimum length of the `NOTIFICATION` message is 21 bytes (including message header).

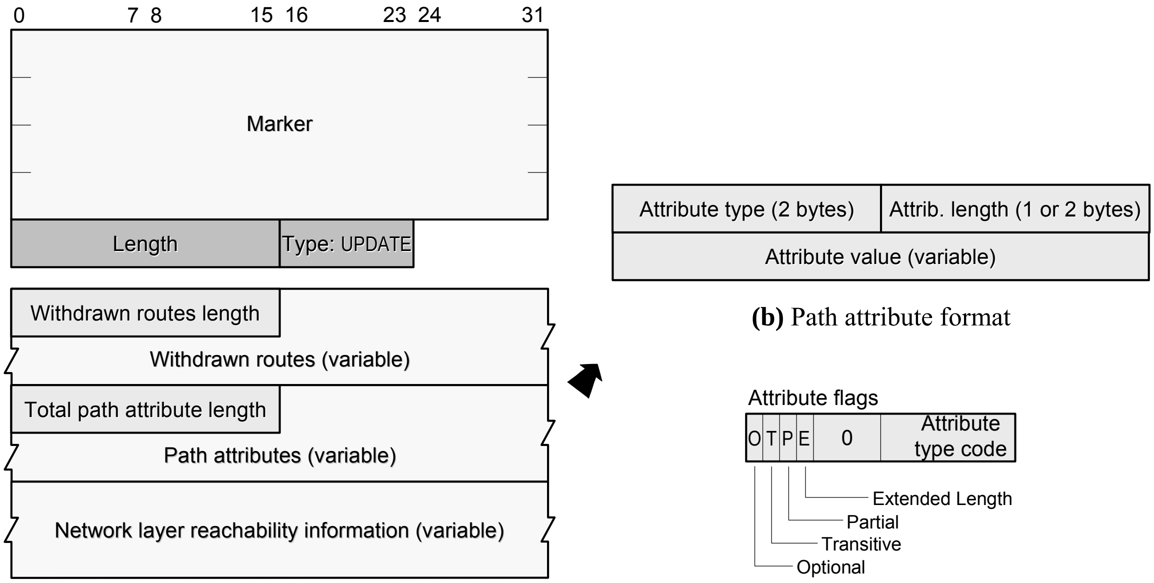
- **`UPDATE` Messages (Figure 9-13)**

After the connection is established, BGP peers exchange routing information by using the `UPDATE` messages. The information in the `UPDATE` messages is used by the path-vector routing algorithm (Section 1.4.5) to construct a graph that describes the connectivity of the Autonomous Systems. By applying logical rules, routing information loops and some other anomalies may be detected and removed from inter-AS routing.

An `UPDATE` message is used to advertise feasible routes that share common path attributes to a peer, and to withdraw multiple unfeasible routes from service. The `UPDATE` message always includes the fixed-size BGP header, and other fields, some of which may not be present in every `UPDATE` message (Figure 9-13(a)).

Withdrawn Routes Length indicates the total length of the Withdrawn Routes field in bytes. A value of zero indicates that no routes are being withdrawn from service, and that the Withdrawn Routes field is not present in this `UPDATE` message.

The Withdrawn Routes field contains a variable-length list of IP-address prefixes for the routes that are being withdrawn from BGP routing tables. Each prefix is encoded as a 2-tuple of the form <*length*, *prefix*>. The *Length* field indicates the length (in bits) of the prefix. A length of


0
7 8
15 16
23 24
31
Marker
Length
Type: UPDATE
Withdrawn routes length
Withdrawn routes (variable)
Total path attribute length
Path attributes (variable)
Network layer reachability information (variable)
Attribute type (2 bytes)
Attrib. length (1 or 2 bytes)
Attribute value (variable)
Attribute flags
O T P E
0
Attribute type code
Extended Length
Partial
Transitive
Optional

(a) BGP UPDATE message format

(b) Path attribute format

(c) Attribute type format

Figure 9-13: Format of BGP UPDATE message.

zero indicates a prefix that matches all IP addresses. The *Prefix* field contains an IP-address prefix, possibly followed by padding bits to make the field length a multiple of 8 bits.

Total Path Attribute Length indicates the total length of the Path Attributes field in bytes. A value of zero indicates that neither the Network Layer Reachability Information field nor the Path Attribute field is present in this UPDATE message.

A BGP router uses the *Path Attributes* and *Network Layer Reachability Information* (NLRI) fields to advertise a route. The NLRI field contains a list of IP-address prefixes that can be reached by this route. The NLRI is encoded as one or more 2-tuples of the form <*length*, *prefix*>. This is the path-vector information used by the path-vector routing algorithm (Section 1.4.5).

A variable-length sequence of Path Attributes is present in every UPDATE message, except for an UPDATE message that carries only the withdrawn routes. Each path attribute is a triple <*type*, *length*, *value*> of variable length (Figure 9-13(b)).

Attribute Type field that consists of the Attribute Flags byte, followed by the Attribute Type Code byte (Figure 9-13(c)). The high-order bit (bit 0) of Attribute Flags is the Optional bit. It defines whether the attribute is optional (if set to 1) or well-known (if set to 0). The second bit is the Transitive bit. It defines whether an optional attribute is transitive (if set to 1) or non-transitive (if set to 0). For well-known attributes, the Transitive bit must be set to 1. The third bit is the Partial bit that defines whether the information contained in the optional transitive attribute is partial (if set to 1) or complete (if set to 0). For well-known attributes and for optional non-transitive attributes, the Partial bit must be set to 0. The fourth bit of Attribute Flags is the Extended Length bit. It defines whether the following Attribute Length field is one byte (if set to 0) or two bytes (if set to 1). The lower-order four bits of Attribute Flags are unused. They are set to zero by the sender, and ignored by the receiver.

The Attribute Type Code field contains the Attribute Type Code. Attribute Type Codes defined in RFC-4271 are discussed next.

BGP Path Attributes

This section discusses the path attributes of the UPDATE message (Figure 9-13). A BGP route announcement (UPDATE message) has a set of attributes associated with each destination prefix. Path attributes can be classified as: (1) well-known mandatory; (2) well-known discretionary; (3) optional transitive; and, (4) optional non-transitive. A BGP router must recognize all well-known attributes. Some of these attributes are mandatory and must be included in every UPDATE message that contains Network Layer Reachability Information (NLRI). Others are discretionary and may or may not be sent in a particular UPDATE message. Once a BGP peer has updated any well-known attributes, it must pass these attributes to its peers in any updates it transmits.

In addition to well-known attributes, each path may contain one or more optional attributes. It is not required or expected that all BGP implementations support all optional attributes. The handling of an unrecognized optional attribute is determined by the value of the Transitive flag (Figure 9-13(c)).

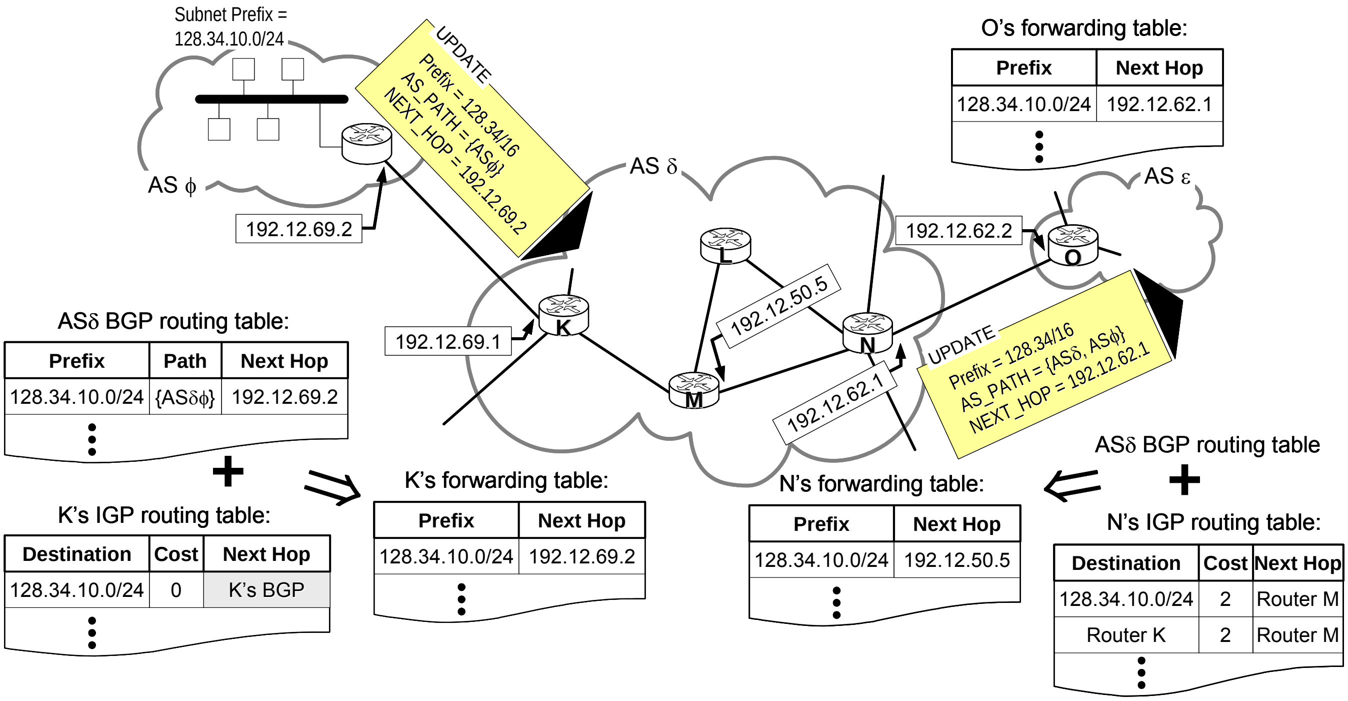
• **ORIGIN (type code 1)** is a well-known mandatory attribute. The ORIGIN attribute describes how BGP at the origin AS came to know about destination addresses aggregated in a prefix. The allowed values are: code 1 (IGP) means that the prefix was learned from an interior gateway protocol (IGP); code 2 (EGP) means that the prefix was learned from an exterior gateway protocol; and, code 3 (INCOMPLETE) represents another source, usually a manually configured static route. The value of ORIGIN should not be changed by any other speaker.

• **AS_PATH (type code 2)** is a well-known mandatory attribute. This attribute lists the autonomous systems through which this UPDATE message has traversed (in reverse order). This list is called a **path vector** and hence BGP is a *path vector protocol* (Section 1.4.5). Every router through the message passes prepends its own AS number (ASN) to AS_PATH and propagates the UPDATE message on (subject to its route filtering rules). An example is shown in Figure 1-58 and a detail in Figure 9-14. When a given BGP speaker advertises the route to an internal peer (over iBGP), the advertising speaker does not modify the AS_PATH attribute.

BGP uses the AS_PATH attribute to detect a potential routing loop. When an external UPDATE message is received (over eBGP), if the ASN of this BGP speaker is already contained in the path vector, then the route should be rejected.

The speaker that modifies AS_PATH may prepend more than one instance of its own ASN in the AS_PATH attribute. This is controlled via local configuration.

• **NEXT_HOP (type code 3)** is a well-known mandatory attribute. It defines the IP address of the next-hop speaker to the destinations listed in the UPDATE message (in NLRI). As the UPDATE message propagates across an AS boundary, the NEXT_HOP attribute is changed to the IP address of the speaker from which this announcement was received (Figure 9-14). (The reader should check RFC-4271 about detailed rules for modifying the NEXT_HOP attribute.)



O's forwarding table:

Prefix	Next Hop
128.34.10.0/24	192.12.62.1
⋮	

ASδ BGP routing table:

Prefix	Path	Next Hop
128.34.10.0/24	{ASδϕ}	192.12.69.2
⋮		

K's IGP routing table:

Destination	Cost	Next Hop
128.34.10.0/24	0	K's BGP
⋮		

K's forwarding table:

Prefix	Next Hop
128.34.10.0/24	192.12.69.2
⋮	

N's forwarding table:

Prefix	Next Hop
128.34.10.0/24	192.12.50.5
⋮	

N's IGP routing table:

Destination	Cost	Next Hop
128.34.10.0/24	2	Router M
Router K	2	Router M
⋮		

Figure 9-14: Example of BGP UPDATE message propagation, originating from ASϕ.

When sending UPDATE to an internal peer, if the route is originated externally to this AS, the BGP speaker should not modify the NEXT_HOP attribute unless it has been explicitly configured to announce its own IP address as the NEXT_HOP. This is because UPDATE messages to internal peers are sent by iBGP over the TCP connection, which runs on top of an IGP. On the other hand, when announcing a route to a local destination within this AS, the BGP speaker should use as the NEXT_HOP the IP address of the first router that announced the route.

As Figure 9-14 shows, all BGP speakers in an AS have the same BGP routing table (ASδ BGP routing table is shown). The forwarding table is created based on the AS BGP routing table and the IGP routing table (which is, naturally, different for each router).

The immediate next-hop address is determined by performing a recursive route lookup operation for the IP address in the NEXT_HOP attribute, using the contents of the Routing Table, selecting one entry if multiple entries of equal cost exist. The Routing Table entry that resolves the IP address in the NEXT_HOP attribute will always specify the outbound interface. If the entry specifies an attached subnet, but does not specify a next-hop address, then the address in the NEXT_HOP attribute should be used as the immediate next-hop address. If the entry also specifies the next-hop address, this address should be used as the immediate next-hop address for packet forwarding.

- **MULTI_EXIT_DISC (type code 4)** is an optional non-transitive attribute that is intended to be used on external (inter-AS) links to discriminate among multiple exit or entry points to the same neighboring AS. To motivate the need for this attribute, consider the example in Figure 9-15 (extracted from Figure 1-58), where Autonomous Systems ASα and ASδ are linked at multiple points. Suppose that router *A* in ASα receives a data packet from ASχ that is destined for ASη. ASα would prefer to get rid of the packet in a hurry ("hot-potato" routing) and simply forward it

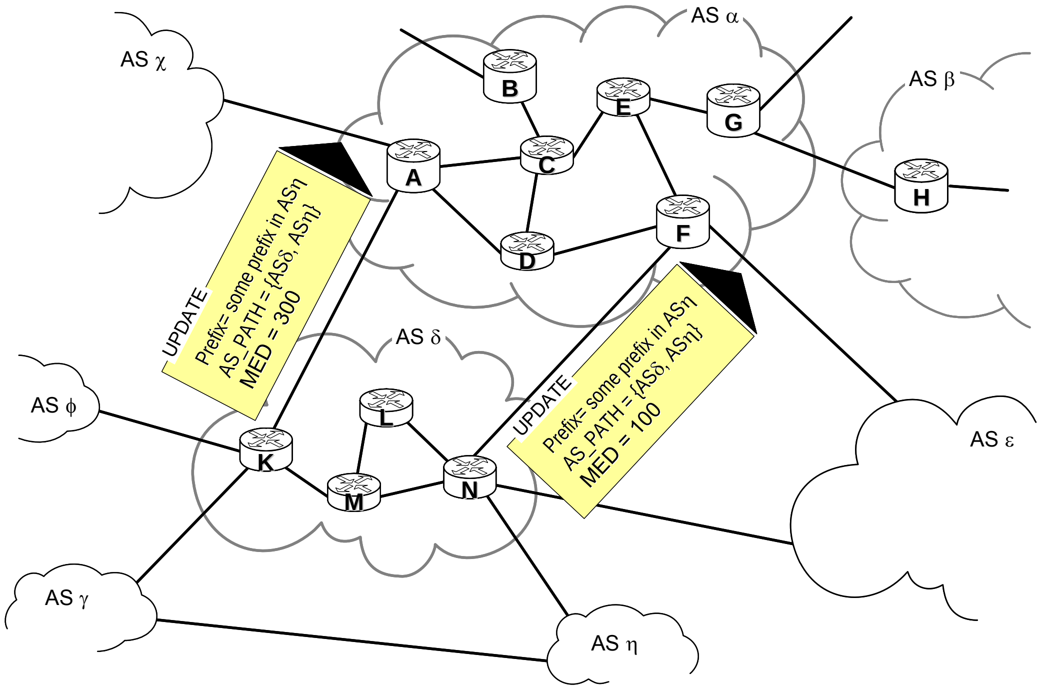


Figure 9-15: Example for BGP `MULTI_EXIT_DISC` (`MED`) attribute.

to router *K* in ASδ. However, ASδ would prefer to receive packets for ASη on the other point (router *N*), because this is less expensive for ASδ—the packet would be immediately forwarded by *N* to ASη instead of traversing ASδ's interior routers. If there were no financial settlement involved, ASα would simply implement its own preferences. However, because ASδ pays ASα for transit service (Figure 1-56), ASα has to honor ASδ's preferences. Earlier we described how a BGP router uses the attribute `LOCAL_PREF` to integrate the received routing advertisement into its routing table. However, because `LOCAL_PREF` expresses the local preferences, it is not useful to express another AS's preferences. For this purposed the `MULTI_EXIT_DISCRIMINATOR` (`MED`) attribute is used.

The value of the `MULTI_EXIT_DISC` (`MED`) attribute is a four-byte integer, called a *metric*, and the standard does not prescribe how to choose the `MED` metric. A common practice is to derive the `MED` metric from the local IGP metric. All other factors being equal, the exit point with the lower metric should be preferred. In Figure 9-15, router *N* advertises a prefix in ASη with `MED` = 100, but router *K* advertises the same prefix with `MED` = 300. Based on this, ASα should deliver packets destined for ASη to router *N*. One the other hand, ASδ would prefer to receive destined packets for ASγ or ASφ on router *K*, so it will choose opposite values of `MED` attribute when advertising prefixes in ASγ or ASφ.

In peering relationships between ASs, (Section 1.4.5), the `MED` attribute is usually ignored. The usage of the `MED` attribute becomes complicated when a third AS advertises the same route, because the IGP metrics used by different ASs can be different. In such a case, comparing a `MED` metric received from one AS with another `MED` metric received from another AS makes no sense.

There are three more path attributes (`LOCAL_PREF`, `ATOMIC_AGGREGATE`, and `AGGREGATOR`), and the interested reader should check RFC-4271. Table 9-3 summarizes how a BGP router that learned about more than route to a prefix selects one. The router will select the

Table 9-3: Priority of rules by which BGP speaker selects routes from multiple choices.

Priority	Rule	Comments
1	LOCAL_PREF	E.g., LOCAL_PREF specifies the order of preference as customer > peer > provider If more than one route remains after this step, go to the next step.
2	AS_PATH	Select shortest AS_PATH length (i.e., the list with the smallest number of ASNs, *not* smallest number of hops or lowest delay!)
3	MED	Select the route with the lowest MULTI_EXIT_DISC value, if there is financial incentive involved.
4	IGP path	Select the route for which the NEXT_HOP attribute, for which the cost in the IGP routing table is lowest, i.e., use hot-potato routing.
5	eBGP > iBGP	Select the route which is learned from eBGP over the one learned by iBGP (i.e., prefer the route learned first hand)
6	Router ID	Select the BGP router with the smallest IP address as the next hop.

next hop along the first route that meets the criteria of the logical rule, starting with the highest priority (1) and going down to the lowest priority (6).

9.2.4 Multicast Routing Protocols

Multicast Group Management

Internet Group Management Protocol (IGMP) is used by an end-system to declare membership in particular multicast group to the nearest router(s). IGMP v3 (current) is defined by RFC-3376.

Version 1: Timed-out Leave (Joining Host send IGMP Report; Leaving Host does nothing; Router periodically polls hosts on subnet using IGMP Query; Hosts respond to Query in a randomized fashion)

Version 2: Fast, Explicit Leave (ADDS to Version 1: Group Specific Queries; Leave Group Message; Host sends Leave Group message if it was the one to respond to most recent query; Router receiving Leave Group message queries group.)

Version 3: Per-Source Join (ADDS to Version 2: Group-Source Specific Queries, Reports and Leaves; Inclusion/Exclusion of sources)

Multicast Route Establishment

Protocol Independent Multicast (PIM), RFC-4601 defines Protocol Independent Multicast – Sparse Mode (PIM-SM); RFC-3973 defines Protocol Independent Multicast – Dense Mode (PIM-DM). PIM operates independently of the underlying unicast protocol, such as IS-IS or OSPF. It supports applications that operate with fewer servers transmitting to multiple destinations (called the *dense mode*) or numerous small workgroups operating in different multicast groups (called the *sparse mode*).

Distance Vector Multicast Routing Protocol (DVMRP), defined in RFC-1075. DVMRP is an enhancement of Reverse Path Forwarding (RPF, Section 3.3.2) that: Uses Distance Vector routing packets for building tree; Prunes broadcast tree links that are not used (non-membership reports); Allows for Broadcast links (LANs).

Multicast Forwarding in DVMRP: 1. check incoming interface: discard if not on shortest path to source; 2. forward to all outgoing interfaces; 3. do not forward if interface has been pruned; 4. prunes timeout every minute.

Source-Specific Multicast (SSM), defined in RFC-3569 and RFC-4607.

Multicast Open Shortest Path First (MOSPF); RFC-1584 defines multicast extensions to OSPF.

For independent Autonomous Systems, Border Gateway Multicast Protocol (BGMP) was abandoned due to the lack of support within the Internet service provider community. Multiprotocol Extensions for BGP4, defined in RFC-2858 defines the codes for Network Layer Reachability Information (NLRI) that allow BGP to carry multicast information. This information is used by other (i.e., multicast) protocols for multicast forwarding. Multicast Source Discovery Protocol (MSDP) defined in RFC-3618, can be used to connect together rendezvous points in different PIM sparse mode domains (see RFC-4611 for MSDP deployment scenarios).

Additional information: see RFC-3170 – "IP Multicast Applications: Challenges & Solutions"

An excellent overview of the current state of multicast routing in the Internet is RFC-5110 (January 2008).

9.3 Address Translation Protocols

9.3.1 Address Resolution Protocol (ARP)

In Chapter 1 we saw that different protocol layers use different addressing systems. The network-layer Internet Protocol uses IPv4 or IPv6 addresses that are assigned by an Internet authority. Link-layer protocols (Ethernet and Wi-Fi) use MAC addresses that are assigned by the hardware manufacturer. One may wonder, why cannot we use a single addressing system, e.g., MAC addresses that are assigned to all network interface cards? Computer hardware and software are abstract so everything may seem possible to make. To understand better why we need two (or more) addressing systems, let us look at real-world physical objects, such as vehicles. (See also Sidebar 1.3 in Section 1.4.4.) As shown in Figure 9-16, every vehicle comes with a *vehicle identification number* (VIN) that is assigned by the manufacturer and engraved at several locations in the vehicle. Every vehicle also has a registration plate with a unique number. Both numbers can be considered "addresses" of the car. So, why not to use the VIN number for the registration plates, as well? Because VINs are assigned by the manufacturer and vehicles of the same manufacturer are bought by customers all around the world, it is impossible to embed into

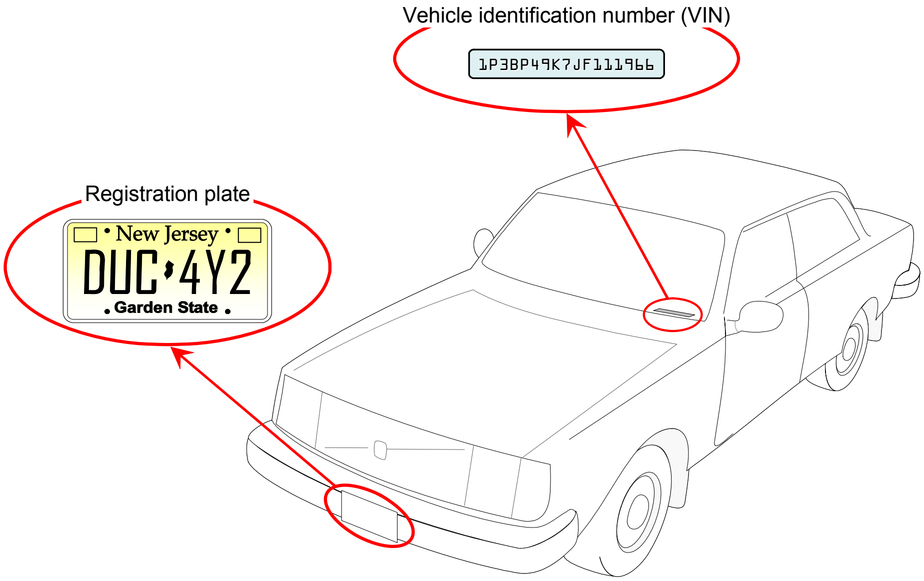


Figure 9-16: Why multiple addressing conventions are necessary.

the address anything specific to a geographic locale or organization that owns cars. Having registration plate number assigned by a local authority makes possible to have for location-specific addresses. Therefore, we need both: the manufacturer needs to be able to distinguish their different products and organizations need location-specific addresses.

A key benefit of location-specific network addresses is the ability to aggregate many addresses by address prefixes, which reduces the amount of information carried in routing advertisements or stored in the forwarding tables of routers.

Let us assume that a node has a packet for a certain destination. The network-layer protocol (IP) at the node looks up the forwarding table and determines the IP address of the next-hop node. Before calling the `send()` method of the link-layer protocol (see Listing 1-1 in Section 1.1.4), the network-layer `send()` must translate the next-hop IP address to the link-layer address of the next-hop node. Recall that Point-to-Point Protocol (PPP, Section 1.5.1) does not use link-layer addresses because it operates over a point-to-point link directly connecting two nodes, one on each end of the link. However, broadcast-based link-layer protocols, such as Ethernet (Section 1.5.2) or Wi-Fi (Section 1.5.3) must use link-layer addresses because many nodes are simultaneously listening on the channel. Because broadcast-based link-layer protocols implement Medium Access Control (MAC), these addresses are called **MAC addresses**. To send the packet to the next-hop node, the node needs a mechanism to translate from a (network-layer) IP address to a (link-layer) MAC address. This is the task of the Address Resolution Protocol.

Address Resolution Protocol (ARP) translates an IP address to a MAC address for a node that is on the same broadcast local-area network (or, subnet). ARP cannot translate addresses for hosts that are not on the same subnet; if such attempt is made, ARP returns an error. When a sender wants a translation, it looks up an **ARP table** on its node, which contains mappings of network-layer IP addresses to MAC addresses.

If the ARP table does not contain an entry for the given IP address, it broadcasts a query **ARP packet** on the LAN (Figure 9-17). (The MAC broadcast address in hexadecimal notation is

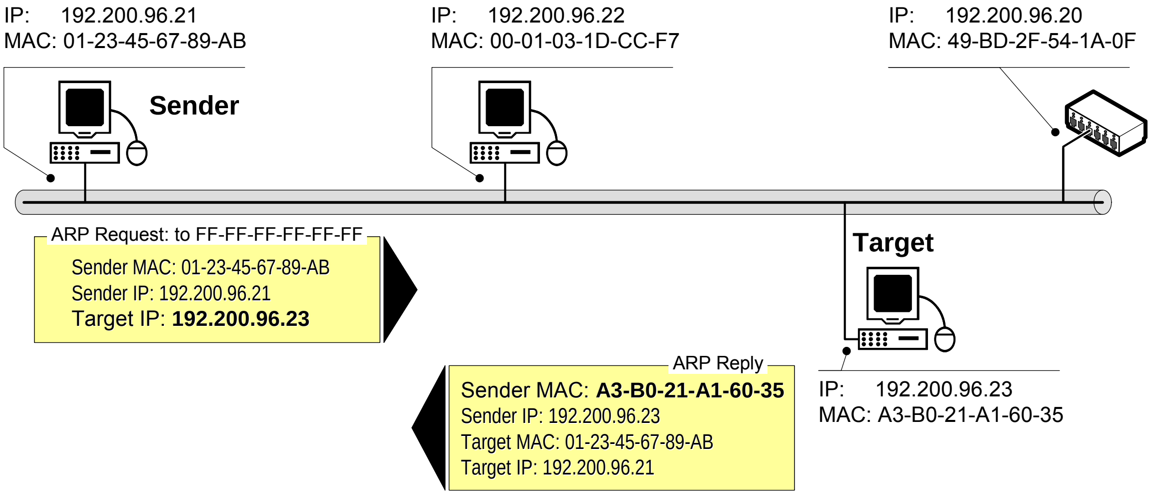


Figure 9-17: ARP request and response example.

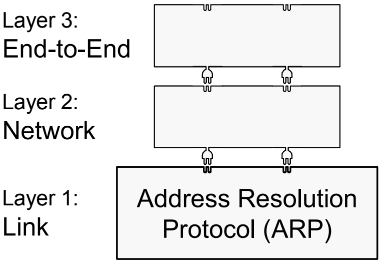
FF-FF-FF-FF-FF-FF.) The node that owns the IP address replies with an ARP packet containing the responder's MAC address. The reply is sent to the querier's MAC address, available from the ARP request.

Figure 9-18 shows the ARP packet format for mapping IPv4 addresses to MAC addresses. In this case, the ARP packet size is 28 bytes. ARP can be used for other kinds of mappings, with different address sizes, as standardized by the IANA (http://iana.org/). The packet fields are:

• Hardware Type: specifies the link-layer protocol type. For example, the code for Ethernet is 1.

• Protocol Type: specifies the upper layer protocol for which the ARP request is intended. For example, IPv4 is encoded as 0x0800.

• Hardware Length: length (in bytes) of a hardware address. Ethernet addresses size is 6 bytes (48 bits).



• Protocol Length: length (in bytes) of a logical address of the network-layer protocol. IPv4 address size is 4 bytes (32 bits).

• Operation: specifies the operation what the sender is performing: 1 for request, 2 for reply.

• Sender Hardware Address: hardware (MAC) address of the sender.

• Sender Protocol Address: upper-layer protocol address of the sender, e.g. IP.

• Target Hardware Address: hardware (MAC) address of the intended receiver. This field is ignored in request operations.

• Target Protocol Address: upper layer protocol address of the intended receiver.

The graphic on the right shows ARP as a link-layer protocol. This is somewhat controversial because ARP uses another link-layer protocol for sending ARP packets, and ARP deals with network-layer, IP addresses. The reason to classify ARP as a link-layer protocol is that it operates over a single link connecting nodes on the same local-area network. Unlike network-layer protocols, it does not span multiple hops and does not send packets across intermediate nodes.

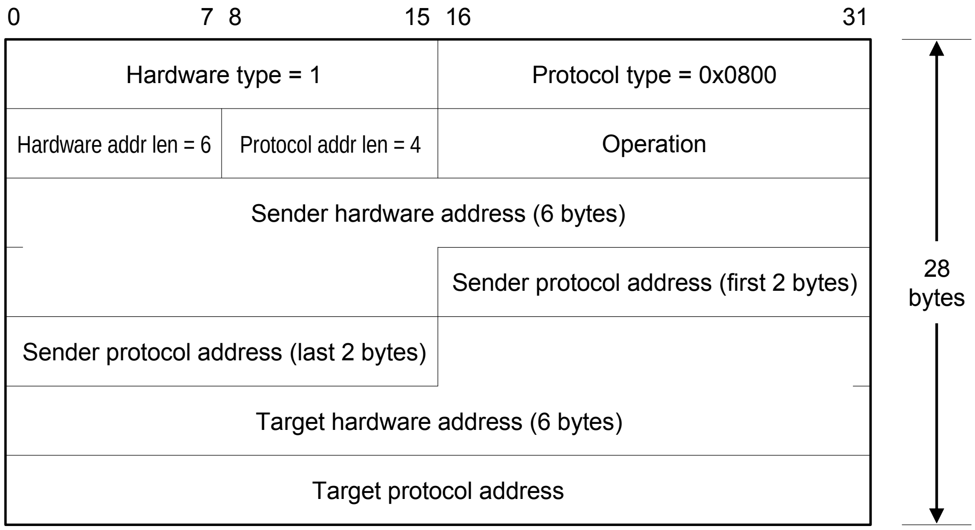

0
7
8
15
16
31
Hardware type = 1
Protocol type = 0x0800
Hardware addr len = 6
Protocol addr len = 4
Operation
Sender hardware address (6 bytes)
Sender protocol address (first 2 bytes)
Sender protocol address (last 2 bytes)
Target hardware address (6 bytes)
Target protocol address
28 bytes

Figure 9-18: ARP packet format for mapping IPv4 addresses into MAC addresses.

ARP solves the problem of determining which MAC address corresponds to a given IP address. Sometimes the reverse problem has to be solved: Given a MAC address, determine the corresponding IP address. An early solution was to use *Reverse ARP* (*RARP*) for this task. RARP is now obsolete and it is replaced by Dynamic Host Configuration Protocol (DHCP), reviewed in the next section.

In the next generation Internet Protocol, IPv6, ARP's functionality is provided by the Neighbor Discovery Protocol (NDP).

9.3.2 Dynamic Host Configuration Protocol (DHCP)

To send a packet on the Internet, a computer must have a network address (IP address). This address is associated with the location of the computer, specifically with the network to which the computer is attached (Section 1.4.4). Dynamic Host Configuration Protocol (DHCP) supports automatic assignment of IP addresses to new hosts, known as plug-and-play. When a new computer is attached to a local-area network, the computer broadcasts a DHCP message asking "Router, give me a network address." The router maintains a pool of free network addresses and assigns one to this computer with a specified time-to-live (say one hour). The computer can then start using the standard Internet applications. As the time-to-live becomes close to zero, the computer asks the router for an extension, which is normally granted. If the user unplugs the computer, there will be no message asking for extension, so the router will return this network address to the pool of free addresses.

9.3.3 Network Address Translation (NAT)

Network Address Translation (NAT) is an Internet Engineering Task Force (IETF) standard used to allow multiple computers on a private network (using private address ranges such as 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16) to share a single, globally routable IPv4 address.

NATs are often deployed because public IPv4 addresses are becoming scarce. Companies often use NAT devices to share a single public IPv4 address among dozens or hundreds of systems that use private, often duplicated IPv4 addresses. These private IPv4 addresses cause problems if inadvertently leaked across the public Internet by private IP-based networks.

The NAT router translates traffic coming into and leaving the private network. NAT allows a single device, such as a router, to act as agent between the Internet (or, "public network") and a local (or, "private") network. This means that only a single unique IP address is required to represent an entire group of computers to anything outside their network.

NAT is an immediate but temporary solution to the IPv4 address exhaustion problem that will eventually be rendered unnecessary with IPv6 deployment. However, the shortage of IP addresses in IPv4 is only one reason to use NAT. Two other reasons are:

· Security

· Administration

Implementing dynamic NAT automatically creates a firewall between your internal network and outside networks or the Internet. *Dynamic NAT* allows only connections that originate inside the stub domain. Essentially, this means that a computer on an external network cannot connect to your computer unless your computer has initiated the contact. NAT provides a simple packet filtering function by forwarding only solicited traffic to private network hosts. Solicited traffic is traffic that was requested by a private network host. For example, when a private host computer accesses a Web page, the private host computer requests the page contents from the Web server. The traffic for the Web page contests is solicited traffic. By default, a NAT does not forward unsolicited traffic to private network hosts. Therefore, you can browse the Internet and connect to a site, even download a file. However, somebody else cannot simply latch onto your IP address and use it to connect to a port on your computer.

Static NAT, also called *inbound mapping*, allows connections initiated by external devices to computers on the stub domain to take place in specific circumstances. For instance, you may wish to map an inside global address to a specific inside local address that is assigned to your Web server. Static NAT (inbound mapping) allows a computer on the stub domain to maintain a specific address when communicating with devices outside the network.

9.3.4 Mobile IP

Mobility is the quality of being capable of movement or moving readily from place to place. Wireless devices provide this kind of untethered freedom, but mobility means more than the lack of a network cable. Many terms describe mobility, but this chapter uses the terms mobility and roaming to describe the act of moving between access points.

Defining or characterizing the behavior of roaming stations involves two forms:

* Seamless roaming

* Nomadic roaming

Seamless roaming is best analogized to a cellular phone call. For example, suppose you are using your cellular phone as you drive your car on the freeway. A typical global system for mobile

(GSM) communications or time-division multiple access (TDMA) cell provides a few miles of coverage area, so it is safe to assume that you are roaming between cellular base stations as you drive. Yet as you roam, you do not hear any degradation to the voice call (that is what the cellular providers keep telling us). There is no noticeable period of network unavailability because of roaming. This type of roaming is deemed seamless because the network application requires constant network connectivity during the roaming process.

Nomadic roaming is different from seamless roaming. Nomadic roaming is best described as the use of an 802.11-enabled laptop in an office environment. As an example, suppose a user of this laptop has network connectivity while seated at his desk and maintains connectivity to a single AP. When the user decides to roam, he undocks his laptop and walks over to a conference room. Once in the conference room, he resumes his work. In the background, the 802.11 client has roamed from the AP near the user's desk to an AP near the conference room. This type of roaming is deemed nomadic because the user is not using network services when he roams, but only when he reach his destination.

The Nomadic Mobility Problem

Internet hosts are widely known by their IP addresses. We also know that routers use the CIDR scheme to aggregate sets of contiguous IP addresses and, therefore, simplify the task of routing messages (Section 1.4.4). Imagine you are traveling and you need a means to allow your friends to send you messages while you are away. If this is a short travel, your visiting address(es) will not enter the "infrastructure" records, such as those of government agencies or public registries. You could designate a care-of agent to whom the post office will deliver your mail, which you will collect when returning back. However, if you want your mail forwarded to your visiting location, you need to let the postal office know your visiting address, which may be difficult if you do not know where you will be staying or will be staying at different locations only briefly and unpredictably. One option is that you explicitly notify your care-of agent every time you arrive at a visiting address. Note that there is a built-in inefficiency when relying on a care-of agent instead of the infrastructure registries, because every communication must first travel to your home address (known to the infrastructure registries), and then be redirected by your care-of agent to your visiting address. In the worst case, you may be located near the sender and far away from home, so the mail needs to make an unnecessary trip to your home and back. We will see that the Internet functions in a similar manner.

The Mobile IP Protocol

Mobile IP is a network-layer protocol (or Layer-3 protocol in the OSI architecture). We have already seen in Section 1.5.3 how extended service set (ESS) specification supports mobility of Wi-Fi hosts in IEEE 802.11 networks (Figure 1-76). ESS is a link-layer (or Layer-2 protocol in the OSI architecture) mechanism for mobility support. Mobile IP allows location-independent routing of datagrams to a mobile host that is identified by its home address. The home address will not change no matter which visiting network the mobile terminal is connected to. When the mobile terminal roams to a visiting network, the visiting network will assign a care-of address to the mobile terminal. The information of this care-of address is sent back to the home agent (HA) in the home network (Figure 9-19). The home agent keeps the association of the care-of address and the mobile terminal's home address. The IP tunnel may be built to connect the mobile

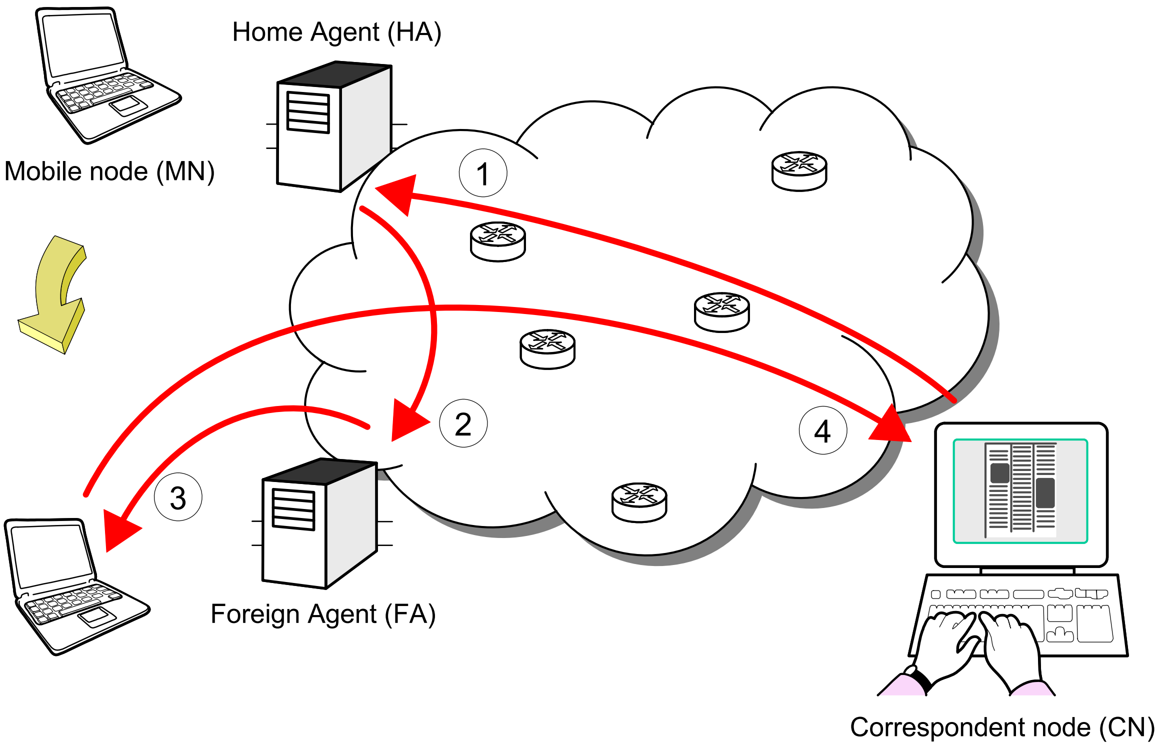


Figure 9-19: Mobile IP.

terminal and the home agent through the Internet cloud. For any packets received in the home agent, the home agent will forward them to the mobile terminal through the tunnel. Mobile IP provides an efficient mechanism for roaming within the Internet. Using Mobile IP, nodes may keep its connection to the Internet without changing their home IP address. Therefore, the location changing in the network is transparent to the correspondent node (CN). Node mobility is realized without the need to propagate the changed location on the network.

9.4 Domain Name System (DNS)

To facilitate network management and operations, the Internet Community has defined the Domain Name System (DNS). An application that wants to send a message to another application on remote computer uses DNS to find the remote computer's IP address given its name. A *domain* refers to a subdivision of a wide area network. Names are hierarchical and a major subdivision is the *top-level domain*, which is broken into organizational and geographic domains. The geographic or country domains use two letters to identify a country and there are 225 country domain labels. For example, *.us* stands for the United States, *.ru* stands for Russia, and *.it* stands for Italy. A name like `university.edu` is found registered with a `.edu` registrar, and within the associated network other names like `mylab.department.university.edu` can be defined, with obvious hierarchy. Security extensions allow a registry to sign the records it contains and in this way demonstrate their authenticity.

Domain Name System is a kind of an address translation protocol (Section 8.3). As shown in Figure 1-40 and Figure 1-41, it translates from the computer name (application-layer address) to a network-layer address. We consider it separately because of its importance and complexity. Unlike the protocols described in Section 8.3, which translate addresses for nodes on the same subnet, DNS resolves host names for hosts anywhere in the Internet.

9.5 Network Management Protocols

Network management tools allow network administrators to monitor network performance, failures, security, and help with accounting management. A basic requirement is to support isolating, diagnosing, and reporting problems to facilitate quick repair and recovery. More advanced features include support for data analytics to predict potential problems, so the network manager can take action before the problem occurs. A number of communication protocols exist for gathering information from network devices. This section reviews some of them.

9.5.1 Internet Control Message Protocol (ICMP)

Internet Control Message Protocol (ICMP) provides a mechanism for communicating control messages and error reports. Both routers and hosts use ICMP to transmit problem reports about datagrams back to the datagram source. In addition, ICMP includes an echo request/reply that can be used to determine if a destination is reachable and if so, is responding. ICMP specifies the format of control messages and when routers should send them. ICMP messages are delivered by IP, as the payload of IP datagrams.

9.5.2 Simple Network Management Protocol (SNMP)

SNMP is the protocol that provides the query language for gathering the information and for sending it to the console. The current version is SNMPv3. In general, the SNMP management system will discover the topology of the network automatically and will display it on the management console in the form of a graph. From this display, the human network manager can select a particular segment of the network to view its status in greater detail.

Each network device hosts a software *agent* that gathers information about the status of that device into a Management Information Base (MIB) and sends it to the *network management system* (NMS), as shown in Figure 9-20(a). SNMP defines seven message types for accessing management information in a client-server relationship (Figure 9-20(b)). Here the NMS is the client and the agent is the server. The message types are as follows:

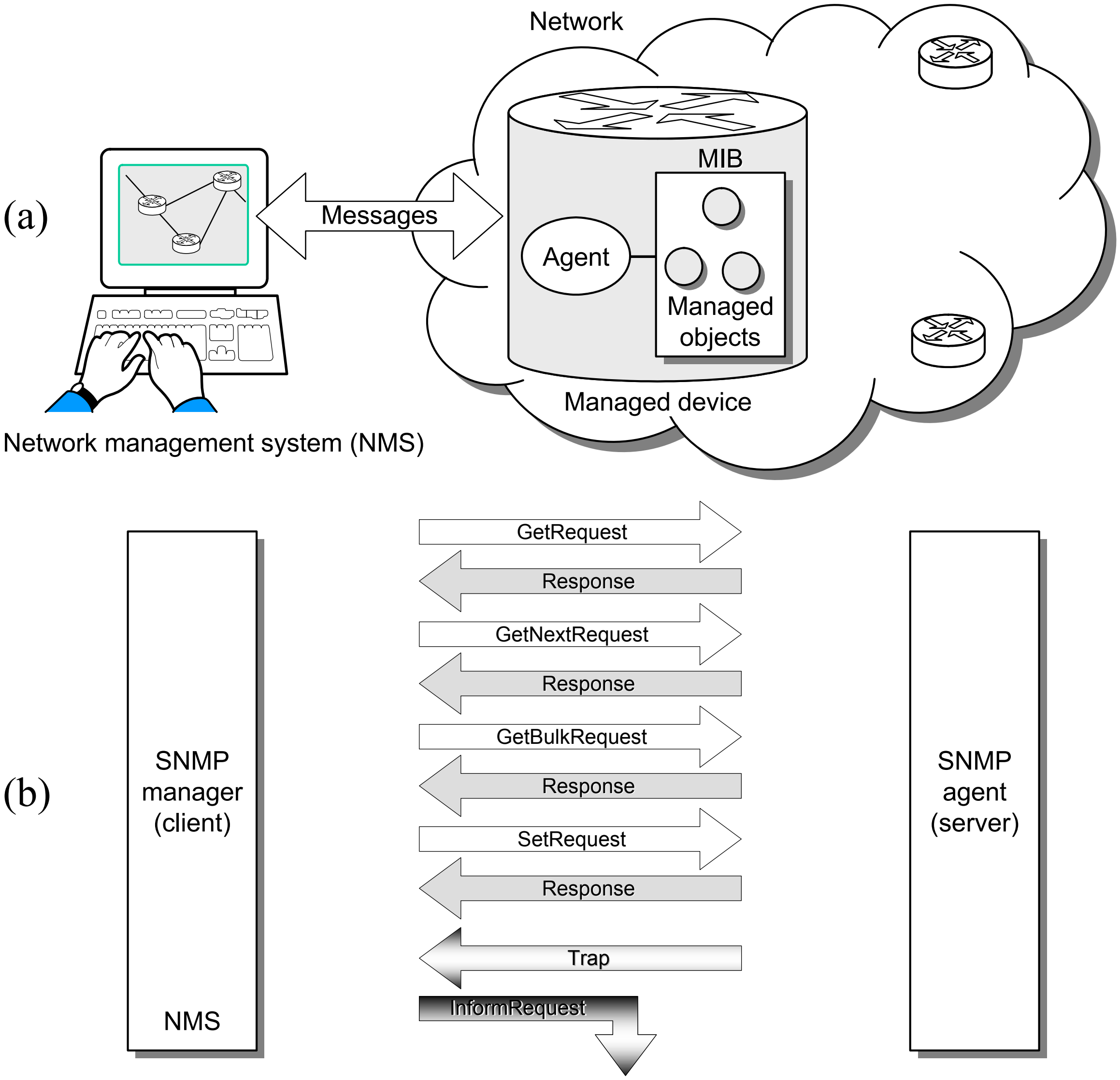


Figure 9-20: (a) SNMP architecture. (b) SNMPv3 message types and their flows.

• *GetRequest*: This is the most commonly used SNMP message and is sent from the manager (NMS) to the agent to retrieve the value of a specific management variable. The manager must send out one GetRequest for each value of a variable that needs to be retrieved.

• *GetNextRequest*: This message is used by an NMS for requesting the next variable in a list or table of variables. It is used mostly to retrieve the values of entries in the MIB if the network manager does not know how many variables there are in the MIB for a certain event.

• *GetBulkRequest*: (not in SNMPv1) This message is sent by an NMS to retrieve large blocks of data, such as multiple rows in a table.

• *Response*: An agent sends this message to a manger/NMS in response to GetRequest or GetNextRequest messages. It contains the value of the variable requested by the manager.

• *SetRequest*: The manager/NMS sends a SetRequest message to an agent to create, store, or modify an information variable. The agent must reply using a Response message.

• *Trap*: An agent sends a Trap message to an NMS to report an event when a certain set of circumstances arises. This is done autonomously, without any request from the manager. For example, if the agent resides in a router that is rebooted, the agent informs the manager of this event and reports the time of the rebooting.

• *InformRequest*: (not in SNMPv1, supports MoM architectures) This message is sent by an NMS to notify another NMS of information in a MIB table that is remote to the receiving manager.

SNMPv2 and SNMPv3 enhanced the original version of SNMP (SNMPv1) with additional message types for establishing multiple manager entities within a single network to support distributed management. *Distributed management* means that NMSs and agents are spread out across the internetwork. A hierarchical distributed arrangement can be used, whereby distributed NMSs send data to more powerful central NMSs using a Manager-of-Managers (MoM) architecture. A centralized system that manages distributed NMSs is sometimes called an *umbrella NMS*. The advantages of distributed management include resilience to failures and the ability to reduce network-management overhead by filtering data at distributed NMSs before sending them to the central stations. On the downside, distributed management is complex and hard to operate, and more susceptible to security breaches.

SNMP is an application-layer protocol and it operates over the user datagram protocol (UDP). As a result, SNMP is a connectionless protocol. No continuous connections exist between a management console and its agent so that each message between them is a separate transaction.

9.6 Multimedia Application Protocols

Multimedia application protocols include RTP and RTCP, which are described in Section 3.4.

9.6.1 Session Description Protocol (SDP)

Peer ends of a multimedia application use the Session Description Protocol (SDP), to offer and accept (or not) codecs, decide the port number and IP address for where each endpoint wants to receive their RTP packets (Section 3.4). SDP packets are transported by SIP.

9.6.2 Session Initiation Protocol (SIP)

The Session Initiation Protocol is an application-layer control (signaling) protocol for creating, modifying and terminating multimedia sessions on the Internet, meant to be more scalable than H.323. Multimedia sessions can be voice, video, instant messaging, shared data, and/or subscriptions of events. SIP can run on top of TCP, UDP, SCTP, or TLS over TCP. SIP is independent of the transport layer, and independent of the underlying IPv4/v6 version. In fact, the transport protocol used can change as the SIP message traverses SIP entities from source to destination. SIP itself does not choose whether a session is voice or video—the SDP does it.

9.7 Summary and Bibliographical Notes

The best source of information about the Internet protocols are Requests for Comments (RFCs) published by the Internet Engineering Task Force (IETF), which can be found online at: http://www.ietf.org/rfc.html. For a complete listing of all protocols in the IP stack, their standards statuses, and reference to their RFC documents, see http://www.rfc-editor.org/rfcxx00.html, which is updated daily.

Section 9.1: Internet Protocol Version 6 (IPv6)

The IETF solicited proposals for a next generation Internet Protocol (IPng) in July of 1992. A number of proposals were received and by 1994 the design and development of a suite of protocols and standards now known as Internet Protocol Version 6 (IPv6) was initiated. A major milestone was reached in 1995 with the publication of RFC-1752 ("The Recommendation for the IP Next Generation Protocol"). Overall specification of IPv6 is defined in RFC-2460 [Deering & Hinden, 1998]. The address structure of IPv6 is defined in RFC-2373. The format of IPv6 global unicast addresses is defined in RFC-3587 (obsoletes RFC-2374). Great deal of information about IPv6 can be found at the IPv6.com webpage, at http://www.ipv6.com. Also [Huitema, 1998]

At the time of this writing (2010), IPv6 is not widely adopted. One of the greatest obstacles to wider adoption of IPv6 is that it lacks backwards compatibility with IPv4. However, fewer than 10% of IPv4 addresses remain unallocated and industry experts predict the rest of the IPv4 address supply will run out in 2012. Therefore, it can be expected that the adoption rate will grow rapidly. The IPv6 Forum (http://www.ipv6forum.com/) verifies protocol implementation and validates interoperability of IPv6 products. The IPv6 Forum has a service called IPv6 Ready Logo (http://www.ipv6ready.org/), which is a qualification program that assures devices they test are IPv6 capable. Actual testing in the U.S. is performed by the IPv6 Testing Consortium at the University of New Hampshire (http://www.iol.unh.edu/services/testing/ipv6/), which is a pioneer in IPv6 testing.

Section 9.2: Routing Protocols

The Routing Information Protocol (RIP) was the initial routing protocol in the ARPAnet. RIP was originally designed for Xerox PARC Universal Protocol (where it was called GWINFO) and used in the Xerox Network Systems (XNS) protocol suite. RIP became associated with both UNIX and TCP/IP in 1982 when the Berkeley Software Distribution (BSD) version of UNIX began shipping with a RIP implementation referred to as `routed` (pronounced "route dee"). RIP, which is still a very popular routing protocol in the Internet community, is formally defined in the XNS Internet Transport Protocols publication (1981) and in RFC-1058 (1988). RIP version 2 (for IPv4) is defined in RFC-1723. This document does not change the RIP protocol per se; rather, it provides extensions to the message format that allows routers to share important additional information.

Open Shortest Path First (OSPF) is a link-state routing protocol defined in RFC-2328. OSPF was designed to advertise the subnet mask of with the network prefix. OSPF supports variable-length subnet masks (VLSM), disjointed subnets, and supernetting.

Border Gateway Protocol (BGP) was designed largely to handle the transition from a single administrative entity (NSFNet in the US in 1980s) to multiple backbone networks run by competitive commercial entities. Border Gateway Protocol version 4 (BGP4) is defined in RFC-4271 [Rekhter *et al*., 2006], which obsoletes RFC-1771. See also RFC-4274 and RFC-4276. [Stewart, 1999] provides a concise overview of BGP4, although it does not include the latest updates. BGP4 and CIDR (Section 1.4.4) have played a key role in enabling the Internet to scale to its current size. Large ISPs use path aggregation in their BGP advertisements to other Autonomous Systems. An ISP can use CIDR to aggregate the addresses of many customers into a single advertisement, and thus reduce the amount of information required to provide routing to customers.

IETF specified routing protocols that work with IPv6 include RIP for IPv6 [RFC-2080], IS-IS for IPv6 [RFC-5308], OSPF for IPv6 [RFC-5340], and BGP-4 for IPv6 [RFC-2545].

Section 9.3: Address Translation Protocols

Address Resolution Protocol (ARP) is defined in RFC-826. Reverse Address Resolution Protocol (RARP) is defined in RFC-903. Inverse Address Resolution Protocol (InARP) is defined in RFC-2390. RARP is now obsolete (succeeded by DHCP), and InARP is primarily used in Frame Relay and ATM networks. Neighbor Discovery (NDP) which is used for discovery of other nodes on the link and determining their link layer addresses for IP version 6 (IPv6) is described in RFC-4861.

Network Address Translation (NAT) is described in RFC-2663 [Srisuresh & Holdrege, 1999] and RFC-3022 [Srisuresh & Egevang, 2001].

Section 9.4: Domain Name System (DNS)

The Domain Name System (DNS) is defined in RFC-1034 and RFC-1035.

Section 9.5: Network Management Protocols

The first version of Simple Network Management Protocol, SNMPv1, was defined by RFC-1067 in 1988. SNMPv2 was introduced in 1993 and updated in 1996. SNMPv1 and SNMPv2 supported monitoring network statistics, but had no security features. SNMPv3 is the current standard version (see RFC-3411). SNMPv3 offers security features and is expected to displace the earlier versions. SNMPv3 supports user authentication to prevent unauthorized users from executing network management functions, and message encryption to prevent eavesdropping on the messages exchanged between NMSs and managed devices. SNMP is supported by most commercial network management systems (NMSs) and many networking devices, including switches, routers, servers, and workstations.

The current version of management information base (MIB) for SNMP is defined in RFC-3418.

The statistics of RMON that should be collected are standardized in RFC-1757 and RFC-2021.

Section 9.6: Multimedia Application Protocols

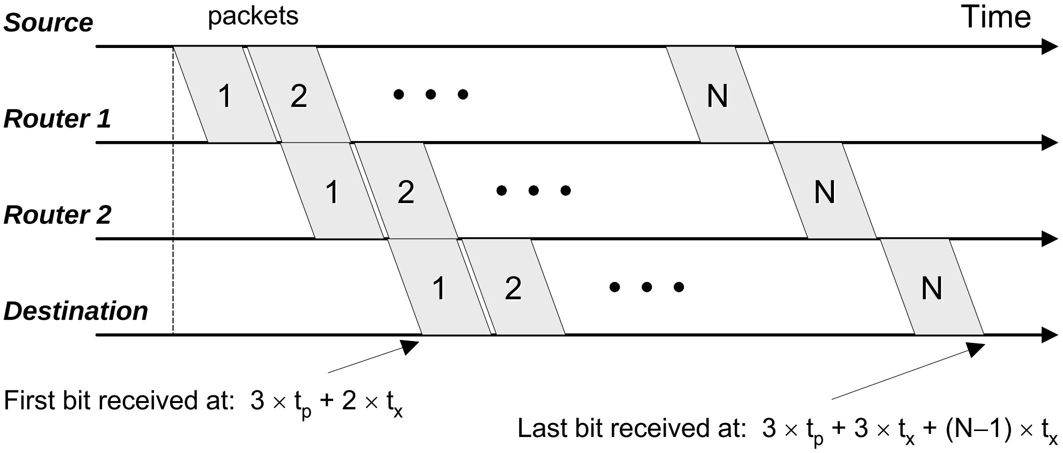
The Session Initiation Protocol is defined in RFC-3261, RFC-3265, RFC-3853, RFC-4320, RFC-4916, RFC-5393, and RFC-5621.

Problems

Solutions to Selected Problems

Problem 1.1 — Solution

Let t_x denote the transmission time for a packet L bits long and let t_p denote the propagation delay on each link. Each packet crosses three links (Link-1: source to router1; Link-2: router1 to router2; Link-3: router2 to destination).


Source
packets
Time
1
2
• • •
N
Router 1
1
2
• • •
N
Router 2
1
2
• • •
N
Destination
First bit received at: 3 × t_p + 2 × t_x
Last bit received at: 3 × t_p + 3 × t_x + (N–1) × t_x

(a)

The total transmission time equals: $3 \times t_p + 3 \times t_x + (N - 1) \times t_x = 3 \times t_p + (N + 2) \times t_x$.

(b)

The transmission time is ½ t_x, the propagation time remains the same, and there are twice more packets, so the total transmission time equals:

$3 \times t_p + (2\times N + 2) \times (t_x/2) = 3 \times t_p + (N + 1) \times t_x$.

(c)

The total delay is smaller in case (b) by t_x because of greater parallelism in transmitting shorter packets. If we use, for example, four times shorter packets ($L/4$), then the total transmission time equals: $3 \times t_p + (N + 1/2) \times t_x$.

On the other hand, if we use two times longer packets, i.e., $N/2$ packets each $2\times L$ bits long, then the total transmission time equals:

$3 \times t_p + (N/2 + 2) \times (2\times t_x) = 3 \times t_p + (N + 4) \times t_x$.

which is longer by $2\times t_x$ then in case (a).

Problem 1.2 — Solution

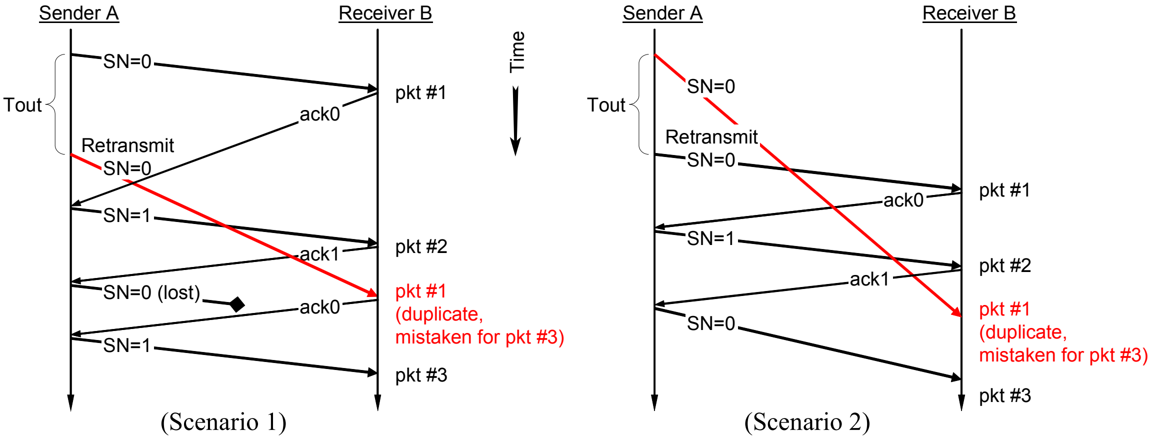
Problem 1.3 — Solution

(a)

Having only a single path ensures that all packets will arrive in order, although some may be lost or damaged due to the non-ideal channel. Assume that *A* sends a packet with SN = 1 to *B* and the packet is lost. Because *A* will not receive ACK within the timeout time, it will retransmit the packet using the same sequence number, SN = 1. Because *B* already received a packet with SN = 1 and it is expecting a packet with SN = 0, it concludes that this is a duplicate packet.

(b)

If there are several alternative paths, the packets can arrive out of order. There are many possible cases where *B* receives duplicate packets and cannot distinguish them. Two scenarios are shown below, where either the retransmitted packet or the original one gets delayed, e.g., by taking a longer path. These counterexamples demonstrate that the alternating-bit protocol cannot work over a general network.

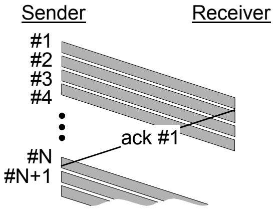


Problem 1.4 — Solution

Problem 1.5 — Solution

Problem 1.6 — Solution

Recall that the utilization of a sender is defined as the fraction of time the sender is actually busy sending bits into the channel. Because we assume errorless communication, the sender is maximally used when it is sending without taking a break to wait for an acknowledgement. This happens if the first packet of the window is acknowledged before the transmission of the last packet in the window is completed. That is,



$$(N-1) \times t_x \geq \text{RTT} = 2 \times t_p$$

where t_x is the packet transmission delay and t_p is the propagation delay. The left side represents the transmission delay for the remaining $(N-1)$ packets of the window, after the first packet is sent. Hence, $N \geq \left\lceil \frac{2 \times t_p}{t_x} \right\rceil + 1$. The ceiling operation $\lceil \cdot \rceil$ ensures integer number of packets.

In our case, ℓ = 10 km, $v \approx 2 \times 10^8$ m/s, R = 1 Gbps, and L = 512 bytes.

Hence, the packet transmission delay is: $t_x = \frac{L}{R} = \frac{512 \times 8 \text{ (bits)}}{1 \times 10^9 \text{(bits / sec)}} = 4.096\ \mu\text{s}$

The propagation delay is: $t_p = \frac{\ell}{v} = \frac{10000 \text{ m}}{2 \times 10^8 \text{ m/s}} = 50\ \mu\text{s}$

Finally, $N \geq \lceil 24.41 \rceil + 1 = 26$ packets.

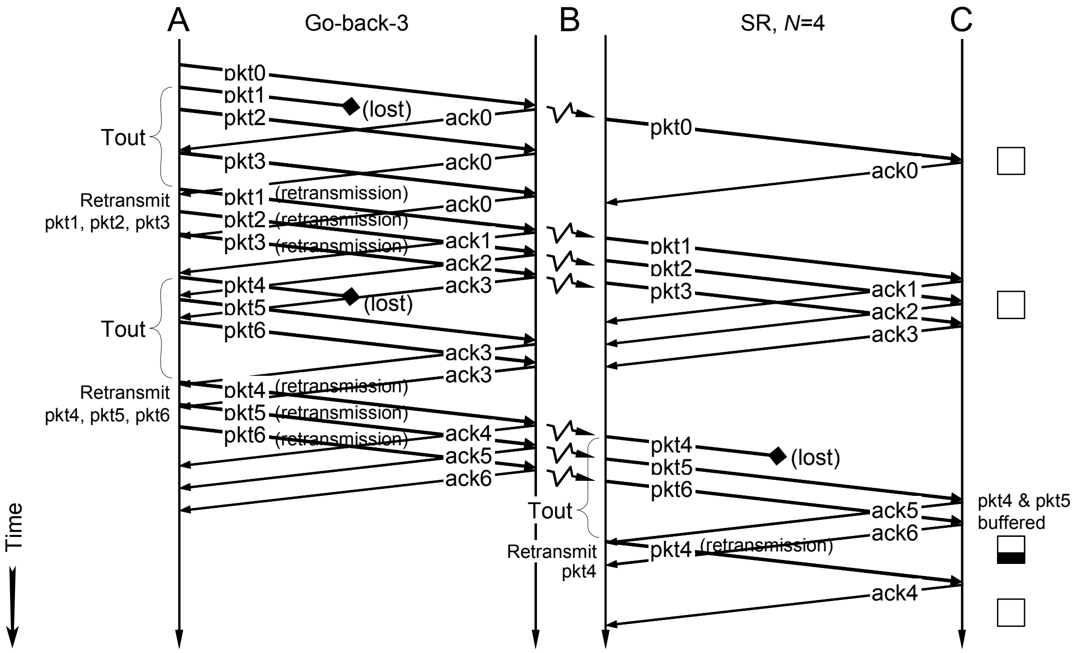
Problem 1.7 — Solution

str. 284

Problem 1.8 — Solution

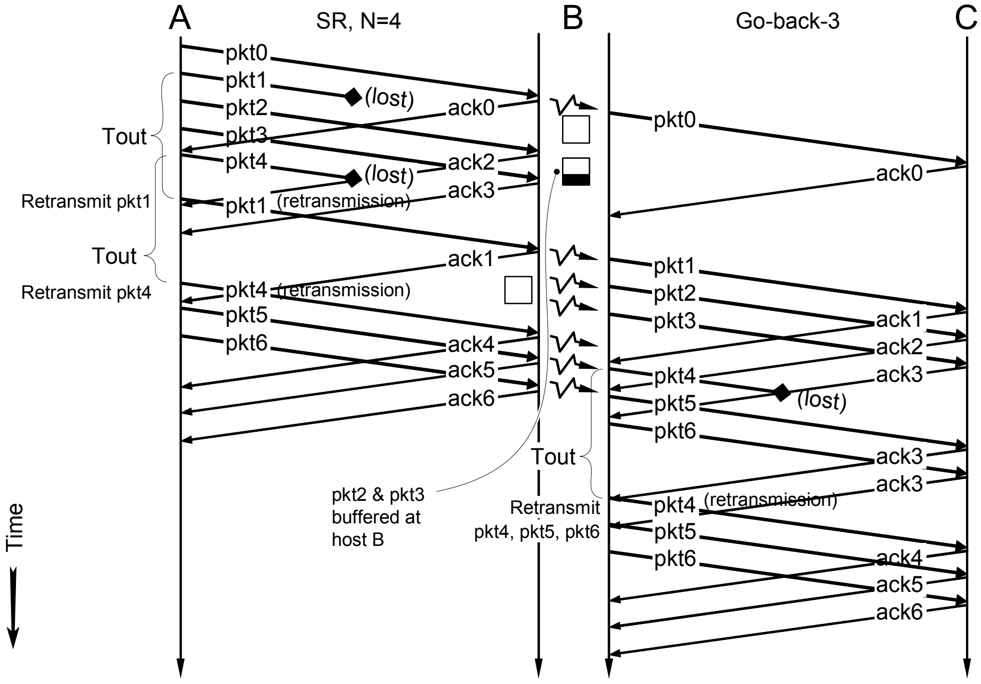
The solution is shown in the following figure. We are assuming that the retransmission timer is set appropriately, so ack0 is received before the timeout time expires. Note that host *A* simply ignores the duplicate acknowledgements of packets 0 and 3, i.e., ack0 and ack3.

There is no need to send source-to-destination acknowledgements (from *C* to *A*) in this particular example, because both $\overline{AB}$ and $\overline{BC}$ links are reliable ***and*** there are no alternative paths from *A* to *C* but via *B*. The reader should convince themselves that should alternative routes exist, e.g., via another host *D*, then we would need source-to-destination acknowledgements in addition to (or instead of) the acknowledgements on individual links.



Problem 1.9 — Solution

The solution is shown in the following figure.



Problem 1.10 — Solution

It is easy to get tricked into believing that the second, (b), configuration would offer better performance, because the router can send in parallel in both directions. However, this is not true, as will be seen below.

(a)

Propagation delay = 300 m / $(2 \times 10^8) = 1.5 \times 10^{-6}$ s = 1.5 μs

Transmission delay per data packet = $2048 \times 8 / 10^6 = 16.384 \times 10^{-3}$ = 16.384 ms

Transmission delay per ACK = $10^8 / 10^6 = 0.08 \times 10^{-3}$ = 0.08 ms

Transmission delay for $N = 5$ (window size) packets = $16.384 \times 5 < 82$

$82 + 0.08 + 0.0015 \times 2 = 82.083$

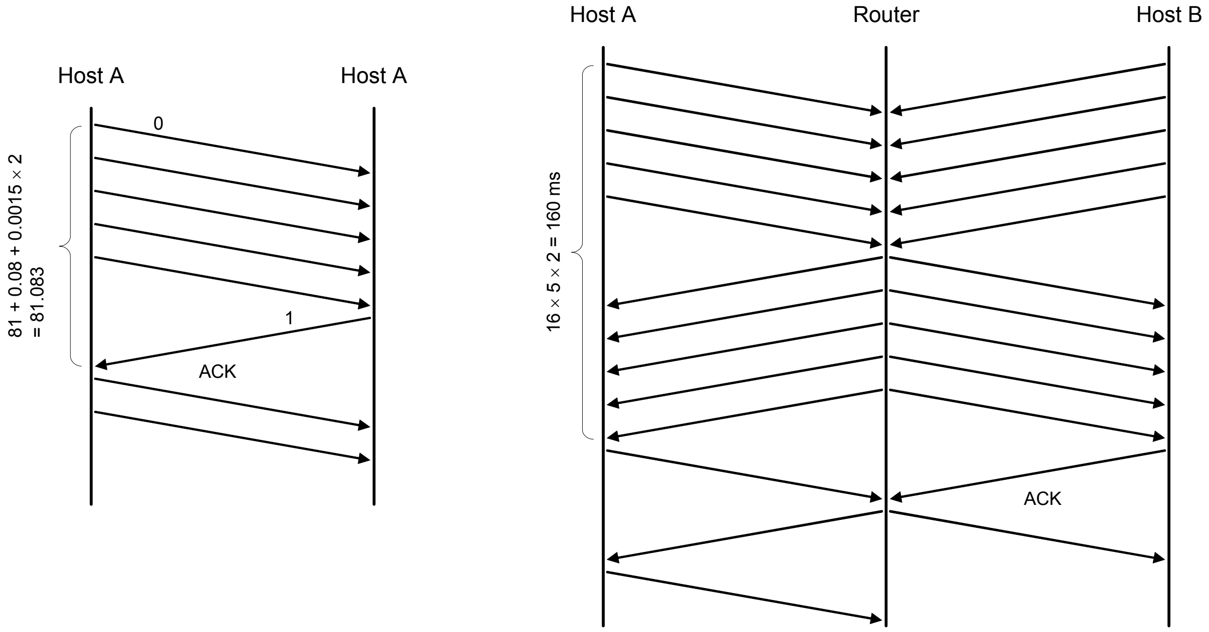
Subtotal time for 100 packets in one direction = $100 \times 82.083 / 5 = 1641.66$ ms

Total time for two ways = $1641.66 \times 2 = 3283.32$ ms

(b)

If host A (or B) sends packets after host B (or A) finishes sending, then the situation is similar to (a) and the total time is about 3283.32 ms.

If hosts A and B send a packet each simultaneously, the packets will be buffered in the router and then forwarded. The time needed is roughly **double** (!), as shown in this figure.



(a) (b)

Problem 1.11 — Solution

Go-back-*N* ARQ.

Packet error probability = p_e for data packets; ACKs error free. Successful receipt of a given packet with the sequence number *k* requires successful receipt of all previous packets in the sliding window. In the worst case, retransmission of frame *k* is always due to corruption of the earliest frame appearing in its sliding window. So, $p_{\text{succ}} = \prod_{i=k-N}^{k} (1 - p_e) = (1 - p_e)^N$, where *N* is the sliding window size. An upper bound estimate of $E\{n\}$ can be obtained easily using Eq. (1.8) as $E\{n\} = \frac{1}{p_{\text{succ}}} = \frac{1}{(1 - p_e)^N}$. On average, however, retransmission of frame *k* will be due to an error in frame (*k*–LAR)/2, where LAR denotes the sequence number of the *Last Acknowledgement Received*.

(a)

Successful transmission of one packet takes a total of $t_{\text{succ}} = t_x + 2 \times t_p$

The probability of a failed transmission in one round is $p_{\text{fail}} = 1 - p_{\text{succ}} = 1 - (1 - p_e)^N$

Every failed packet transmission takes a total of $t_{\text{fail}} = t_x + t_{\text{out}}$

(assuming that the remaining *N*–1 packets in the window will be transmitted before the timeout occurs for the first packet).

Then, using Eq. (1.9) the expected (average) total time per packet transmission is:

$$E\{T_{\text{total}}\} = t_{\text{succ}} + \frac{p_{\text{fail}}}{1 - p_{\text{fail}}} \cdot t_{\text{fail}}$$

(b)

If the sender operates at the maximum utilization (see Problem 1.6 — Solution), then the sender waits for *N*–1 packet transmissions for an acknowledgement, $t_{\text{out}} = (N\text{–}1) \cdot t_x$, before a packet is retransmitted. Hence, the expected (average) time per packet transmission is:

$$E\{T_{\text{total}}\} = t_{\text{succ}} + \frac{p_{\text{fail}}}{1 - p_{\text{fail}}} \cdot t_{\text{fail}} = 2 \cdot t_p + t_x \cdot \left(1 + (N - 1) \cdot \frac{p_{\text{fail}}}{1 - p_{\text{fail}}}\right)$$

Problem 1.12 — Solution

(a)

Packet transmission delay equals = 1024 × 8 / 64000 = 0.128 sec; acknowledgement transmission delay is assumed to be negligible. Therefore, the throughput S_{MAX} = 1 packet per second (pps).

(b)

To evaluate $E\{S\}$, first determine how many times a given packet must be (re-)transmitted for successful receipt, $E\{n\}$. According to Eq. (1.8), $E\{n\} = 1/p \cong 1.053$. Then, the expected throughput is $E\{S\} = \dfrac{S_{\mathrm{MAX}}}{E\{n\}} = 0.95$ pps.

(c)

The fully utilized sender sends 64Kbps, so $S_{\mathrm{MAX}} = \dfrac{64000}{1024 \times 8} = 7.8125$ pps.

(d)

Again, we first determine how many times a given packet must be (re-)transmitted for successful receipt, $E\{n\}$. The sliding window size can be determined as $N = 8$ (see Problem 1.6 — Solution). A lower bound estimate of $E\{S\}$ can be obtained easily by recognizing that $E\{n\} \leq 1/p^8 \cong 1.5$ (see Problem 1.11 — Solution). Then, $S_{\mathrm{MAX}} \times p^8 \cong (7.8125) \times (0.6634) \cong 5.183$ pps represents a non-trivial lower bound estimate of $E\{S\}$.

Problem 1.13 — Solution

The packet size equals to transmission rate times the slot duration, which is 1500 bps × 0.08333 s = 125 bits. This wireless channel can transmit a maximum of 12 packets per second, assuming that in each slot one packet is transmitted. (Recall that slotted ALOHA achieves maximum throughput when $G = 1$ packet per slot). Of these, some will end up in collision and the effective throughput will be equal to 0.368 × 12 = 4.416 packets per second. Because this is aggregated over 10 stations, each station can effectively transmit 4.416 / 10 = 0.4416 packets per second, or approximately 26 packets per minute, at best.

Problem 1.14 — Solution

(a) Given the channel transmission attempt rate G, the probability of success was derived in Eq. (1.12) as e^{-G}, which is the probability that no other station will transmit during the vulnerable period.

(b) The probability of exactly K collisions and then a success is

$$(1 - e^{-G})^k \cdot e^{-G}$$

which is derived in the same manner as Eq. (1.7) in Section 1.3.1.

Problem 1.15 — Solution

(a)

The solution is given in Figure 1-29(b): there will be $G{\cdot}P_0$ successfully transmitted packets, and $G{\cdot}(1-P_0)$ collided packets. Slotted ALOHA operates under maximum efficiency when G=1, that is when on average one packet is transmitted per each slot. This means that, on average, there will be no idle slots—every slot, on average, will be used for a transmission. Of those transmissions, there will be on average $\lambda = 1/e$ fresh packet arrivals as well as $1/e$ slots with successful transmissions (some of the successful transmissions may be retransmissions of backlogged packets), and there will be on average $(1 - 1/e)$ slots with collisions.

(b)

A slotted ALOHA system would operate with a less-than-maximum efficiency if $G \neq 1$. When $G < 1$ there are on average less than one transmission attempts per packet time, so we say that the system is *underloaded*. Conversely, when $G > 1$ there are on average more than one transmission attempts per packet time, so we say that the system is *overloaded*. In both cases, the arrival rate will be $\lambda < 1/e$ (see Figure 1-31).

The system would be *underloaded* when some stations do not have packets in some slots to transmit or retransmit. As a result, there will be a non-zero fraction of idle slots. In the underloaded case, there will be on average $1/e - \lambda > 0$ idle slots, λ successful slots, and the remaining $\leq(1 - 1/e)$ slots with collisions.

The system would be *overloaded* if the arrival rate became temporarily $\lambda > 1/e$, i.e., greater than the optimal value. This will result in many collisions and there will be many stations that become backlogged, trying to retransmit previously collided packets. The backlogged stations will not accept fresh packets, which effectively means that the arrival rate will drop to $\lambda < 1/e$. (Note that λ becomes small not because users are not generating new packets but because many stations are backlogged and do not accept fresh packets). In the overloaded case, there will be on average no idle slots, $\lambda < 1/e$ successful slots, and the remaining $>(1 - 1/e)$ slots with collisions.

(c)

Both the maximum-efficiency and underloaded cases are stable. In both cases for a steady arrival rate λ, the system will remain in a stable condition.

The overloaded case is *unstable*: initially the arrival rate will be $\lambda > 1/e$, but then many stations will become backlogged and will not accept new packet arrivals so λ will drop to $\lambda < 1/e$. These backlogged stations will keep retransmitting their packets and eventually the backlogged packets will clear. If after this clearing the arrival rate remains $\lambda \leq 1/e$, then the system will remain maximally efficient ($\lambda = 1/e$) or it will remain underloaded ($\lambda < 1/e$). Otherwise, it will again become temporarily overloaded.

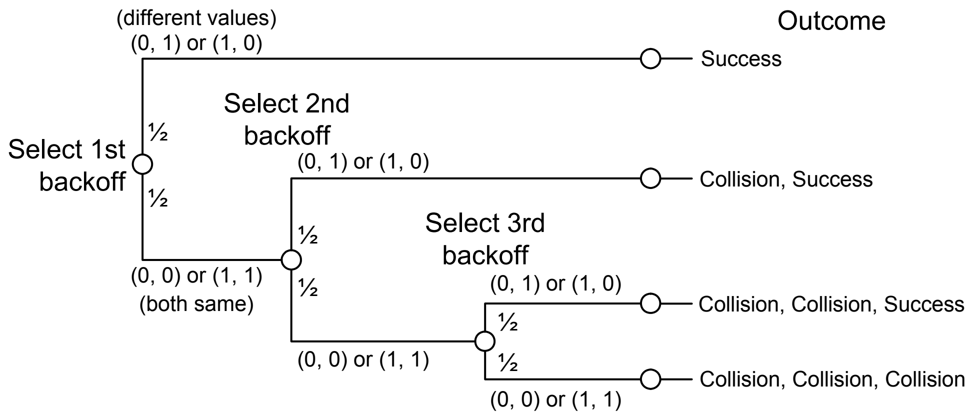
Problem 1.16 — Solution

Problem 1.17 — Solution

The stations will collide only if both select the same backoff value, i.e., either both select 0 or both select 1. The “tree diagram” of possible outcomes is shown in the figure below. To obtain the probabilities of different outcomes, we just need to multiply the probabilities along the path to each outcome.

(a)

Probability of transmission $p = ½$, the success happens if either station transmits alone:



$$P_{\text{success}} = \binom{2}{1} \cdot (1-p)^{2-1} \cdot p = 2 \times ½ \times ½ = 0.5$$

(b)

The first transmission ends in collision if both stations transmit simultaneously

$$P_{\text{collision}} = 1 - P_{\text{success}} = 0.5$$

Because the waiting times are selected *independent* of the number of previous collisions (i.e., the successive events are independent of each other), the probability of contention ending on the second round of retransmissions is

$$P_{\text{collision}} \times P_{\text{success}} = 0.5 \times 0.5 = 0.25$$

(c)

Similarly, the probability of contention ending on the third round of retransmissions is

$$P_{\text{collision}} \times P_{\text{success}} \times P_{\text{success}} = 0.5 \times 0.5 \times 0.5 = 0.125$$

(d)

Regular nonpersistent CSMA with the normal binary exponential backoff algorithm works in such a way that if the channel is busy, the station selects a random period to wait. When the waiting period expires, it senses the channel. If idle, transmit; if busy, wait again for a random period, selected from the *same range*. And so on until the channel becomes idle and transmission occurs.

If the transmission is successful, the station goes back to wait for a new packet.

If the transmission ends in collision, *double the range* from which the waiting period is drawn and repeat the above procedure.

Unlike the regular nonpersistent CSMA, in the modified nonpersistent CSMA there are only two choices for waiting time. Therefore, half of the backlogged stations will choose one way and the other half will choose the other way.

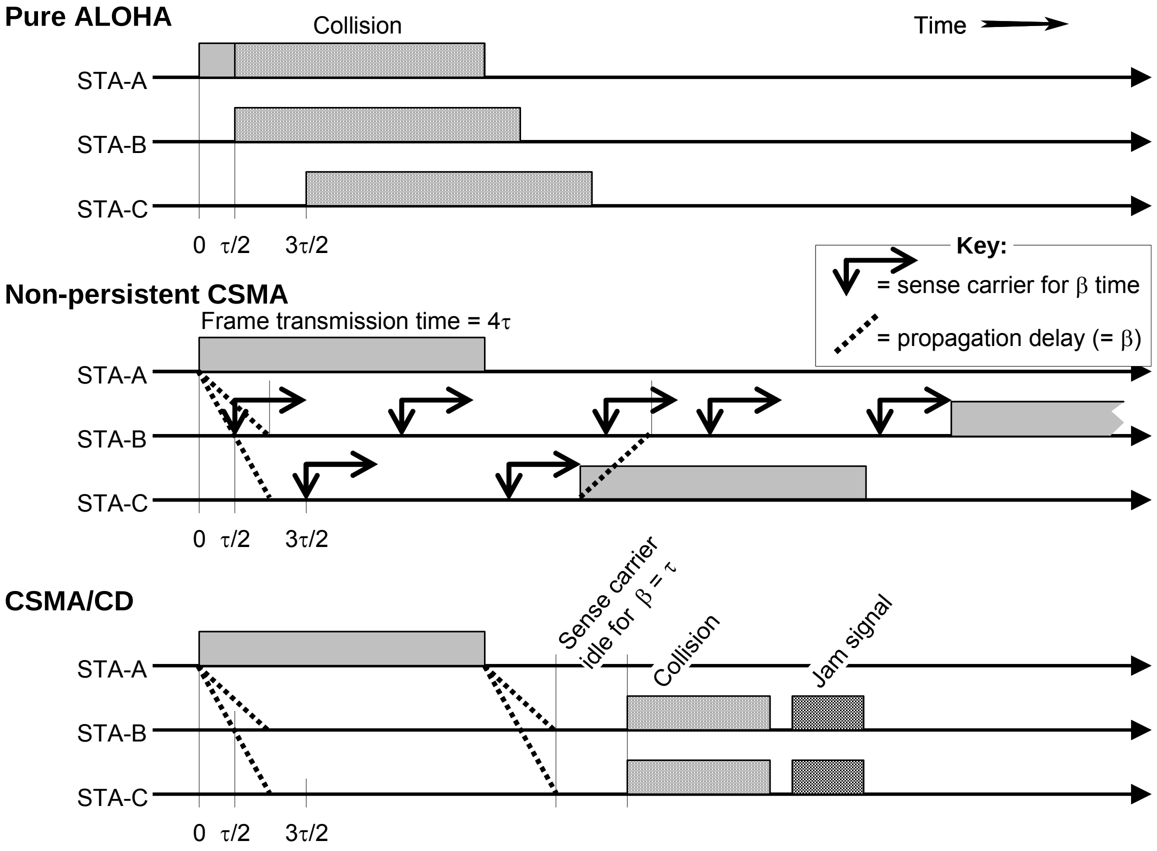
In conclusion, regular nonpersistent CSMA performs better than the modified one under heavy loads and the modified algorithm performs better under light loads.

Problem 1.18 — Solution

The solution is shown in the figure below. Recall that nonpersistent CSMA operates so that if the medium is idle, it transmits; and if the medium is busy, it waits a random amount of time and senses the channel again. The medium is decided idle if there are no transmissions for time duration *β*, which in our case equals *τ*. Therefore, although station *B* will find the medium idle at the time its packet arrives, (*τ*/2), because of *τ* propagation delay, the medium will become busy during the *τ* sensing time. The station must sense the channel state (idle or busy) for a full slot duration—sensing takes time: it cannot "figure out" the correct state in the middle of a slot.

The figure below shows the case where station *C* happens to sense the channel idle before *B* so station *C* transmits first.

For CSMA/CD, the smallest frame size is twice the propagation time, i.e., 2*τ*, which is the duration of the collision.



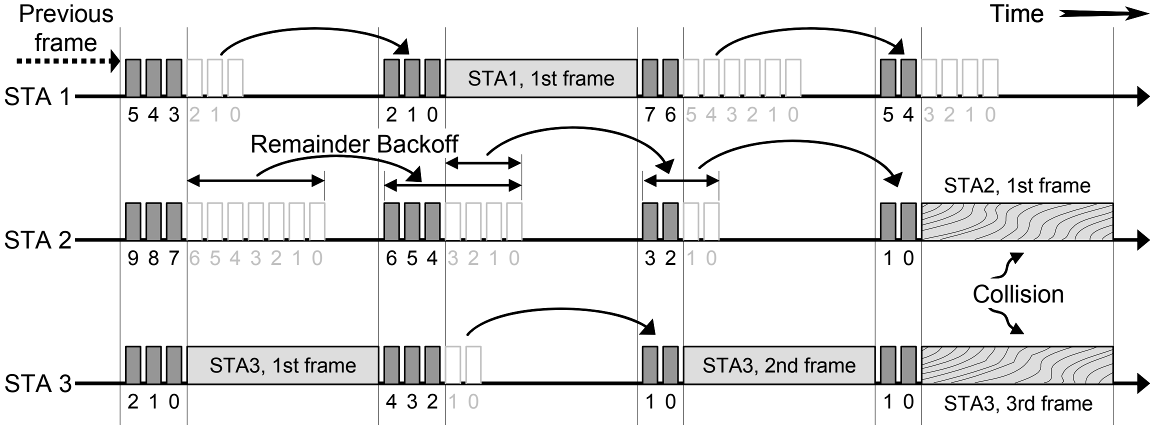
Problem 1.19 — Solution

Problem 1.20 — Solution

The solution for the three stations using the CSMA/CA protocol is shown in the figure below. The third station has the smallest initial backoff and transmits the first. The other two stations will freeze their countdown when they sense the carrier as busy. After the first frame, STA3 randomly chooses the backoff value equal to 4, and the other two stations resume their previous countdown.

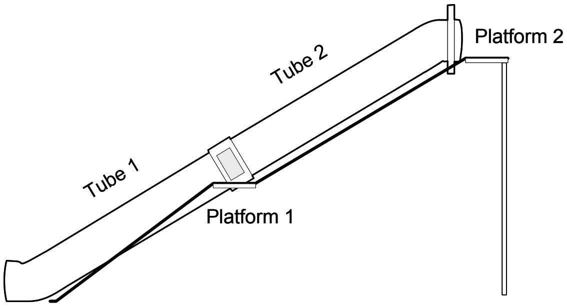
STA1 reaches zero first and transmits, while STA2 and STA3 freeze their countdown waiting for the carrier to become idle. STA3 transmits next its second frame and randomly chooses the backoff value equal to 1, and the other two stations again resume their previous countdown. STA2 finally transmits its first frame but STA3 simultaneously transmits its third frame and there is a collision.

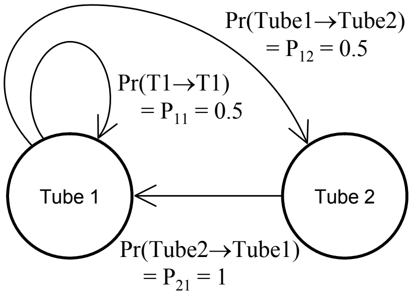
(Note: The reader may wish to compare this with Figure 1-35 which illustrates a similar scenario for three stations using the CSMA/CD protocol, as well as with Problem 1.46 for IEEE 802.11.)



Problem 1.21 — Solution

The analogy with a slide (Figure 1-87) is shown again in the figure below. We define three probabilities for a station to transition between the backoff states. P_{11} represents the probability that the kid will enter the slide on Platform-1, therefore after sliding through Tube-1 he again slides through Tube-1. P_{12} represents the probability that the kid will enter the slide on Platform-2, therefore after sliding through Tube-1 he first slides through Tube-2. Finally, P_{21} represents the probability that the kid will slide through Tube-1 after sliding through Tube-2, and this equals 1 because the kid has no choice. Below I will present three possible ways to solve this problem.





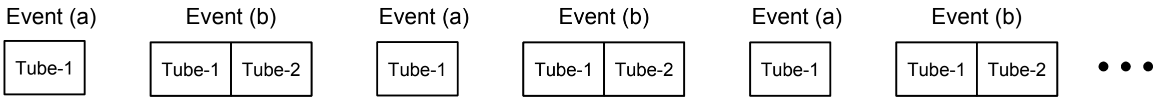
Solution 1 (Statistical)

There are two events:

(a) The kid enters the slide on Platform-1 and slides through Tube-1

(b) The kid enters the slide on Platform-2 and slides through Tube-2, then through Tube-1

Because the probabilities of choosing Platform-1 or proceeding to Platform-2 are equal, these two events will, statistically speaking, happen in a regular alternating sequence:



By simple observation we can see that the kid will be found two thirds of time in Tube-1 and one third of time in Tube-2. Therefore, P_{T1} = 2/3 and P_{T2} = 1/3 and these correspond to the distribution of steady-state probabilities of backoff states.

Solution 2 (Algebraic)

We introduce two variables, $x_1(k)$ and $x_2(k)$ to represent the probabilities that at time k the kid will be in Tube-1 or Tube-2, respectively. Then we can write the probabilities that at time k + 1 the kid will be in one of the tubes as:

$$\begin{aligned} x_1(k+1) &= P_{11} \cdot x_1(k) + P_{21} x_2(k) &&= \tfrac{1}{2} x_1(k) + x_2(k) \\ x_2(k+1) &= P_{12} \cdot x_1(k) &&= \tfrac{1}{2} x_1(k) \end{aligned}$$

Write the above equations using matrix notation:

$$\mathrm{X}(k+1) = \begin{bmatrix} \frac{1}{2} & 1 \\ \frac{1}{2} & 0 \end{bmatrix} \cdot \mathrm{X}(k) = \mathrm{A} \cdot \mathrm{X}(k)$$

Find the eignevalues of the matrix **A**:

$$\left| \mathrm{A} - \lambda \cdot \mathrm{I} \right| = \begin{vmatrix} \frac{1}{2} - \lambda & 1 \\ \frac{1}{2} & -\lambda \end{vmatrix} = \lambda^2 - \tfrac{1}{2}\lambda - \tfrac{1}{2} = 0 \qquad \Rightarrow \qquad \lambda_1 = 1, \quad \lambda_2 = -\tfrac{1}{2}$$

and eigenvectors of the matrix **A**:

$$\lambda = \lambda_1 = 1 \qquad \Rightarrow \qquad \begin{bmatrix} -\frac{1}{2} & 1 \\ \frac{1}{2} & -1 \end{bmatrix} \cdot \left[v_1 \right] = 0 \qquad \Rightarrow \qquad v_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\lambda = \lambda_2 = -\tfrac{1}{2} \qquad \Rightarrow \qquad \begin{bmatrix} 1 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \cdot \left[v_2 \right] = 0 \qquad \Rightarrow \qquad v_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

So,

$$S = \left[v_1 \quad v_2 \right] = \begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{and} \quad S^{-1} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & -\frac{2}{3} \end{bmatrix}, \qquad \Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -\frac{1}{2} \end{bmatrix}$$

Then we solve the backoff-state equation:

$$\mathrm{X}(k) = \mathrm{A}^k \cdot \mathrm{X}(0) = S \cdot \Lambda^k \cdot S^{-1} = \begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} (1)^k & 0 \\ 0 & (-\frac{1}{2})^k \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & -\frac{2}{3} \end{bmatrix} \cdot \mathrm{X}(0)$$

$$= \begin{bmatrix} \tfrac{2}{3}\cdot(1)^k + \tfrac{1}{3}\cdot(-\tfrac{1}{2})^k & \tfrac{2}{3}\cdot(1)^k - \tfrac{2}{3}\cdot(-\tfrac{1}{2})^k \\ \tfrac{1}{3}\cdot(1)^k - \tfrac{1}{3}\cdot(-\tfrac{1}{2})^k & \tfrac{1}{3}\cdot(1)^k + \tfrac{2}{3}\cdot(-\tfrac{1}{2})^k \end{bmatrix} \cdot \mathrm{X}(0)$$

The steady state solution is obtained for $k \to \infty$:

$$\mathrm{X}(k) = \begin{bmatrix} \tfrac{2}{3} & \tfrac{2}{3} \\ \tfrac{1}{3} & \tfrac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix}$$

We obtain that regardless of the initial conditions $x_1(0)$ and $x_2(0)$, the backoff state probabilities are $x_1(k) = 2/3$ and $x_2(k) = 1/3$.

Problem 1.22 — Solution

Problem 1.23 — Solution

(a)

The lower boundary for the vulnerable period for *A*'s transmission is any time before *A*'s start (at t_A) that overlaps with *A*'s transmission. This is equal to packet transmission time (t_x), which for data rate 1 Mbps and packet length of 44 bytes gives 352 μs or 18 backoff slots. To account for the cases when *A* selects its backoff countdown $b_A < 18$ slots, we can write the lower bound of the vulnerable period as: $max\{t_A - t_x, 0\}$.

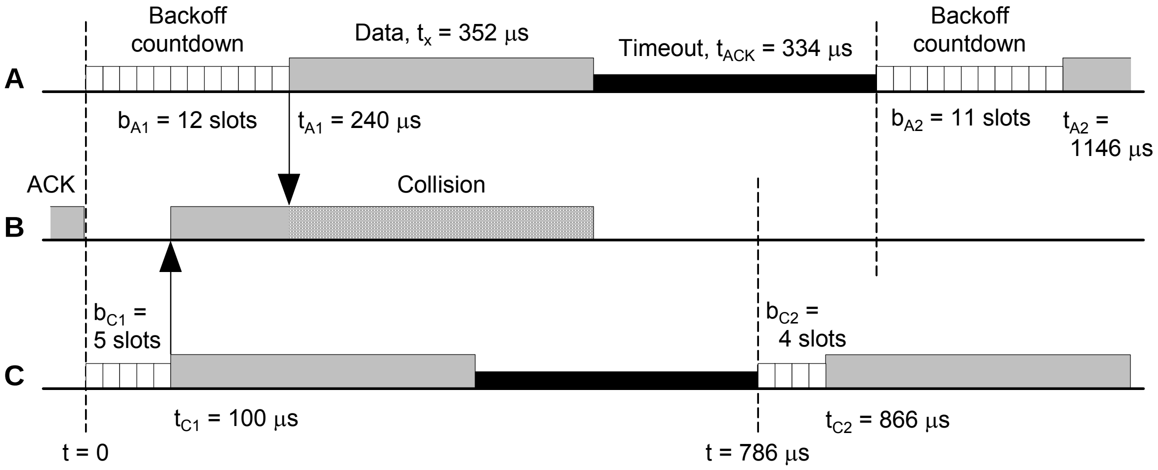
Similarly, the upper bound is the end of *A*'s packet transmission, which is at $t_A + t_x$. To account for the limited size of the backoff contention window *CW*, we can write the lower bound of the vulnerable period as: $min\{t_A + t_x, CW\}$.

Therefore, if *A* starts transmission at t_A, then the vulnerable period for the reception of this packet at the receiver *B* is $[max\{t_A - t_x, 0\}, min\{t_A + t_x, CW\}]$. In our specific example, the vulnerable period is $[max\{12 \times 20\ \mu\text{s} - 352\ \mu\text{s}, 0\}, max\{12 \times 20\ \mu\text{s} + 352\ \mu\text{s}, 32 \times 20\ \mu\text{s}\}] = [0, 592\ \mu\text{s}]$.

(b)

The timing diagram is shown in the figure below.

Note that for the first transmission, *A* and *B* will start their backoff countdown at the same time (synchronized with each other). Conversely, for the second transmission, *A* and *B* will start their backoff countdown at different times (desynchronized with each other).



(c)

Problem 1.24 — Solution

(a)

At Kathleen's computer, the TCP layer will slice the 16-Kbytes letter into payload segments of:

TCP segment size = 512 – 20(TCP hdr) – 20 (IP hdr) = 472 bytes

Total number of IP datagrams generated is: 16,384 ÷ 472 = 34 × 472 + 1 × 336 = 35 IP datagrams

Note that the payload of the first 34 IP datagrams is: 20(TCP hdr) + 472(user data) = 492 bytes and the last one has 356-bytes payload.

(b)

There will be no fragmentation on Link 2, but there will be on Link 3. The IP datagram size allowed by MTU of Link 3 is 256, which means that the payload of each datagram is up to 256 – 20(IP hdr) = 236 bytes. Because the fragment offset is measured in 8-byte chunks, not in bytes, the greatest number divisible by 8 without remainder is 232, which means that each of the 34 incoming datagrams will be split into 3 new fragments: 492 = 232 + 232 + 28, and the last one 35th will be split into two fragments: 356 = 232 + 124.

Total number of fragments on Link 3 is: 34 × 3 + 1 × 2 = 104 datagrams, of which 69 have payload of 232 bytes, 34 have payload of 28 bytes, and 1 has payload of 124 bytes (this is the *last* one). The reader should recall that the IP layer does not distinguish any structure in its payload data. Each original IP datagram sent by Kathleen's computer contains a TCP header and user data (this structure is unknown to IP). The TCP header will be end up in each first fragment and there will be 212 bytes of user data, and the remaining two fragments of each datagram will contain only user data. Again, IP is not aware of the payload structure and treats the whole payload in the same way (i.e., it does not do anything special with the fragment containing the TCP header).

(c)

Recall that the first datagram ID = 672. There are 35 IP datagrams leaving Kathleen's computer and the ID of the last datagram is 672 + 35 − 1 = 706. The offset of the second fragment is calculated as: (1st fragment's payload)/8-bytes = 232/8 = 29. Then,

First 4 packets that Joe receives	Last 5 packets that Joe receives
#1: Length = 232, ID = 672, MF = 1, Offset = 0	#100: Length = 232, ID = 705, MF = 1, Offset = 0
#2: Length = 232, ID = 672, MF = 1, Offset = 29	#101: Length = 232, ID = 705, MF = 1, Offset = 29
#3: Length = 28, ID = 672, MF = 0, Offset = 58	#102: Length = 28, ID = 705, MF = 0, Offset = 58
#4: Length = 232, ID = 673, MF = 1, Offset = 0	#103: Length = 232, ID = 706, MF = 1, Offset = 0
	#104: Length = 124, ID = 706, MF = 0, Offset = 29

(d)

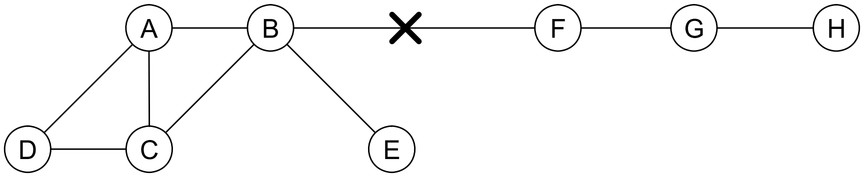
If the very last (104th) fragment is lost, Joe's computer cannot reassemble the last (35th) IP datagram that was sent by Kathleen's computer. Therefore, the TCP sender will not receive an acknowledgement for the last segment that it transmitted and, after the retransmission timer expires, it will resend the last segment of 336 bytes of data + 20 bytes TCP header. The IP layer will create a datagram with the payload of 356 bytes. So, Kathleen's computer will retransmit only 1 IP datagram. This datagram will be fragmented at Link 3 into 2 IP datagrams. So, Joe's computer will receive 2 IP datagrams. The relevant parameters for these two fragments are:

#105. Length = 232, ID = 707, MF = 1, Offset = 0
#106. Length = 124, ID = 707, MF = 0, Offset = 29

Note that the ID of the retransmitted segment is different from the original, to avoid confusion between the fragments of two different datagrams.

Problem 1.25 — Solution

After the steps (a) through (e), the network will look like:



The link-state advertisements flooded after the step (a) are as follows:

Node A:

Seq#	Neighbor	Cost
1	B	
	C	

Node B:

Seq#	Neighbor	Cost
1	A	1
	C	1
	F	∞

Node C:

Seq#	Neighbor	Cost
1	A	1
	B	1

Node F:

Seq#	Neighbor	Cost
1	B	∞
	G	1

Node G:

Seq#	Neighbor	Cost
1	F	1

[We assume that the sequence number starts at 1 in step (a), although a valid assumption would also be that it is 1 before step (a).]

In the remaining steps, we show only those LSAs that are different from the previous step. (Note that the sequence number changes in every step for all LSAs, including the ones not shown.)

Step (b):

Node G:

Seq#	Neighbor	Cost
2	F	1
	H	1

Node H:

Seq#	Neighbor	Cost
2	G	1

Step (c):

Node C:

Seq#	Neighbor	Cost

Node D:

Seq#	Neighbor	Cost

3	A	1
	B	1
	D	1

3	C	1

Step (d):

Node B:

Seq#	Neighbor	Cost
4	A	1
	C	1
	E	1

Node E:

Seq#	Neighbor	Cost
4	B	1

Step (e):

Node A:

Seq#	Neighbor	Cost
5	B	1
	C	1
	D	1

Node D:

Seq#	Neighbor	Cost
5	A	1
	C	1

Step (f):

Node B:

Seq#	Neighbor	Cost
6	A	1
	C	1
	E	1
	F	1

Node F:

Seq#	Neighbor	Cost
6	B	1
	G	1

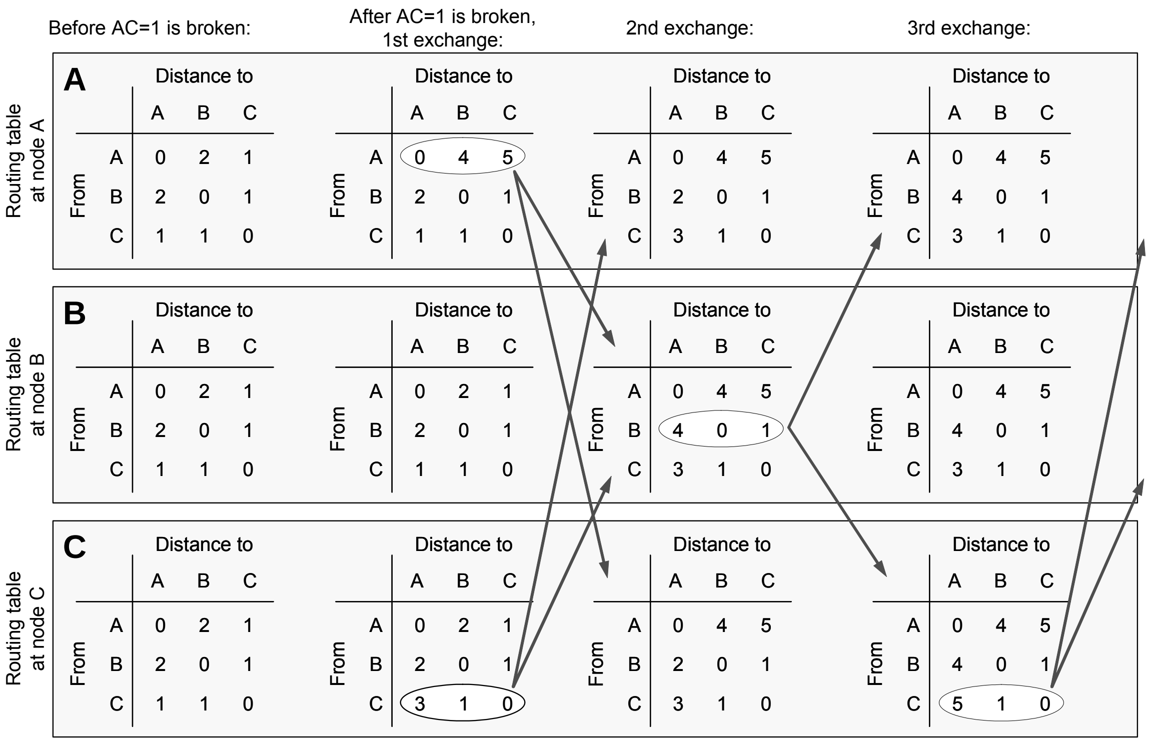
Problem 1.26 — Solution

Problem 1.27 — Solution

The tables of distance vectors at all nodes after the network stabilizes are shown in the leftmost column of the figure below. Note that, although, there are two alternative $\overline{AC}$ links, the nodes select the best available, which is $\overline{AC}$=1.

When the link $\overline{AC}$ with weight equal to 1 is broken, both A and C detect the new best cost $\overline{AC}$ as 50.

1. A computes its new distance vector as
   $$D_A(B) = \min\{c(A,B) + D_B(B),\ c(A,C) + D_C(B)\} = \min\{4+0,\ 50+1\} = 4$$
   $$D_A(C) = \min\{c(A,C) + D_C(C),\ c(A,B) + D_B(C)\} = \min\{50+0,\ 4+1\} = 5$$
   Similarly, C computes its new distance vector as
   $$D_C(A) = \min\{c(C,A) + D_A(A),\ c(C,B) + D_B(A)\} = \min\{50+0,\ 1+2\} = 3$$
   $$D_C(B) = \min\{c(C,B) + D_B(B),\ c(C,A) + D_A(B)\} = \min\{1+0,\ 50+2\} = 1$$
   Having a global view of the network, we can see that the new cost $D_C(A)$ via B is *wrong*. Of course, C does not know this and therefore a *routing loop* is created.
   Both B and C send their new distance vectors out, each to their own neighbors, as shown in the second column in the figure (**first exchange**).
2. Upon receiving C's distance vector, A is content with its current d.v. and makes no changes. Ditto for node C.
   B computes its new distance vector as
   $$D_B(A) = \min\{c(B,A) + D_A(A),\ c(B,C) + D_C(A)\} = \min\{4+0,\ 1+3\} = 4$$



A: Routing table at node A	Before AC=1 is broken: Distance to			After AC=1 is broken, 1st exchange: Distance to			2nd exchange: Distance to			3rd exchange: Distance to		
From	A	B	C	A	B	C	A	B	C	A	B	C
A	0	2	1	0	4	5	0	4	5	0	4	5
B	2	0	1	2	0	1	2	0	1	4	0	1
C	1	1	0	1	1	0	3	1	0	3	1	0

B: Routing table at node B	Before AC=1 is broken: Distance to			After AC=1 is broken, 1st exchange: Distance to			2nd exchange: Distance to			3rd exchange: Distance to		
From	A	B	C	A	B	C	A	B	C	A	B	C
A	0	2	1	0	2	1	0	4	5	0	4	5
B	2	0	1	2	0	1	4	0	1	4	0	1
C	1	1	0	1	1	0	3	1	0	3	1	0

C: Routing table at node C	Before AC=1 is broken: Distance to			After AC=1 is broken, 1st exchange: Distance to			2nd exchange: Distance to			3rd exchange: Distance to		
From	A	B	C	A	B	C	A	B	C	A	B	C
A	0	2	1	0	2	1	0	4	5	0	4	5
B	2	0	1	2	0	1	2	0	1	4	0	1
C	1	1	0	3	1	0	3	1	0	5	1	0

$$D_B(C) = \min\{c(B,C) + D_C(C),\ c(B,A) + D_A(C)\} = \min\{1+0,\ 4+5\} = 1$$

B sends out its new distance vector to *A* and *C* (**second exchange**).

3. Upon receiving *B*'s distance vector, *A* does not make any changes so it remains silent. Meanwhile, *C* updates its distance vector to the correct value for $D_C(A)$ and sends out its new distance vector to *A* and *B* (**third exchange**).
4. *A* and *B* will update the *C*'s distance vector in their own tables, but will *not* make further updates to their own distance vectors. There will be no further distance vector exchanges related to the $\overline{AC}$ breakdown event.

Problem 1.28 — Solution

Problem 1.29 — Solution

(a)

After the network has converged, the routing tables on all routers will be as shown below.

Routing table on node **A**

From	To: A	B	C	D
A	0	7	8	9
B	7	0	1	2

Routing table on node **D**

From	To: A	B	C	D
C	8	1	0	1
D	9	2	1	0

Routing table on node **B**

Routing table on node **C**

To:

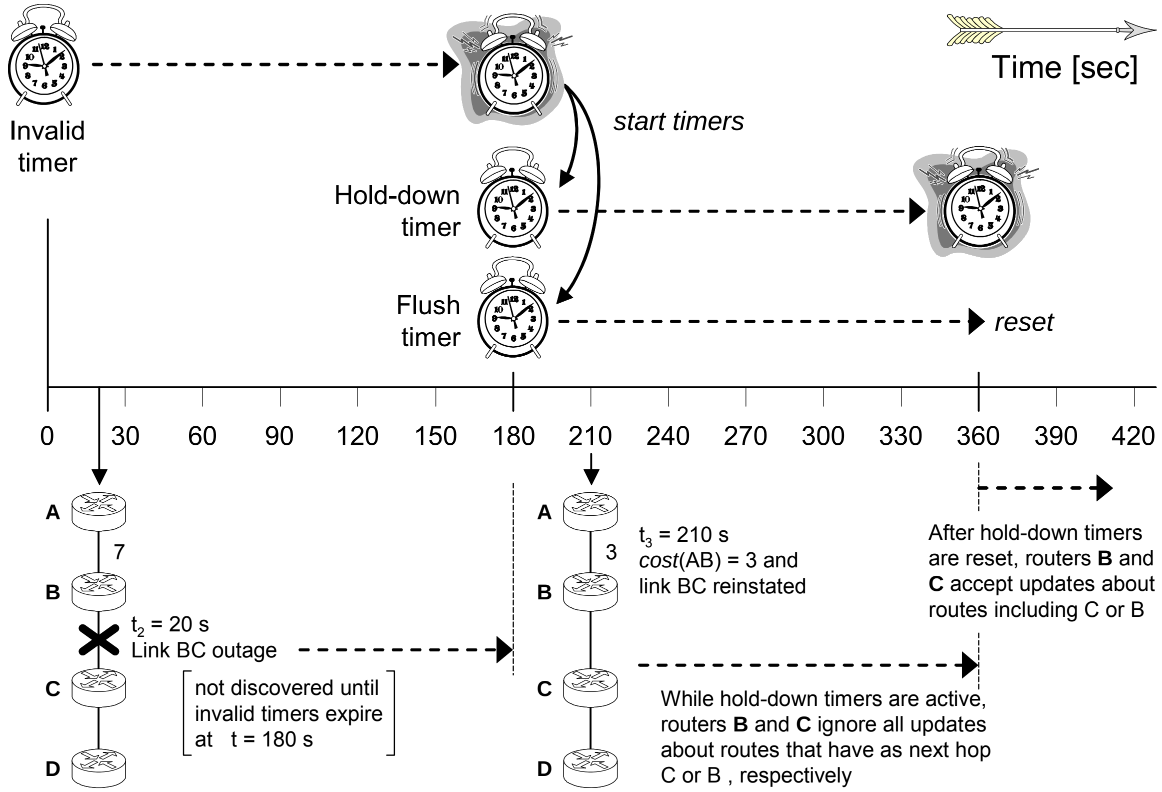
From:	A	B	C	D
A	0	7	8	9
B	7	0	1	2
C	8	1	0	1

To:

From:	A	B	C	D
B	7	0	1	2
C	8	1	0	1
D	9	2	1	0

(b)

The list of messages exchanged between all routers from 0 to 450 seconds is as describe below. The following figure may be useful for tracking the events:



t_2 = 20 s
Link BC outage

not discovered until invalid timers expire at t = 180 s

t_3 = 210 s
cost(AB) = 3 and link BC reinstated

While hold-down timers are active, routers **B** and **C** ignore all updates about routes that have as next hop C or B , respectively

After hold-down timers are reset, routers **B** and **C** accept updates about routes including C or B

Although the link BC failed at 20 seconds, the failure is not detected until the routing protocol misses enough keep-alive messages to declare its neighbor down. Therefore, all routers will send periodic keep-alive updates as if everything were normal:

t = 30 s ⟨periodic updates⟩ A→B, B→A, B→C (lost), C→B (lost), C→D, D→C

Note that B and C will not receive periodic updates from each other, but they will not react immediately because their invalid timers were reset last time at *t* = 0, and have *not* yet expired. Thus, they will neither mark any routes using link BC as "invalid" nor start hold-down and flush timers.

t = 60 s ⟨periodic updates⟩ A→B, B→A, B→C (lost), C→B (lost), C→D, D→C

t = 90 s ⟨periodic updates⟩ A→B, B→A, B→C (lost), C→B (lost), C→D, D→C

t = 120 s ⟨periodic updates⟩ A→B, B→A, B→C (lost), C→B (lost), C→D, D→C

t = 150 s ⟨periodic updates⟩ A→B, B→A, B→C (lost), C→B (lost), C→D, D→C

The invalid timers on routers B and C will keep counting (will *not* be reset), because no new updates will be received for the routes that include link BC. Note that on routers A and D the invalid timers for such routes will be reset because they will receive all periodic updates without any indication about problems with link BC.

At *t* = 180 s, the *invalid timers* will expire on routers B and C for all routes that include link BC. Those routes will be marked as "invalid," and their *hold-down* and *flush timers* will be initiated. In addition, these messages will be sent:

t = 180 s ⟨triggered updates⟩ B→A: { distance(BC=∞), distance(BD=∞) }
C→D: { distance(CB=∞), distance(CA=∞) }

Note that the routers using distance vector routing advertise distance vectors, rather than the state of individual links. B and C will continue sending periodic updates to each other after discovering the link BC outage, so to detect if the link BC becomes reinstated. All subsequent periodic updates will carry infinite distance for routes including link BC (as first advertised by the triggered update at *t* = 180 s).

At *t* = 210 s, the link between the routers B and C is reinstated and a new cost of link AB becomes 3. The periodic updates from B to C and vice versa will now be successful:

t = 210 s ⟨periodic updates⟩ A→B, B→A, B→C, C→B, C→D, D→C

However, because their hold-down timers are started, routers B and C will *ignore everything* new about that route, better metric or worse from the original source node, until the hold-down timers expire. What about routers B and C informing their respective neighbors A and D about the restored link BC? Recall again that with distance vector routing, routers do *not* advertise link state; rather they advertise distance vectors. Given that the hold-down timer is active for all routes including link BC, B and C will ***not*** update their own entries and will ***not*** notify their neighbors. In addition, router B will discover new cost of link AB as 3, and will inform C about the decreased distance from B to A. C will *ignore* this information: it will neither update its own routing entry for destination A, nor will it notify D about the new distance to A.

t = 240 s ⟨periodic updates⟩ A→B, B→A, B→C, C→B, C→D, D→C

t = 270 s ⟨periodic updates⟩ A→B, B→A, B→C, C→B, C→D, D→C

t = 300 s ⟨periodic updates⟩ A→B, B→A, B→C, C→B, C→D, D→C

t = 330 s ⟨periodic updates⟩ A→B, B→A, B→C, C→B, C→D, D→C

At *t* = 360 s, several events will happen at once on routers B and C, so we again order them as instructed in the problem statement. First, the hold-down timers will expire on routers B and C for routes that include link BC. Second, the regular periodic updates will be exchanged, with the same content as for previous periods:

t = 360 s ⟨periodic updates⟩ A→B, B→A, B→C, C→B, C→D, D→C

No action will be taken in response to the expiration of *hold-down timers*, but B and C will from now start accepting updates for paths including BC. Meanwhile, the flush timers will continue counting for another 60 s. If no such updates arrived, all associated "invalid" route would be removed from the routing table. However, as instructed in the problem statement, third, the periodic updates will arrive without delays. These updates will reset the flush timers and cause B and C to modify their distance vectors, so the new distances will be dist(B,C) = 1, dist(B,D) = 2; dist(C,B) = 1; dist(C,A) = 4. The updated distance vectors on B and C will result in triggered update messages:

t = 360 s ⟨triggered updates⟩ B→A: { distance(BC=1), distance(BD=2) }
C→D: { distance(CB=1), distance(CA=4) }

The remaining two periods for the observed interval will be at *t* = 390 s and *t* = 420 s. They will bring no news compared to the previous (triggered) update:

t = 390 s ⟨periodic updates⟩ A→B, B→A, B→C, C→B, C→D, D→C

t = 420 s ⟨periodic updates⟩ A→B, B→A, B→C, C→B, C→D, D→C

(c)

At *t* = 60 s, the routing tables will be as follows:

t = 60 s

Routing table on node **B**

To:

From:	A	B	C	D
A	0	7	8	9
B	7	0	1	2
C	8	1	0	1

Routing table on node **C**

To:

From:	A	B	C	D
B	7	0	1	2
C	8	1	0	1
D	9	2	1	0

As seen, nothing has yet changed, because their invalid timers have not yet expired.

At *t* = 180 s, B's and C's invalid timers have expired, *after* the triggered updates have been processed the routing tables will be as:

t = 180 s

Routing table on node **B**

To:

From:	A	B	C	D	
A	0	3	∞	∞	
B	3	0	∞	∞	
C	8	1	0	1	←invalid

Routing table on node **C**

To:

	From:	A	B	C	D
invalid→	B	7	0	1	2
	C	∞	∞	0	1
	D	∞	∞	1	0

The hold-down timers will be started and the tables remain unchanged until *t* = 360 s.

At *t* = 360 s, *after* the triggered updates have been processed the routing tables will be as:

t = 360 s

Routing table on node **B**

To:

From:	A	B	C	D
A	0	3	4	5
B	3	0	1	2
C	4	1	0	1

Routing table on node **C**

To:

From:	A	B	C	D
B	3	0	1	2
C	4	1	0	1
D	5	2	1	0

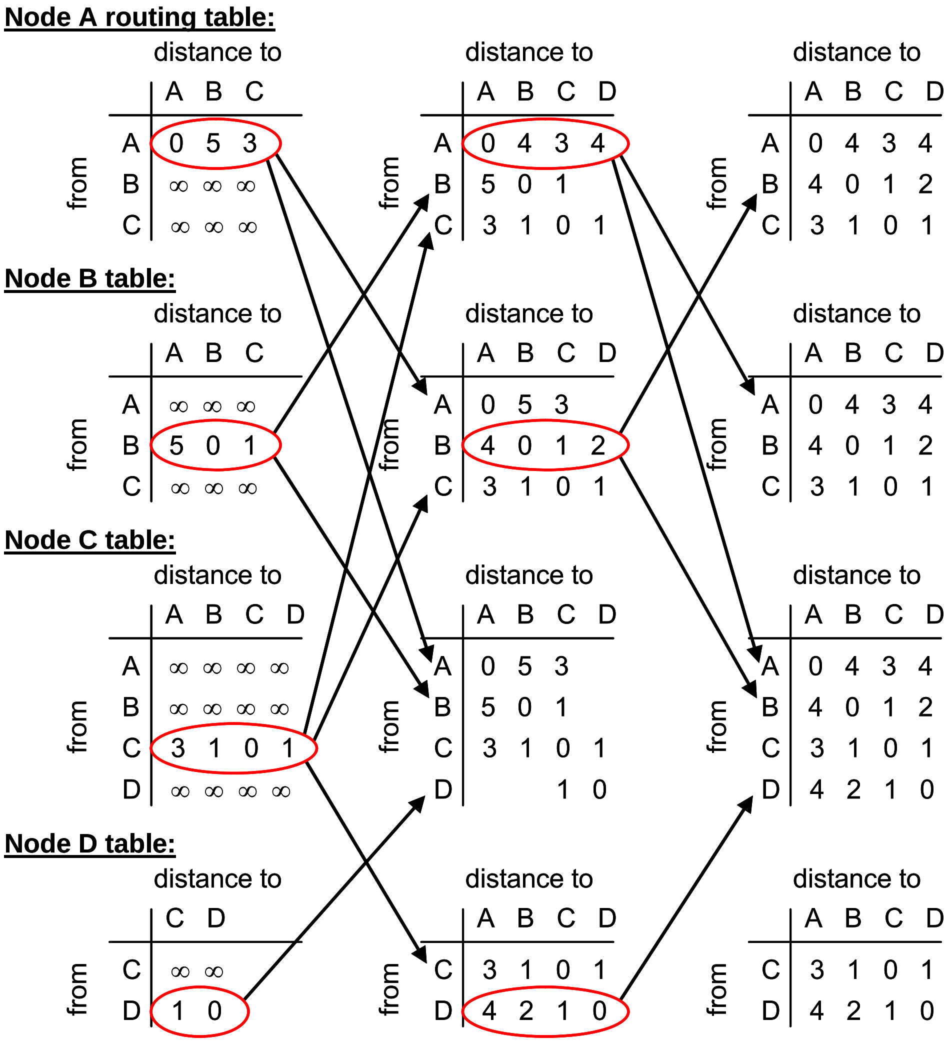
The tables will remain unchanged for the rest of the observed interval.

Additional information about hold-down timer behavior in a real network can be found here: https://learningnetwork.cisco.com/thread/20422 and here: http://blog.ine.com/2010/04/15/how-basic-are-rip-timers-test-your-knowledge-now/ .

Problem 1.30 — Solution

(a)

The following figure shows how the routing tables are constructed until they stabilize.



Node A routing table:

distance to

from	A	B	C
A	0	5	3
B	∞	∞	∞
C	∞	∞	∞

distance to

from	A	B	C	D
A	0	4	3	4
B	5	0	1	
C	3	1	0	1

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

Node B table:

distance to

from	A	B	C
A	∞	∞	∞
B	5	0	1
C	∞	∞	∞

distance to

from	A	B	C	D
A	0	5	3	
B	4	0	1	2
C	3	1	0	1

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

Node C table:

distance to

from	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	3	1	0	1
D	∞	∞	∞	∞

distance to

from	A	B	C	D
A	0	5	3	
B	5	0	1	
C	3	1	0	1
D			1	0

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1
D	4	2	1	0

Node D table:

distance to

from	C	D
C	∞	∞
D	1	0

distance to

from	A	B	C	D
C	3	1	0	1
D	4	2	1	0

distance to

from	A	B	C	D
C	3	1	0	1
D	4	2	1	0

(b)

The forwarding table at node *A* after the routing tables stabilize is shown in the figure below. Note that the forwarding table at each node is kept separately from the node's routing table.

Node A forwarding table:

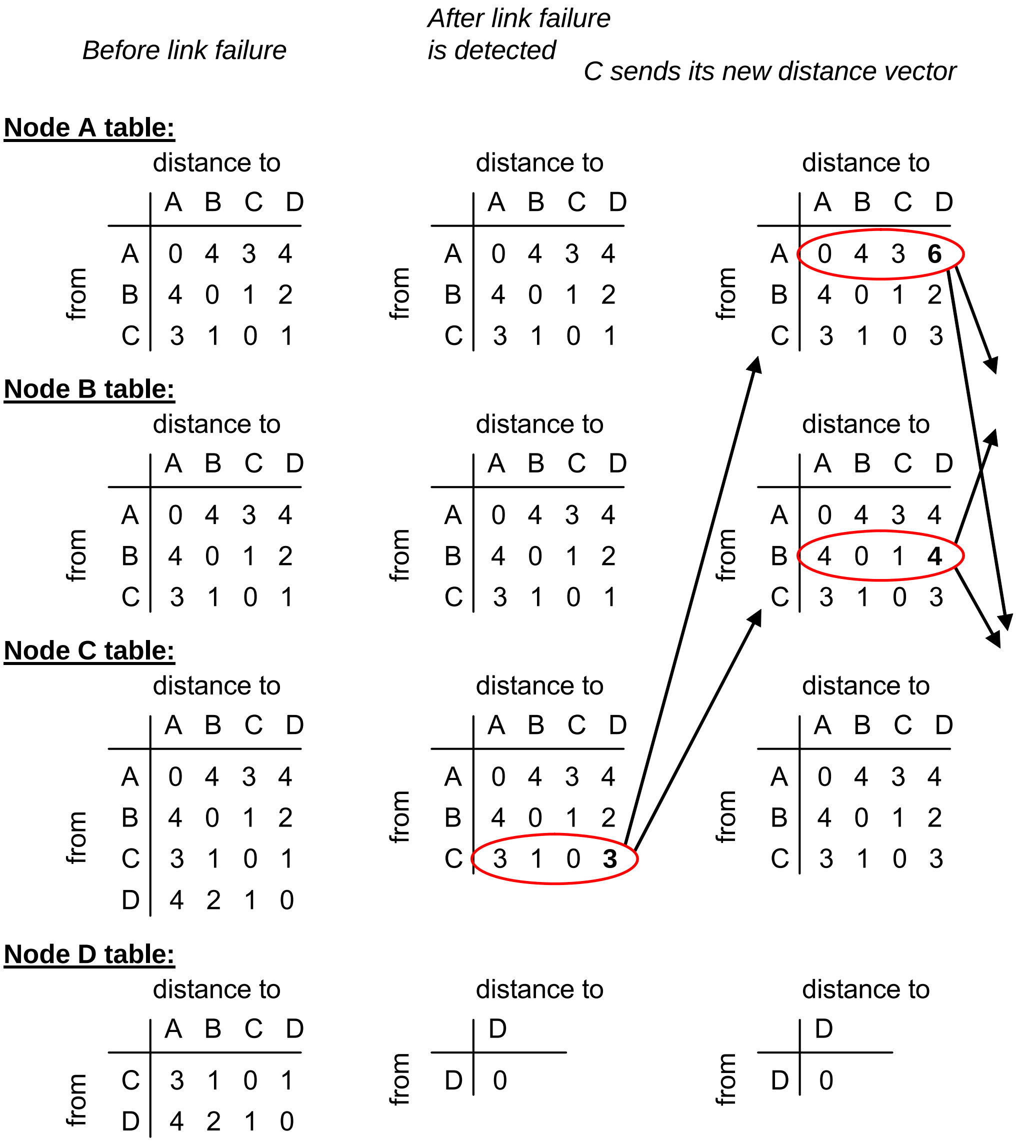
destination	interface
A	--
B	AC
C	AC
D	AC

(c)

See also Problem 1.27 — Solution.

First, consider the figure below. *C* is directly connected to *D*, so after *C* detected that link *CD* is broken, it will set its distance to *D* as infinity. Then *C* recalculates its distance vector using equation (1.14). The new minimum distance to *D* is via B, because $c(C, B) = 1$ and $D_B(D) = 2$. Now *C* thinks that the new shortest distance to *D* is 3 (via *B*). It sends its updated distance vector to its neighbors (*A* and *D*) and they update their distance vectors, as shown in the figure.

A routing loop is formed because a packet sent from *A* to *D* would go to *C* and *C* would send it to *B* (because *C*'s new shortest path to *D* is via *B*). *B* would return the packet to *C* because *B*'s shortest path to *D* is still via *C*. This looping of the packet to *D* would continue forever.



Before link failure

After link failure is detected

C sends its new distance vector

Node A table:

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

distance to

from	A	B	C	D
A	0	4	3	**6**
B	4	0	1	2
C	3	1	0	3

Node B table:

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	**4**
C	3	1	0	3

Node C table:

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1
D	4	2	1	0

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	**3**

distance to

from	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	3

Node D table:

distance to

from	A	B	C	D
C	3	1	0	1
D	4	2	1	0

distance to

from	D
D	0

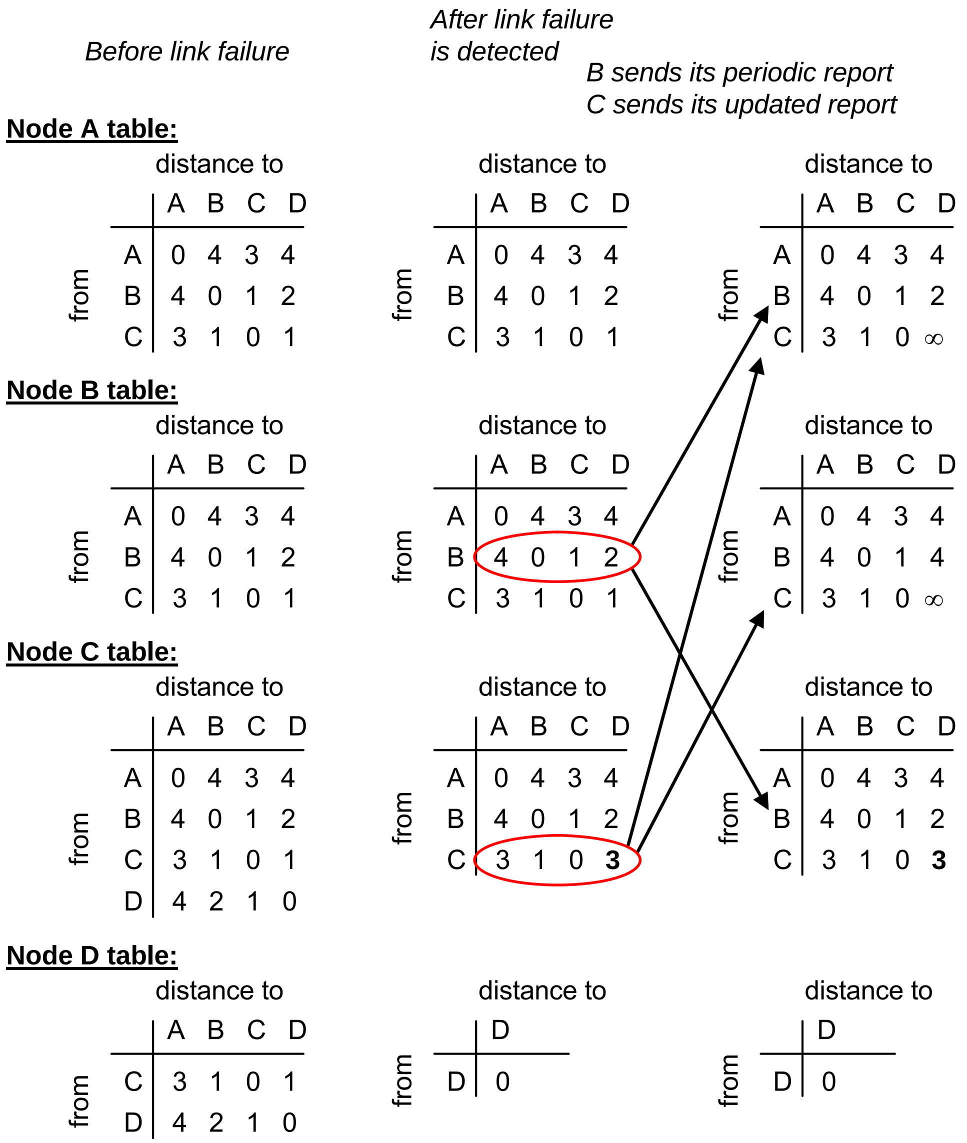
distance to

from	D
D	0

(d)

If the nodes use split-horizon routing, then neither *A* nor *B* would advertise to *C* their distances to *D*, because *C* is the next hop for both of them on their paths to *D*. Therefore, in principle *D* would not think there is an alternative path to *D*.

Even in this case, it is possible that a routing loop forms. The key to a routing loop formation in this case is the periodic updates that nodes running a distance vector protocol are transmitting. Thus, the cycle shown below is due to the pathological case where *B* transmits its periodic report some time in the interval after the link *CD* crashes, but before *B* receives update information about the outage to *D* from node *C*.



Before link failure

After link failure is detected

B sends its periodic report
C sends its updated report

Node A table:

Before link failure:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

After link failure is detected:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

B sends its periodic report / C sends its updated report:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	∞

Node B table:

Before link failure:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

After link failure is detected:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1

B sends its periodic report / C sends its updated report:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	4
C	3	1	0	∞

Node C table:

Before link failure:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	1
D	4	2	1	0

After link failure is detected:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	**3**

B sends its periodic report / C sends its updated report:

from \ distance to	A	B	C	D
A	0	4	3	4
B	4	0	1	2
C	3	1	0	**3**

Node D table:

Before link failure:

from \ distance to	A	B	C	D
C	3	1	0	1
D	4	2	1	0

After link failure is detected:

from \ distance to	D
D	0

B sends its periodic report / C sends its updated report:

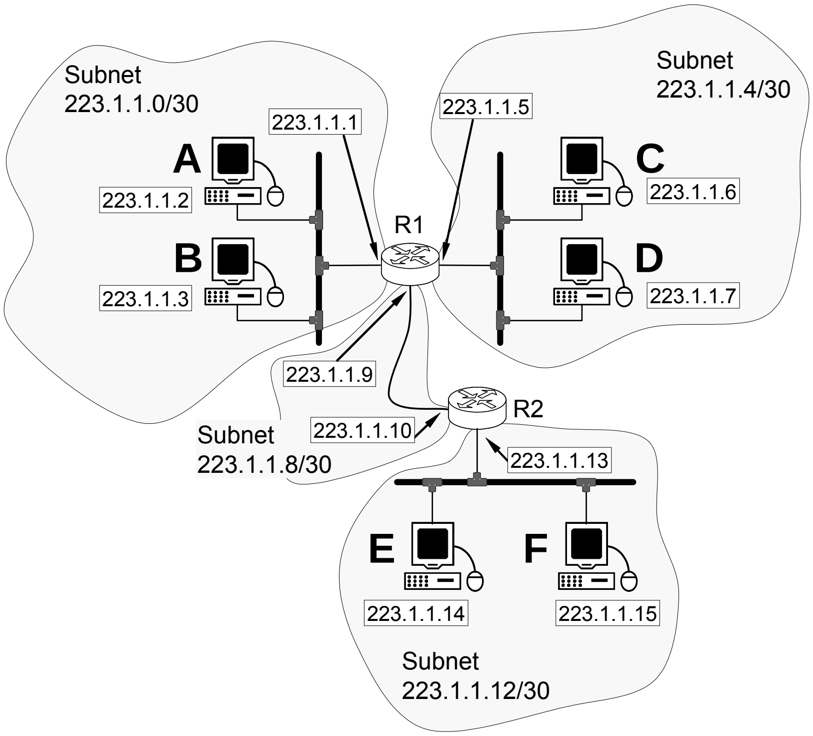
from \ distance to	D
D	0

Problem 1.31 — Solution

Problem 1.32 — Solution

(a)

Example IP address assignment is as shown:



(b)

Routing tables for routers R1 and R2 are:

Router R1	**Destinat. IPaddr / Network prefix**	**Next hop**	**Output interface**
	223.1.1.2 (A)	223.1.1.2 (A)	223.1.1.1
	223.1.1.3 (B)	223.1.1.3 (B)	223.1.1.1
	223.1.1.6 (C)	223.1.1.6 (C)	223.1.1.5
	223.1.1.7 (D)	223.1.1.7 (D)	223.1.1.5
	223.1.1.12/30	223.1.1.10	223.1.1.9

Router R2	**Destinat. IPaddr / Network prefix**	**Next hop**	**Output interface**
	223.1.1.14 (E)	223.1.1.14 (E)	223.1.1.13
	223.1.1.15 (F)	223.1.1.15 (F)	223.1.1.13
	223.1.1.0/30	223.1.1.9	223.1.1.10
	223.1.1.4/30	223.1.1.9	223.1.1.10

Problem 1.33 — Solution

Recall that in CIDR the *x* most significant bits of an address of the form *a.b.c.d*/*x* constitute the network portion of the IP address, which is referred to as ***prefix*** (or *network prefix*) of the address. In our case the forwarding table entries are as follows:

Network prefix	Binary form	Next hop
223.92.32.0/20	**11011111 01011100 0010**0000 00000000	A
223.81.196.0/12	**11011111 0101**0001 11000100 00000000	B
223.112.0.0/12	**11011111 0111**0000 00000000 00000000	C
223.120.0.0/14	**11011111 011110**00 00000000 00000000	D
128.0.0.0/1	**1**0000000 00000000 00000000 00000000	E
64.0.0.0/2	**01**000000 00000000 00000000 00000000	F
32.0.0.0/3	**001**00000 00000000 00000000 00000000	G

Note that the network prefix is shown in bold face, whereas the remaining 32–*x* bits of the address are shown in gray color. When forwarding a packet, the router considers only the leading *x* bits of the packet's destination IP address, i.e., its network prefix.

	Packet destination IP address			Longest prefix match	Next hop
(a)	195.145.34.2	=	11000011 10010001 00100010 00000010	1	E
(b)	223.95.19.135	=	11011111 01011111 00010011 10000111	11011111 0101	B
(c)	223.95.34.9	=	11011111 01011111 00100010 00001001	11011111 0101	B
(d)	63.67.145.18	=	00111111 01000011 10010001 00010010	001	G
(e)	223.123.59.47	=	11011111 01111011 00111011 00101111	11011111 011110	D
(f)	223.125.49.47	=	11011111 01111101 00110001 00101111	11011111 0111	C

One may wonder if the next hop packet destination for destination (f) should be D, instead of C. The IP address (f) has one-bit longer prefix match with the network prefix for D than it does with C. The reason that C is the best match is because the destination address has to match the *entire* prefix *A*/*x* (i.e., all *x* bits), and in this sense C, not D, is the best match for (f).

Problem 1.34 — Solution

The packet forwarding associations are given as follows:

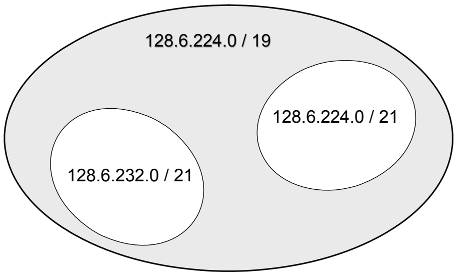
	Destination IP address	Binary representation	Next hop
(a)	128.6.4.2 (cs.rutgers.edu)	10000000 00000110 00000100 00000010	A
(b)	128.6.236.16 (caip.rutgers.edu)	10000000 00000110 11101100 00010000	B
(c)	128.6.29.131 (ece.rutgers.edu)	10000000 00000110 00011101 10000011	C
(d)	128.6.228.43 (toolbox.rutgers.edu)	10000000 00000110 11100100 00001010	D

From this representation, we can reconstruct the forwarding table as:

Network Prefix	Binary form	Next Hop
128.6.0.0 / 20	10000000 00000110 0000	A
128.6.232.0 / 21	10000000 00000110 11101	B
128.6.16.0 / 20	10000000 00000110 0001	C

128.6.224.0 / 19	10000000 00000110 111	D

Note that in the last row we could have used the prefix "128.6.224.0 / 21," or in binary form "10000000 00000110 11100." However, it suffices to use only 19 most significant bits because the router forwards to the *longest possible match*, as explained next.



If we calculate the range of address associated with the prefix 128.6.224.0/19 (last row in the above forwarding table), we find the range as 128.6.224.0 to 128.6.255.254. The addresses associated with 128.6.232.0/21 (second row) are 128.6.232.1 to 128.6.239.254. As seen, the latter set of addresses is a subset of the former set, so one may think that the router will route all addresses starting with 128.6.224.0/19 to the next hop *D*. This is not the case. When a packet arrives with 128.6.228.43 in its header, the router will find that the longest match is 128.6.224.0/19 (last row in the above table). Hence, this packet will be forwarded to *D*. Alternatively, when a packet arrives with 128.6.236.16 in its header, the router will find that there are two matches in the forwarding table: 128.6.224.0/19 (last row) and 128.6.232.0/21 (second row). Of these two, the latter is longer by two bits, so the router will not get confused and route this packet to *D*. Rather, the router will correctly route the packet to *B*.

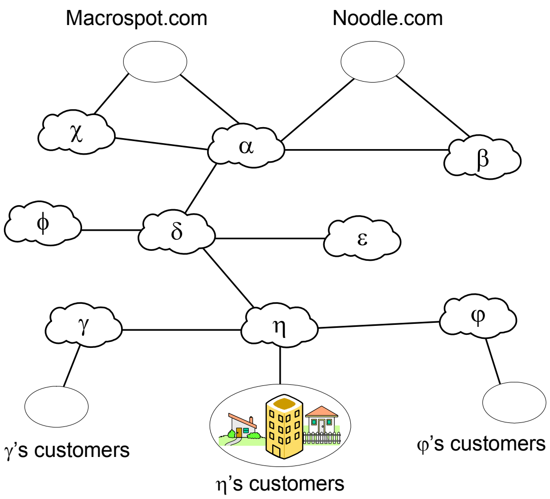
Problem 1.35 — Solution

Problem 1.36 — Solution

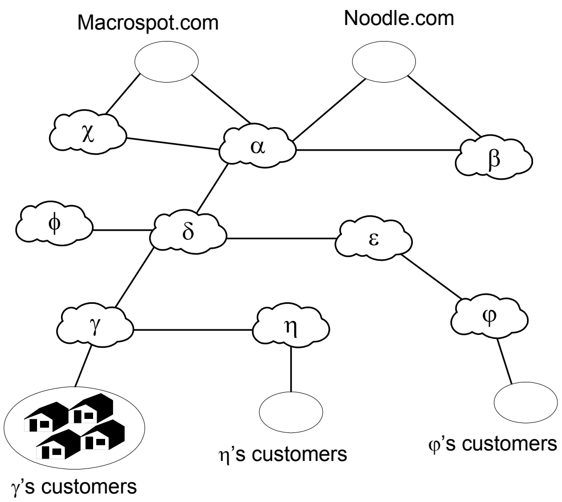
The observed topology of an internetwork of autonomous systems depends on the vantage point. The view from ASφ's vantage point is shown in the problem statement. When solving the problem for other stub ASs, it is key to keep in mind that ASs do not like to provide transit without being paid. ASs will happily provide transit to their paying customers, or will provide access of their customers to their peers, but will not provide free transit to their peers.

The solutions for the remaining vantage points are as follows:

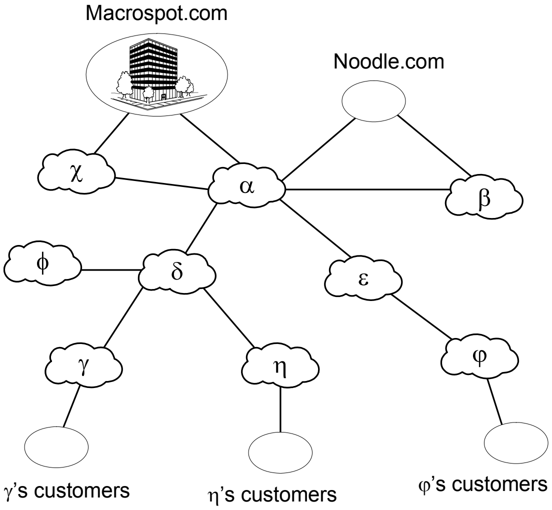
Topology view from η's customers



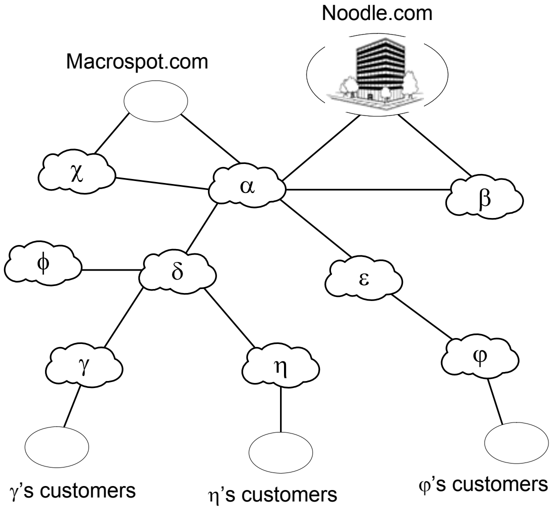
Topology view from γ's customers



Topology view from Macrospot.com



Topology view from Noodle.com



Problem 1.37 — Solution

(a)

Recall that autonomous systems use path vector routing for inter-domain routing (Section 1.4.5). ASα will receive three routes to ASβ, along router links *CH*, path: $\langle 1 \mid \beta \rangle$; *DK*, path: $\langle 1 \mid \beta \rangle$; and *DE*, path: $\langle 2 \mid \gamma, \beta \rangle$.

(b)

X→*Y* traffic will take link *CH*, because this is the shortest path when crossing ASα. Recall that internally ASs use hot-potato routing, so when a packet from *X* arrives at router *A*, the shortest path across ASα to ASβ is *AC*, then *CH*. Note that this strategy minimizes cost to the source of the traffic (i.e., ASα) and is not optimal to other ASs along the path, such as ASβ.

By the same reasoning, $Y \to X$ traffic will take link *KD*. The resulting paths are $X \to A \to C \to H \to I \to J \to Y$ and $Y \to J \to K \to D \to B \to A \to X$. In both cases, hot-potato routing within the given AS does not look at the overall path length, but only the path length within this AS.

(c) To have all $X \to Y$ traffic take link *DK*, the Exterior Gateway Protocol of ASα could simply be configured with a routing policy that prefers link *DK* in all cases. The Exterior Gateway Protocol of ASα then would simply not tell the Interior Gateway Protocol (IGP) of ASα that it is possible to reach ASβ via the speaker *C*.

The only general solution, though, is for ASα to accept into its routing tables some of the internal structure of ASβ, so that the IGP protocol of ASα, for example, knows where *Y* is relative to links *CH* and *DK*. (In our example both alternative paths $X \to Y$ (or $Y \to X$) are equally long, so both ASs would need to break the tie in the same way.)

Of course, if one of the links *CH* or *DK* (or an attached speaker) goes down, then all $X \to Y$ traffic and $Y \to X$ traffic will be forced to take the same path.

(d)
If ASα were configured with a routing policy to prefer AS paths through ASγ, or to avoid AS paths involving links direct links to ASβ, then ASα might route to β via γ.

Problem 1.38 — Solution

Problem 1.39 — Solution

(a)

The path table entries for the destination ASδ before the link βδ outage:

From:		To: δ
	α	⟨3 | α, γ, β, δ⟩
	β	⟨1 | β, δ⟩
	γ	⟨2 | γ, β, δ⟩
	δ	⟨0 | δ⟩

(b)

From the table in part (a) we can see that every AS uses the link βδ as part of the shortest path to ASδ, so every AS will be affected by its failure.

(c)

First, ASβ and ASδ will discover that the link βδ has failed. We know from part (b) that all AS's will be affected by βδ outage, but ASβ and ASδ have no way of knowing who will be affected and how. Therefore, they have to send an "unreachable" message to all of their neighbors to which they previously advertised this path, stating that path ⟨β, δ⟩ is not available:

β→α and β→γ: ⟨"unreachable" | β, δ⟩

δ→γ: ⟨"unreachable" | δ, β⟩

On receiving the message from ASβ, ASα and ASγ will realize that their path to ASδ was through ASβ (see the table above), and will mark this path as unavailable. On the other hand, when ASγ receives the message from ASδ (second row above), it will realize that its path to ASβ was *not* through ASδ, so ASγ will ignore the message.

When ASγ receives the message from ASβ, it will realize that it can reach ASδ via a direct link, so it will inform ASβ of its path to ASδ:

$\gamma \rightarrow \beta$: $\langle 7 \mid \gamma, \delta \rangle$

In addition, ASα and ASγ will forward the “unreachable” message to all of their “downstream” neighbors, in case they are also using ASβ on a path to destination ASδ. The only “downstream” neighbor for ASα is ASγ, while ASβ is “upstream” in the sense that the message arrived from there. ASγ will not inform its other neighbors (i.e., ASα) about the alternative path to ASδ, because it would be ignored as inferior to their existing path to ASδ. They need first to receive “unreachable” message, before they are ready to consider an inferior path. Therefore,

$\alpha \rightarrow \gamma$: $\langle \text{“unreachable”} \mid \beta, \delta \rangle$ and $\gamma \rightarrow \alpha$: $\langle \text{“unreachable”} \mid \beta, \delta \rangle$

Both ASα and ASγ will realize that they have already seen this message and ignore it. However, ASγ will in response inform ASα of its direct path to ASδ. Meanwhile, at about the same time ASβ will inform ASα about its new path to ASδ via ASγ:

$\gamma \rightarrow \alpha$: $\langle 7 \mid \gamma, \delta \rangle$ and $\beta \rightarrow \alpha$: $\langle 8 \mid \beta, \gamma, \delta \rangle$

ASα will choose the better option (i.e., $\langle 8 \mid \alpha, \gamma, \delta \rangle$ and with this step, the system has reached a new equilibrium.

(d)

The new equilibrium entries for ASδ after the outage of βδ:

		To: δ
From:	α	$\langle 8 \mid \alpha, \gamma, \delta \rangle$
	β	$\langle 8 \mid \beta, \gamma, \delta \rangle$
	γ	$\langle 7 \mid \gamma, \delta \rangle$
	δ	$\langle 0 \mid \delta \rangle$

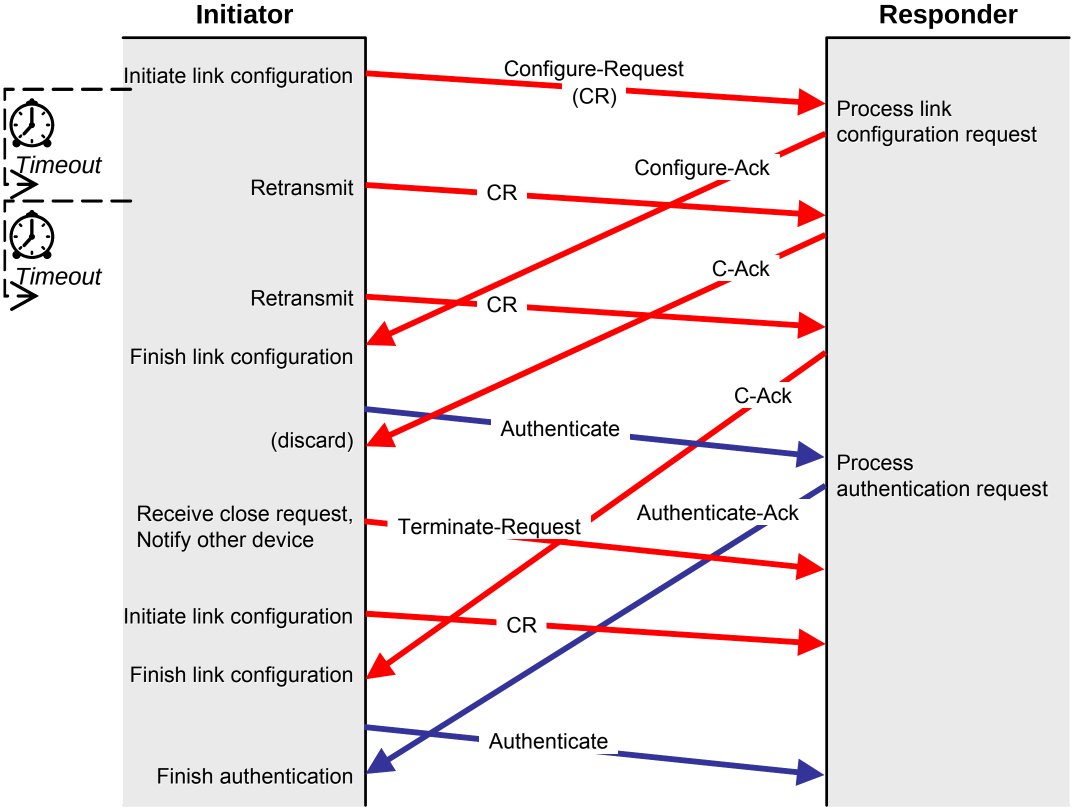
Problem 1.40 — Solution

Problem 1.41 — Solution

PPP and Link Control Protocol (LCP)

(a)

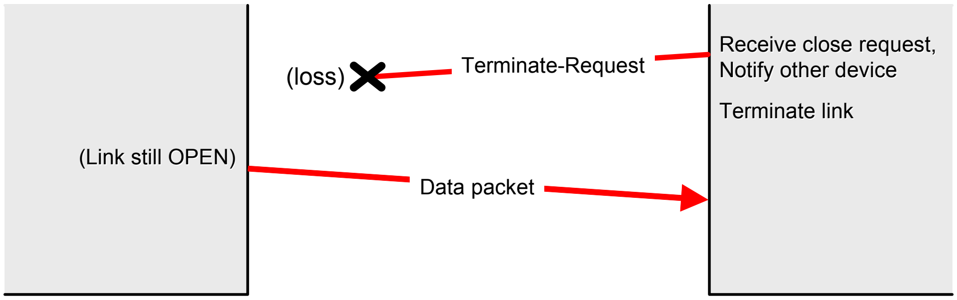
If packets are not identified with unique identifiers, then the following sequence may happen, due to delayed acknowledgements from Responder. The delayed ACKs are falsely interpreted as being ACKs for a subsequent disconnect and link configuration. This in turn causes a failure in authentication.



Note: I was not able to find in the literature explicitly stated the rationale for unique identifiers in LCP frames. This example is based on [Bertsekas & Gallager, 1992], Section 2.7.2, Figure 2.44, which is an example of initialization failure for the HDLC protocol. Because PPP is derived from HDLC, I assume that its designers anticipated this problem. Another example is also from [Bertsekas & Gallager, 1992], Section 2.7.4, Figure 2.46, which happens in case of a sequence of node failures.

(b)

If it is not necessary to acknowledge a Terminate-Request, then one side may terminate the link, while the other side still considers the link open and keeps sending data.

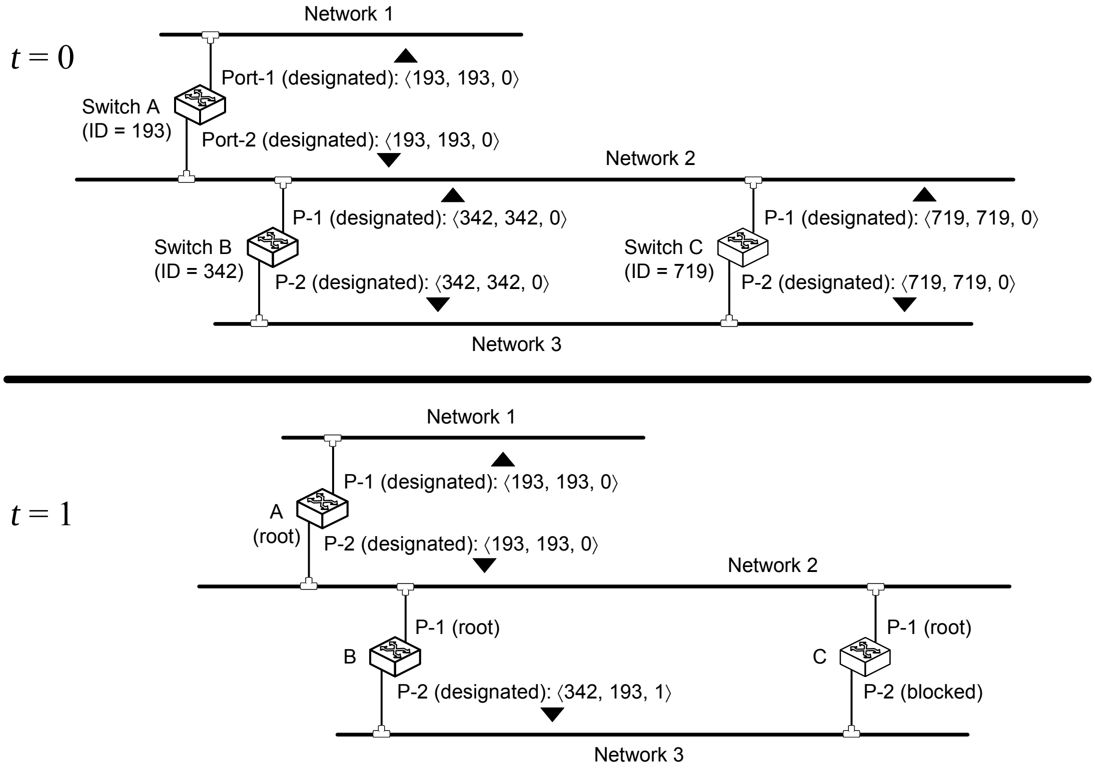


Problem 1.42 — Solution

Problem 1.43 — Solution

(a) and (b)

See the figure below for the configuration messages that are sent and how the ports are labeled. The network converges to its spanning tree in just two iterations.


$t = 0$
Network 1
Port-1 (designated): ⟨193, 193, 0⟩
Switch A
(ID = 193)
Port-2 (designated): ⟨193, 193, 0⟩
Network 2
P-1 (designated): ⟨342, 342, 0⟩
Switch B
(ID = 342)
P-2 (designated): ⟨342, 342, 0⟩
P-1 (designated): ⟨719, 719, 0⟩
Switch C
(ID = 719)
P-2 (designated): ⟨719, 719, 0⟩
Network 3
$t = 1$
Network 1
P-1 (designated): ⟨193, 193, 0⟩
A
(root)
P-2 (designated): ⟨193, 193, 0⟩
Network 2
P-1 (root)
B
P-2 (designated): ⟨342, 193, 1⟩
P-1 (root)
C
P-2 (blocked)
Network 3

At $t = 0$:

Initially each switch selects itself as the "root switch" and for all the attached networks as the "designated switch." The following notation ⟨x, y, z⟩ symbolizes a message carrying the sending switch's identifier x, the identifier y of the switch that x believes is the root switch, and the distance z (in hops) from the sending switch to the root switch.

Messages (each switch on both ports): Switch *A*: ⟨193, 193, 0⟩; Switch *B*: ⟨342, 342, 0⟩; Switch *C*: ⟨719, 719, 0⟩.

At $t = 1$:

Switch *A* selects itself as the "designated switch" for Network segments 1 and 2 because it has the lowest ID on both segments. Both of its ports become "designated."

Switch *B* selects *A* as the "designated switch" for Network 2 and selects itself as the "designated switch" for Network 3, because these are the lowest ID on their respective network segments. Its Port-1 becomes "root port" and Port-2 becomes "designated port."

Switch *C* selects *A* as the “designated switch” for Network 2 and selects *B* as the “designated switch” for Network 3, because they have the lowest ID on their respective network segments. Its Port-1 becomes “root port” and its Port-2 becomes blocked.

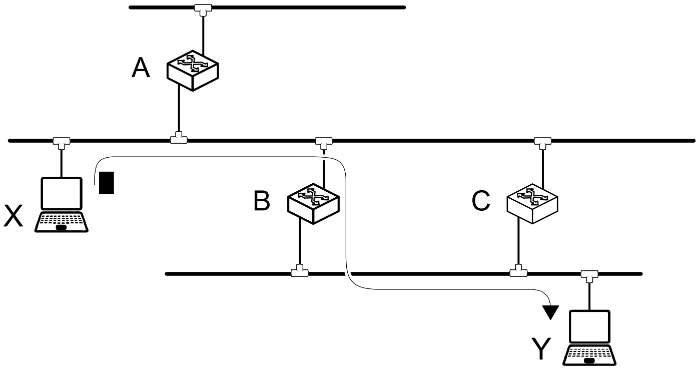
Messages: Switch *A*: ⟨193, 193, 0⟩ on both ports; Switch *B*: ⟨342, 193, 1⟩ on Port-2; Switch *C*: none.

At $t = 2$:

Messages: Switch *A*: ⟨193, 193, 0⟩ on both ports; Switch *B*: ⟨342, 193, 1⟩ on Port-2; Switch *C*: none.

(c)
The network reached the stable state in two iterations. After this, only switch *A* (root) will generate configuration messages and switch *B* will forward these messages only over its Port-2 for which it is the designated switch.

(d)
After the network stabilizes, a frame sent by station *X* to station *Y* will traverse the path shown in the figure below. Note that, unlike routing, in LAN switches the frame is not first transmitted to the “next hop” and then relayed by that hop. That is, switch *A* does not “relay” a packet from host *X* to host *Y*, although *A* is elected as the root of the spanning tree. This is because the spanning tree protocol (STP) operates transparently to the backward learning algorithm, which learns the switching tables based on the network topology configured by STP unknown to the backward learning algorithm. In our example, only switch *B* will “relay” a packet from host *X* to host *Y*.



Problem 1.44 — Solution

Problem 1.45 — Solution

(a)

Let us name the LANs so that LAN-*X* denotes the LAN to which host *X* is connected. Switch *S*1 will be elected the root switch because of its smallest identifier.

Switch	Root port	Designated ports	Blocked ports
S1	–	–	–

S2	1	2, for LAN-*EF*	
S3	1	2, for LAN-*C* | 4, for LAN-*D*	3
S4	1	2, for LAN-*FG*	3
S5	~~1 or~~ 2 ~~or 3~~ (tie)		1, 3

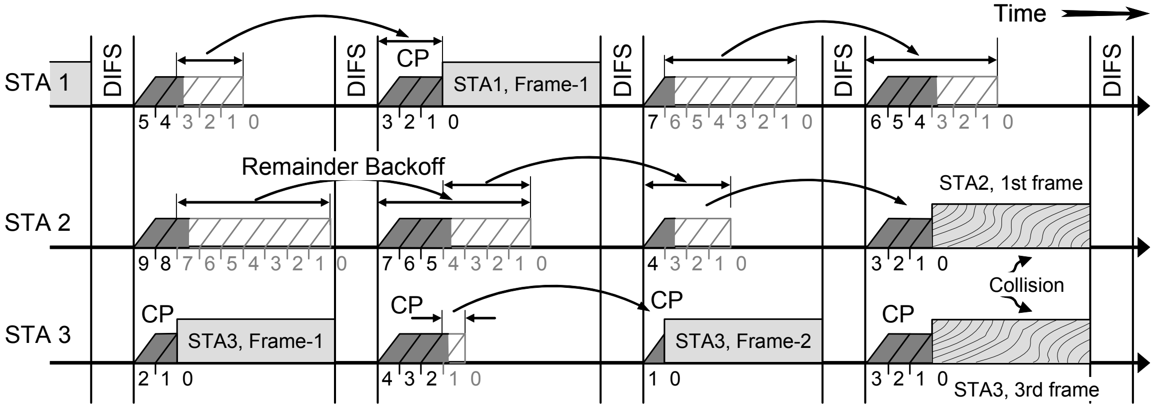
In case of *S*5, there is a tie for its "root" port. Port 2 will be selected as "root" because it is connecting this switch to the neighbor switch with the lowest identifier, i.e., *S*2.

(b)

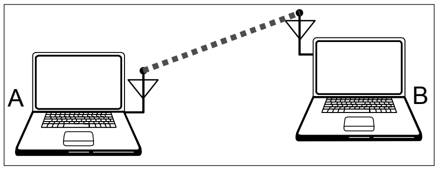
After switch *S*3 suffers a breakdown, the "designated" ports of the remaining switches will be selected as: *S*2 port 2 for LAN-*EF*; *S*4 port 2 for LAN-*FG*; *S*5 port 1 for LAN-*D*; and LAN-*C* will be disconnected from the rest of the network.

Problem 1.46 — Solution

The solution for three stations using the 802.11 protocol is similar to Problem 1.20 — Solution and is shown in the figure below. Again, the third station has the smallest initial backoff and transmits the first. The other two stations will freeze their countdown when they sense the carrier as busy. After the first frame STA3 randomly chooses the backoff value equal to 4, and the other two stations resume their previous countdown. STA1 reaches zero first and transmits, while STA2 and STA3 freeze their countdown waiting for the carrier to become idle. Next, STA3 transmits its second frame and randomly chooses the backoff value equal to 3, and the other two stations again resume their previous countdown. STA2 finally transmits its first frame but STA3 simultaneously transmits its third frame and there is a collision.

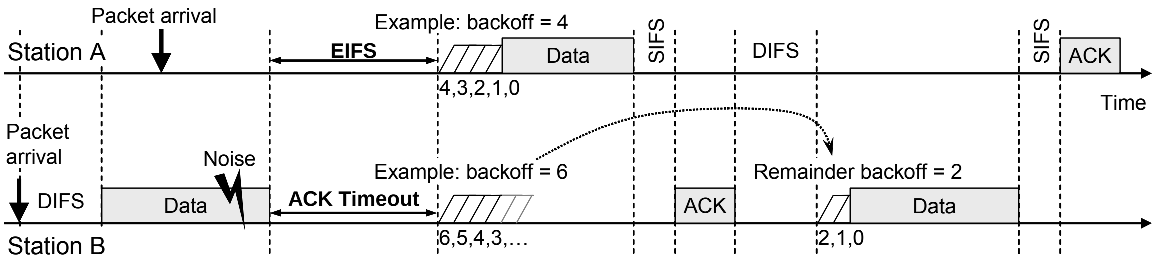


Problem 1.47 — Solution



There is no access point, as this is an independent BSS (IBSS). Remember that 802.11 stations cannot transmit and receive simultaneously, so once a station starts receiving a packet, it cannot contend for transmission until the next transmission round.

The solution is shown in the figure below (check also Figure 1-82 in Example 1.6). We make arbitrary assumption that packet arrives first at *B* and likewise that after the first transmission *A* claims the channel before *B*. Note that the durations of EIFS and ACK_Timeout are the same (Table 1-6). Station *B* selects the backoff=6 and station *A* transmits the first, while station *B* carries over its frozen counter and resumes the countdown from backoff=2.



Problem 1.48 — Solution

Problem 2.1 — Solution

(a)

Scenario 1: the initial value of `TimeoutInterval` is picked as 3 seconds.

At time 0, the first segment is transmitted and the initial value of `TimeoutInterval` is set as 3 s. The timer will expire before the ACK arrives, so the segment is retransmitted and this time the RTO timer is set to twice the previous size, which is 6 s. The ACK for the initial transmission arrives at 5 s, but the sender cannot distinguish whether this is for the first transmission or for the retransmission. This does not matter, because the sender simply accepts the ACK, doubles the congestion window size (it is in slow start) and sends the second and third segments. The RTO timer is set at the time when the second segment is transmitted and the value is 6 s (unchanged). At 8 s a duplicate ACK will arrive for the first segment and it will be ignored, with no action taken. At 10 s, the ACKs will arrive for the second and third segments, the congestion window doubles and the sender sends the next four segments. `SampleRTT` is measured for both the second and third segments, but we assume that the sender sends the fourth segment immediately upon receiving the ACK for the second. This is the first `SampleRTT` measurement so

$$\texttt{EstimatedRTT} = \texttt{SampleRTT} = 5 \text{ s}$$

$$\texttt{DevRTT} = \texttt{SampleRTT} / 2 = 2.5 \text{ s}$$

and the RTO timer is set to

$$\texttt{TimeoutInterval} = \texttt{EstimatedRTT} + 4 \cdot \texttt{DevRTT} = 15 \text{ s}$$

After the second `SampleRTT` measurement (ACK for the third segment), the sender will have

$$\texttt{EstimatedRTT} = (1\text{–}\alpha) \cdot \texttt{EstimatedRTT} + \alpha \cdot \texttt{SampleRTT}$$

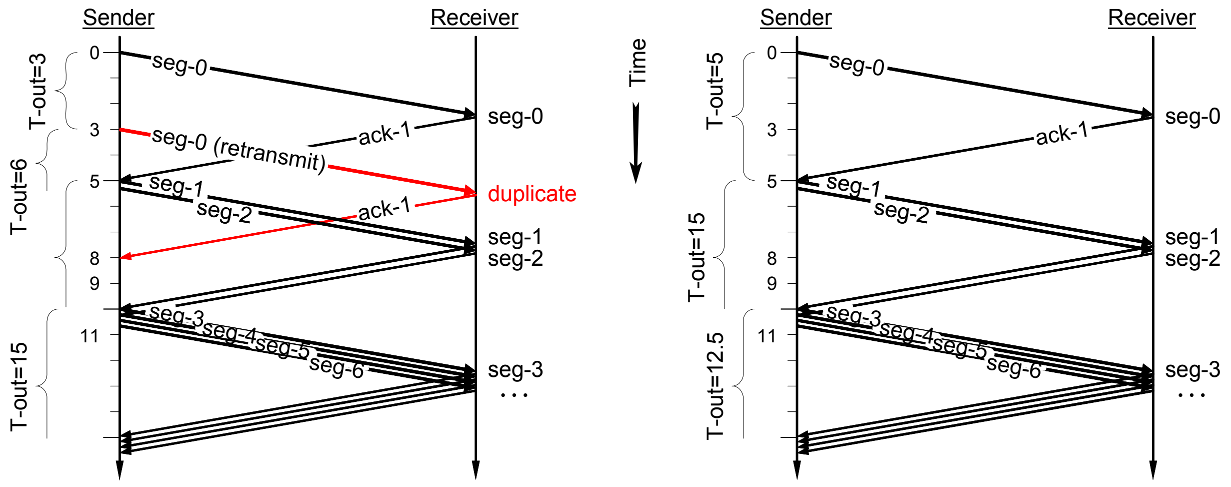
$$= 0.875 \times 5 + 0.125 \times 5 = 5 \text{ s}$$

$$\texttt{DevRTT} = (1-\beta)\cdot\texttt{DevRTT} + \beta\cdot|\,\texttt{SampleRTT} - \texttt{EstimatedRTT}\,|$$

$$= 0.75 \times 2.5 + 0.25 \times 0 = 1.875 \text{ s}$$

but the RTO timer is already set to 15 s and remains so while the fifth, sixth, and seventh segments are transmitted. The following table summarizes the values that `TimeoutInterval` is set to for the segments sent during the first 11 seconds:

Times when the RTO timer is set	RTO timer values
t = 0 s (first segment is transmitted)	`TimeoutInterval` = 3 s (initial guess)
t = 3 s (first segment is retransmitted)	`TimeoutInterval` = 6 s (RTO doubling)
t = 10 s (fourth and subsequent segments)	`TimeoutInterval` = 15 s (estimated value)



Scenario 1: initial T-out = 3 s

Scenario 2: initial T-out = 5 s

(b)

As shown in the above figure, the sender will transmit seven segments during the first 11 seconds and there will be a single (unnecessary) retransmission.

(c)

Scenario 2: the initial value of `TimeoutInterval` is picked as 5 seconds.

This time the sender correctly guessed the actual RTT interval. Therefore, the ACK for the first segment will arrive before the RTO timer expires. This is the first `SampleRTT` measurement and, as above, `EstimatedRTT` = 5 s, `DevRTT` = 2.5 s. When the second segment is transmitted, the RTO timer is set to `TimeoutInterval` = `EstimatedRTT` + 4 · `DevRTT` = 15 s.

After the second `SampleRTT` measurement (ACK for the *second* segment), the sender will have, as above, `EstimatedRTT` = 5 s, `DevRTT` = 1.875 s.

When the fourth segment is transmitted, the RTO timer is set to

$$\texttt{TimeoutInterval} = \texttt{EstimatedRTT} + 4\cdot\texttt{DevRTT} = 12.5 \text{ s}$$

After the third `SampleRTT` measurement (ACK for the *third* segment), the sender will have

$$\texttt{EstimatedRTT} = 0.875 \times 5 + 0.125 \times 5 = 5 \text{ s}$$

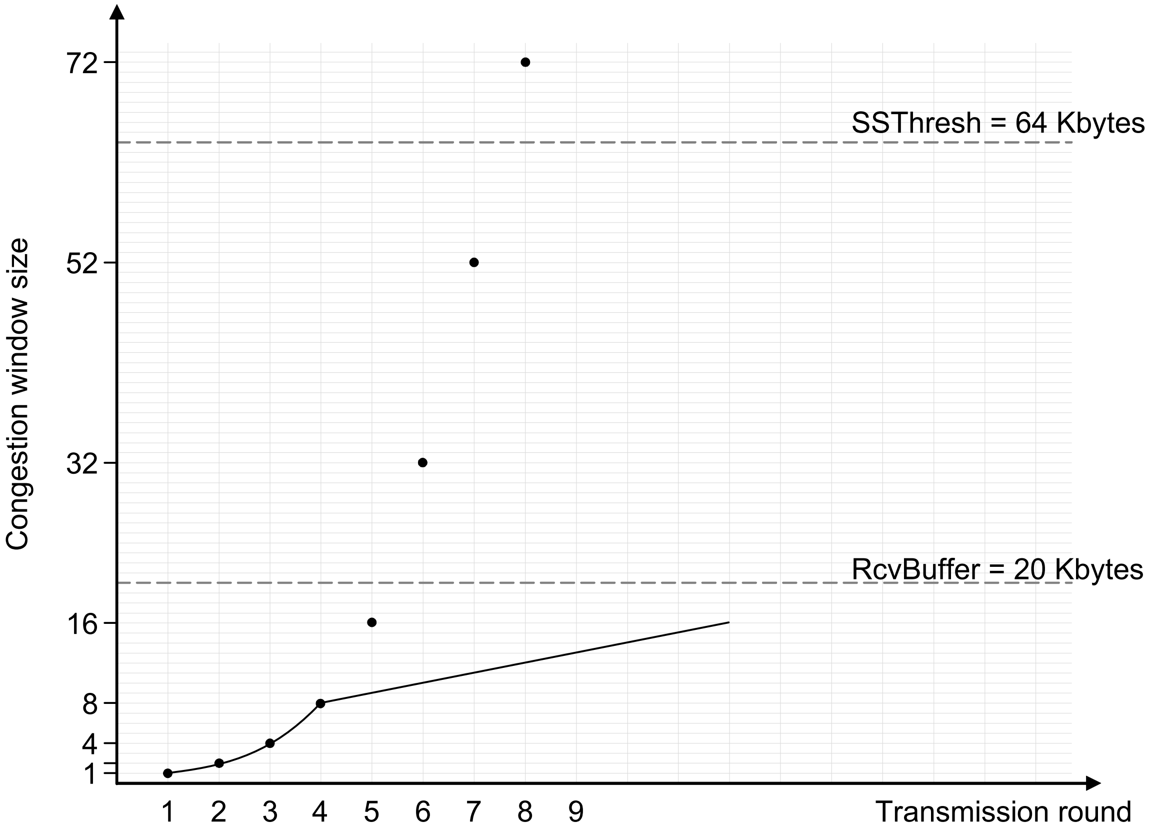
$$\texttt{DevRTT} = 0.75 \times 1.875 + 0.25 \times 0 = 1.40625 \text{ s}$$

but the RTO timer is already set to 12.5 s and remains so while the fifth, sixth, and seventh segments are transmitted. The following table summarizes the values that `TimeoutInterval` is set to for the segments sent during the first 11 seconds:

Times when the RTO timer is set	RTO timer values
t = 0 s (first segment is transmitted)	`TimeoutInterval` = 3 s (initial guess)
t = 5 s (second segment is transmitted)	`TimeoutInterval` = 15 s (estimated value)
t = 10 s (fourth and subsequent segments)	`TimeoutInterval` = 12.5 s (estimated val.)

Problem 2.2 — Solution

The congestion window diagram is shown in the figure below.



First, note that because both hosts are fast and there is no packet loss, the receiver will never buffer the received packets, so sender will always get notified that `RcvWindow` = 20 Kbytes, which is the receive buffer size.

The congestion window at first grows exponentially. However, in transmission round #6 the congestion window of 32 × `MSS` = 32 Kbytes exceeds `RcvWindow` = 20 Kbytes. At this point the sender will send only min{`CongWin`, `RcvWindow`} = 20 segments and when these get acknowledged, the congestion window grows to 52 × `MSS`, instead of 64 × `MSS` under the exponential growth. Thereafter, the sender will keep sending only 20 segments and the congestion window will keep growing by 20 × MSS.

It is very important to observe that the growth is *not exponential* after the congestion window becomes 32 × MSS.

In transmission round #8 the congestion window grows to 72 × MSS, at which point it exceeds the slow start threshold (initially set to 64 Kbytes), and the sender enters the *congestion avoidance* state.

This diagram has the same shape under different network speeds, the only difference being that a transmission round lasts longer, depending on the network speed.

Problem 2.3 — Solution

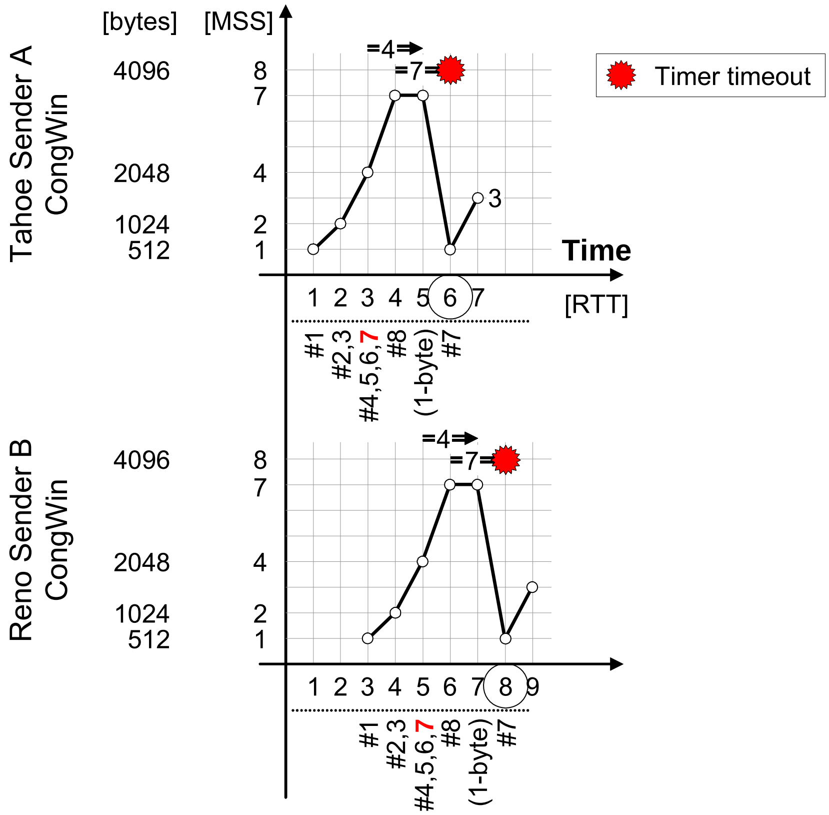
Problem 2.4 — Solution

Recall that a TCP sender keeps a single RTO retransmission timer for all outstanding packets. Every time a regular, non-duplicate ACK is received, the timer is reset if there remain outstanding packets. Thus, although sender *A* sets a timer for segments sent in round 3×RTT, including segment #7, the timer is **reset** at time 4×RTT because packet #7 is unacknowledged. This is why the figure below shows the start of the timer for segment #7 at time 4×RTT, rather than at 3×RTT.

At time 5×RTT, sender *A* has not yet detected the loss of #7 (neither the timer expired, nor three dupACKs were received), so CongWin = 7×MSS (remains constant). There are two segments in flight (segments #7 and #8), so at time = 5×RTT sender *A* could send up to

$$\texttt{EffctWin} = \min\{\texttt{CongWin}, \texttt{RcvWindow}\} - \texttt{FlightSize} = \min\{7, 64\} - 2 = 5\times\texttt{MSS}$$

but it has nothing left to send, *A* sends a 1-byte segment to keep the connection alive. Recall that TCP guarantees *reliable transmission*, so although sender sent all data it cannot close the connection until it receives acknowledgement that all segments successfully reached the receiver. Ditto for sender *B* at time = 7×RTT.

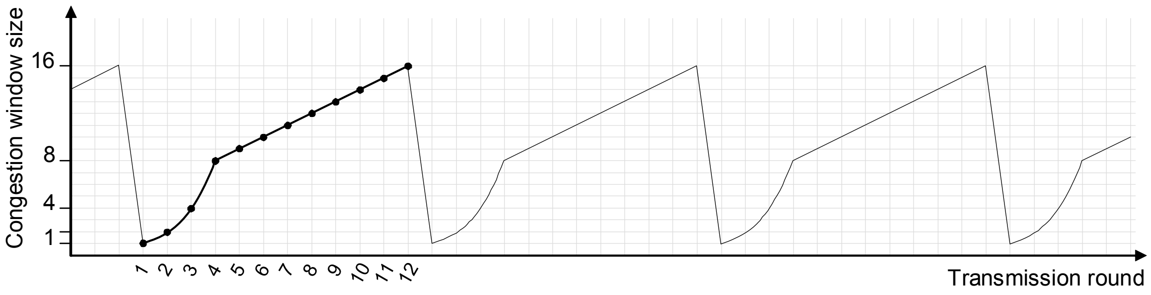


The *A*'s timer times out at time = 6×RTT (before three dupACKs are received), and *A* re-sends segment #7 and enters slow start. At 7×RTT the cumulative ACK for both segments #7 and #8 is received by *A* and it, therefore, increases `CongWin` = 1 + 2 = 3×`MSS`, but there is nothing left to send (recall that the problem statement says that each sender has only 3.6 KB of data to send).

The Reno sender, *B*, behaves in the same way as the Tahoe sender because the loss is detected by the expired RTO timer. Both types of senders enter slow start after timer expiration. (Recall that the difference between the two types of senders is only in Reno implementing *fast recovery*, which takes place after three dupACKs are received.)

Problem 2.5 — Solution

The range of congestion window sizes is [1, 16]. Because the loss is detected when `CongWindow` = 16×`MSS`, `SSThresh` is set to 8×`MSS`. Thus, the congestion window sizes in consecutive transmission rounds are: 1, 2, 4, 8, 9, 10, 11, 12, 13, 14, 15, and 16 `MSS` (see the figure below). This averages to 9.58×`MSS` per second (recall, a transmission round is RTT = 1 sec), and a mean utilization of $\frac{9.58 \times 8}{128}$ [Kbps/Kbps] = 0.59875, or about 60%.



Problem 2.6 — Solution

The solution of Problem2.5 is an idealization that cannot occur in reality. A better approximation is as follows. The event sequence develops as follows:

packet ***loss happens at a router*** (last transmitted segment), current `CongWin` = 16×`MSS`.

the sender receives 16-1=15 ACKs which is not enough to grow `CongWin` to 17

but it still sends 16 new segments, last one will be lost

the sender receives 15 dupACKs, ***loss detected at the sender***

retransmit the oldest outstanding packet, `CongWin` ← 1

the sender receives cumulative ACK for 16 recent segments, except for the last one

`CongWin` ← 2, `FlightSize` = 1×`MSS`, send one new segment

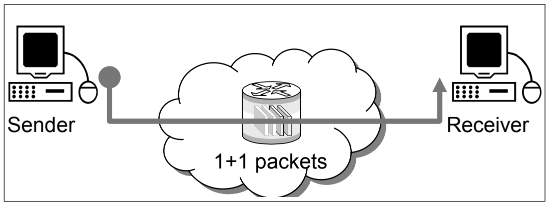
the sender receives 2 dupACKs, `FlightSize` = 3×`MSS`, `EffctWin` = 0, sends one 1-byte segment

the sender receives 3rd dupACK, retransmits the oldest outstanding packet, CW ← 1

the sender receives cumulative ACK for 4 recent segments (one of them was 1-byte), `FlightSize` ← 0

`CongWin` ← 2, the sender resumes slow start

Problem 2.7 — Solution

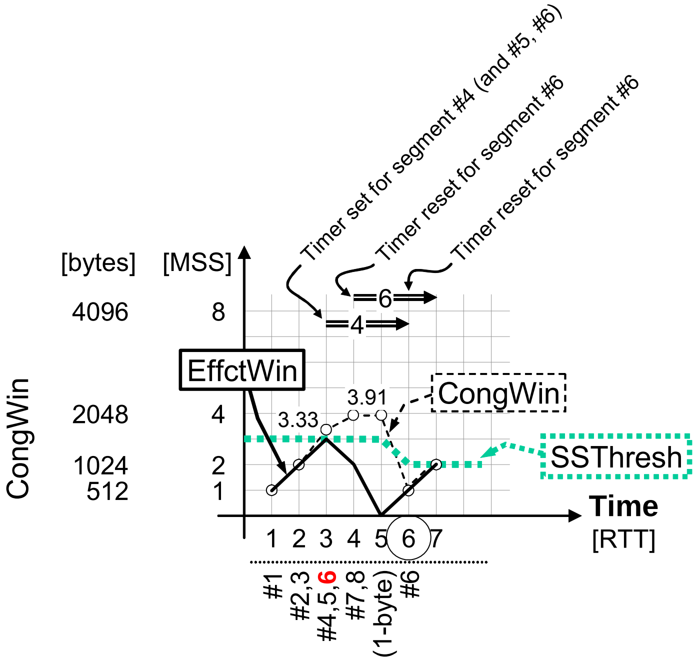


`MSS` = 512 bytes

`SSThresh` = 3×`MSS`

`RcvBuffer` = 2 KB = 4×`MSS`

`TimeoutInterval` = 3×RTT



At time = 3×RTT, after receiving the acknowledgement for the 2nd segment, the sender's congestion window size reaches `SSThresh` and the sender enters additive increase mode. Therefore, the ACK for the 3rd segment is worth $\text{MSS} \times \frac{\text{MSS}}{\text{CongWindow}_{t\text{-}1}} = 1 \times \frac{1}{3} =$ 0.33 MSS. `CongWindow`$_{3\times\text{RTT}}$ = 3.33×`MSS`. Therefore, the sender sends 3 segments: 4th, 5th, and 6th. The router receiver 3 segments but can hold only 1+1=2. It will start transmitting the 4th segment, store the 5th segment in its buffer, and discard the 6th segment due to the lack of buffer space (there is a waiting room for one packet only). The acknowledgements for the 4th and 5th segments add $1 \times \frac{1}{3.33} = 0.3 \times$`MSS` and $1 \times \frac{1}{3.63} = 0.28 \times$`MSS`, respectively, so `CongWindow`$_{4\times\text{RTT}}$ = 3.91×`MSS`. The effective window is smaller by one segment because the 6th segment is outstanding:

$$\texttt{EffctWin} = \lfloor \min\{\texttt{CongWin}, \texttt{RcvWindow}\} - \texttt{FlightSize} \rfloor = \lfloor \min\{3.91, 4\} - 1 \rfloor = 2\times\text{MSS}$$

At time = 4×RTT the sender sends two segments, both of which successfully reach the receiver. Acknowledgements for 7th and 8th segments are duplicate ACKs, but this makes only two duplicates so far, so the loss of #6 is still not detected. Note that at this time the receive buffer stores two segments (`RcvBuffer` = 2 KB = 4 segments), so the receiver starts advertising `RcvWindow` = 1 Kbytes = 2 segments.

The sender computes

$$\texttt{EffctWin} = \lfloor \min\{\texttt{CongWin}, \texttt{RcvWindow}\} - \texttt{FlightSize} \rfloor = \lfloor \min\{3.91, 2\} - 3 \rfloor = 0$$

so it sends a 1-byte segment at time = 5×RTT.

At time = 6×RTT, the loss of the 6th segment is detected *via three duplicate ACKs*. Recall that the sender in fast retransmit does not use the above formula to determine the current `EffctWin`—it simply retransmits the segment that is suspected lost. That is why the above figure shows

EffctWin = 2 at time = 6×RTT. The last EffctWin, at time = 7×RTT, equals 2×MSS but there are no more data left to send.

Therefore, the answers are:

(a)

The first loss (segment #6 is lost in the router) happens at 3×RTT, so

$\text{CongWindow}_{3\times\text{RTT}} = 3.33\times\text{MSS}$.

(b)

The loss of the 6th segment is detected via three duplicate ACKs at time = 6×RTT.

At this time, not-yet-acknowledged segments are: 6th, 7th, and 8th, a total of three.

Problem 2.8 — Solution

In solving this problem, we should keep in mind that the receive buffer size is set relatively small to 2Kbytes = 8×MSS. The sender is a regular TCP Reno.

In the transmission round *i*, the sender sent segments *k*, *k*+1, …, *k*+7, of which the segment *k*+3 is lost. The receiver receives the four segments *k*+4, …, *k*+7 out of order, and buffers them and sends back four duplicate acknowledgements. In addition, the receiver notifies the sender that the new RcvWindow = 1 Kbytes = 4×MSS.

At *i*+1, the sender first receives three regular (non-duplicate!) acknowledgements for the first three successfully transferred segments, so the new CongWin = 11×MSS. Then, four duplicate acknowledgements will arrive while FlightSize = 5. After receiving the first three dupACKs, Reno sender reduces the congestion window size according to Eq. (2.7):

$\text{CongWin} = \max\{½\,\text{FlightSize}, 2\times\text{MSS}\} + 3\times\text{MSS} = 5/2 + 3 = 5.5\times\text{MSS}$.

The new value of $\text{SSThresh} = \max\{½\,\text{FlightSize}, 2\times\text{MSS}\} = 2.5\times\text{MSS}$, by Eq. (2.6).

Because Reno sender enters *fast recovery*, each dupACK received after the first three increments the congestion window by one MSS. Therefore, CongWin = **6.5**×MSS. The effective window is:

$$\begin{aligned}\text{EffctWin} &= \lfloor \min\{\text{CongWin}, \text{RcvWindow}\} - \text{FlightSize} \rfloor \\ &= \lfloor \min\{6.5, 4\} - 5 \rfloor = -1\end{aligned} \qquad (\#)$$

Thus, the sender is allowed to send no more than the oldest unacknowledged segment, *k*+3, which is suspected lost.

There is an interesting observation to make here, as follows. Knowing that the receive buffer holds the four out-of-order segments and it has four more slots free, it may seem inappropriate to use the formula (#) above to determine the effective window size. After all, there are four *free* slots in the receive buffer, so that should not be a limiting parameter! The sender's current knowledge of the network tells it that the congestion window size is 6×MSS so this should allow sending more!? Read on.

The reason that the formula (#) is correct is that you and I know what receiver holds and where the unaccounted segments currently are residing. But the sender does not know this! It only knows that currently `RcvWindow` = 4×`MSS` and there are five segments *somewhere* in the network. As far as the sender knows, they still may show up at the receiver. So, it must not send anything else.

At *i*+2, the sender receives ACK asking for segment *k*+8, which means that all five outstanding segments are acknowledged at once. Because the congestion window size is still below the `SSThresh`, the sender increases `CongWin` by 5 to obtain `CongWin` = 11×`MSS`. Note that by now the receiver notifies the sender that the new `RcvWindow` = 2 Kbytes = 8×`MSS`, because all the receive buffer space freed up.

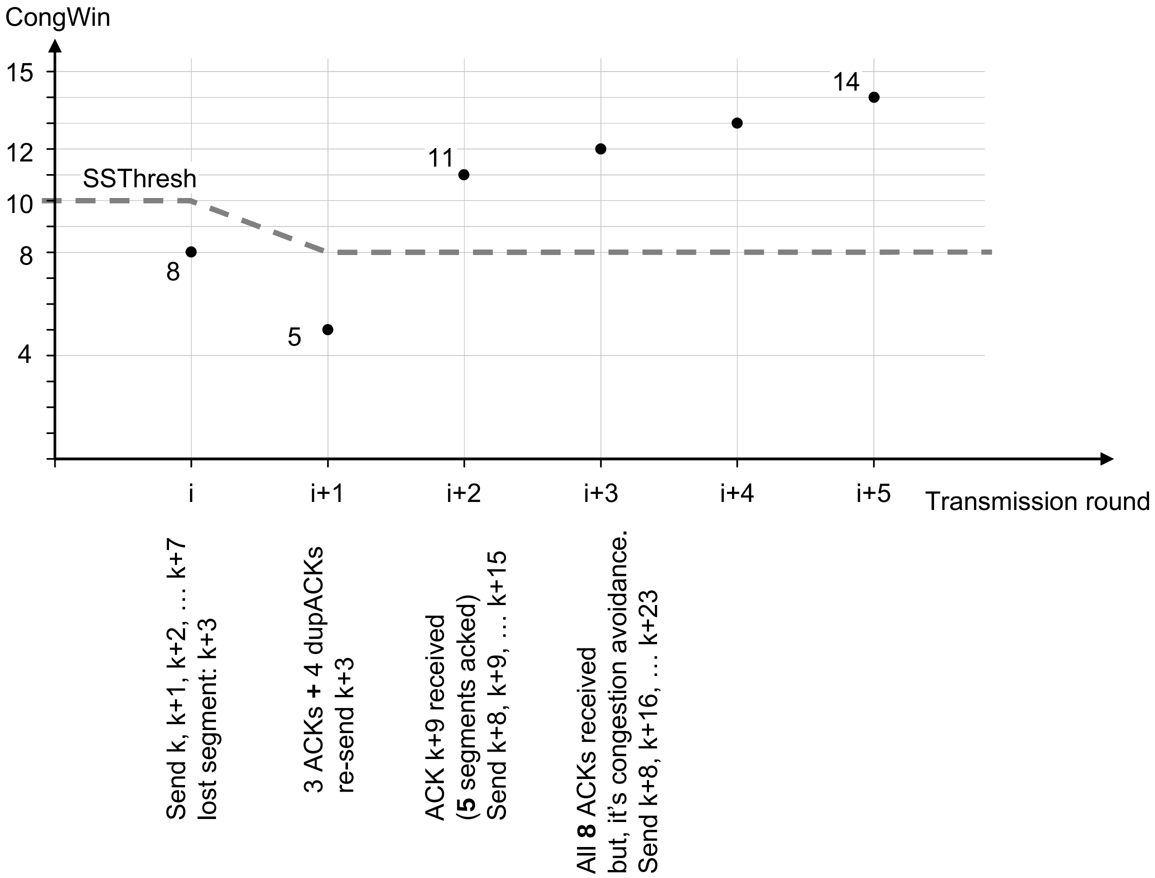
The new effective window is:

$$\texttt{EffctWin} = \lfloor \min\{\texttt{CongWin}, \texttt{RcvWindow}\} - \texttt{FlightSize} \rfloor = \lfloor \min\{11, 8\} - 0 \rfloor = 8 \times \texttt{MSS}$$

so the sender sends the next eight segments, *k*+8, …, *k*+15.

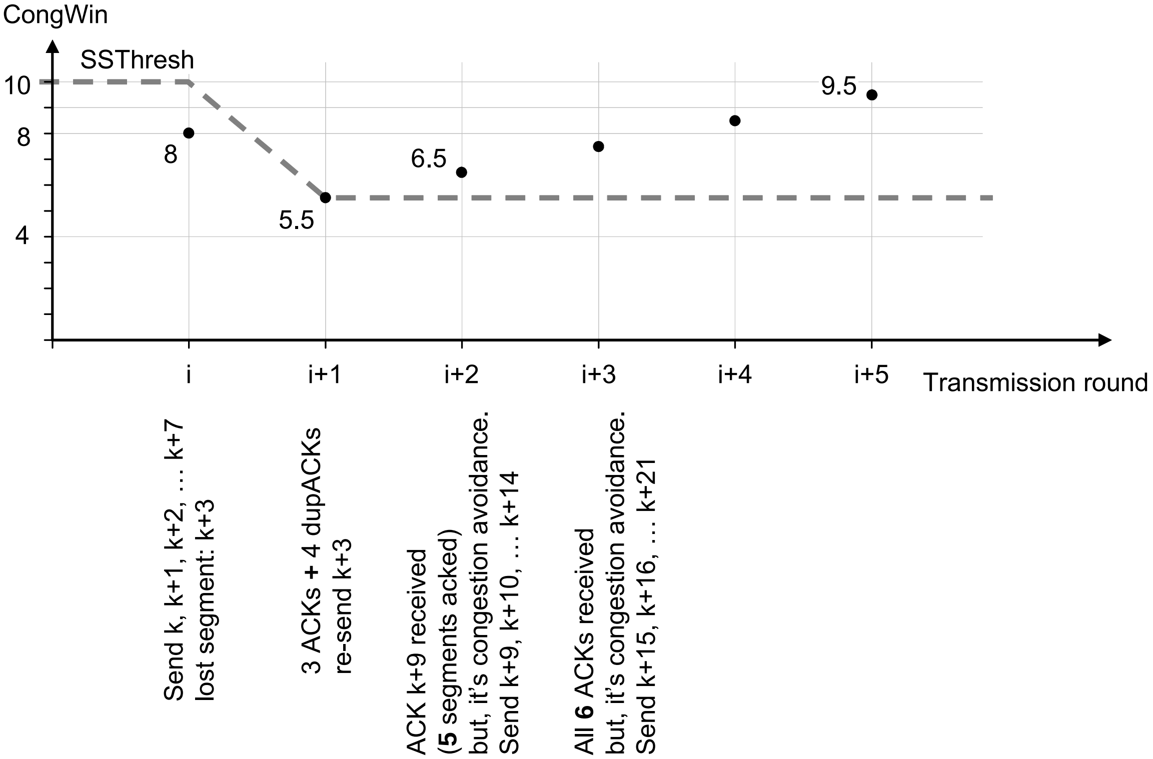
Next time the sender receives ACKs, it's already in *congestion avoidance* state, so it increments `CongWin` by 1 in every transmission round (per one RTT).

Note that, although `CongWin` keeps increasing, the sender will keep sending only **eight** segments per transmission round because of the receive buffer's space limitation.



Some networking books give a simplified formula for computing the slow-start threshold size after a loss is detected as `SSThresh` = `CongWin` / 2 = 5.5×`MSS`. Rounding `CongWin` down to the next integer multiple of `MSS` is often not mentioned and neither is the property of fast recovery to increment `CongWin` by one `MSS` for each dupACK received after the first three that triggered the retransmission of segment *k*+3.

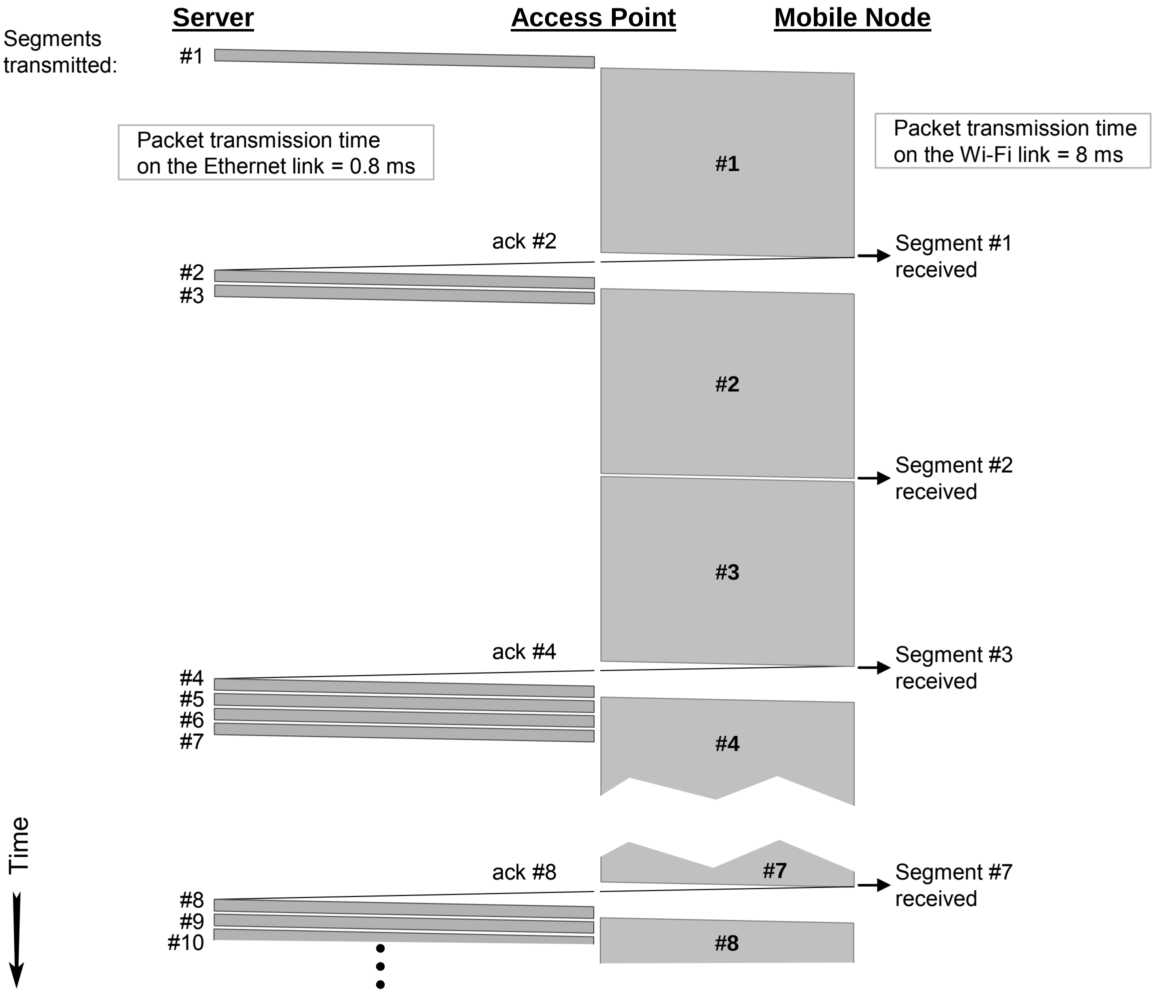
In this case, the sender in our example would immediately enter congestion avoidance, and the corresponding diagram is as shown in the figure below.



Problem 2.9 — Solution

We can ignore the propagation times because they are negligible relative to the packet transmission times (mainly due to short the distance between the transmitter and the receiver). Also, the transmission times for the acknowledgements can be ignored. Because the transmission just started, the sender is in the slow start state. Assuming that the receiver sends only *cumulative acknowledgements*, the total time to transmit the first 15 segments of data is (see the figure):

$$4 \times 0.8 + 15 \times 8 = 123.2 \text{ ms.}$$

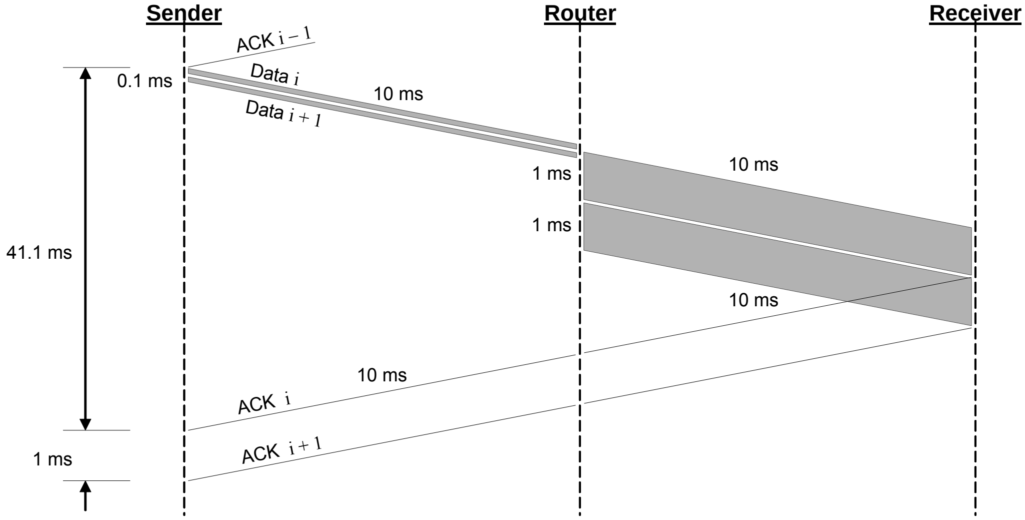


The timing diagram is as shown in the figure.

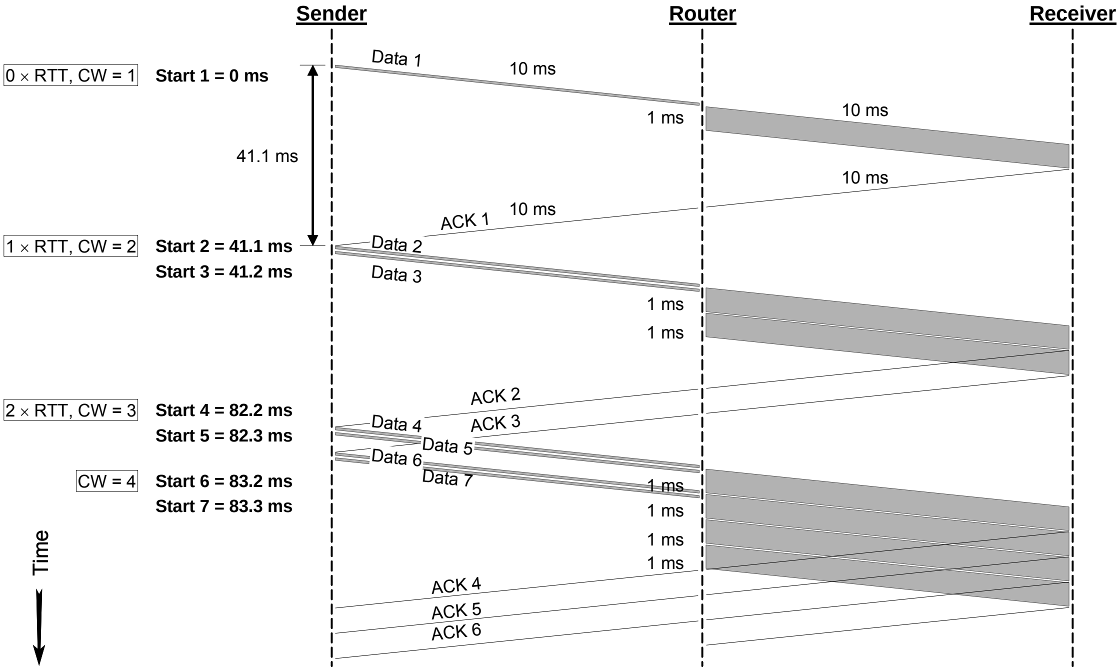
Problem 2.10 — Solution

(a)

During the slow start phase, an incoming acknowledgment will allow the sender to send two segments. The acknowledgements to these two segments will be separated by 1 ms, as shown in the figure below.



(b)



The start times for the transmissions of the first seven segments are as shown in the figure:

(c)

We can make the following observations:

o1. Except for the first round, packets always arrive in pairs at the router. The reason for this was explained under item (a); that is, during the slow start phase, each acknowledgement will allow the sender to send two segments back-to-back.

o2. The time gap between the packet pairs is 1 ms, because the time gap between the acknowledgements (during slow start) is 1 ms. This gives enough time to the router to transmit one packet before the arrival of the next pair.

o3. We can think conceptually that from each pair, one packet is relayed by the router and the other remains in the router's buffer. (This is *not* true, because the router transmits packets in the order of their arrival (FIFO), so it will first transmit any packets remaining from previous pairs, but we can think this way conceptually.)

o4. Because router can hold 9+1=10 packets, the first loss will happen when ≥ 20 packets is sent on one round. In 5th round, time = 5×RTT, the `CongWin` = 32, so this is when the first loss will happen

o5. The packets sent in this round will find the following number of packets already at the router (packet pairs are separated by ||):

0, 1 || 1, 2 || 2, 3 || 3, 4 || 4, 5 || 5, 6 || 6, 7 || 7, 8 || 8, 9 || 9, 10 || 9, 10 || 9, 10 || 9, 10 || 9, 10 || 9, 10 || 9, 10 .

o6. Therefore, the 20th packet of this round will find 10 packets already at the router and this packet will be lost. This is the 41st packet from the start of sending at time = 0.

o7. By the time the next pair arrives, the router will have transmitted one packet, so 21st packet finds 9 packets already at the router, but its companion in the pair, 22nd packet finds 10 packets and is lost

o8. This pattern repeats until the last, which is 32nd packet of this round.

o9. A total of 7 packets will be lost, starting with 20th, 22nd, 24th, …, and 32nd.

o10. At time, 6×RTT, the congestion window will grow up to 32 + 19 = 51

o11. After this ≥ 3 × dupACKs will be received and the sender will go into the multiplicative decrease phase

Therefore, the congestion window sizes for the first 6×RTT are: 1, 2, 4, 8, 16, 31, 51.

(d)

As shown in (c), the first packet will be lost in the 5th round, and it is the 20th packet of this round. Its ordinal number is #51, determined as follows:
1 + 2 + 4 + 8 + 16 = 31 (from previous four rounds) + 20 (from the 5th round) = 51

(e)

…

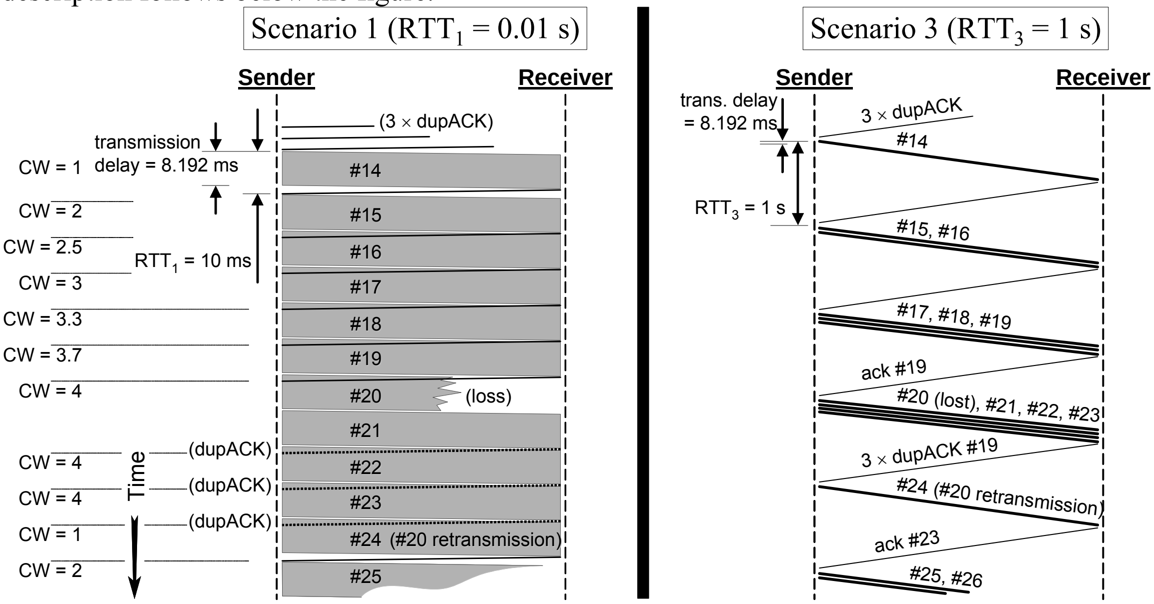
Problem 2.11 — Solution

Transmission delay for all three scenarios is:

$$t_x = \frac{\text{packet length}}{\text{bandwidth}} = \frac{8192 \text{ bits}}{1000000 \text{ bits per second}} = 8.192 \text{ ms}$$

In the first scenario (RTT_1 = 0.01 sec), the round-trip time is about the same as transmission delay. The sender can send up to one segment and start with a second one before it receives the acknowledgement for the first segment. Conversely, in the third scenario (RTT_3 = 1 sec), the sender can send a burst of up to 122 segments before receiving the acknowledgement for the first segment of this burst.

We ignore the initial slow-start phase and consider that the sender settled into a congestion-avoidance state. Because the network exhibits periodic behavior where every tenth packet is lost, we need to determine the sender's cyclic behavior. The figure illustrates the solution and the description follows below the figure.



To detect a lost 10th segment, the sender must have sent at least three more to receive three duplicate acknowledgements. Let us assume the sender detects the loss (via 3 dupACKs) after it has sent 13 segments. The 14th transmission will be retransmission of the 10th segment (the lost one). The receiver fills the gap and cumulatively acknowledges up to the segment #13, requesting the sender to send the next segment (#14). This table shows how the first scenario sender sends segments and receives acknowledgements:

packet number	10th	11th	12th	13th	14th	15th	16th	17th	18th	19th	20th	21st	22nd	23rd	24th	25th
segment seq. num.	10 (×)	11	12	13	10	14	15	16	17	18	19 (×)	20	21	22	19	23
ACK	_□	**10**	**10**	**10**	14	15	16	17	18	19	_□	**19**	**19**	**19**	23	24
CongWin				**1**	2	2.5	3	3.3	3.7	3.7		**3.7**	**3.7**	**1**	2	2.5

This cycle will repeat. The sender is sending segments back-to-back. The only overhead it experiences is that every tenth packet is lost, which implies that during 10 transmission delays it successfully sends 9 segments. Hence, the average data rate the sender achieves is: (9/10) × 1 Mbps = 900 Kbps.

Let us now consider the third scenario with RTT$_3$ = 1 sec. Because the round-trip time is much longer than transmission time and every 10th packet is lost, we can consider that burst transmissions are clocked to RTT intervals.

RTT		n–1			n	n+1		n+2			n+3				n+4
packet number	10th	11th	12th	13th	14th	15th	16th	17th	18th	19th	20th	21st	22nd	23rd	24th
segment seq. num.	10 (×)	11	12	13	10	14	15	16	17	18	19 (×)	20	21	22	19
ACK	_□	**10**	**10**	**10**	14	15	16	17	18	19	_□	**19**	**19**	**19**	23
CongWin				1	2		2.5, 3			3.3, 3.7, 4				1	2

In this scenario, the cycle lasts 4×RTT and the sender successfully sends 9 segments of which 6 are acknowledged and 3 are buffered at the receiver and will be acknowledged in the next cycle. Hence, the average data rate the sender achieves is:

$$\frac{9 \times t_x}{4 \times \mathrm{RTT}} \times 1\,\mathrm{Mbps} = \frac{0.073728}{4} \times 1000000 = 18432\ \mathrm{bps} \approx 18.4\ \mathrm{Kbps}$$

Obviously, the sender in the third scenario achieves much lower rate than the one in the first scenario. The reason for this is that the first-scenario sender receives feedback quickly and reacts quickly. Conversely, the third-scenario sender receives feedback very slowly and accordingly reacts slowly—it is simply unable to reach the potentially achievable rate.

Problem 2.12 — Solution

Object size,	O = 1 MB = 2^{20} bytes = 1048576 bytes = 8388608 bits
Segment size `MSS` = 1 KB,	S = `MSS` × 8 bits = 8192 bits
Round-trip time	RTT = 100 ms
Transmission rate,	R = as given for each individual case (see below)

There are a total of $L = \frac{1\,MB}{1\,KB} = \frac{2^{20}}{2^{10}} = 2^{10} = 1024$ segments (packets) to transmit.

(a)

Bottleneck bandwidth, R = 1.5 Mbps, data sent continuously:

$$\frac{O}{R} = \frac{2^{20} \times 8}{1.5 \times 10^{6}} = \frac{1048576 \times 8}{1500000} = \frac{8388608}{1500000} = 5.59\ \text{sec}$$

latency = 2 × RTT + O / R = 200 × 10^{-3} + 5.59 sec = 5.79 sec

(b)

R = 1.5 Mbps, Stop-and-wait

latency $= 2 \times \mathrm{RTT} + \left(\frac{S}{R} + \mathrm{RTT}\right) \times L = 0.2 + \left(\frac{8192}{1500000} + 0.1\right) \times 1024 = 108.19$ sec

(c)

R = ∞, Go-back-20

Because transmission time is assumed equal to zero, all 20 packets will be transmitted instantaneously and then the sender waits for the ACK for all twenty. Thus, data will be sent in chunks of 20 packets:

latency $= 2 \times \mathrm{RTT} + \mathrm{RTT} \times \frac{L}{20} = 0.2 + 5.12 = 5.22$ sec

(d)

$R = \infty$, TCP Tahoe

Because the transmission is error-free and the bandwidth is infinite, there will be no loss, so the congestion window will grow exponentially (*slow start*) until it reaches the slow start threshold `SSThresh` = 65535 bytes = 64 × `MSS`, which is the default value. From there on it will grow linearly (*additive increase*). Therefore, the sequence of congestion window sizes is as follows:

Congestion window sizes:

$$\underbrace{1,\ 2,\ 4,\ 8,\ 16,\ 32,\ 64,}_{\substack{\text{Slow start} \\ \text{-- total 7 bursts}}}\ \ \underbrace{65,\ 66,\ 67,\ 68,\ 69,\ 70,\ \ldots}_{\substack{\text{Additive increase} \\ \text{-- total 13 bursts}}}$$

Then, slow start phase consists of 7 bursts, which will transfer the first 127 packets. The additive increase for the remaining 1024 − 127 = 897 packets consists of at most 897/64 ≈ 14 bursts. Quick calculation gives the following answer:

Assume there will be thirteen bursts during the additive increase.

With a constant window of 64 × `MSS`, this gives 13 × 64 = 832.

On the other hand, additive increase adds 1 for each burst, so starting from 1 this gives $1+2+3+ \ldots + 13 = \frac{13 \times (13+1)}{2} = 91$ packets.

Therefore, starting with the congestion window size of 64 × `MSS`, the sender can in 13 bursts send up to a total of 832 + 91 = 923 packets, which is more than 832.

Finally, sender needs total of 7 + 13 = 20 bursts:

latency = 2 × RTT + 20 × RTT = 2.2 sec.

Problem 2.13 — Solution

It is easier to solve this problem if we represent the relevant parameters in a tabular form as in the table below (see also the figure below).

First, we know that this is a TCP Reno sender, currently in the slow start phase, with `CongWin`(t_i) = 400 bytes. Given that `MSS` = 200 bytes, the sender sends a “burst” of two segments back-to-back. According to Listing 2-1 (summarized on pages 168-169), when sending a segment, the sender needs to set the RTO timer if it is not already running. The RTO timer is not running at t_i because all previous segments were successfully acknowledged. So, using equation (2.1), the sender obtains:

`TimeoutInterval`(t_i) = `EstimatedRTT`(t_i) + 4 · `DevRTT`(t_i) = 100.8 + 4×9 = 136.8 ms

where `EstimatedRTT`(t_i) and `DevRTT`(t_i) are determined as follows (using default values for α = 0.125 and β = 0.25):

EstimatedRTT(t_i) = $(1-\alpha)$ · EstimatedRTT(t_{i-1}) + α · SampleRTT(t_i) = $0.875\times100 + 0.125\times106 = 100.8$ ms

DevRTT(t_i) = $(1-\beta)$ · DevRTT(t_{i-1}) + β · |SampleRTT(t_i) − EstimatedRTT(t_{i-1})| = $0.75\times10 + 0.25\times|106 - 100| = 9$ ms

The sender does not modify the RTO timer when sending the second segment, because the timer is already running.

Note that the top row of the table shows the *times when each segment is* ***transmitted***. Although the problem statement does not require determining these, we can easily determine them as follows. We assume for simplicity that $t_i = 0$ ms. Given that $t_x \ll t_p$ and the propagation times are in the range from 50 ms to 100 ms, we can assume for simplicity that the segment transmission time equals $t_x = 1$ ms. However, assuming any other small number for t_x would not make a difference for the problem solution.

The second segment is sent right after the first one, so $t_{i+1} = t_i + 1 = 1$ ms. Note that at t_{i+1} there is no acknowledgement that arrived, so the sender does not calculate EstimatedRTT(t_{i+1}) and DevRTT(t_{i+1}) and these entries are left empty in the table.

Time t [ms]	t_i (=0 ms)	t_{i+1} (=1 ms)	t_{i+2} (=105)	t_{i+3} (=106)	t_{i+4} (=107)	t_{i+5} (=108)	t_{i+6} (=237.5)	t_{i+7} (=288)	$t_{i+7}+t_x$ (=289)
CongWin(t)	2×MSS		4×MSS				1×MSS	3×MSS	
SSThresh(t)	64 KBytes						2×MSS		
RTT(t)	105	93	179	182	165	193	154	171	
SampRTT(t)	106		105						
EstimRTT(t)	100.8		101.3						
DevRTT(t)	9		7.8						
TimeoutInt	136.8 ms		132.5				265		
Transmiss'n type	Burst #1		Burst #2				Re-transmit	Burst #3	

The sender then waits idle until an acknowledgement arrives for the two segments that it sent.

The ACK for the first segment (shown in the figure as ACK #3) will arrive at the time t_i + RTT(t_i) = 0 + 105 = 105 ms.

The ACK for the second segment (shown in the figure as ACK #1 - duplicate) will arrive at time t_{i+1} + RTT(t_{i+1}) = 1 + 93 = 94 ms. This is earlier than the ACK for the first segment by 11 ms and we assume that the second segment also arrived before the first one to the receiver. Because segment #2 is out of order, the receiver will immediately send a duplicate ACK, asking again for segment #1. The sender will do nothing when it receives the duplicate ACK. (Recall that the TCP sender retransmits only when it receives ≥3×dupACKs, and it does not sample RTT for duplicate ACKs.)

When the first segment arrives, the receiver will acknowledge both outstanding segments by sending a *cumulative* ACK #3. When the sender receives ACK #3 at 105 ms, it will increment its congestion window by 2×MSS and send four more segments at times t_{i+2}, $t_{i+3}=t_{i+2}+t_x$, $t_{i+4}=t_{i+3}+t_x$, and $t_{i+5}=t_{i+4}+t_x$.

At the time of arrival of ACK #3, the sender measures the sample RTT. This time corresponds to t_{i+2} and SampleRTT(t_{i+2}) = RTT(t_i) = 105 ms. The sender calculates EstimatedRTT(t_{i+2}) and DevRTT(t_{i+2}) as shown in the table.

Because ACK #3 acknowledged all the outstanding segments and the RTO timer was reset, the sender will calculate the new value TimeoutInterval(t_{i+2}) = 132.5 ms and set the RTO timer when it sends the third segment at t_{i+2} = 105 ms. Three more segments will be sent in this burst at times t_{i+3}=106 ms, t_{i+4}=107 ms, and t_{i+5}=108 ms.

Given that RTT(t_{i+4})=165 ms is the shortest for all the segments from the second burst, we will again assume that segment #5 arrived out of order and the receiver will immediately send a duplicate ACK (shown in the figure as ACK #3 - duplicate). The acknowledgements for the four segments of the second burst will arrive at these times (see also the figure):

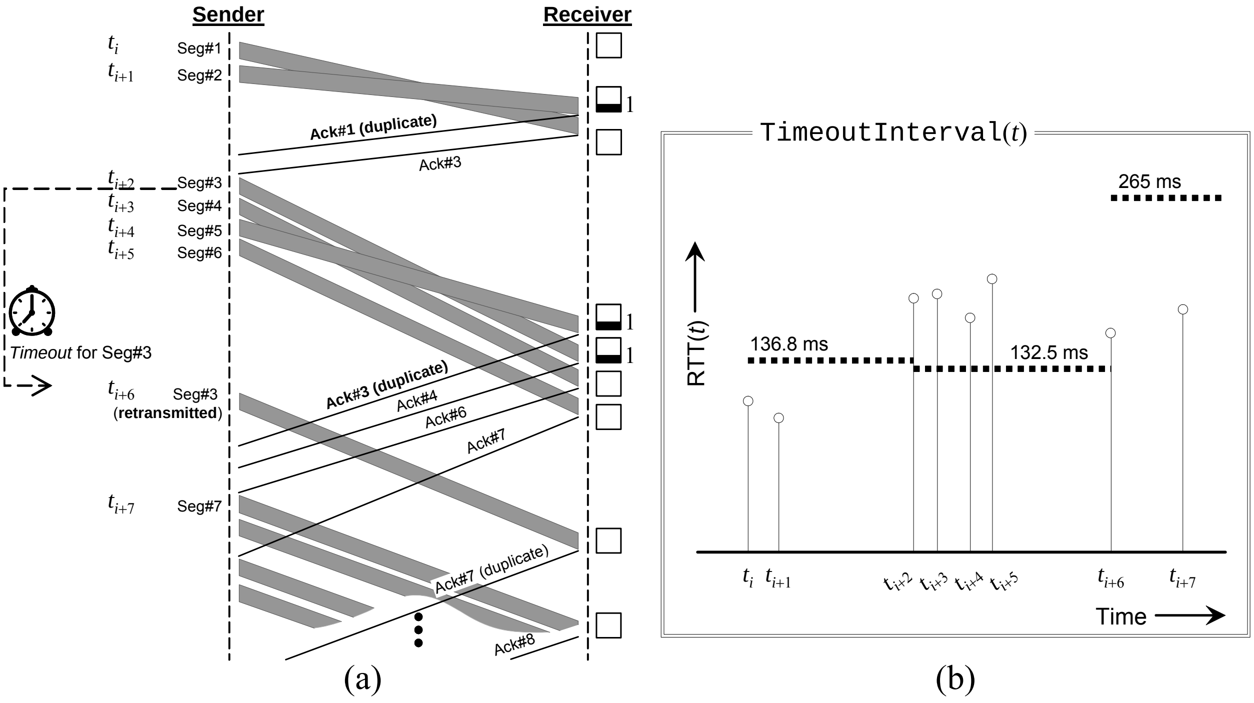
For segment #5 (ACK #3 - duplicate) arrives at: t_{i+4} + RTT(t_{i+4}) = 107 + 165 = 272 ms.

For segment #3 (ACK #4) arrives at: t_{i+2} + RTT(t_{i+2}) = 105 + 179 = 284 ms.

For segments #4 and #5 (ACK #6) arrives at: t_{i+3} + RTT(t_{i+3}) = 106 + 182 = 288 ms.

For segment #6 (ACK #7) arrives at: t_{i+5} + RTT(t_{i+5}) = 108 + 193 = 301 ms.

All of the above times are too late for the RTO timer, which will expire at time t_{i+6} = t_{i+2} + TimeoutInterval(t_{i+2}) = 105 + 132.5 = 237.5 ms. Note that both TCP Tahoe and TCP Reno behave the same on RTO timer expiration. They both enter slow start, so there will be no difference in behavior and CongWin(t_{i+6}) = 1×MSS = 200 bytes. Also, according to equation (2.1), the timeout interval is set to twice the previous value, so TimeoutInterval(t_{i+6}) = 2 × TimeoutInterval(t_{i+2}) = 265 ms.



Because of the expired RTO timer, the sender will retransmit the oldest outstanding segment at time t_{i+6} = 237.5 ms. This segment is a copy of the third segment, but we count it as the seventh transmission. Given that RTT(t_{i+6}) = 154 ms, the corresponding acknowledgement will arrive at the time t_{i+6} + RTT(t_{i+6}) = 237.5 + 154 = 391.5 ms.

Meanwhile, the duplicate ACK #3 will arrive and the sender will ignore it. Next, the ACK for segment #3 (shown in the figure as ACK #4) will arrive at time equal 284 ms. Note that this is the ACK for the original segment #3, not the retransmitted one. Although at this time segment #5 is already at the receiver, it is still out of order: there is a gap because segment #4 still did not arrive, so the receiver is asking for #4. The sender thinks that the receiver is acknowledging the retransmitted segment, so it increments CongWin to 2×MSS. There are still three outstanding segments (#4, #5 and #6), so EffectiveWindow = CongWindow – FlightSize = 0 and the sender remains quiet. Note also that now CongWindow reached SSThresh, so the sender enters *congestion avoidance*.

At the time equal 288 ms, ACK #6 arrives acknowledging segments #4 and #5. Because the sender is in congestion avoidance and CongWin = 2×MSS, the two-segment ACK will count as one by equation (2.5), so CongWin becomes 3×MSS. There is one more segment unaccounted (#6), so FlightSize = 1 and EffectiveWindow = 2 and the sender transmits two segments at times t_{i+7} and t_{i+7} + t_x.

(a)
The table below shows the congestion window sizes CongWin(t) and the sequence numbers of the segments transmitted from the sender. (Recall that the sender's sequence number for the segment that will be transmitted at t_i equals 30 and MSS = 200 bytes.)

Time t [ms]	t_i (=0 ms)	t_{i+1} (=1 ms)	t_{i+2} (=105)	t_{i+3} (=106)	t_{i+4} (=107)	t_{i+5} (=108)	t_{i+6} (=237.5)	t_{i+7} (=288)	t_{i+7}+t_x (=289)
CongWin(t)	2×MSS		4×MSS				1×MSS	3×MSS	
SeqNum(t)	30	230	430	630	830	1030	**430**	1230	1430

(b)
To determine the times when the ACKs will arrive, we simply add the corresponding RTT to the time the segment departed, as was already done above for most of the segments. The table below shows the times of ACK arrivals, the sequence numbers of the ACKs, and the values of RcvWindow as carried in each ACK segment (recall that the receiver's buffer size RcvWindow(t_i) = 1000 bytes and also see how the figure above indicates the receive buffer occupancy).

Time t [ms]	t_i (=0 ms)	t_{i+1} (=1 ms)	t_{i+2} (=105)	t_{i+3} (=106)	t_{i+4} (=107)	t_{i+5} (=108)	t_{i+6} (=237.5)	t_{i+7} (=288)	t_{i+7}+t_x (=289)
ACK arrives	105 ms	94 ms	284	288	278	301	391.5	459	
ACKSeqNo	430	30/**dup**	630	1030	430/**dup**	1230	1230/**dup**	1430	
RcvWindow	1000	800	800	1000	800	1000	1000	1000	

(c)
The values of EstimatedRTT(t) and DevRTT(t) are shown in the first table above. Recall that the TCP retransmission-timer management algorithm measures SampleRTT for segments that have been transmitted *once* and *not* for segments that have been retransmitted. It also ignores dupACKs. An extract from the first table shows that EstimatedRTT(t) and DevRTT(t) are calculated only at t_i and t_{i+2}:

Time t [ms]	t_i (=0 ms)	t_{i+1} (=1 ms)	t_{i+2} (=105)	t_{i+3} (=106)	t_{i+4} (=107)	t_{i+5} (=108)	t_{i+6} (=237.5)	t_{i+7} (=288)	$t_{i+7}+t_x$ (=289)
RTT(t)	105	93	179	182	165	193	154	171	
SampRTT(t)	106		105						
EstimRTT(t)	100.8		101.3						
DevRTT(t)	9		7.8						

(d)

The TCP sender will set its retransmission timer three times during the considered interval, and the values of `TimeoutInterval`(t) are as follows:

Time t [ms]	t_i (=0 ms)	t_{i+1} (=1 ms)	t_{i+2} (=105)	t_{i+3} (=106)	t_{i+4} (=107)	t_{i+5} (=108)	t_{i+6} (=237.5)	t_{i+7} (=288)	$t_{i+7}+t_x$ (=289)
TimeoutInt	136.8 ms		132.5				265		

Problem 2.14 — Solution

Problem 2.15 — Solution

During the specified period of time, the sender will receive only 5 acknowledgements for new data. This will happen at times: at 89 for segment #33 (which was transmitted earlier, at time t = 82); at 96 for the retransmitted segment #35; at 104 for segment #37; at 113 for segment #39; and at 120 for segment #41. Check the pseudocode at the end of Section 2.1.3 to see that all other ACKs are ignored because they are acknowledging already acknowledged data (they are called "duplicate ACKs").

By examining Figure 2-26, we can see that the retransmitted segments #33, #35, and #37 arrive at an idle router, so they will be transmitted first and their measured RTT values equal 6. However, segment #39 will find segment #73 in front of it at the router, and segment #41 will find segments #80 and #81 in front of it at the router. The measured values of `SampleRTT`(t) as read from Figure 2-26 are:

`SampleRTT`(89) = 6; `SampleRTT`(96) = 6; `SampleRTT`(104) = 6; `SampleRTT`(113) = 7; `SampleRTT`(120) = 8;

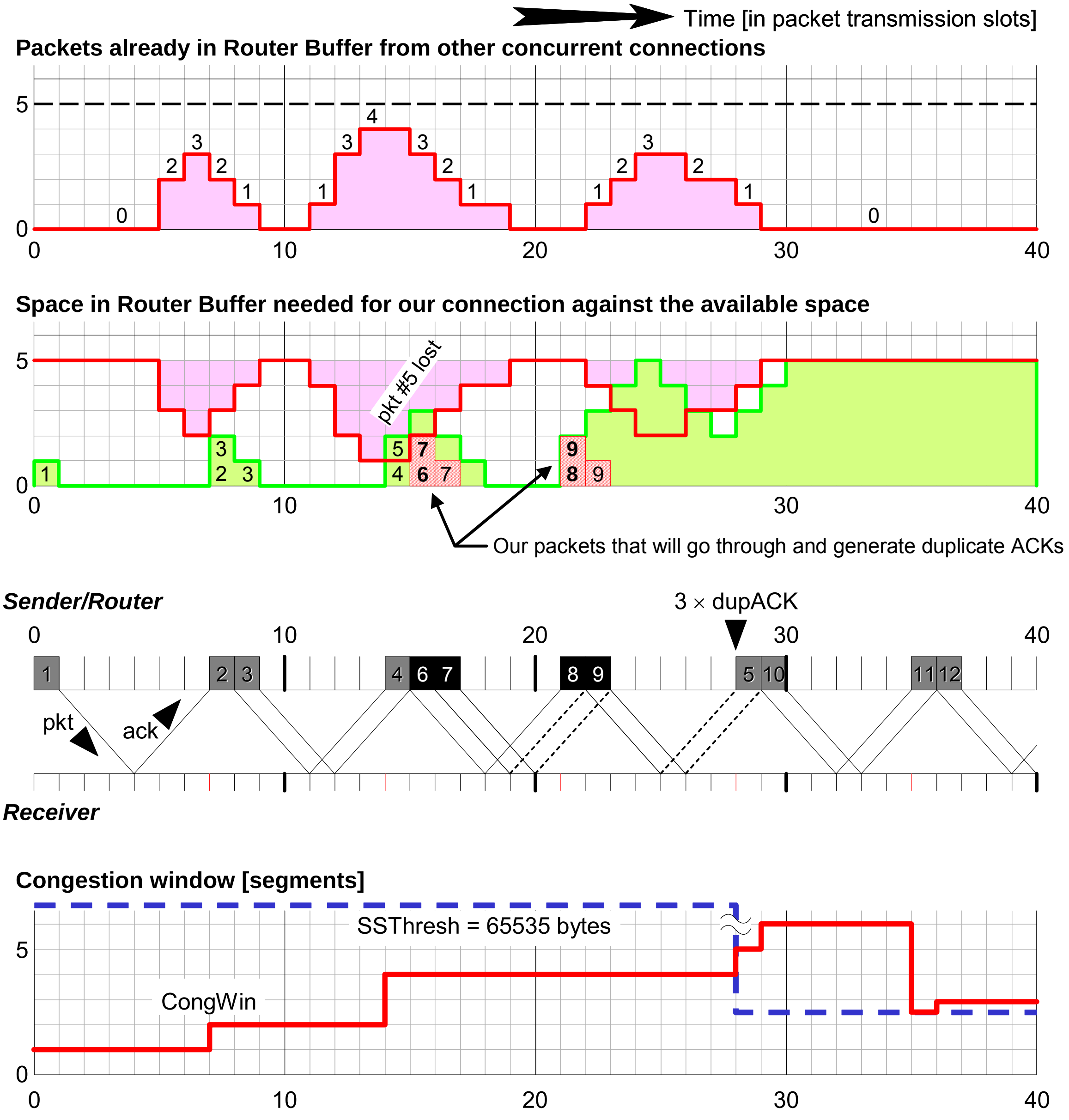
The calculation of `TimeoutInterval`(t) is straightforward using Eq. (2.1).

Problem 2.16 — Solution

(a)

The solution is shown in the figure below. (Recall that we are working with a TCP NewReno sender.) We mirror vertically the router buffer demand from other flows (top row of the figure, copied from the problem statement) and obtain the upper limit of buffer space available for our connection, shown in the second row of the figure below. Recall that the problem statement says that other packets compete with our packets only for memory space, but will wait for different output links. Therefore, our segments #2 and #3 will depart the router at times 8 and 9, respectively, as if they were the only packets in the router.

We can see that the first packet from our flow that will be above the upper limit curve is segment #5 and it will be lost at time $t = 14$. Then at time 15, two subsequent packets from our connection (#6 and #7) will go through the router. These packets are shaded differently in the second row of the figure below. The current congestion window (`CongWin` = 4) does not allow the sender to send more packets, so our sender is shut until the ACK for segment #4 arrives at time 21. This is a regular ACK and the sender is still in slow start, so it sends segments #8 and #9. Meanwhile, #6 and #7 are received out of order and will result in two duplicate ACKs at times 22 and 23. Two is not enough for loss detection, so the sender keeps waiting until more dupACKs arrive at 28 and 28. Hence, the sender will first detect a lost packet at time 28.



(b)

The congestion window sizes are shown in the bottom row of the above figure. The sequence numbers of the sent segments are shown in the third row above. (Note that the correct sequence numbers should be shown in bytes, but we keep them as packet numbers for simplicity.)

When the sender discovers the loss at time 28, it sets the congestion window size according to equation (2.7) from Section 2.2.2:

$$\begin{aligned}\texttt{CongWin} &= \lfloor \max\{\,½\,\texttt{FlightSize}, 2\times\texttt{MSS}\} + 3\times\texttt{MSS}\rfloor \\ &= \lfloor \max\{\,½ \times 5, 2\,\} + 3\rfloor = 5\end{aligned}$$

and the congestion window is incremented by one for each new duplicate acknowledgement that is received after the first three, so it becomes 6 at time 29 and `EffectiveWin` = `CongWin` – `FlightSize` = 6 – 5 = 1 and segment #10 is sent. Also, according to equation (2.6), at time 28 `SSThresh` becomes equal to 2.5.

At time 35, the ACK for retransmitted segment #5 is received. This is a cumulative ACK that acknowledges all segments that were transmitted before the loss was detected, i.e., up to and including segment #9. At this point, according to equation (2.8), `CongWin` = `SSThresh` = 2.5 and sender exits fast recovery and enters *congestion avoidance*. There is one segment in flight (#10) and the effective window is one, so segment #11 can be sent. When the ACK for segment #10 arrives at time 36, it will be counted according to equation (2.5):

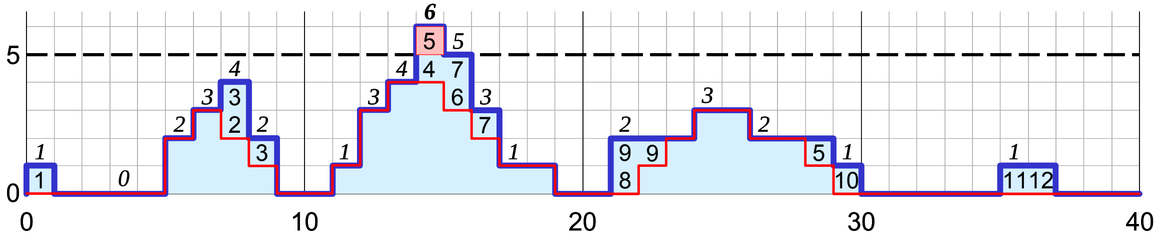
$$\text{CongWin}(t=36) = \text{CongWin}(35) + \frac{\text{MSS}^2}{\text{CongWin}(35)} = 2.5 + \frac{1}{2.5} = 2.9\ \text{MSS}$$

Given that one segment is already in flight (#11), one more segment will be sent (#12).

(c)

The total buffer occupancy at the router from all flows combined is shown below:

Total packets in Router Buffer from all connections



Recall that the problem statement said that the times when our packets will leave the router are independent of packets from other flows, because they are heading in different direction and are competing only for router memory space.

Problem 2.17 — Solution

Problem 3.1 — Solution

Problem 3.2 — Solution

Problem 3.3 — Solution

(a)

We assume that the packet with sequence number is the first packet, sent by the source at time $t_1 = 20$ ms. Its playout time is then: $p_1 = t_1 + q = 170$ ms. The playout times for the other packets are shown in the table. Packet #5 is discarded because it missed its playout time.

Packet sequence number	Arrival time r_i [ms]	Playout time p_i [ms]
#1	95	170
#2	145	190
#3	170	210
#4	135	230
#6	160	250
#5	275	discarded (>270)
#7	280	290
#8	220	310
#9	285	330
#10	305	350

(b)

The minimum propagation delay given in the problem statement is 50 ms. Given a fixed playout delay of $q = 150$ ms, any packet cannot spend more than 100 ms in the receiver's jitter buffer waiting for its playout. Because the source generates a packet every 20 ms, the maximum number of packets that can arrive during 100 ms is 5. Therefore, the required size of memory buffer at the destination is 6 × 160 bytes = 960 bytes. (The buffer should be able to hold 6 packets, rather than 5, because we assume that the last arriving packet is first buffered and then the earliest one is removed from the buffer and played out.)

Problem 3.4 — Solution

The length of time from when the first packet in this talk spurt is generated until it is played out is:

$$q_k = \hat{\delta}_k + K \cdot \hat{\upsilon}_k = 90 + 4 \times 15 = 150 \text{ ms}$$

The playout times for the packets including k+9th are obtained by adding this amount to their timestamp, *because they all belong to the same talk spurt*. Note the timestamp discontinuity between k+9th and k+10th packets—there is no packet with the timestamp 600 ms and all other timestamps are equally spaced by 20 ms. We see that the packet with the sequence number *k*+5 is missing (lost in transport), but this is not interpreted as the beginning of a new talk spurt because its timestamp would presumably fit the regularly spaced sequence. Also, when calculating $\hat{\delta}_{k+6}$ we are missing $\hat{\delta}_{k+5}$, but we just use $\hat{\delta}_{k+4}$ in its stead.

The new talk spurt starts at *k*+10, because there is no gap in sequence numbers, but the difference between the timestamps of subsequent packets is $t_{k+10} - t_{k+9} = 40$ ms > 20 ms, which indicates the beginning of a new talk spurt. The length of time from when the first packet in this new talk spurt is generated until it is played out is:

$$q_{k+10} = \hat{\delta}_{k+10} + K \cdot \hat{\upsilon}_{k+10} = 92.051 + 4 \times 15.9777 = 155.9618 \text{ ms} \approx 156 \text{ ms}$$

and this is reflected on the playout times of packets *k*+10 and *k*+11.

Packet seq. #	Timestamp t_i [ms]	Arrival time r_i [ms]	Playout time p_i [ms]	Average delay $\hat{\delta}_i$ [ms]	Average deviation $\hat{\upsilon}_i$
k	400	480	550	90	15
k+1	420	510	570	90	14.85
k+2	440	570	590	90.4	15.0975
k+3	460	600	610	90.896	15.4376
k+4	480	605	630	91.237	15.6209
k+7	540	645	690	91.375	15.6009
k+6	520	650	670	91.761	15.8273
k+8	560	680	710	92.043	15.9486
k+9	580	690	730	92.223	15.9669
k+10	620	695	776	92.051	15.9777
k+11	640	705	796	91.78	16.0857

Problem 3.5 — Solution

Given the playout delay, we can determine the constant *K* using Eq. (3.3) to find out which value of K gives the playout delay of 300 ms. Then, we use the chart shown in Figure A-7 in Appendix A to guess approximately what is the percentage of packets that will arrive to late to be played out.

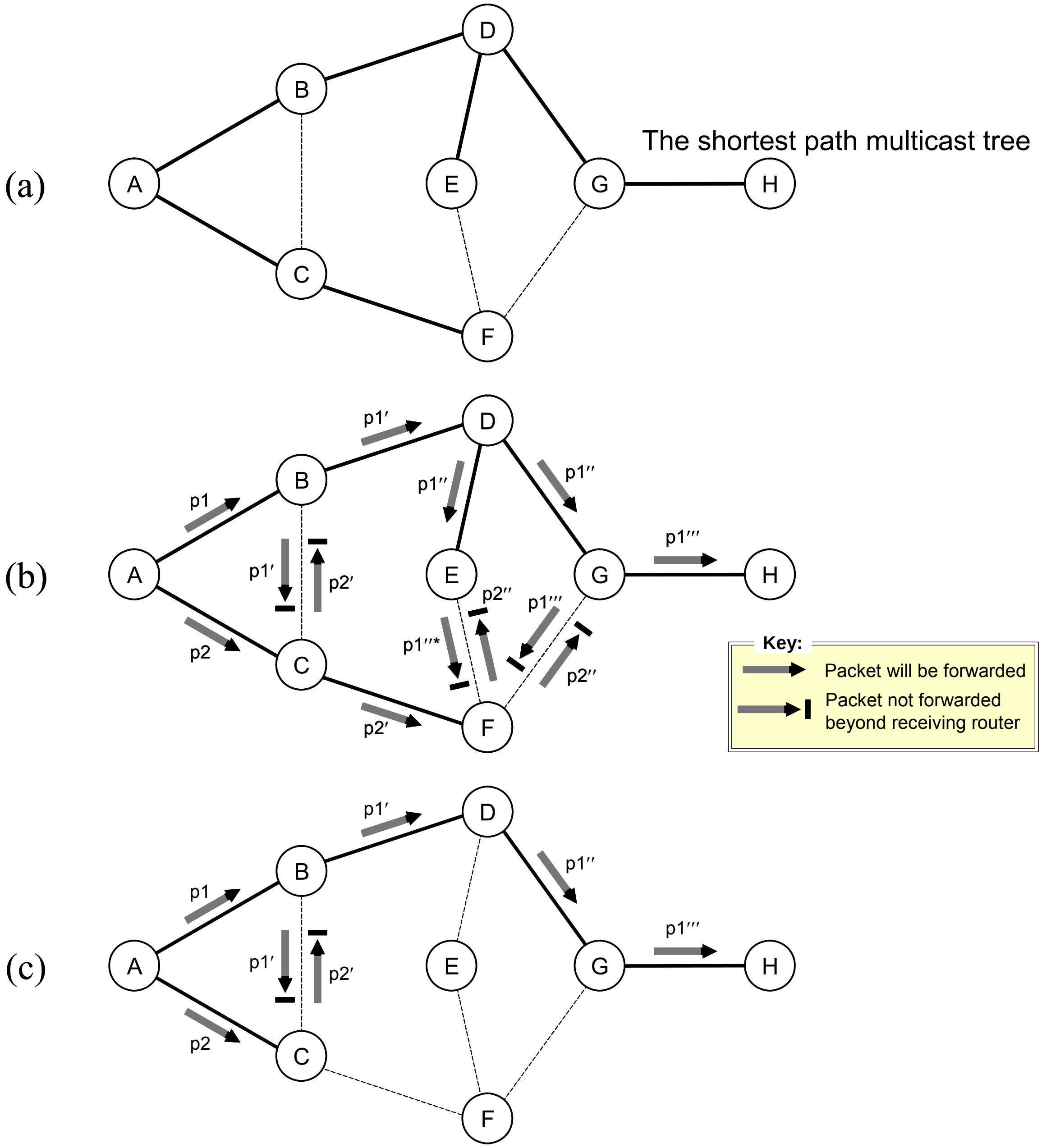
Problem 3.6 — Solution

Problem 3.7 — Solution

The solutions for (a) and (b) are shown in the figure below.

(c)

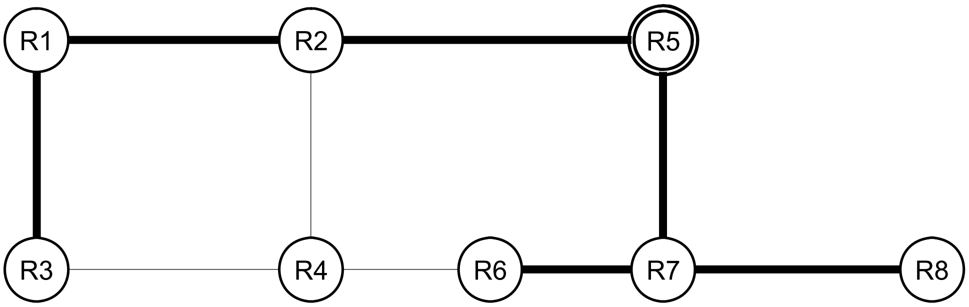
If the RPF uses pruning and routers *E* and *F* do not have attached hosts that are members of the multicast group, then there will be 6 packets less forwarded in the entire network per every packet sent by *A*, compared to the case (b).



Problem 3.8 — Solution

(a)

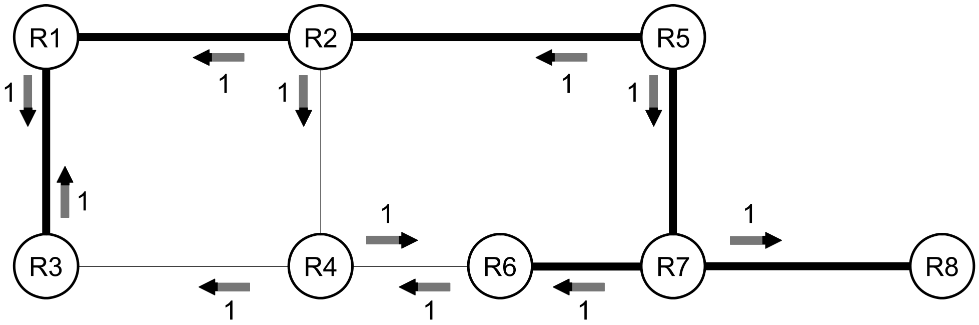
RPF from source *D* attached to the root router R5. R3 can be reached via R4 or via R1; say we resolve this tie by selecting R1 as preferred because it has a lower number identifier. Then R4 can be pruned because no host is attached to it.



(b)

The answer is 11.

Forward the packet if and only if arrived on the link that is on its own shortest unicast path to the source. R8 can be pruned after its host *E* departed.



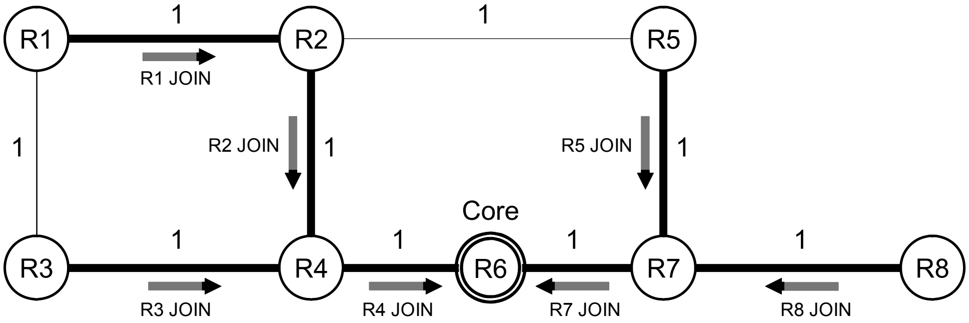
(c)

The answer is 10.

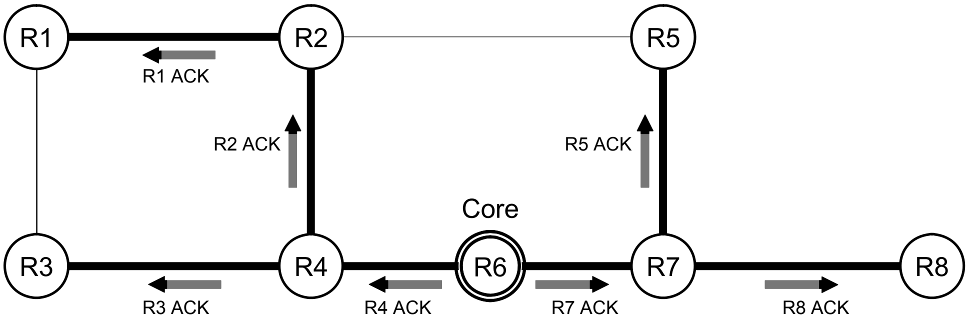
Problem 3.9 — Solution

(a)

This is a spanning tree multicast problem, using core-based trees (CBTs). The *group-shared multicast tree* is shown in the figure below with bold lines (assuming that *R*6 is the core node):

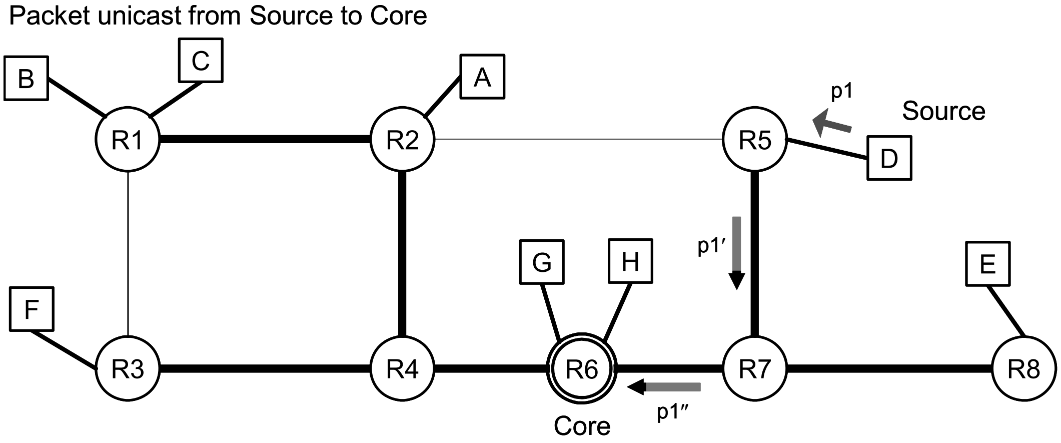


As described in Figure 3-11, the JOIN requests from router *R*1 will not be forwarded by *R*2, because *R*2 will have already sent its own JOIN request and is waiting for an acknowledgement. Similarly, *R*2's and *R*3's JOIN requests will not be forwarded by *R*4, and *R*5's and *R*8's requests will not be forwarded by *R*7. As a result, there will be a total of *seven* JOIN packets sent. When the Core *R*6 receives JOIN requests from *R*4 and *R*7, it sends an ACK, which is then propagated to all other routers in response to their JOIN requests. Again, there will be *seven* ACK packets sent.

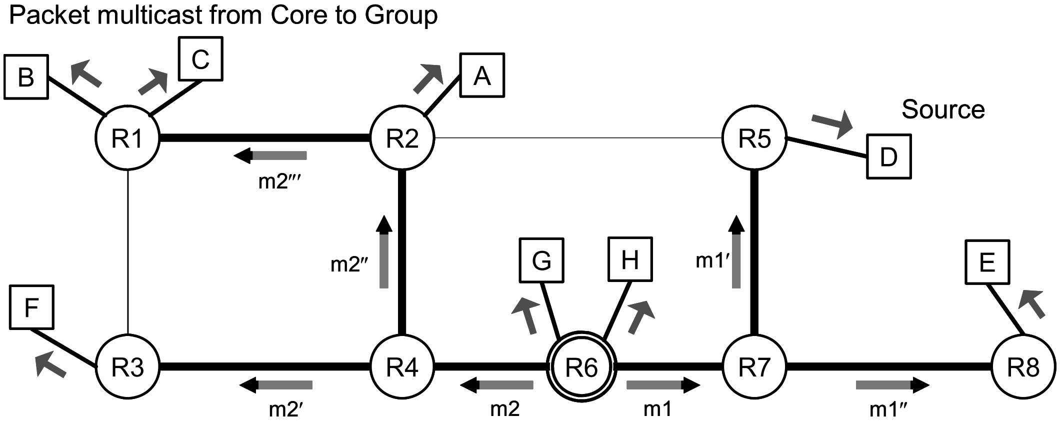


(b)

For the solution, also check Figure 3-12. We assume that the network implements standard CBT (no performance optimization), so *D* sends a packet first to its designated router, in our case *R*5. *R*5 encapsulates the packet into a unicast IP packet and sends it to the core R6, as shown here:



When the packet reaches R1, the core removes the unicast IP header and forwards it down the spanning tree:



The total number of packets forwarded in the entire network per each packet sent by the source *D* is can be counted from the figures above as:

Unicast forwarding: 2 packets

Multicast forwarding: 7 packets

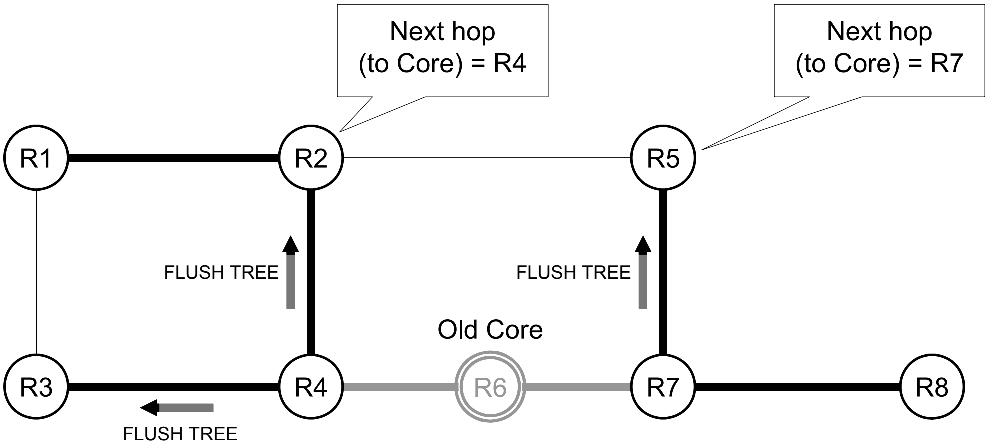
If forwarding packets to the endpoint hosts are counted, then multicast forwarding counts 7 + 8 packets.

(c)

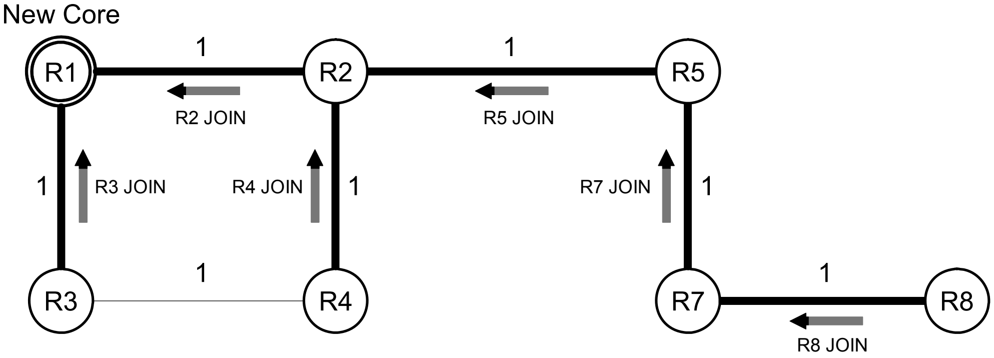
If host E leaves the group, router R8 will leave as well because it will not have hosts in the group any more. As a result, there will be 6 + 7 packets multicast in total.

(d)

Assuming that *R*6 crashed, its associated hosts *G* and *H* would not be part of the group any more. Router *R*1 becomes the new core the old spanning tree must be torn down by sending "flush-tree" messages, as shown (also see Figure 3-13):



After this, the messages sent during rebuilding a new spanning tree are similar to the solution in part (a), and the figure below shows only JOIN requests, which will be followed by the corresponding ACKs.



Problem 3.10 — Solution

Problem 3.11 — Solution

To calculate the RTCP transmission interval, we follow the algorithm shown in Figure 3-23. Although we expect that all three participants will talk at some time during the conference call, we assume that at most one will talk at any given time. (There may be brief periods when two or more participants start talking simultaneously, but we assume that these conflicting situations will be quickly resolved.) Therefore, the expected percentage of senders is 1/3 = 33%.

One audio stream (for one sender at a time) using PCM encoding requires the bandwidth of 64 Kbps. Given that RTCP is allowed 5% of the RTP data bandwidth, this yields 3.2 Kbps for RTCP bandwidth. The total required conferencing session bandwidth is 64+3.2 = 67.3 Kbps.

We calculate the average size of RTC packets by assuming that all participants will send compound RTCP reports (although only one at a time will act as sender). Each compound packet will contain both Sender Report and two Receiver Reports, because each participant is a sender and a receiver to the two other senders. From Figure 3-19 we see that a typical Receiver Report has two 32-bit words for the RTC header and two report blocks, each six 32-bit words long. Similarly, from Figure 3-22 we see that a typical Sender Report has thirteen 32-bit words (including the RTC header). Therefore, after we add 28 bytes per packet for the UDP and IPv4 headers (Figure 3-18), we have (2+6+6)×4 + 13×4 + 28 = 136 bytes. Finally, we find the interval for sending compound RTCP reports:

$$T = \frac{(136 \times 8\, bits) \times 3}{3200\, bps} = 1.02 \text{ sec}$$

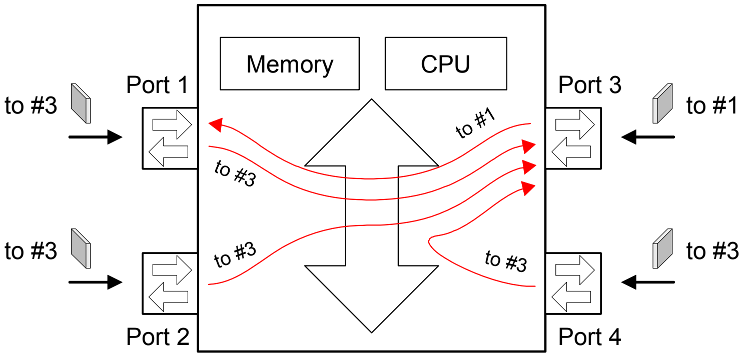
Problem 3.12 — Solution

Problem 4.1 — Solution

Problem 4.2 — Solution

(a)

Worst-case scenario: packets arriving at all four ports at a maximum rate and then from *three* ports heading to the same output port. (Note that *not* all four ports can send packets to the same output port, because otherwise packets from one input port would be transmitted back on the same link over which they arrived, which normally never happens.) An example is shown in the figure below, where three packets are heading to port 3:



Aggregate maximum input rate is 4 Mbps and the system bus data rate is also 4 Mbps. During one packet time on network links, four packets will arrive on the router (the packet time on the system bus equals one quarter of the network packet time). In the next network packet time, all four packets will be transported over the system bus. (Recall that in a second-generation router, a packet traverses the system bus only once.) They will empty one packet slot in each NFE memory and, at the same time, a new packet will arrive on the input port of each NFE. If all packets are heading to different output ports, then there will be no packet loss because each NFE can hold up to two packets.

However, if packets from three input ports are heading to the same output port (as in the above figure), then there will be three packets arriving on the same output port during the same network packet time, in addition to the one that arrived on the input port. Because each NFE can hold up to two packets, *two packets will be lost*.

(b)

Under the worst-case scenario, the first packet will be lost during the second network packet time. The network packet time is:

$$t_x = \frac{L}{R} = \frac{1024 \times 8[\text{bits}]}{1 \times 10^6[\text{bps}]} = 8.192 \text{ milliseconds}$$

so the first packet will be lost in the interval (8.192 ms, 16.384 ms).

(c)

Under the worst-case scenario, during each network packet time one port will lose two packets and the remaining three ports will lose none. Therefore, the maximum average loss is $\frac{2+0+0+0}{4} = 0.5$, i.e., 50 % of packets will be lost under the worst-case scenario. (Note that we ignored the first network packet time during which no packets will be ever lost.)

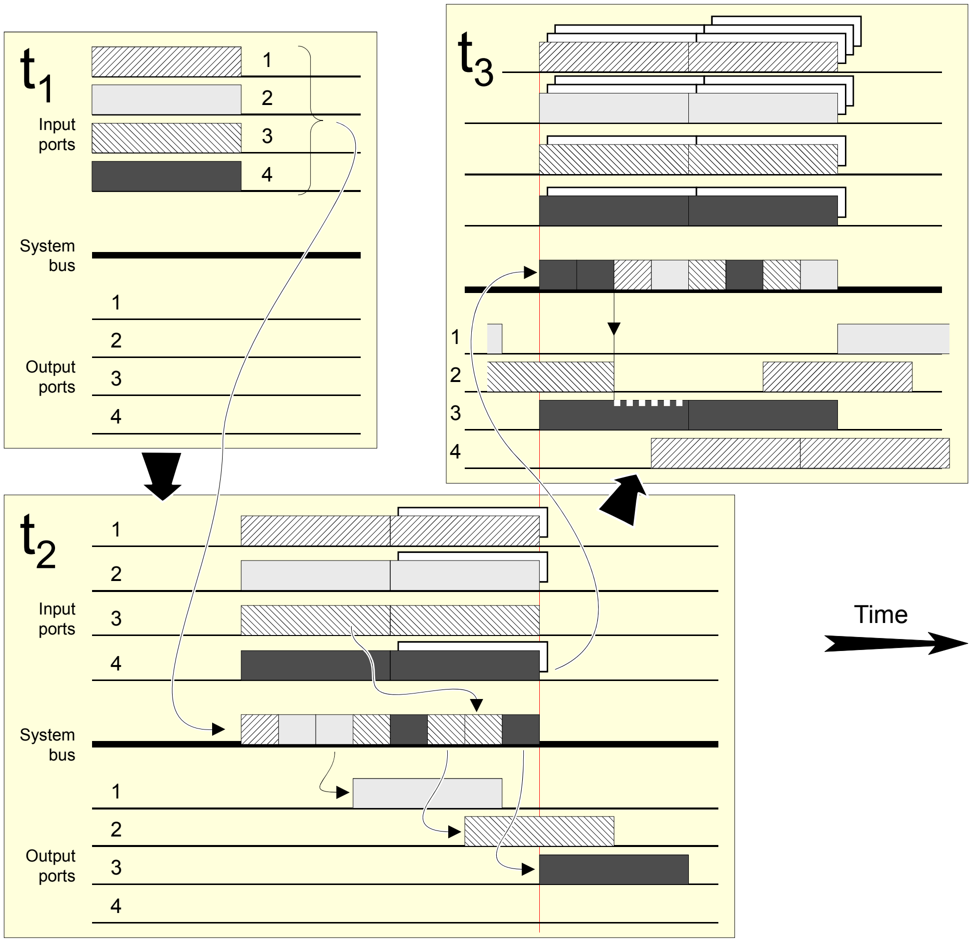
Problem 4.3 — Solution

The architecture of a first-generation router is shown in Figure 4-5(a). As shown in Figure 4-6, in this architecture every packet must cross the system bus two times on its way from the input port to the output port. We assume that processing times in line cards and the CPU are negligible.

(a)

If packets arrive simultaneously on all four ports, it will take $T_4 = 4 \times 2 \times \frac{L}{R}$ time units to move the packets from their input ports to their output ports. During this time, two packets can arrive on each input port. If packets continue arriving at the link data rate R, which is the *peak rate*, then during each cycle two packets will arrive on an input port and only one will be moved to the output port. The remaining packets will accumulate and the delay can grow arbitrarily high, depending on how long the period of peak-rate arrivals lasts.

The figure below illustrates an example of peak rate behavior. We assume that packets will arrive on all input ports simultaneously. However, to move the packet to the CPU, the input ports will access the system bus randomly, without priority access. Similarly, CPU will randomly access the system bus to move the packets to their output ports. As already stated, it takes T_4 time units move four incoming packets to their output ports, but at the same time up to 2 new packets can arrive at each input port. As illustrated in the figure, there will be a queue buildup on each input port. Note also that there may be queue buildup on output ports, for example, if packets from different input ports are heading to the same output port. Because of random access to the system bus, even packets from the same port may need to queue, as illustrated for output port 3 in the figure below at time t_3.



(b)

Technically speaking, there is *neither head-of-line blocking nor output blocking* in this system. Head-of-line blocking happens when one packet may be heading to an idle port, but in front of it may be another packet headed for an output port that is currently busy, and the former packet must wait until the one in front of it departs. This assumes that packets are not moved to their output port until their output port becomes idle. However, in the first generation routers, both input and output ports must be able to queue packets. (An input port must be able to queue a newly arriving packet while some previously arrived packets are waiting for access to the system bus, to be moved to the CPU for forwarding table lookup. The input port does not know where the incoming packet is heading, so all incoming packets are queued in FCFS manner to wait for access to the system bus and transfer to CPU.

CPU will examine the packet's header and decide to which output port it goes. Once CPU decides the output port, the packet is lined up in a FCFS queue, where the same queue is for *all* output ports. However, the head-of-line packet is never waiting for its output port to become idle; rather, it is waiting for the system bus access to get moved to its output port (regardless of whether this output port is currently idle or busy).

Output blocking occurs if two or more packets (all from different input ports) are headed to the same output port and the switching fabric is unable to move the simultaneously. The first generation routers are based on system bus as their switching fabric, so packets are always moved sequentially, one-by-one, and *never* simultaneously. Therefore, any delays that packet experience at their input ports neither qualify neither as head-of-line blocking nor as output blocking.

This is not to say that there are no queuing delays in this system. Quite opposite, there will be a major buildup of packets both at input and output queues during peak-arrival-rate periods, as explained above in part (a).

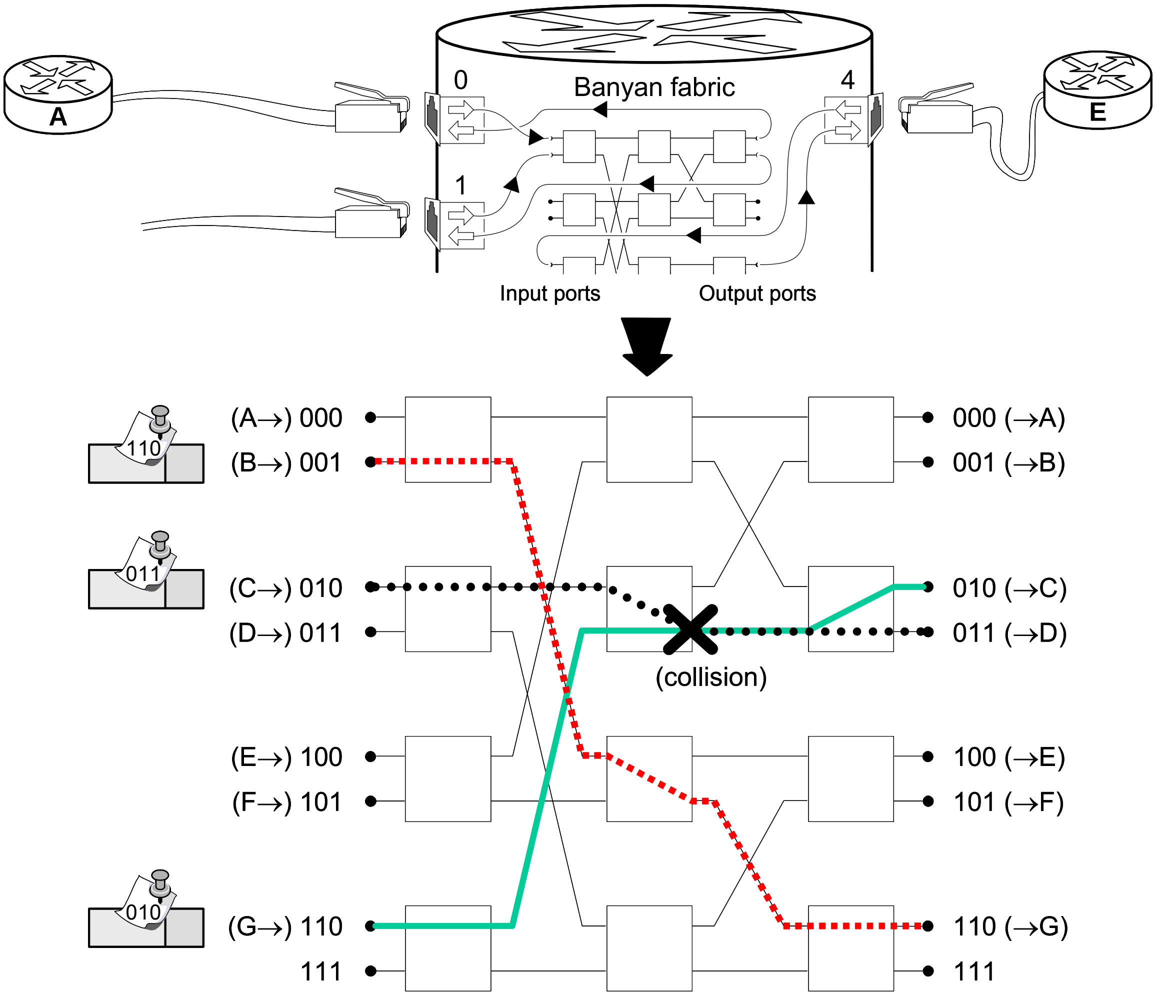
Problem 4.4 — Solution

Problem 4.5 — Solution

Check Problem 1.33 — Solution to see that the next hop router for the arrived packets is as follows:

Packet arrived from	Packet destination IP address	Next hop	Output port
B	63.67.145.18	G	6 (tag: 110)
C	223.123.59.47	D	3 (tag: 011)
G	223.125.49.47	C	2 (tag: 010)

The packets will traverse the Banyan switch as shown in the figure below. The top part of the figure shows how the router's network ports are wired to the switching fabric. Every network port is bidirectional and connected by a full-duplex link to another router.



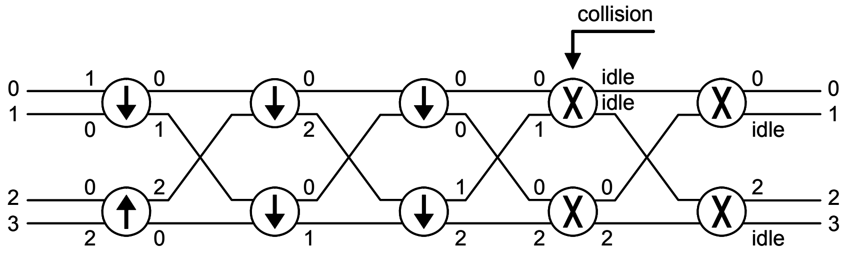
Note that although all packets go to different output ports, there will be a collision in the second stage of the Banyan fabric because packets from ports 2 and 6 are trying to go to the same output of the second-stage 2×2 switching element.

Problem 4.6 — Solution

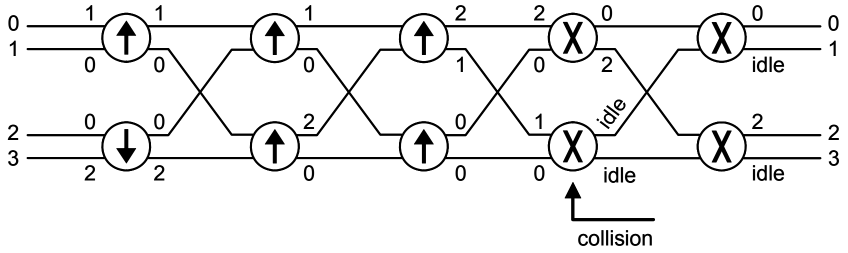
Problem 4.7 — Solution

Two alternative solutions for the Batcher-banyan fabric are shown:

(a)



(b)



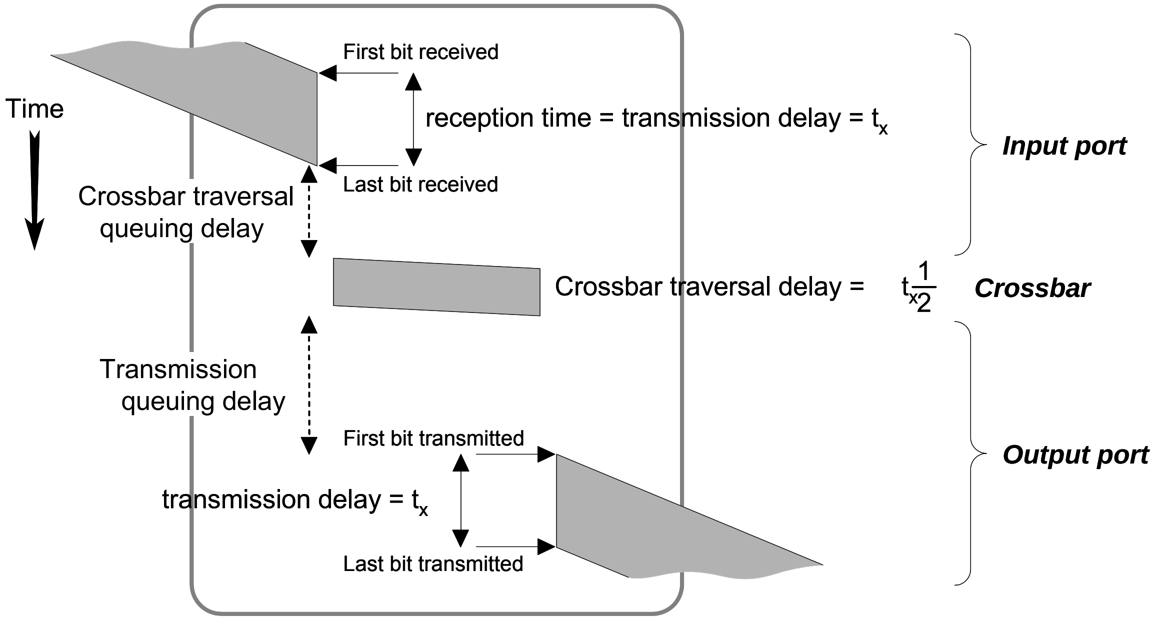
Result: the packets at inputs I0 and (I1 or I2) are lost.

The reader may notice that the above 4 × 4 Batcher network looks different from that in Figure 4-10(b). This just means that the same sorting problem can be solved in different ways, with different Batcher networks.

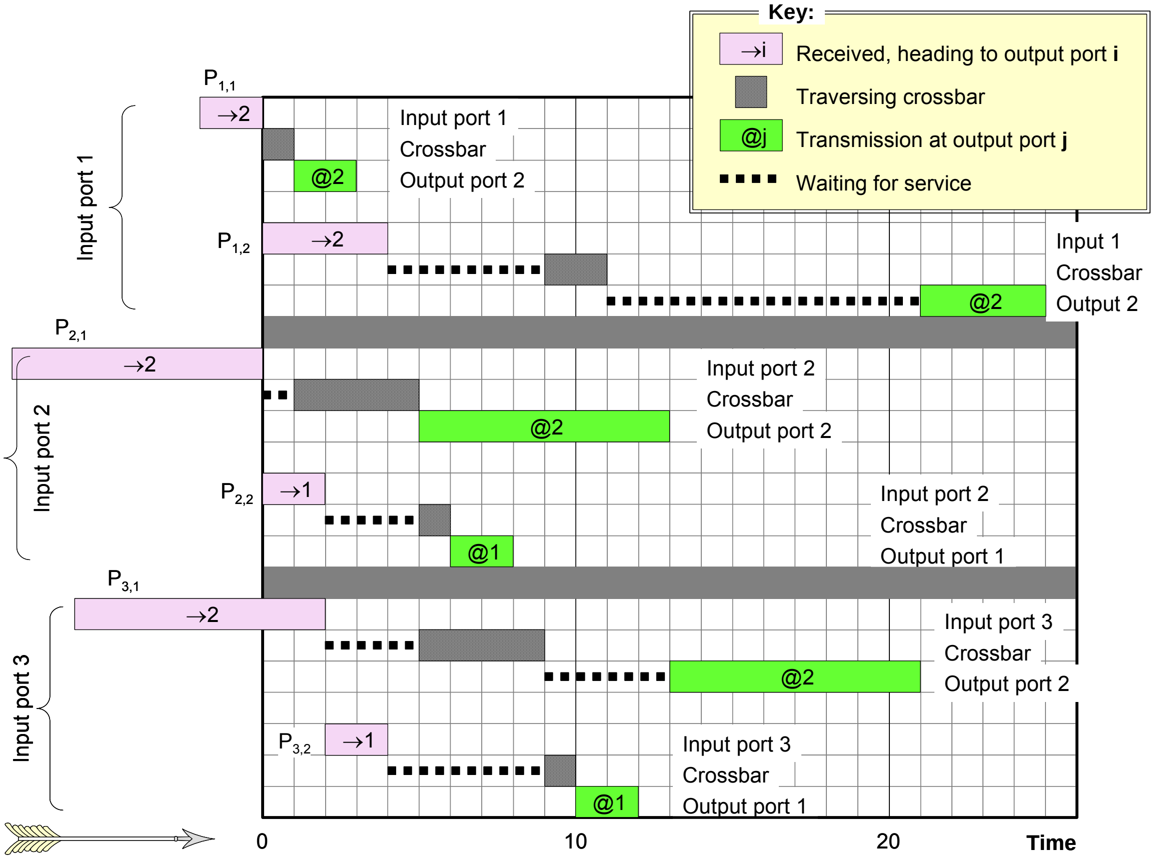
Problem 4.8 — Solution

Problem 4.9 — Solution

The figure below shows the components of the router datapath delay (compare to Figure 4-13). We ignore the forwarding decision delay. Transmission delays as well as reception delays are the same on all communication lines. Crossbar traversal delay equals $\frac{1}{2}t_x$.


Time
First bit received
reception time = transmission delay = t_x
Last bit received
Input port
Crossbar traversal queuing delay
Crossbar traversal delay = $\frac{1}{2} t_x$
Crossbar
Transmission queuing delay
First bit transmitted
transmission delay = t_x
Last bit transmitted
Output port

The timing diagram of packet handling in the router is shown in the figure below.



Only the first packet on input port 1 ($P_{1,1}$) will experience no waiting at all. All other packets will experience some form of blocking. The second packet on input port 1 ($P_{1,2}$) must wait before traversing the crossbar because the first packet on input port 2 ($P_{2,1}$) is currently traversing the crossbar and then it must wait for $P_{3,1}$ (although $P_{3,1}$ is on a higher-index port, it arrived before

$P_{1,2}$)—so $P_{1,2}$ is experiencing *output blocking*. $P_{2,1}$ is also experiencing output blocking because it has to wait for $P_{1,1}$ to traverse the crossbar. Finally, packet $P_{3,1}$ is also experiencing output blocking because it has to wait for $P_{2,1}$ to traverse the crossbar.

Packets $P_{2,2}$ and $P_{3,2}$ are experiencing *head-of-line blocking*, because they could traverse the crossbar if it were not for packets $P_{2,1}$ and $P_{3,1}$, respectively, which are if front of them in their respective queue and are blocking their access to the crossbar.

Output blocking and head-of-line blocking both prevent crossbar traversal and therefore cause queuing delay before crossbar traversal.

Note also that packets $P_{1,2}$ and $P_{3,1}$ must wait at the output port for their turn for transmission. This is indicated as transmission queuing delay in the first figure above.

Problem 4.10 — Solution

Problem 4.11 — Solution

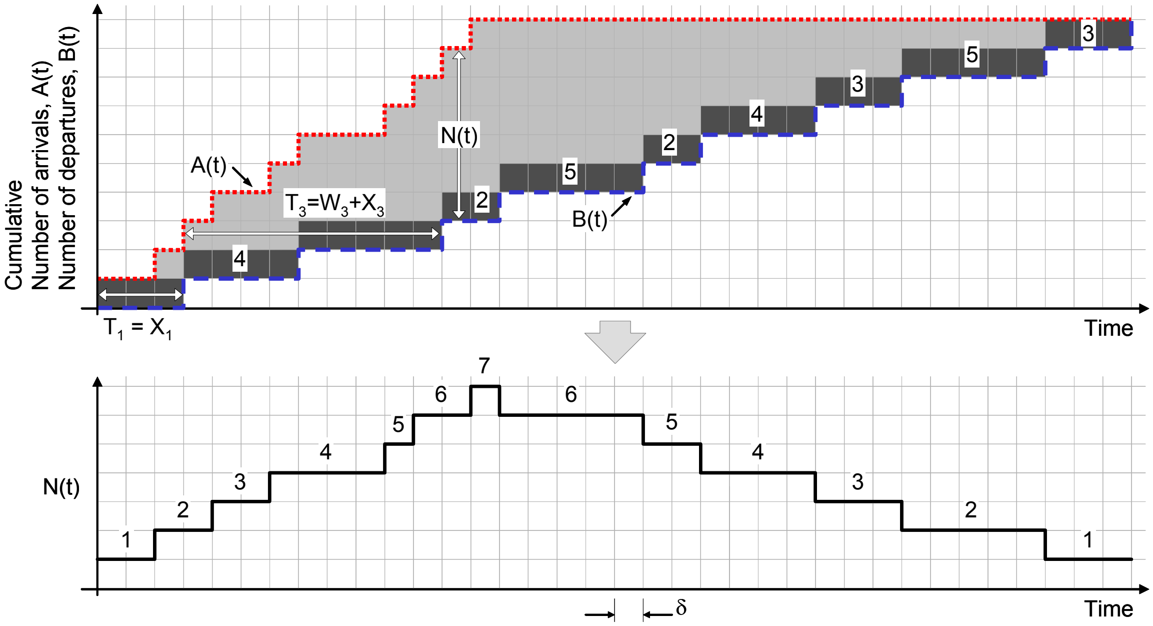
Problem 4.12 — Solution

Problem 4.13 — Solution

The problem statement gives arrival times A_i and service times X_i. From these we need to determine total delays T_i in the system for each customer. Note that there is a single server, so if a new customer arrives while one customer is served, the new customer should join the waiting line. Figure 4-17 illustrates that the total delay for a customer consists of waiting plus service time, i.e., $T_i = W_i + X_i$.

(a)

The following figure illustrates the arrivals $A(t)$, delays T_i and departures $B(t)$. The first customer arrives at an idle server, so immediately goes into service, i.e., $W_1 = 0$ and $T_1 = X_1$. The second customer arrives at time $t = 2$ and finds the first customer in service, so has to wait one time unit, $W_2 = 1$ and $T_2 = W_2 + X_2$. The length of the observed interval is 36 time units, until the last customer that arrived during the observed interval (customer #10) departs.



(b)

The lower chart indicates the current number of customers in the system, $N(t)$. We can calculate the average number N over the observed interval as follows

$$N = \frac{(5\times 1)+(7\times 2)+(5\times 3)+(8\times 4)+(3\times 5)+(7\times 6)+(1\times 7)}{36} = \frac{130}{36} = 3.6\dot{1}\ \text{customers}$$

The average delay T per customer over the observed interval is

$$T = \frac{3+(1+4)+(4+5)+(8+2)+(8+5)+(12+2)+(11+4)+(14+3)+(16+5)+(21+3)}{10} = \frac{131}{10} = 13.1$$

Assuming that the arrival rate is λ = 1 customer/unit-of-time, the Little's Law should yield $N = \lambda \cdot T$. However, over the observed interval we have $3.61 \neq 1\times 12.9$. Therefore, the system does *not* satisfy the Little's Law over the observed interval.

Problem 4.14 — Solution

Problem 4.15 — Solution

Problem 4.16 — Solution

(a)

This is an $M/M/1$ queue with the arrival rate λ = 950,000 packets/sec and service rate μ = 1,000,000 packets/sec. The expected queue waiting time is:

$$W = \frac{\lambda}{\mu\cdot(\mu-\lambda)} = \frac{950000}{1000000\times(1000000-950000)} = 19\times 10^{-6}\ \text{sec}$$

(b)

The time that an average packet would spend in the router if no other packets arrive during this time equals its service time, which is $\frac{1}{\mu} = \frac{1}{1000000} = 1 \times 10^{-6}$ sec

(c)

By Little's Law, the expected number of packets in the router is

$$N = \lambda \cdot T = \lambda \cdot \left(W + \frac{1}{\mu} \right) = 950000 \times 20 \times 10^{-6} = 19 \text{ packets}$$

Problem 4.17 — Solution

Problem 4.18 — Solution

Given

Data rate is 9600 bps ⇒ the average service time is $\frac{1}{\mu} = \frac{\text{average packet length}}{\text{link data rate}} = \frac{1000 \times 8}{9600} = 0.83$

$\therefore\ \mu = 1.2$

Link is 70% utilized ⇒ the utilization rate is $\rho = 0.7$

For exponential message lengths: $M/M/1$ queue with $\mu = 1.2$, $\rho = 0.7$, the average waiting time is $W = \frac{\rho}{\mu \cdot (1-\rho)} = \frac{0.7}{1.2 \times 0.3} = 1.94 \text{ sec}$.

For constant-length messages we have $M/D/1$ queue and the average waiting time is derived in Problem 4.21 — Solution, part (b), as: $W = \frac{\rho}{2 \cdot \mu \cdot (1-\rho)} = \frac{0.7}{2 \times 1.2 \times 0.3} = 0.97 \text{ sec}$.

It is interesting to observe that constant-length messages have 50 % shorter expected queue waiting time than the exponentially distributed length messages.

Problem 4.19 — Solution

The single repairperson is the server in this system and the customers are the machines. Define the system state to be the number of operational machines. This gives a Markov chain, which is the same as in an $M/M/1/m$ queue with arrival rate μ and service rate λ. The required probability is simply p_m for such a queue. Because the sum of state probabilities is $\sum_{i=0}^{m} p_i = 1$, the fraction of time the system spends in state m equals p_m. From Eq. (4.8), we have the steady-state proportion of time where there is no operational machine as $p_m = \frac{\rho^m \cdot (1-\rho)}{1-\rho^{m+1}}$.

Problem 4.20 — Solution

This can be modeled as an *M*/*M*/1/*m* system, because the there are a total of *K* users, and there can be up to *K* tasks in the system if their file requests coincide. The average service time is $\frac{1}{\mu} = \frac{\text{average packet length}}{\text{throughput rate}} = \frac{A \times R}{R} = A$ and the service rate is μ = 1/*A*. The user places the request, but may need to wait if there are already pending requests of other users. Let *W* denote the waiting time once the request is placed but before the actual transmission starts, which is unknown. Every user comes back, on average, after *A*+*B*+*W* seconds. Hence, the arrival rate is λ = $\frac{K}{A+B+W}$.

From Little's Law, given the average number *N* of customers in the system, the average waiting delay per customer is $W = T - A = \frac{N}{\lambda} - A$. The time *T* is from the moment the user places the request until the file transfer is completed, which includes waiting after the users who placed their request earlier but are not yet served, plus the time it takes to transfer the file (service time), which on average equals *A* seconds. (Only one customer at a time can be served in this system.)

Then, $\lambda = \frac{N}{W+A} = \frac{K}{A+B+W}$ and from here: $W = \frac{N\cdot(A+B) - K\cdot A}{K-N}$

For an *M*/*M*/1/*m* system, the average number *N* of users requesting the files is:

$$N = \frac{\rho}{1-\rho} - \frac{(K+1)\cdot\rho^{K+1}}{1-\rho^{K+1}}$$

where $\rho = \lambda / \mu$ is the utilization rate. Finally, the average time it takes a user to get a file since completion of his previous file transfer is ***A* + *B* + *W***.

Problem 4.21 — Solution

This is an *M*/*D*/1 queue with deterministic service times. Recall that *M*/*D*/1 is a sub-case of *M*/*G*/1. Given: Service rate, μ = 1/4 = 0.25 items/sec; arrival rate, λ = 0.2 items/sec.

(a)

Mean service time $\overline{X}$ = 4 sec.

$$\overline{X}^2 = \frac{1}{\mu^2} \text{ and } N_Q = \frac{\lambda^2 \cdot \overline{X}^2}{2\cdot(1-\rho)} = \frac{\lambda^2}{2\cdot\mu^2\cdot(1-\lambda/\mu)} = 16$$

The second moment of service time for the deterministic case is obtained as

$$0 = E\{x-\mu\}^2 = E\{x^2\} - E^2\{x\} \text{ and from here, we have } E\{x^2\} = E^2\{x\} = \overline{X}^2 = \frac{1}{\mu^2}$$

(b)

The total time spent by a customer in the system, T, is $T = W + \overline{X}$, where W is the waiting time in the queue $W = \dfrac{\rho}{2 \cdot \mu \cdot (1-\rho)} = 8 \text{ sec}$ so the total time $T = 12$ sec.

Problem 4.22 — Solution

Problem 4.23 — Solution

Problem 5.1 — Solution

Problem 5.2 — Solution

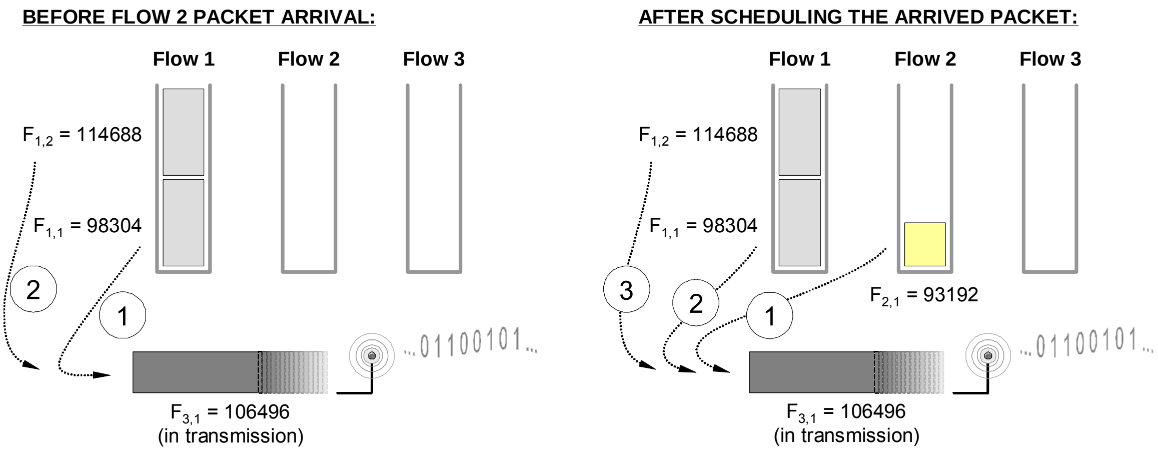
Problem 5.3 — Solution

Problem 5.4 — Solution

Recall that packet-by-packet FQ is *non-preemptive*, so the packet that is already in transmission will be let to finish regardless of its finish number. Therefore, the packet of class 3 currently in transmission can be ignored from further consideration. It is interesting to observe that the first packet from flow 1 has a smaller finish number, so we can infer that it must have arrived *after* the packet in flow 3 was already put in service.

The start round number for servicing the currently arrived packet equals the current round number, because its own queue is empty. Hence, $F_{2,1} = R(t) + L_{2,1} = 85000 + 1024\times8 = 93192$.

Therefore, the order of transmissions under FQ is: $pkt_{2,1} < pkt_{1,1} < pkt_{1,2}$; that is, the newly arrived packet goes first (after the one currently in service is finished).

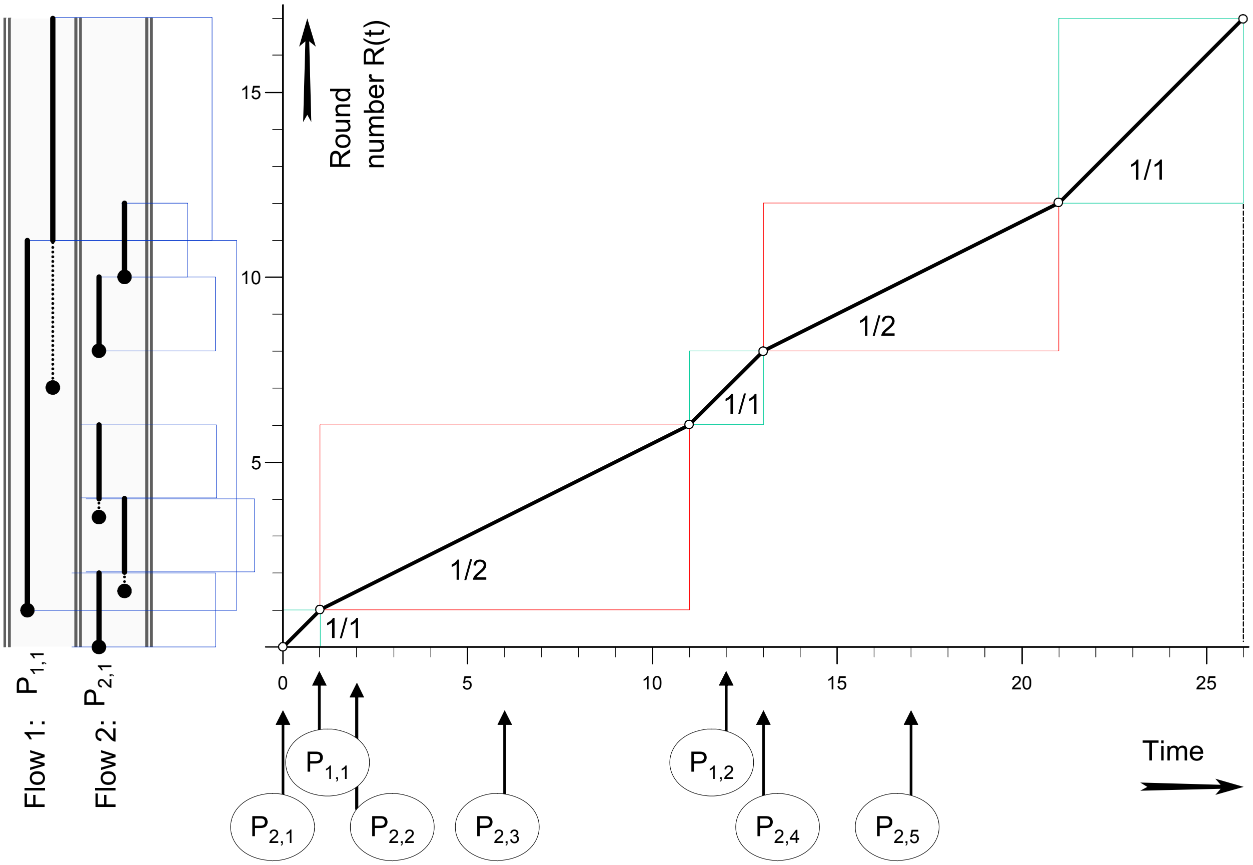


Problem 5.5 — Solution

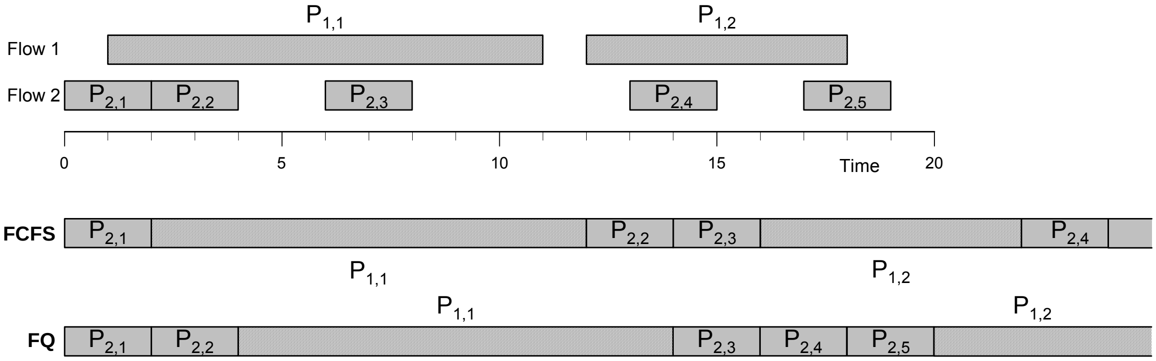
Problem 5.6 — Solution

(a)

We first determine the round numbers under bit-by-bit GPS, as shown in this figure:



Based on the finish round numbers, the timetable of packet departures under FCFS and FQ scheduling disciplines is shown in the figure below. Note that although packets $P_{2,3}$ and $P_{2,4}$ have smaller finish numbers than packet $P_{1,1}$, $P_{1,1}$ will be transmitted first, because when the transmission of $P_{2,2}$ is completed $P_{1,1}$ is the only packet waiting for service and other packets will arrive in the future. When they arrive, the service of a previous packet with a greater finish number is not interrupted, because FQ is a *non-preemptive* queuing discipline.



(b)

The waiting delays experienced by each packet are as follows:

Packet #	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{2,3}$	$P_{2,4}$	$P_{2,5}$
FCFS	1	4	0	9	8	9	7
FQ	3	8	0	0	8	3	1

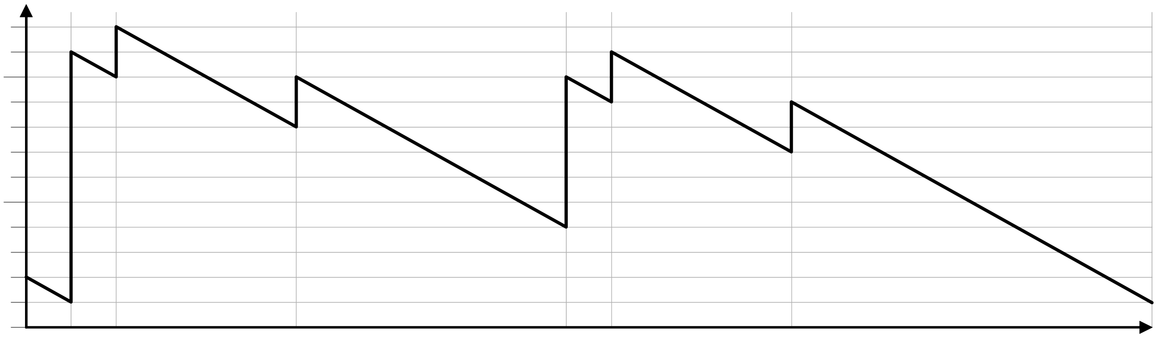
Average delay for Flow 1 under FCFS = (1+4) / 2 = 5/2 = 2.5
Average delay for Flow 1 under FQ = (3+8) / 2 = 11/2 = 5.5

Average delay for Flow 2 under FCFS = (0+9+8+9+7) / 5 = 33/5 = 6.6
Average delay for Flow 2 under FQ = (0+0+8+3+1) / 5 = 12/5 = 2.4

Flow 2 appears to fare better under FQ, but over a long run, the average delays for both flows under FQ would be approximately equal.

(c)

The curve representing the packet delay that would be experienced by a hypothetical packet that arrived at any time on either flow under the FCFS scheduling discipline is:

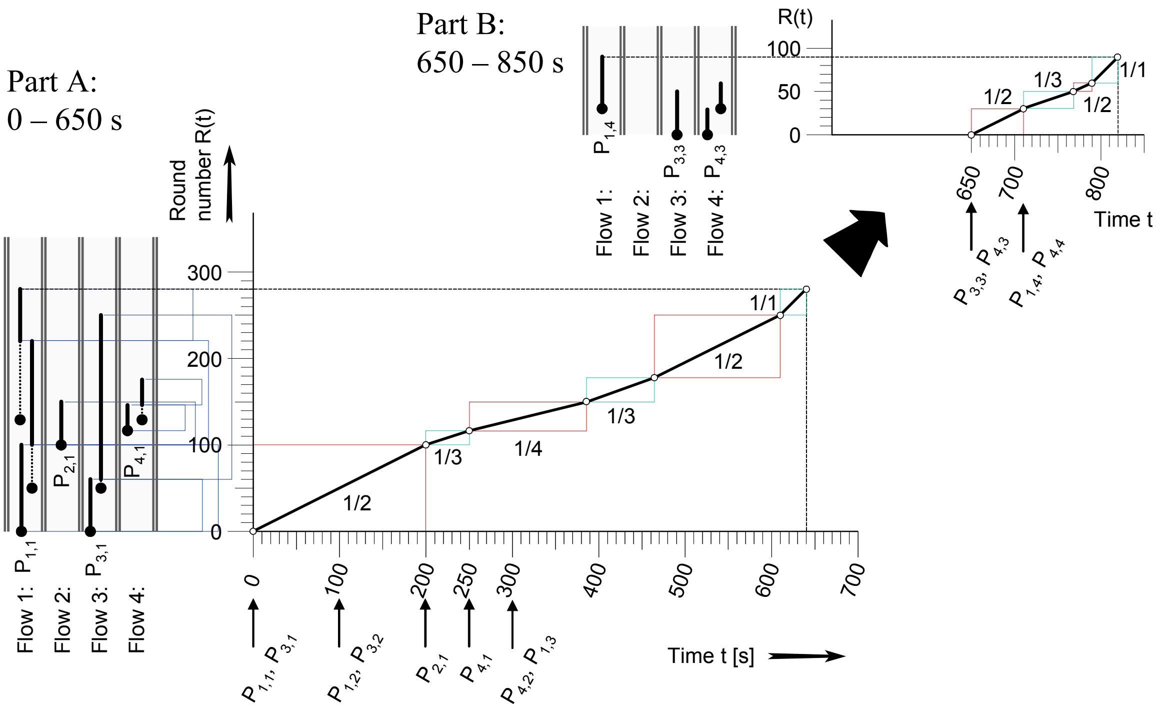


For the delays under FQ, we need to know information about the hypothetical arriving packet: which flow the packet arrives on and what is its size.

Problem 5.7 — Solution

(a) Packet-by-packet FQ

The following figure helps to determine the round numbers, based on bit-by-bit GPS. The packets are grouped in two groups, as follows. Regardless of the scheduling discipline, all of the packets that arrived by 300 s will be transmitted by 640 s. It is easy to check this by using a simple FIFO scheduling. Therefore, the round number $R(t)$ can be considered independently for the packets that arrive up until 300 s vs. those that arrive thereafter. This is shown as Part A and B in the above figure. (Resetting the round number is optional, only for the sake of simplicity.)



The packet arrivals on different flows are illustrated on the left hand side of the figure, in the round number units. Thus, e.g., packet $P_{2,1}$ arrives at time $t_{2,1}$ = 200 s or round number $R(t_{2,1})$ = 100. The following table summarizes all the relevant computations for packet-by-packet FQ.

Arrival times & state	Parameters	Values under packet-by-packet FQ
$t = 0$: $\{P_{1,1}, P_{3,1}\}$ arrive server idle, q's empty	Finish numbers	$R(0) = 0$; $F_{1,1} = L_{1,1} = 100$; $F_{3,1} = L_{3,1} = 60$
	Transmit periods	Start/end($P_{3,1}$): 0→60 sec; Start/end($P_{1,1}$): 60→160 s
$t = 100$: $\{P_{1,2}, P_{3,2}\}$ $P_{1,1}$ in transmission All queues empty	Finish numbers	$R(t) = t{\cdot}C/N = 100{\times}1/2 = 50$ $F_{1,2} = \max\{F_{1,1}, R(t)\} + L_{1,2} = 100 + 120 = 220$; $F_{3,2} = \max\{0, R(t)\} + L_{3,2} = 50 + 190 = 240$
	Transmit periods	Start/end($P_{1,2}$): 160→280 s; Queued packets: $P_{3,2}$
$t = 200$: $\{P_{2,1}\}$ arrives $P_{1,2}$ in transmission $P_{3,2}$ in queue	Finish numbers	$R(t) = t{\cdot}C/N = 200{\times}1/2 = 100$ $F_{2,1} = \max\{0, R(t)\} + L_{2,1} = 100 + 50 = 150$ $F_{3,2} = 240$ (unchanged);
	Transmit periods	$P_{1,2}$ ongoing; Queued packets: $P_{2,1} < P_{3,2}$
$t = 250$: $\{P_{4,1}\}$ arrives $P_{1,2}$ in transmission $\{P_{2,1}, P_{3,2}\}$ in queues	Finish numbers	$R(t) = (t{-}t'){\cdot}C/N + R(t') = 50{\times}1/3 + 100 = 116\ 2/3$ $F_{2,1} = 150$ (unchanged); $F_{3,2} = 240$ (unchanged); $F_{4,1} = \max\{0, R(t)\} + L_{4,1} = 116.6\dot{7} + 30 = 146.6\dot{7}$
	Transmit periods	Start/end($P_{4,1}$): 280→310 s; Queued pkts: $P_{2,1} < P_{3,2}$
$t = 300$: $\{P_{4,2}, P_{1,3}\}$ $P_{4,1}$ in transmission $\{P_{2,1}, P_{3,2}\}$ in queues	Finish numbers	$R(t) = (t{-}t'){\cdot}C/N + R(t') = 50{\times}1/4 + 116.6\dot{7} = 129\ 1/6$ $F_{1,3} = \max\{0, R(t)\}{+}L_{1,3} = 129.1\dot{6} + 60 = 189.1\dot{6}$; $F_{2,1} = 150$ (unchanged); $F_{3,2} = 240$ (unchanged); $F_{4,2} = \max\{0, R(t)\} + L_{4,2} = 129.1\dot{6} + 30 = 159.1\dot{6}$
	Transmit periods	$P_{4,1}$ ongoing; Queued packets: $P_{2,1} < P_{4,2} < P_{1,3} < P_{3,2}$ Start/end($P_{2,1}$): 310→360 s; s/e($P_{4,2}$): 360→390 s;

		Start/end($P_{1,3}$): 390→450 s; s/e($P_{3,2}$): 450→640 s.
At t = 640 s, round number reset, $R(t) = 0$, because the system becomes idle.		
t = 650: $\{P_{3,3}, P_{4,3}\}$ server idle, q's empty	Finish numbers	$R(0) = 0$; $F_{3,3} = L_{3,3} = 50$; $F_{4,3} = L_{4,3} = 30$
	Transmit periods	Start/end($P_{4,3}$): 650→680 sec; s/e($P_{3,3}$): 680→730 s.
t = 710: $\{P_{1,4}, P_{4,4}\}$ $P_{3,3}$ in transmission All queues empty	Finish numbers	$R(t) = (t-t')\cdot C/N + R(t') = 110\times1/2 + 0 = 55$ $F_{1,4} = \max\{0, R(t)\} + L_{1,4} = 55 + 60 = 115$; $F_{4,4} = \max\{30, R(t)\} + L_{4,4} = 55 + 30 = 85$
	Transmit periods	$P_{3,3}$ ongoing; Queued packets: $P_{4,4} < P_{1,4}$ Start/end($P_{4,4}$): 730→760 s; s/e($P_{1,4}$): 760→820 s.

(b) Packet-by-packet WFQ; Weights for flows 1-2-3-4 are 4:2:1:2

The round number computation, based on bit-by-bit GPS, remains the same as in the figure above. The only difference is in the computation of finish numbers under packet-by-packet WFQ, see Eq. (5.3), as summarized in the following table.

Packets $P_{1,4}$ and $P_{4,4}$ end up having the same finish number (70); the tie is broken by a random drawing so that $P_{1,4}$ is decided to be serviced first, ahead of $P_{4,4}$.

Arrival times & state	**Parameters**	**Values under packet-by-packet WFQ**
t = 0: $\{P_{1,1}, P_{3,1}\}$ arrive server idle, q's empty	Finish numbers	$R(0) = 0$; $F_{1,1} = L_{1,1}/w_1 = 100/4 = 25$; $F_{3,1} = 60$
	Transmit periods	Start/end($P_{1,1}$): 0→100 s; Start/end($P_{3,1}$): 100→160 s
t = 100: $\{P_{1,2}, P_{3,2}\}$ $P_{3,1}$ in transmission All queues empty	Finish numbers	$R(t) = t\cdot C/N = 100\times1/2 = 50$ $F_{1,2} = \max\{F_{1,1}, R(t)\} + L_{1,2}/w_1 = 100 + 120/4 = 130$; $F_{3,2} = \max\{0, R(t)\} + L_{3,2}/w_3 = 50 + 190/1 = 240$
	Transmit periods	Start/end($P_{1,2}$): 160→280 s; Queued packets: $P_{3,2}$
t = 200: $\{P_{2,1}\}$ arrives $P_{1,2}$ in transmission $P_{3,2}$ in queue	Finish numbers	$R(t) = t\cdot C/N = 200\times1/2 = 100$ $F_{2,1} = \max\{0, R(t)\} + L_{2,1}/w_2 = 100 + 50/2 = 125$ $F_{3,2} = 240$ (unchanged);
	Transmit periods	$P_{1,2}$ ongoing; Queued packets: $P_{2,1} < P_{3,2}$
t = 250: $\{P_{4,1}\}$ arrives $P_{1,2}$ in transmission $\{P_{2,1}, P_{3,2}\}$ in queues	Finish numbers	$R(t) = (t-t')\cdot C/N + R(t') = 50\times1/3 + 100 = 116\ 2/3$ $F_{2,1} = 125$ (unchanged); $F_{3,2} = 240$ (unchanged); $F_{4,1} = \max\{0, R(t)\} + L_{4,1}/w_4 = 116.6\dot{7} + 30/2 = 131.6\dot{7}$
	Transmit periods	Start/end($P_{2,1}$): 280→330 s; Queued pkts: $P_{4,1} < P_{3,2}$
t = 300: $\{P_{4,2}, P_{1,3}\}$ $P_{2,1}$ in transmission $\{P_{3,2}, P_{4,1}\}$ in queues	Finish numbers	$R(t) = (t-t')\cdot C/N + R(t') = 50\times1/4 + 116.6\dot{7} = 129\ 1/6$ $F_{1,3} = \max\{0, R(t)\} + L_{1,3}/w_1 = 129.1\dot{6} + 60/4 = 144.1\dot{6}$; $F_{3,2} = 240$ (unchanged); $F_{4,1} = 131.6\dot{7}$ (unchanged); $F_{4,2} = \max\{131.6\dot{7}, R(t)\} + L_{4,2}/w_4 = 146.6\dot{7}$
	Transmit periods	$P_{2,1}$ ongoing; Queued packets: $P_{4,1} < P_{1,3} < P_{4,2} < P_{3,2}$ Start/end($P_{4,1}$): 330→360 s; s/e($P_{1,3}$): 360→420 s; Start/end($P_{4,2}$): 420→450 s; s/e($P_{3,2}$): 450→640 s.
At t = 640 s, round number reset, $R(t) = 0$, because the system becomes idle.		
t = 650: $\{P_{3,3}, P_{4,3}\}$ server idle, q's empty	Finish numbers	$R(0) = 0$; $F_{3,3} = L_{3,3}/w_3 = 50$; $F_{4,3} = L_{4,3}/w_4 = 15$
	Transmit periods	Start/end($P_{4,3}$): 650→680 sec; s/e($P_{3,3}$): 680→730 s.
t = 710: $\{P_{1,4}, P_{4,4}\}$	Finish numbers	$R(t) = (t-t')\cdot C/N + R(t') = 110\times1/2 + 0 = 55$

$P_{3,3}$ in transmission All queues empty		$F_{1,4} = \max\{0, R(t)\} + L_{1,4}/w_1 = 55 + 60/4 = 70$; $F_{4,4} = \max\{30, R(t)\} + L_{4,4}/w_4 = 55 + 30/2 = 70$
	Transmit periods	$P_{3,3}$ ongoing; Queued pkts: $P_{1,4} = P_{4,4}$ (tie ⇒ random) Start/end($P_{1,4}$): 730→790 s; s/e($P_{4,4}$): 790→820 s.

Finally, the following table summarizes the order/time of departure:

Packet #	Arrival time [sec]	Packet size [bytes]	Flow ID	Departure order/ time under FQ	Departure order/ time under WFQ
1	0	100	1	#2 / 60 s	#1 / 0 s
2	0	60	3	#1 / 0 s	#2 / 100 s
3	100	120	1	#3 / 160 s	#3 / 160 s
4	100	190	3	#8 / 450 s	#8 / 450 s
5	200	50	2	#5 / 310 s	#4 / 280 s
6	250	30	4	#4 / 280 s	#5 / 330 s
7	300	30	4	#6 / 360 s	#7 / 420 s
8	300	60	1	#7 / 390 s	#6 / 360 s
9	650	50	3	#10 / 680 s	#10 / 680 s
10	650	30	4	#9 / 650 s	#9 / 650 s
11	710	60	1	#12 / 760 s	#11 / 730 s (tie)
12	710	30	4	#11 / 730 s	#11 / 790 s (tie)

Problem 5.8 — Solution

Let us assign the high-priority queue index 1, and the other two queues have indices 2 and 3. We modify Eq. (5.2) as follows. For a high priority packet $P_{1,j}$, the finish number is:

$$F_{1,j} = \max\{F_{1,j-1}, R(t_a)\} + L_{1,j} \tag{5.2$'$}$$

If the priority packet is at the head of its own queue and no other packet is currently serviced, then $F_{1,j} = R(t_a) + L_{1,j}$. If a non-priority packet $P_{i,k}$, $i \neq 1$, is currently in service, then we use its finish number $F_{i,k}$ to compute $F_{1,j} = F_{i,k} + L_{1,j}$, because a currently serviced packet is not preempted.

For a non-priority packet $P_{i,j}$, $i \neq 1$:

$$F_{i,j} = \max\{F_{1,\text{last}}, F_{i,j-1}, R(t_a)\} + L_{i,j}; \quad i \neq 1 \tag{5.2$''$}$$

where $F_{1,\text{last}}$ symbolizes the finish number of the *last* packet currently in the priority queue.

Any time new packet arrives (whether to a priority or non-priority queue), we must recompute the finish numbers and sort the packets in the ascending order of their finish numbers.

The order and departure times are as follows:

$P_{2,1}$ at $t = 0$ (not preempted) | $P_{1,1}$ at $t = 6$ | $P_{1,2}$ at $t = 8$ | $P_{3,1}$ at $t = 10$ | $P_{3,2}$ at $t = 12$ | $P_{2,2}$ at $t = 13$ (the tie between $P_{2,2}$ and $P_{3,2}$ is resolved by flipping a coin).

Problem 5.9 — Solution

Problem 5.10 — Solution

Problem 5.11 — Solution

Problem 5.12 — Solution

Problem 5.13 — Solution

(a)

Use the recommended $\gamma = 0.002$. At most 1 packet departing the router per unit of time.

At time $t = 1$, we calculate:

$\text{AvgQL}(1) = (1-\gamma)\cdot \text{AvgQL}(0) + \gamma\cdot\text{MeasuredQL}(1) = 0.998\times6.75 + 0.002\times9 = 6.7545$

$\text{P}_{\text{temp}}(\text{AvgQL}) = P_{max}\cdot(\text{AvgQL}-\text{Thr}_{\text{min}})/(\text{Thr}_{\text{max}}-\text{Thr}_{\text{min}}) = 0.1\times(6.7545-0)/(7-0)\approx0.0965.$

$\text{P}(\text{AvgQL}) = \text{P}_{\text{temp}}(\text{AvgQL}) / (1 - \text{count}\times\text{P}_{\text{temp}}(\text{AvgQL})) = 0.0965 / 0.9035 \approx 0.1068.$

We continue using the same formulas and decide to drop a packet only if the calculated probability exceeds 0.15. The calculations are shown in the table below. The column named "Measured QL" shows the current number of packets queued in the router buffer. The notation *a*–*b*+*c*–*d*, such as 9–1+1–1 in the last row, indicates that there were *a* packets queued in the previous time unit, of which *b* departed the router, *c* new packets arrived, and *d* were dropped.

Time	Measured QL	AvgQL	P_{temp}	count	P	Drop decision
init	9	6.75		0		
1	9–1+1 = 9	6.7545	0.0965	1	0.1068	No, queue the packet
2	9–1+1 = 9	6.758991	0.09656	2	0.11967	No, queue the packet
3	9–1+1 = 9	6.763473	0.09662	3	0.13606	No, queue the packet
4	9–1+1 = 9	6.767946	0.09668	4	0.15764	Yes, drop the packet and reset *count* ← 0
5	9–1+1–1 = 8	6.77041	0.09672	1	0.10708	No, queue the packet

(b)

The first packet will be dropped at time $t = 4$.

Problem 5.14 — Solution

The problem statement suggests that, because $\textit{Threshold}_{max}$ is less than *BufferSize*, the "tail" part of the buffer will never be filled because as soon as $\textit{Threshold}_{max}$ is reached, packets will be dropped. This conclusion is not true because RED is based on *average* queue length rather than *instantaneous* (or, measured) queue length. Equation (5.6) that determines the probability of a packet being dropped intentionally uses *AverageQLen*(*t*) instead of *MeasuredQLen*(*t*) to determine the drop probability.

For bursty traffic, the *MeasuredQLen* could quickly become much larger than the *AverageQLen*, as calculated by equation (5.4). The RED algorithm (Listing 5-1) may *not* drop packets in excess of $Threshold_{max}$, if there is buffer capacity to hold these packets. This is because *AverageQLen* could be lower than $Threshold_{max}$ if it has not yet caught up to the size of *MeasuredQLen*. The space between $Threshold_{max}$ and the queue capacity exists to accommodate this bursty traffic. The reason why an effort is made to accommodate bursty traffic is that often traffic that will affect the end user experience is bursty.

Problem 5.15 — Solution

Problem 5.16 — Solution

(a)

The first packet from *A* will not be marked to indicate congestion at any of the routers. Receiver *B* will send a regular ACK. The Sender *A* will receive this regular ACK and, given that it is in slow start, increment its CongWin by one and send two new packets.

Router *R*2 will drop the second packet. The third packet will arrive at Receiver *B* with CE codeword indicating congestion. *B* will send a duplicate ACK with the ECN-Echo flag set. When Sender *A* receives this dupACK with the ECN-Echo flag set, it reacts as if it just received 3× dupACKs and assumes that the oldest outstanding packet (#2) is lost.

Therefore, before *A* discovers packet loss, it will send two more packets: #5 and #6.

(b)

For simplicity we assume that Sender *A* uses ECT(0) in all packets to indicate ECN capable transport. The following table shows the ECN fields of packets that are leaving each of the nodes along the path to Receiver *B*.

Packet		from Sender A	from Router R1	from Router R2
#1	IP header ECN field:	ECT(0)	ECT(0)	ECT(0)
	TCP header ECN field:	--	--	--
#2	IP header ECN field:	ECT(0)	CE	{**dropped**}
	TCP header ECN field:	--	--	--
#3	IP header ECN field:	ECT(0)	CE	CE
	TCP header ECN field:	--	--	--
#4	IP header ECN field:	ECT(0)	CE	CE
	TCP header ECN field:	--	--	--

Receiver *B* will receive only the first packet from *A* in-order and generates a regular ACK. Packets #3 and #4will arrive out-of-order, because #2 was dropped on router *R*2. In response, *B* will generate two duplicate ACKs. Recall that "pure" ACK packets (with zero data payload) *must not* indicate ECN-Capability, i.e., they must use codepoint "00."

ACK		from Receiver B	from Router R1	from Router R2
#2	IP hdr ECN field:	Not-ECT	Not-ECT	Not-ECT
	TCP hdr ECN field:	--	--	--
dupACK	IP hdr ECN field:	Not-ECT	Not-ECT	Not-ECT
#2	TCP hdr ECN field:	ECN-Echo flag = 1	ECE = 1	ECE = 1
dupACK	IP hdr ECN field:	Not-ECT	Not-ECT	Not-ECT
#2	TCP hdr ECN field:	ECE = 1	ECE = 1	ECE = 1

The Sender *A* first receives a regular ACK and, given that it is in slow start, increments its CongWin by one and sends two new packets. After receiving the first duplicate ACK with the ECN-Echo flag set, the sender immediately reacts to congestion (without waiting for 3× dupACKs) and reduces the congestion window and slow-start threshold as if it has received 3× dupACKs. Recall also that retransmitted data packets *must not* indicate ECN-Capability, i.e., they must use codepoint "00."

Packet		from Sender A	from Router R1	from Router R2
#5	IP hdr ECN field:	ECT(0)	ECT(0)	ECT(0)
	TCP hdr ECN field:	--	--	--
#6	IP hdr ECN field:	ECT(0)	ECT(0)	ECT(0)
	TCP hdr ECN field:	--	--	--
retransmit	IP hdr ECN field:	Not-ECT	Not-ECT	Not-ECT
#2	TCP hdr ECN field:	--	--	--

Receiver *B* is still missing packet #2, so it will generate two more duplicate ACKs. Then, when it receives the retransmitted packet #2, it acknowledges all packets up to #6 and requests packet #7. Note that although packets #5, #6, and retransmitted #2 arrived without CE flag set, the receiver keeps setting the ECN-Echo (ECE) flag in all ACKs. The reason is that Explicit Congestion Notification is a *probabilistic mechanism*, so the receiver does not know whether the congestion subsided just because the subsequent packets do not carry the CE codepoint. Recall that the TCP receiver will keep setting the ECE flag in all ACKs until it receives a packet with the Congestion Window Reduced (CWR) flag from the TCP sender.

ACK		from Receiver B	from Router R1	from Router R2
dupACK	IP hdr ECN field:	Not-ECT	Not-ECT	Not-ECT
#2	TCP hdr ECN field:	ECE = 1	ECE = 1	ECE = 1
dupACK	IP hdr ECN field:	Not-ECT	Not-ECT	Not-ECT
#2	TCP hdr ECN field:	ECE = 1	ECE = 1	ECE = 1
#7	IP hdr ECN field:	Not-ECT	Not-ECT	Not-ECT
	TCP hdr ECN field:	--	--	--

When Sender *A* receives the third (and fourth) duplicate acknowledgement, it does not react as if it discovered packet loss, because it knows that it already reacted when it received the first (duplicate) ACK with the ECN-Echo flag set. The sender just does nothing for the duplicate ACKs (or it counts them to inflate the CongWin as prescribed by TCP Reno). Finally, when it receives a new ACK (asking for packet #7), it transmits packet #7 with the Congestion Window Reduced (CWR) flag set.

Packet		from Sender A	from Router R1	from Router R2

#7	IP hdr ECN field:
	TCP hdr ECN field:
		ECT(0)
		CWR = 1
			ECT(0)
			CWR = 1
				ECT(0)
				CWR = 1

...	...	...	...	...

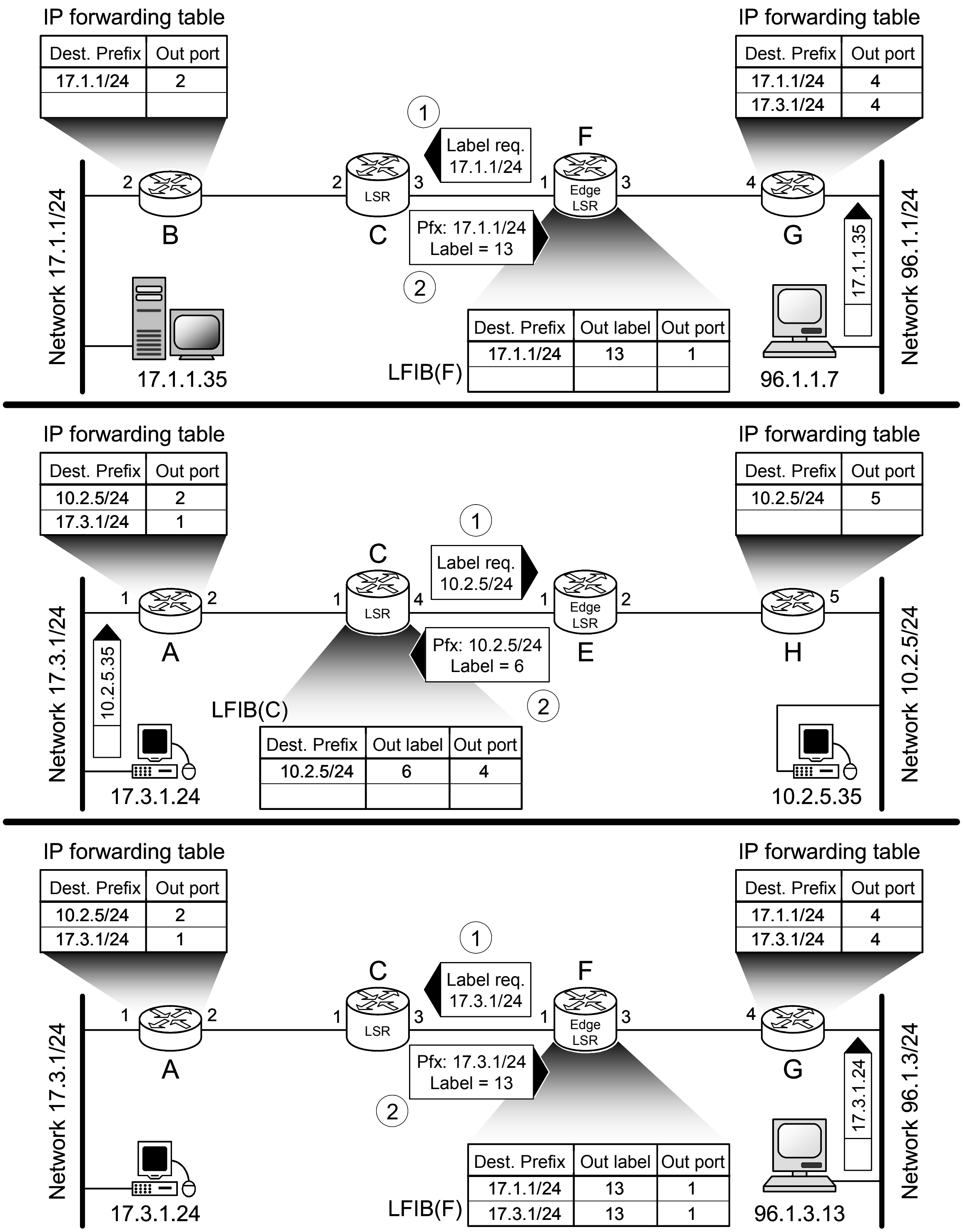
When Sender *B* receives packet #7 with the CWR flag set, it stops setting the ECE flag in the subsequent ACKs. If in the future it receives again a packet with the CE codepoint set in its IP header, it will resume sending ECE-flagged ACKs.

Problem 5.17 — Solution

Problem 5.18 — Solution

(a)

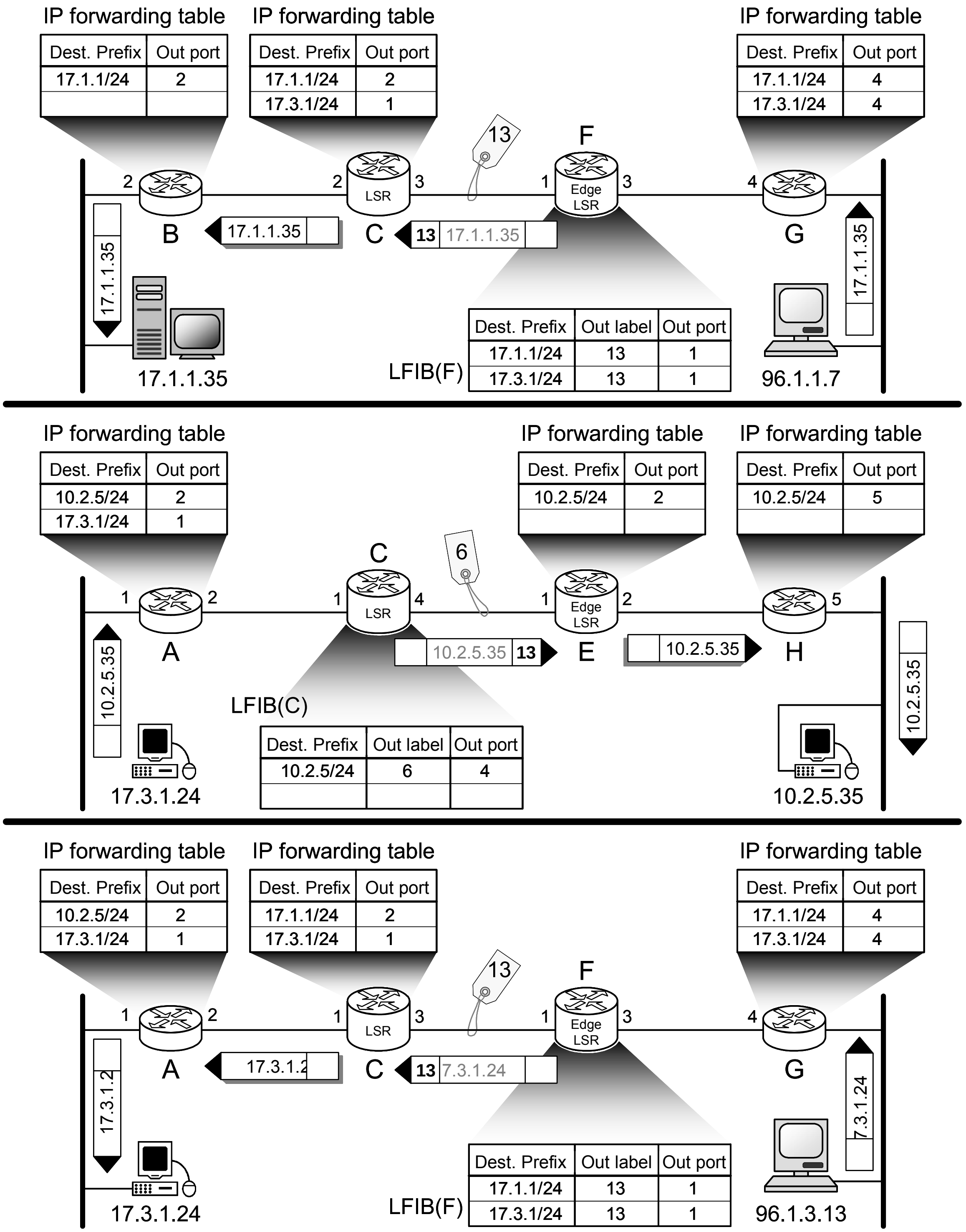
The solution is shown in the figure blow. Each LSR chooses the label numbers for itself. The only condition is that an LSR should *not* choose the label number that it is already using for another LSP. (The same label number can be used by other LSRs, though). Note that LSR *C* can decide that packets for destination `17.1.1/24` and `17.3.1/24` belong to the same FEC (forwarding equivalence class). The reason for this is that they traverse the same path across the MPLS domain (from LSR *F* to LSR *C*) and neither one is assigned preferential treatment. Therefore, in this case there is no reason to build more than a single LSP tunnel for the traffic from *F* to *C*.



(b)

Forwarding will work as illustrated in the figure below. LSR *C* uses its conventional IP forwarding table to forward packets to destinations `17.1.1.35` (first packet) and `17.3.1.24`

(3rd packet). Similarly, LSR *E* uses its IP forwarding table to forward packets to destination `10.2.5.35` (2nd packet).



IP forwarding table

Dest. Prefix	Out port
17.1.1/24	2

IP forwarding table

Dest. Prefix	Out port
17.1.1/24	2
17.3.1/24	1

IP forwarding table

Dest. Prefix	Out port
17.1.1/24	4
17.3.1/24	4

LFIB(F)

Dest. Prefix	Out label	Out port
17.1.1/24	13	1
17.3.1/24	13	1

IP forwarding table

Dest. Prefix	Out port
10.2.5/24	2
17.3.1/24	1

IP forwarding table

Dest. Prefix	Out port
10.2.5/24	2

IP forwarding table

Dest. Prefix	Out port
10.2.5/24	5

LFIB(C)

Dest. Prefix	Out label	Out port
10.2.5/24	6	4

IP forwarding table

Dest. Prefix	Out port
10.2.5/24	2
17.3.1/24	1

IP forwarding table

Dest. Prefix	Out port
17.1.1/24	2
17.3.1/24	1

IP forwarding table

Dest. Prefix	Out port
17.1.1/24	4
17.3.1/24	4

LFIB(F)

Dest. Prefix	Out label	Out port
17.1.1/24	13	1
17.3.1/24	13	1

(c)

The minimum number of FECs is 2 and the minimum number of LSP tunnels that need to be set up is accordingly 2: *F*→*C* and *C*→*E*.

Problem 5.19 — Solution

LSP-2: A → B → C ↘ E → G → J, D ↗

LSP-1: H → G → E → F

(a)

LSP-1: Label(*HG*)=1; Label(*GE*)=1; Label(*EF*)=1

LSP-2: Label(*AC*)=1; Label(*BC*)=2; Label(*CE*)=2; Label(*DE*)=3; Label(*EG*)=2; Label(*GJ*)=1

The selections of values for the MPLS labels are explained in part (b).

(b)

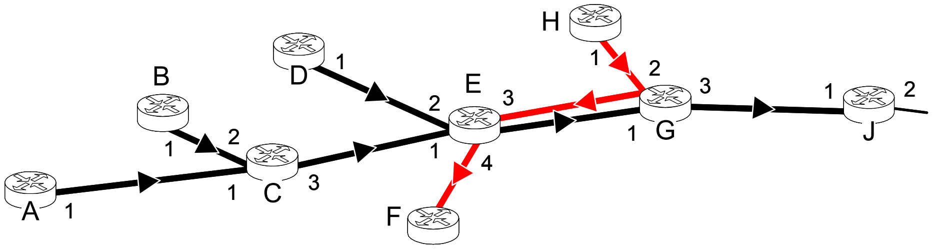
The constraints on MPLS label values in (a) include:

- Label value is selected as an arbitrary 20-bit number, which is true because all values are ≤3
- Label value must not be already used by the downstream router for another tunnel. The same label value can be used by different routers or by the same router on different outgoing ports.

To verify that the second constraint is satisfied, we recall that the link label is assigned by the downstream router. Routers *H* and *F* are downstream for a single flow each, so they can choose "1." Router *G* is downstream for *HG* and *EG*, so it must choose two different values. As shown in part (a), *G* assigns Label(*HG*)=1 and Label(*EG*)=2. Similarly, router *E* is downstream for *GE*, *CE*, and *DE*, so it must choose three different values. As shown in part (a), *E* assigns Label(*GE*)=1, Label(*CE*)=2, and Label(*DE*)=3. The remaining routers are left to the reader as exercise.

(c)

A possible labeling of the router port numbers is shown in the figure below.



Based on the MPLS label values from part (a), the LFIB tables of all LSRs are as follows:

LFIB(*A*)

Dest. prefix	Out label	Out port
192.168.2.1	1	1

LFIB(*B*)

Dest. prefix	Out label	Out port
192.168.2.1	2	1

LFIB(*C*)

In label	Out label	Out port
1	1	3
2	1	3

Question: do we need different input labels in LFIB(*C*) because the packets will be arriving on different incoming ports?

LFIB(*D*)

Dest. prefix	Out label	Out port

LFIB(*E*)

In label	Out label	Out port

LFIB(*F*)

Dest. prefix	Out label	Out port

192.168.2.1	1	1

1	1	4
2	2	3
3	2	3

--.--.--.--	1	3

The IP address of LSP-1 (connected to router *F*) is not specified, so we put “--.--.--.--” denoting an unspecified IP address.

LFIB(*H*)

Dest. prefix	Out label	Out port
--.--.--.--	1	1

LFIB(*G*)

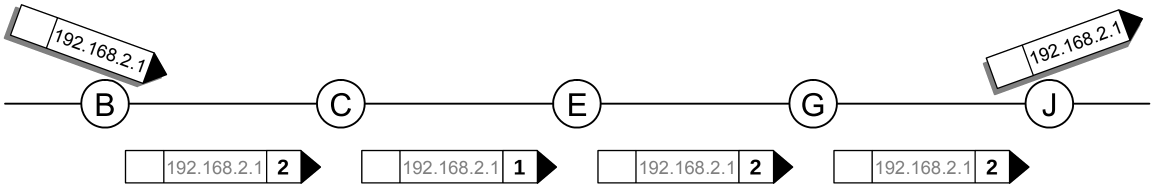
In label	Out label	Out port
1	1	1
2	2	3

LFIB(*J*)

Dest. prefix	In label	Out port
192.168.2.1	1	2

(d)

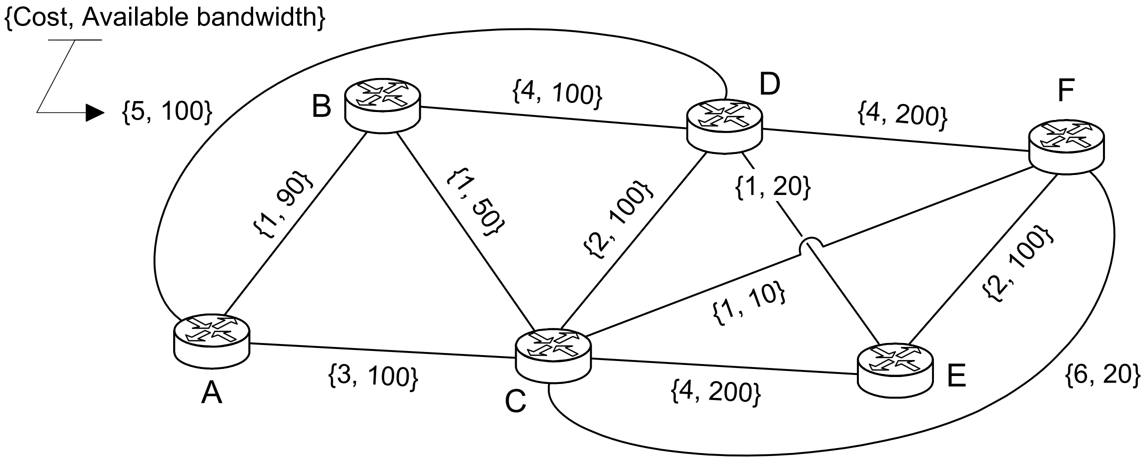
All tunnels are at a single hierarchical level, so each packet will carry a single MPLS label in its stack. For a packet sent to the server from a host on router *B*, the content of the packet’s label stack on links from *B* to *J* is as follows:



Problem 5.20 — Solution

Problem 5.21 — Solution

The network topology from the problem statement is copied here for reader’s reference:



The goal is to find the best path from router *A* to router *F*, such that the minimum bandwidth along this path is 20 Mbps. The CSPF computation proceeds as follows, and the reader should check additional notes after this table:

Step	Confirmed paths	Tentative set	Comments
0	(*A*, 0, –, N/A)	∅	*A* puts “self” on the Confirmed list.

1	(*A*, 0, –, N/A)	(*B*, 1, *B*, 90), (*C*, 3, *C*, 100), (*D*, 5, *D*, 100)	Put *A*'s neighbors in the Tentative(*A*) set.
2	(*A*, 0, –, N/A), (*B*, 1, *B*, 90)	(*C*, 2, *B*, 50), (*D*, 5, *D*, 100)	Move the neighbor with the shortest path length (*B*) to the Confirmed set, and put *B*'s neighbors in the Tentative set. (*C*, 2, *B*, 50) replaced (*C*, 3, *C*, 100) because of its shorter length.
3	(*A*, 0, –, N/A), (*B*, 1, *B*, 90), (*C*, 2, *B*, 50)	(*D*, 4, *B*, 50), (*E*, 6, *B*, 50), (*F*, 8, *B*, 20)	Move *C* to Confirmed and its neighbors (*B*, *D*, *E*, and *F*) to Tentative. *B* is already Confirmed and the path to *C* is via *B*, so be is omitted.
4	(*A*, 0, –, N/A), (*B*, 1, *B*, 90), (*C*, 2, *B*, 50), (*D*, 4, *B*, 50)	(*E*, 5, *B*, 20), (*F*, 8, *B*, 50)	Move *D* to Confirmed and its neighbors (*B*, *C*, *E*, and *F*) to Tentative. *B* and *C* are already in. Existing paths to *E* and *F* are replaced with better alternatives.
5	(*A*, 0, –, N/A), (*B*, 1, *B*, 90), (*C*, 2, *B*, 50), (*D*, 4, *B*, 50), (*E*, 5, *B*, 20)	(*F*, 7, *B*, 20)	Move *E* to Confirmed. Of its neighbors (*C* and *F*) only *F* is offered a better path.
6	(*A*, 0, –, N/A), (*B*, 1, *B*, 90), (*C*, 2, *B*, 50), (*D*, 4, *B*, 50), (*E*, 5, *B*, 20), **(*F*, 7, *B*, 20)**	∅	Move *F* to the Confirmed set. At this point, the tunnel endpoint (*F*) is in Confirmed, so we are done. (Incidentally, the Tentative set is also empty.) END.

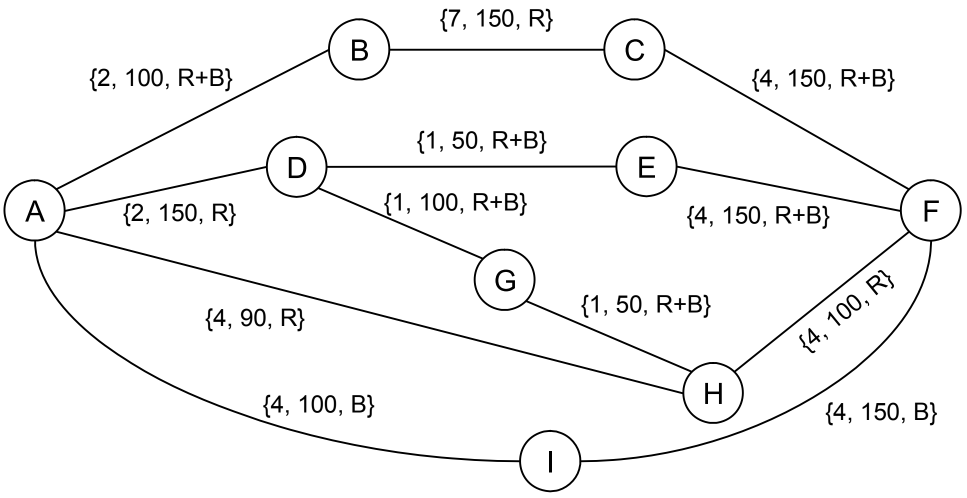
In Step 2, there are two alternative paths from *A* to *C*: *A*→*B*→*C* (length=2, bw=50 Mbps) and *A*→*C* (length=3, bw=100 Mbps); the former is selected because of its shorter length. There are also alternative paths from *A* to *D*: *A*→*B*→*D* (length=5, bw=90 Mbps) and *A*→*D* (length=5, bw=100 Mbps); both have the same length, but the latter is selected because of its larger minimum available bandwidth (and lower hop count). Here we see an instance of applying the CSPF tiebreakers.

In Step 3, there are two alternative paths from *A* to *D*: *A*→*B*→*C*→*D* (length=4, bw=50 Mbps) and *A*→*D* (length=5, bw=100 Mbps); the former is selected because of its shorter length. There are also two alternative links from *C* to *F*, one with cost=1 and bw=10 Mbps, and the other with cost=6 and bw=20 Mbps; the former is left out because it does not meet the minimum required bandwidth of 20 Mbps.

The reader should verify how the last three steps are computed. Finally, LSR *A* picks path A→B→C→D→E→F with the cost of 1+1+2+1+2= 7 and minimum bandwidth of Min{90, 50, 100, 20, 100} = 20 Mbps to build a tunnel to LSR *F*.

Problem 5.22 — Solution

The network topology from the problem statement is copied here for reader's reference:



The goal is to find the best path from router *A* to *F*, such that the minimum bandwidth along this path is 80 Mbps and all links include the color "RED" as their attribute. The CSPF computation proceeds as follows, and the reader should also check additional notes after this table:

Step	Confirmed paths	Tentative set	Comments
0	(*A*, 0, –, N/A, N/A)	∅	*A* puts "self" on the Confirmed list.
1	(*A*, 0, –, N/A, N/A)	(*B*, 2, *B*, 100, R+B), (*D*, 2, *D*, 150, R), (*H*, 4, *H*, 90, R), ~~(*I*, 4, *I*, 100, B)~~	Put *A*'s neighbors in the Tentative(*A*) set only if they satisfy constraints on bandwidth (≥80Mbps) and color ("RED").
2	(*A*, 0, –, N/A, N/A), (*D*, 2, *D*, 150, R)	(*B*, 2, *B*, 100, R+B), (*H*, 4, *H*, 90, R), ~~(*E*, 3, *D*, 50, R)~~, (*G*, 3, *D*, 100, R)	Move the neighbor with the shortest path length (*B* and *D*) and greatest bandwidth (*D*) to the Confirmed set, and put *D*'s neighbors in the Tentative set only if they satisfy constraints on bandwidth (≥80Mbps) and color ("RED").
3	(*A*, 0, –, N/A, N/A), (*D*, 2, *D*, 150, R), (*G*, 3, *D*, 100, R)	(*B*, 2, *B*, 100, R+B), (*H*, 4, *H*, 90, R), ~~(*H*, 4, *D*, 50, R)~~	Move *G* to Confirmed and its neighbor *H* to Tentative. However, it turns out that the path to *H* via *D* does not satisfy the bandwidth constraint. As a result, we remove *D* and *G* from Confirmed.
4	(*A*, 0, –, N/A, N/A), (*B*, 2, *B*, 100, R+B)	(*H*, 4, *H*, 90, R), (*C*, 9, *B*, 100, R)	Move *B* to Confirmed because it has a shorter path (and greater bandwidth) and its neighbor *C* to Tentative.
5	(*A*, 0, –, N/A, N/A), (*B*, 2, *B*, 100, R+B), (*C*, 9, *B*, 100, R)	(*F*, 13, *B*, 100, R)	Move *C* to Confirmed and its neighbor *F* to Tentative.
6	(*A*, 0, –, N/A, N/A), (*B*, 2, *B*, 100, R+B), (*C*, 9, *B*, 100, R), **(*F*, 13, *B*, 100, R)**	∅	Move *F* to the Confirmed set. At this point, the tunnel endpoint (*F*) is in Confirmed, so we are done. END.

Unfortunately, this is not the best path that *A* could have chosen, as seen in the next table.

The following table lists possible paths from router *A* to router *F* and their attributes:

Path	Routers in path	Path length	Min. bandwidth (Mbps)	Path links color
P1	A→B→C→F	2+7+4= 13	Min{100, 150, 150} = 100	{R+B, R, R+B}
P2	A→D→E→F	2+1+4= 7	Min{150, 50, 150} = 50	{R, R+B, R+B}

P3	A→D→G→H→F	2+1+1+4= 8	Min{150, 100, 50, 100} = 100	{R, R+B, R+B, R}
P4	A→H→F	4+4= 8	Min{90, 100} = 90	{R, R}
P5	A→H→G→D→E→F	4+1+1+1+4= 11	Min{150, 100, 100, 50, 150} = 50	{R, R+B, R+B, R+B, R+B}
P6	A→I→F	4+4= 8	Min{100, 150} = 100	{**B**, **B**}

Here is the decision process that illustrates the application of tiebreakers to select the best path:

- Eliminate the paths with minimum bandwidth <80 Mbps (~~P2, P3, and P5~~) because their minimum bandwidth is lower than the minimum required bandwidth of 80 Mbps;
  Remaining paths: P1, P4, and P6
- Eliminate the paths in which at least one link does not include the color attribute "RED" (~~P6~~);
  Remaining paths: P1, P4
- Select the path(s) with the shortest length and eliminate all greater length paths (~~P1~~);
  Remaining paths: P4

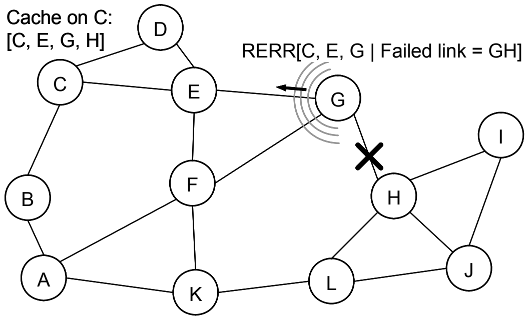
Therefore, LSR *A* should select path P4 as the "best" path to build a tunnel to LSR *F*. However, the actual CSPF computation must follow the steps in the first table and apply the tiebreakers only locally. As a result, it selected a "non-optimal" path P1.

Problem 6.1 — Solution

Problem 6.2 — Solution

(a)

If the next hop link is broken when a data packet is being forwarded, a RouteError (RERR) is generated and propagated backwards. RERR contains the failed link info, in our case link *GH*, and is *unicast* back to the data packet source node *C*.



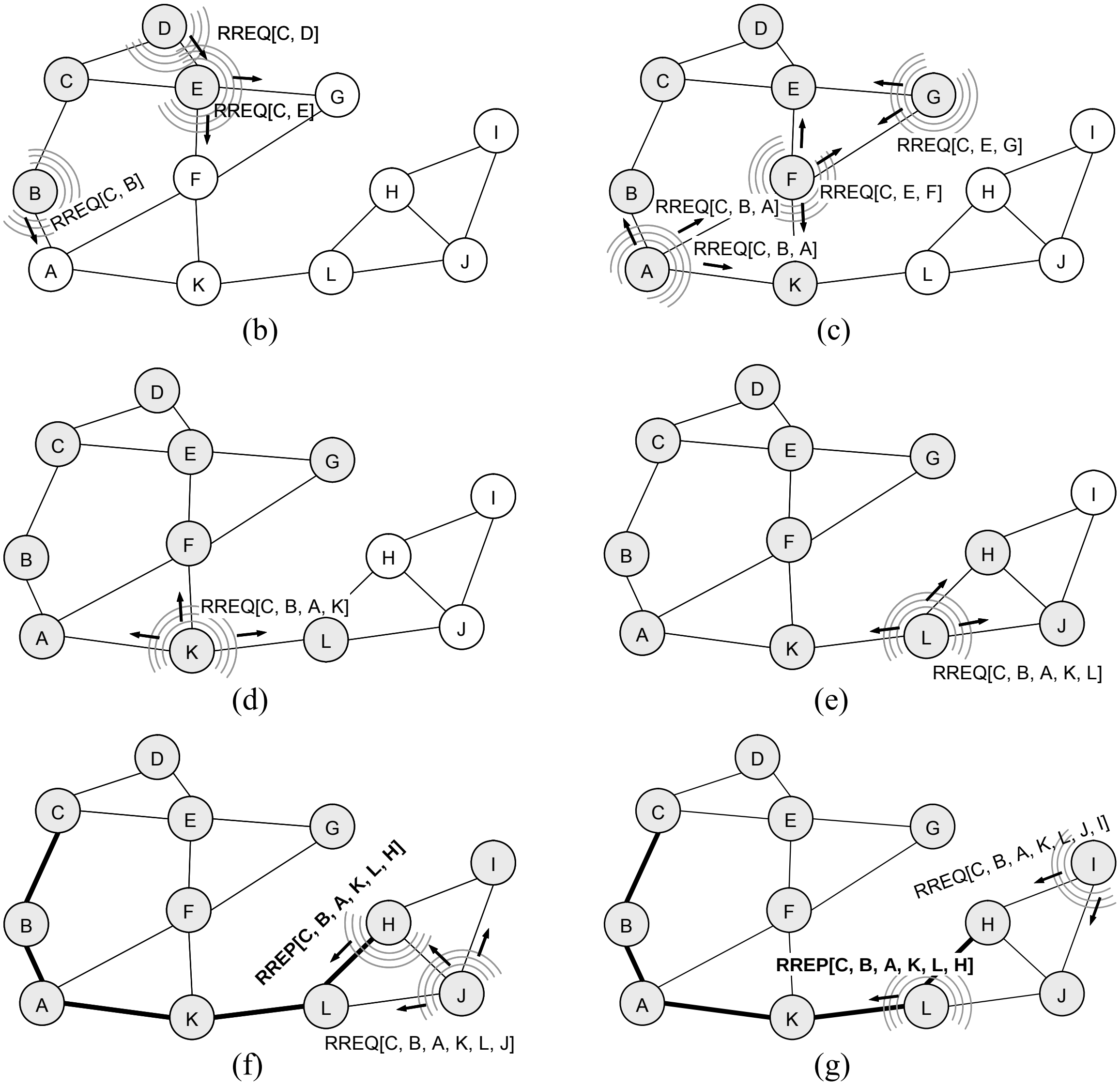
Therefore, the packets list is:

- from *G* to *E*: RERR[*C*, *E*, *G* | Failed link = *GH*]
- from *E* to *C*: RERR[*C*, *E*, *G* | Failed link = *GH*]

When *C* receives the RERR, it erases any cached route that contains the failed link.

(b)

Because *C* has more data to send to *H*, *C* will initiate a new round of route discovery. The first broadcast (from node *C*) is the same as in Figure 6-4(a) and is not shown here. Note also that we ignore the fact that a node cannot simultaneously listen to two or more transmissions, so below in figure (b) *D* and *E* cannot transmit simultaneously and in (c) *F* cannot listen to both *A* and *G*.



(b) (c)

(d) (e)

(f) (g)

In summary, the following messages will be sent:

1. Node *C* broadcasts to its neighbors *B*, *D*, and *E* packet RREQ[C]
2. Node *B* broadcasts to its neighbors *A* and *C* packet RREQ[C, B] figure (b)
3. Node *D* broadcasts to its neighbors *E* and *C* packet RREQ[C, D] figure (b)
4. Node *E* broadcasts to its neighbors *D*, *F*, *G*, and *C* packet RREQ[C, E] figure (b)
5. Node *A* broadcasts to its neighbors *B*, *F*, and *K* packet RREQ[C, B, A] figure (c)
6. Node *F* broadcasts to its neighbors *A*, *E*, *G*, and *K* packet RREQ[C, E, F] figure (c)
7. Node *G* broadcasts to its neighbors *E* and *F* packet RREQ[C, E, G] figure (c)
8. Node *K* broadcasts to its neighbors *A*, *F*, and *L* packet RREQ[C, B, A, K] figure (d)
9. Node *L* broadcasts to its neighbors *K*, *H*, and *J* packet RREQ[C, B, A, K, L] figure (e)
10. Node *H* **unicasts** to node *L* packet RREP[C, B, A, K, L, H] figure (f)

11. Node *J* broadcasts to its neighbors *H*, *I*, and *L* packet RREQ[C, B, A, K, L, J] fig. (f)
12. Node *L* **unicasts** to node *K* packet RREP[C, B, A, K, L, H] figure (g)
13. Node *I* broadcasts to its neighbors *H* and *J* packet RREQ[C, B, A, K, L, J, I] fig. (g)

In step 8, an alternative is that *K* first heard from *F* so it broadcasts to its neighbors *A*, *F*, and *L* packet RREQ[C, E, F, K]. In this case, the remaining steps would be different, so that, for example, in step 9 *L* would broadcast RREQ[C, E, F, K, L].

Three more steps are not shown in the above figure involving the unicasting of packet RREP[C, B, A, K, L, H] from *K* to *A*, then from *A* to *B*, and finally from *B* to *C*.

Therefore, there will be a total of 18 packets sent before *C* has found a new route to *H*.

Problem 6.3 — Solution

Problem 6.4 — Solution

Problem 6.5 — Solution

Problem 6.6 — Solution

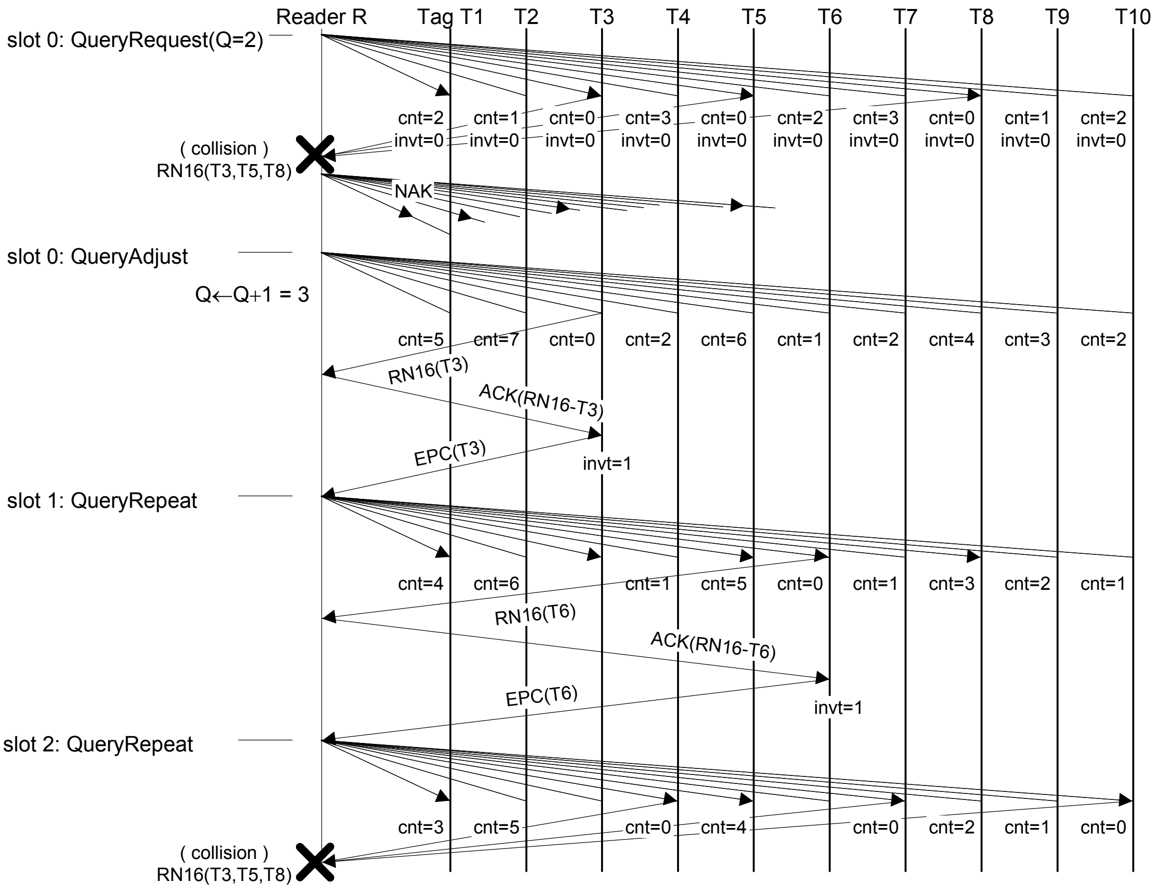
Problem 6.7 — Solution

(a)

First, we define the meaning of “optimum.” On one hand, we want to have *Q* small because then the tags will choose small random numbers and inventorying of all the tags will be completed quickly. On the other hand, larger *Q* means greater range of choices for random numbers and lower probability that two or more tags will select the same number and result in a collision. The value $Q = 4$ yields $2^Q = 16$ choices, which is the smallest range >10, so it is optimum. Smaller values, such as $Q = 3$ would probably soon result in collisions and the protocol would adjust *Q* to 4. It is conceivable that even $Q = 2$ would allow inventorying all ten tags. However, the probability of no-collision slots would be very small (see Problem 6.8 — Solution), so the expected inventorying interval would be very long. A scenario with successful readouts of 10 tags has an extremely low chance to happen under the Gen-2 standard. With probability close to 1, collision would happen and the protocol would adjust *Q* to higher values.

(b)

Here is a possible timeline of message exchanges between the reader and the tags.


Reader R
Tag T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
slot 0: QueryRequest(Q=2)
cnt=2 cnt=1 cnt=0 cnt=3 cnt=0 cnt=2 cnt=3 cnt=0 cnt=1 cnt=2
invt=0 invt=0 invt=0 invt=0 invt=0 invt=0 invt=0 invt=0 invt=0 invt=0
(collision)
RN16(T3,T5,T8)
NAK
slot 0: QueryAdjust
Q←Q+1 = 3
cnt=5 cnt=7 cnt=0 cnt=2 cnt=6 cnt=1 cnt=2 cnt=4 cnt=3 cnt=2
RN16(T3)
ACK(RN16-T3)
EPC(T3)
invt=1
slot 1: QueryRepeat
cnt=4 cnt=6 cnt=1 cnt=5 cnt=0 cnt=1 cnt=3 cnt=2 cnt=1
RN16(T6)
ACK(RN16-T6)
EPC(T6)
invt=1
slot 2: QueryRepeat
cnt=3 cnt=5 cnt=0 cnt=4 cnt=0 cnt=2 cnt=1 cnt=0
(collision)
RN16(T3,T5,T8)

The remaining sequence of packets could be something like this:

Reader →NAK→ all-tags
slot 0: Reader →QueryAdjust→ all-tags Q←Q□+1 = 4
[Count values: T1=1, T2=3, T4=0, T5=9, T7=15, T8=14, T9=10, T10=2]
T4 →RN16(T4)→ R | R →ACK(RN16-T4)→ T4 | T4 →EPC(T4) → R
slot 1: Reader →QueryRepeat→ all-tags
T1 →RN16(T1)→ R | R →ACK(RN16-T1)→ T1 | T1 →EPC(T1)→ R
slot 2: Reader →QueryRepeat→ all-tags
T10 →RN16(T10)→ R | R →ACK(RN16-T10)→ T10 | T10 →EPC(T10)→ R
slot 3: Reader →QueryRepeat→ all-tags
T2 →RN16(T2)→ R | R →ACK(RN16-T2)→ T2 | T2 →EPC(T2)→ R
•••
slot 9: Reader →QueryRepeat→ all-tags
T5 →RN16(T5)→ R | R →ACK(RN16-T5)→ T5 | T5 →EPC(T5)→ R
slot 10: Reader →QueryRepeat→ all-tags
T9 →RN16(T9)→ R | R →ACK(RN16-T9)→ T9 | T1 →EPC(T9)→ R
•••
slot 14: Reader →QueryRepeat→ all-tags
T8 →RN16(T8)→ R | R →ACK(RN16-T8)→ T8 | T8 →EPC(T8)→ R
slot 15: Reader →QueryRepeat→ all-tags
T7 →RN16(T7)→ R | R →ACK(RN16-T7)→ T7 | T7 →EPC(T7)→ R

The rules used to verify the plausibility of the above scenario would include the exact protocol specified by EPCglobal, Inc.—Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID. A list of key rules includes:

- Each tag selects its `count` uniformly randomly from the set {0, …, 2^Q−1}, which is initially {0, 1, 2, 3}. Because there are only four options but ten tags, at seven or more tags will make the same choice as at least one other tag. The maximum possible number of successful replies before a collision is ≤3. The minimum number of tags that will remain unread is ≥7.
- Because of overlapping `count` choices, there will be a collision and the reader will send a QueryAdjust message to increment the Q to 3.
- It is possible, although extremely unlikely, that ≤8 tags remained non-inventoried and with $Q = 3$ all select different `count` values and are successfully inventoried in this cycle. A more likely scenario, shown above, is that there will be another collision and the reader will send another QueryAdjust message to increment the Q to 4.
- With $Q = 4$, yielding 16 choices for `count` values and ≤10 tags remaining unread, it is likely that all tags will be inventoried in this cycle, as shown in the above scenario.

Problem 6.8 — Solution

There are 10 tags and in a slot each may choose to send or not to send. However, the options are not equally likely. The tag will choose to send with probability p_s=1/4 (if `count` = 0) or not to send with probability p_n=3/4 (if `count` ∈ {1, 2, 3}). We assume that the `count` is selected uniformly randomly from the set {0, …, 2^Q−1} = {0, 1, 2, 3}. Assuming that each tag makes a binary choice send/no-send, there are 2^{10} = 1024 possible events. Of these events, only ten have a single tag transmitting and the remaining nine staying quiet, which corresponds to "one of the ten tags responds with no collision in the first slot." We need to weigh the events by the probability of the binary choice for each tag. Because tags choose independently of each other, the probability is:

$$\text{Probability of no collision in the first slot} = 10\times p_s\cdot\underbrace{p_n\cdot\ldots\cdot p_n}_{9} = 10\times p_s\cdot p_n^9 = 10\times\frac{1}{4}\cdot\left(\frac{3}{4}\right)^9 \approx 0.188$$

Problem 7.1 — Solution

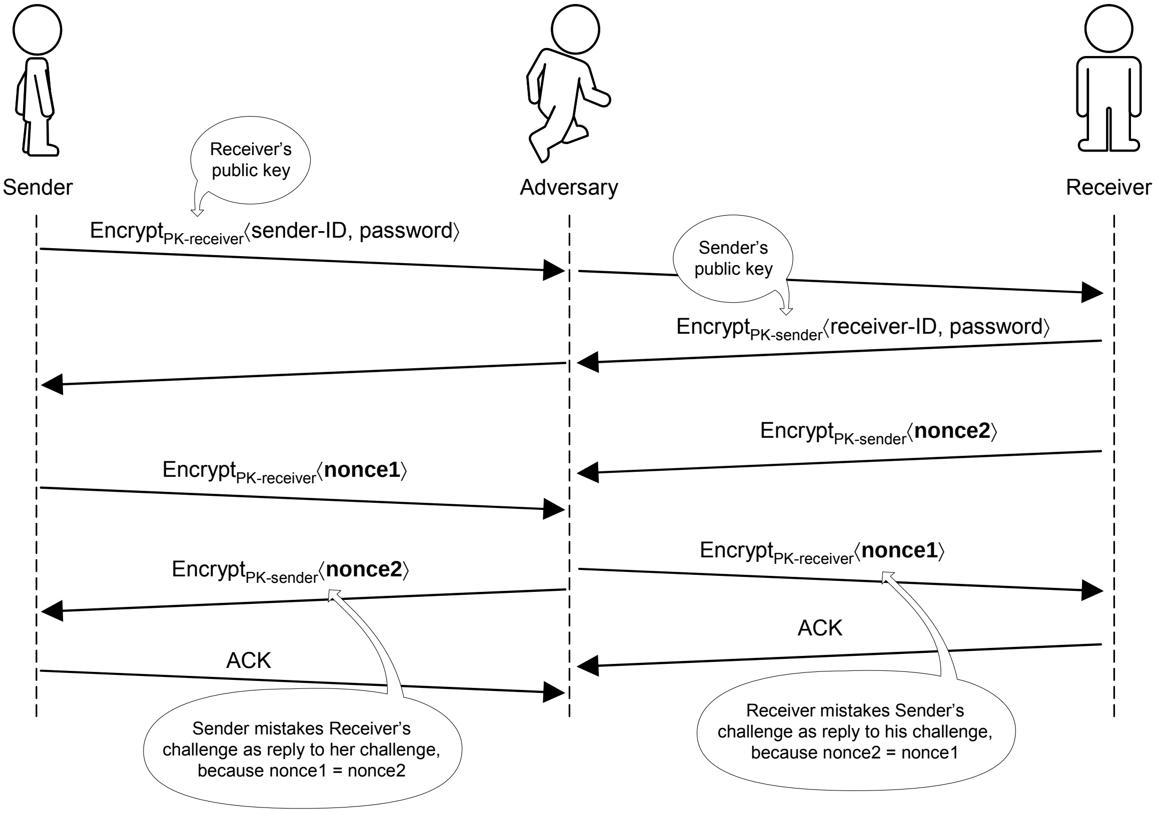
Problem 7.2 — Solution

Problem 7.3 — Solution

The problem occurs if both Sender and Receiver use that same nonce value, although different every time. This may occur if their algorithms for generating the nonces are not well designed and somehow become synchronized. In other words **nonce1 = nonce2**.

Visit http://www.gs1.org/epcglobal/standards for RFID Protocols Class-1 Generation-2

Then, because Sender's nonce value is identical to Receiver's nonce value, Adversary can use their mutual challenges as *responses* to the challenges, as shown in the figure below. Both Sender and Receiver are deceived to think that they have correctly authenticated each other, while Adversary can unnoticed forge any documents and send to either of the unsuspecting endpoints.


Sender
Adversary
Receiver
Receiver's public key
Encrypt$_{\text{PK-receiver}}$⟨sender-ID, password⟩
Sender's public key
Encrypt$_{\text{PK-sender}}$⟨receiver-ID, password⟩
Encrypt$_{\text{PK-sender}}$⟨**nonce2**⟩
Encrypt$_{\text{PK-receiver}}$⟨**nonce1**⟩
Encrypt$_{\text{PK-receiver}}$⟨**nonce1**⟩
Encrypt$_{\text{PK-sender}}$⟨**nonce2**⟩
ACK
ACK
Sender mistakes Receiver's challenge as reply to her challenge, because nonce1 = nonce2
Receiver mistakes Sender's challenge as reply to his challenge, because nonce2 = nonce1

Appendix A: Probability Refresher

Random Events and Their Probabilities

An **experiment** (or, **observation**) is a procedure that yields one of a given set of possible outcomes. **Outcome** is a specific result of an experiment. The **sample space** S of the experiment is the set of possible outcomes that can occur in an experiment. An **event** E is a collection of outcomes, which is a subset of the sample space. For example, tossing a coin results in one of two possible outcomes: heads (H) or tails (T). Tossing a coin twice results in one of four possible outcomes: HH (two heads), HT (a head followed by a tail), TH, or TT (see Figure A-1(a)). Similarly, tossing a coin three times results in one of eight possible outcomes: HHH, HHT, HTH, HTT, THH, THT, TTH, or TTT. Consider the experiment of tossing a coin twice, where we can define a number of possible events:

Event A: The event "two heads" consists of the single outcome A = {HH}. (This event is equivalent to the event "no tails.")

Event B: The event "exactly one head" consists of two outcomes B = {HT, TH}. (This event is equivalent to the event "exactly one tail.")

Event C: The event "at least one head" consists of three outcomes C = {HH, HT, TH}. (This event is equivalent to the events "at most one tail" and "not two tails.")

We also define a null event ("nothing happened"), which is symbolized by $\varnothing$. Given a sample space S, the total number of events that can be defined is the set that contains all of the subsets of S, including both $\varnothing$ and S itself. This set is called the *power set* of set S and is denoted $\mathbb{P}(S)$, or $\mathbb{P}S$, or 2^S. In the example of tossing a coin twice, the power set of S contains $2^4 = 16$ events, including the null event $\varnothing$. The events consisting of a single outcome are: {HH}, {HT}, {TH}, {TT}. The events consisting of pairs of outcomes are:
{HH, HT}, {HH, TH}, {HH, TT}, {HT, TH}, {HT, TT}, {TH, TT}.

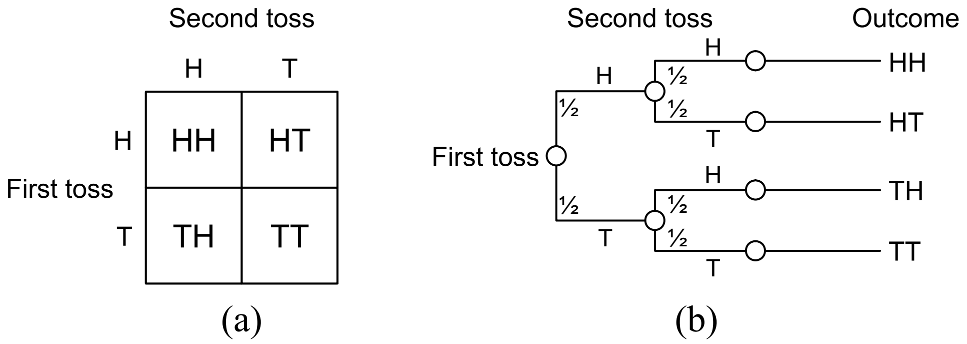


Figure A-1. (a) Possible outcomes of two coin tosses. (b) "Tree diagram" of possible outcomes of two coin tosses.

The events consisting of triples of outcomes are:
{HH, HT, TH}, {HH, HT, TT}, {HH, TH, TT}, {HT, TH, TT}.

Finally, the event consisting of all four outcomes is: {HH, HT, TH, TT}.

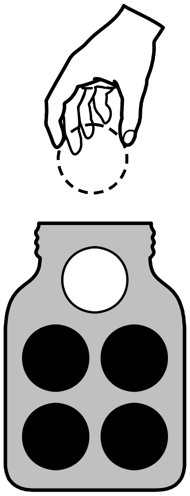


Figure A-2. Jar with black and white balls.

We say that an event is *random* when the result of an experiment is not known before the experiment is performed. For example, a coin toss is random because we do not know if it will land heads or tails. If we knew how it would land, then the event would not be random because its outcome would be predetermined. For a random event, the best we can do is *estimate* the probability of the event.

One way to define the probability of a random event is as the relative frequency of occurrence of an experiment's outcome, when repeating the experiment indefinitely. Consider a jar with five balls: four black and one white (Figure A-2). Imagine that you reach into the jar and retrieve a ball, examine its color, and put it back. If you repeat this experiment many times, then, on average, four out of five times you will retrieve a black ball and one out of five times you will retrieve the white ball. Therefore, the probability of an outcome could be defined as the frequency of the outcome.

We would like to know not only the probability of individual outcomes, but also the probability of events. Let us first consider the case when all outcomes are equally likely, and later we will consider the general case. In the case of equally likely outcomes, the probability of an event is equal to the number of outcomes in an event (cardinality of the event) divided by the number of possible outcomes (cardinality of the sample space): $p(E) = \frac{|E|}{|S|}$. For example, tossing a fair coin has two equally likely outcomes: heads and tails are equally likely on each toss and there is no relationship between the outcomes of two successive tosses. When a coin is tossed twice, the number of outcomes is four (Figure A-1(a)). The probabilities for the three events defined above are:

Event A: The probability of the event "two heads," A = {HH}, equals $p(A) = 1/4$.

Event B: The probability of the event "exactly one head," B = {HT, TH}, equals $p(B) = (1 + 1)/4 = 1/2$.

Event C: The probability of the event "at least one head," C = {HH, HT, TH}, equals $p(C) = (1 + 1 + 1)/4 = 3/4$.

The Principles of Probability Theory

Given a sample space S with finite number of elements (or, outcomes), we assume that a **probability** value $p(x)$ is attached to each element x in S, with the following properties

1. $0 \leq p(x) \leq 1$, for all $x \in S$
2. $\sum_{x \in S} p(x) = 1$

Given an event E, that is, a subset of S, the probability of E is defined as the sum of the probabilities of the outcomes in the event E

$$p(E) = \sum_{x \in E} p(x)$$

Therefore, the probability of the entire sample space is 1, and the probability of the null event is 0.

Events can also be combined. Given two events A and B, their **intersection** or **conjunction** is the event that consists of all outcomes common to both events, and is denoted as $\{A \ and \ B\}$ or $\{A \cap B\}$. For example, the event C defined above ("at least one head") consists of three outcomes C = {HH, HT, TH}. We can define another event, "at least one tail," which also consists of three outcomes D = {HT, TH, TT}. The conjunction of C and D ("at least one head and at least one tail") consists of the outcomes HT and TH. (Note that this event is equivalent to the event "one head and one tail.") Such an event is called a **joint event** (or, compound event) and the probability of such and event is called a **joint probability** (or, compound probability).

Another type of event combination involves outcomes of either of two events. The **union** or **disjunction** of two events consists of all the outcomes in either event, denoted as $\{A \ or \ B\}$ or $\{A \cup B\}$. For example, consider again the event "at least one head," C = {HH, HT, TH}, and the event "at least one tail," D = {HT, TH, TT}. The event "at least one head or at least one tail" consists of $\{A \ or \ B\}$ = {HH, HT, TH, TT}. This disjunction equals the entire sample space S, because there must be at least one head or at least one tail in any coin-toss experiment.

An important property of random events is *independence* of one another. When two events are independent, the occurrence of one of the events gives no information about the probability that the other event occurs. One event does not influence another, or stated formally, the events E and F are **independent** if and only if $p(E \ and \ F) = p(E) \cdot p(F)$.

To illustrate the above concepts, consider the following scenario of two containers with black and white balls (Figure A-3). First you decide randomly from which container to draw a ball from, and then you draw a ball from the selected vessel. To decide the vessel to draw from, you roll a die. If the die comes up 1 or 2, you draw a ball from the jar; otherwise, you draw from the urn (i.e., when the die comes up 3, 4, 5, or 6). Let us define the following events: VJ = {die roll outcomes: 1, 2}, VU = {die roll outcomes: 3, 4, 5, 6}, BB = {black ball taken}, BW = {white ball taken}.

A joint event can be defined as JB = {black ball taken from Jar}, which is a conjunction of $VJ \cap BB$. Another joint event is UW = {white ball taken from Urn}. One can observe that VJ and BB are *not* independent, because the occurrence of one of the events gives useful information about the probability that the other event occurs. That is, we know that the probability of taking a black ball is high when the die comes up 1 or 2, because the fraction of black balls is greater in the jar than in the urn.

Many problems are concerned with a numerical value associated with the outcome of an experiment. For example, we may be interested in the total number of packets that end up with errors when 100 packets are transmitted. To study problems of this type we introduce the concept of a random variable. A **random variable** is a function from the sample space of an experiment to the set of real numbers. That is, a random variable assigns a real number to each possible

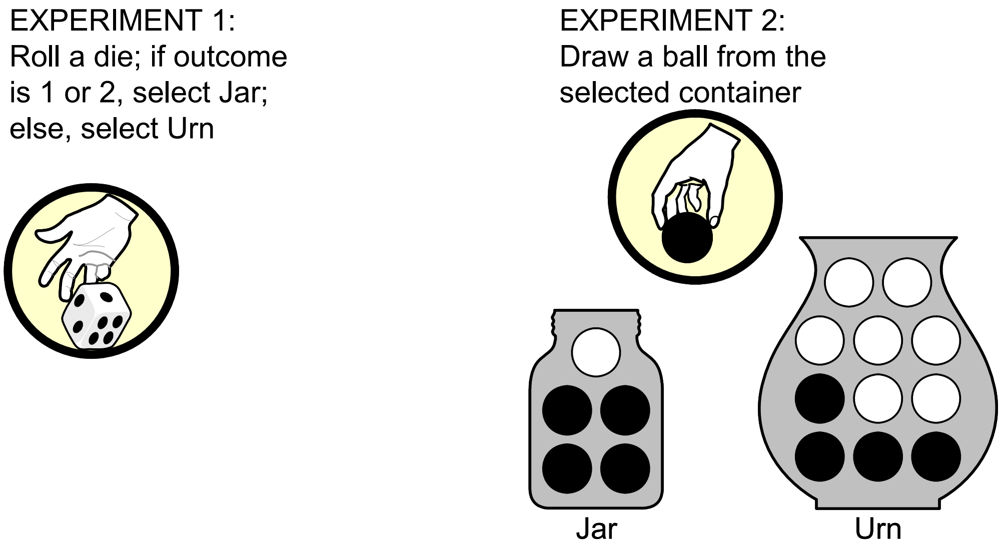

EXPERIMENT 1:
Roll a die; if outcome is 1 or 2, select Jar; else, select Urn
EXPERIMENT 2:
Draw a ball from the selected container
Jar
Urn

Figure A-3. Two experiments using a die, and a jar and urn with black and white balls.

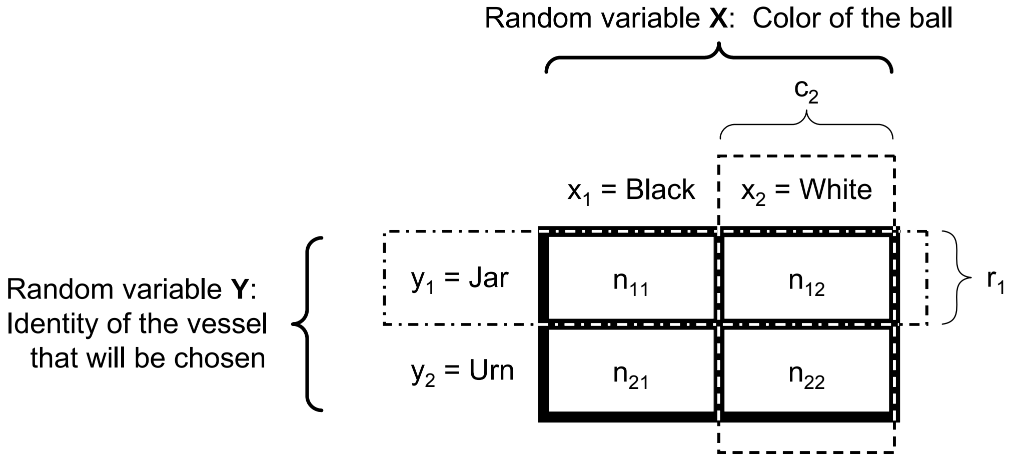

Random variable **X**: Color of the ball
c_2
x_1 = Black
x_2 = White
Random variable **Y**: Identity of the vessel that will be chosen
y_1 = Jar
y_2 = Urn
n_{11}
n_{12}
n_{21}
n_{22}
r_1

Figure A-4. Matrix of probabilities of random variables from Figure A-3.

outcome. (It is worth repeating that a random variable is a function; it is *not* a variable, and it is *not* random!)

In the example of Figure A-3, the identity of the vessel that will be chosen is a random variable, which we shall denote by Y. This random variable can take one of two possible values, namely *jar* or *urn*. Similarly, the color of the ball that will be drawn from the vessel is also a random variable and will be denoted by X. It can take either of the values *black* or *white*.

Consider the matrix representation in Figure A-4. Suppose that we run N times the experiments in Figure A-3 and record the outcomes in the matrix. Each cell of the matrix records the fraction of the total number of samples that turned out in the specific way. For example, n_{11} records the fraction of samples (out of the N total) where the selected vessel was *jar* and the color of the ball taken from the jar was *black*. The **joint probability** of the random variables X and Y is the probability that X will take the value x_i and at the same time Y will take the value y_j, which is written as $p(X = x_i, Y = y_j)$. For instance, in Figure A-4, example joint event is X = *black* AND

$Y = jar$. In the general case where variable X can take M different values, $X = x_i$, $i = 1, \ldots, M$, and variable Y can take L different values, $Y = y_j$, $j = 1, \ldots, L$, we can write the joint probability as

$$p\left(X = x_i, Y = y_j\right) = \frac{n_{ij}}{N} \tag{A.1}$$

(Of course, for this to hold, we are assuming that $N \to \infty$.) Similarly, the probability that X takes the value x_i (e.g., the probability that the ball color is *black*) regardless of the value of Y is written as $p(X = x_i)$ and is given by the fraction of the total number of points that fall in column i, so that

$$p\left(X = x_i\right) = \frac{c_i}{N} \tag{A.2}$$

The number of instances in column i in Figure A-4 is just the sum of the number of instances in each cell of that column. Therefore, we have $c_i = \sum_{j=1}^{L} n_{ij}$. If we plug this to equation (A.2) and then use (A.1), we will obtain

$$p(X = x_i) = \frac{\sum_{j=1}^{L} n_{ij}}{N} = \sum_{j=1}^{L} \frac{n_{ij}}{N} = \sum_{j=1}^{L} p\left(X = x_i, Y = y_j\right) \tag{A.3}$$

This is known as the **sum rule of probability**. The probability $p(X = x_i)$ is sometimes called the **marginal probability**, because it is obtained by marginalizing, or summing out, the other variables (in this case Y).

Let us fix the value of the random variable X so that $X = x_i$, and consider the fraction of such instances for which $Y = y_j$. In the example of Figure A-4, we could assume that X = *white* and consider the fraction of instances for which Y = *jar*. This is written as $p(Y = y_j \mid X = x_i)$ and is called **conditional probability** of $Y = y_j$ given $X = x_i$. It is obtained by finding the fraction of those points in column i that fall in cell i,j and is given as

$$p\left(Y = y_j \mid X = x_i\right) = \frac{n_{ij}}{c_i} \tag{A.4}$$

Starting with equation (A.1) and using equations (A.2) and (A.4), we can derive the following

$$\begin{aligned} p\left(X = x_i, Y = y_j\right) &= \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} \\ &= p\left(Y = y_j \mid X = x_i\right) \cdot p(X = x_i) \end{aligned} \tag{A.5}$$

This relationship is known as the **product rule of probability**.

Statistics of Random Variables

If X is a *discrete random variable*, define $S_X = \{x_1, x_2, \ldots, x_N\}$ as the *range* of X. That is, the value of X belongs to S_X.

Probability mass function (PMF): $P_X(x) = P[X = x]$

Properties of X with $P_X(x)$ and S_X:

a) $P_X(x) \geq 0 \qquad \forall x$

b) $\sum_{x \in S_X} P_X(x) = 1$

c) Given $B \subset S_X$, $P[B] = \sum_{x \in B} P_X(x)$

Define a and b as upper and lower bounds of X if X is a *continuous random variable*.

Cumulative distribution function (CDF): $F_X(x) = P[X \le x]$

Probability density function (PDF): $f_X(x) = \dfrac{dF(x)}{dx}$

Properties of X with PDF $f_X(x)$:

a) $f_X(x) \ge 0 \qquad \forall\ x$

b) $F_X(x) = \int_{-\infty}^{x} f_X(u) \cdot du$

c) $\int_{-\infty}^{\infty} f_X(x) \cdot dx = 1$

Expected value:

The *mean* or first moment.

Continuous RV case: $E[X] = \mu_X = \int_a^b x \cdot f_X(x) \cdot dx$

Discrete RV case: $E[X] = \mu_X = \sum_{k=1}^{N} x_k \cdot P_X(x_k)$

Variance:

Second moment minus first moment-squared: $Var[X] = E\left[(X - \mu_X)^2\right] = E\left[X^2\right] - \mu_X^2$

Continuous RV case: $E\left[X^2\right] = \int_a^b x^2 \cdot f_X(x) \cdot dx$

Discrete RV case: $E\left[X^2\right] = \sum_{k=1}^{N} x_k^2 \cdot P_X(x_k)$

Standard Deviation: $\sigma_X = \sqrt{Var[X]}$

Bayes' Theorem

Consider again the scenario of two containers with black and white balls (Figure A-3), but now modified as in Figure A-5. First, you decide randomly from which container to draw a ball, then you draw a ball from the selected container, and finally you report the ball color to your friend. To decide the container to draw from, you roll a die. If the die comes up 1 or 2, you draw a ball

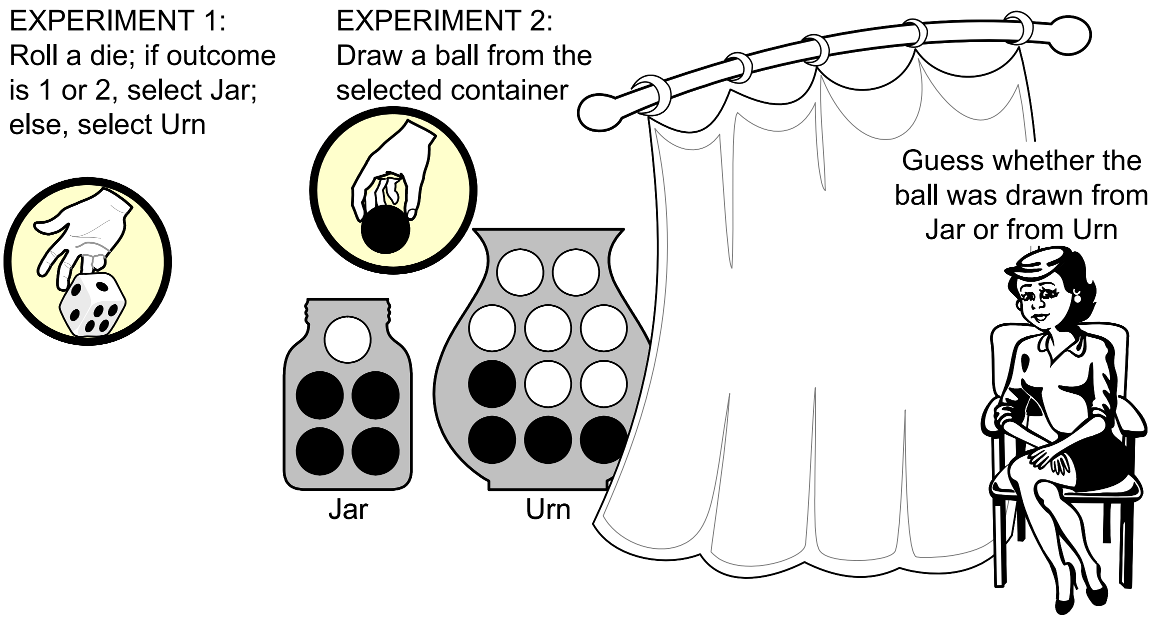

EXPERIMENT 1:
Roll a die; if outcome is 1 or 2, select Jar; else, select Urn
EXPERIMENT 2:
Draw a ball from the selected container
Guess whether the ball was drawn from Jar or from Urn
Jar
Urn

Figure A-5. Two experiments use a die, and a jar and urn with black and white balls. The person behind a curtain is trying to guess from which vessel a ball is drawn, given the color of the ball.

from the jar; otherwise, you draw from the urn (i.e., when the die comes up 3, 4, 5, or 6). Your friend is sitting behind a curtain and cannot observe which container the ball was drawn from. Your friend needs to answer a question such as: "given that a black ball was drawn, is it more likely that the ball was drawn from the jar or urn?". In other words, given that a black ball was drawn, what is the probability that it came from the jar?

On one hand, we know that the fraction of black balls is greater in the jar than in the urn. On the other hand, we know that as the result of the first experiment (the roll of the die), it is twice as likely that you have drawn the ball from the urn. How can we combine the evidence from our sample (black drawing outcome) with our prior belief based on the roll of the die? To help answer this question, consider again the representation in Figure A-4.

$$p(Y \mid X) = \frac{p(X \mid Y) \cdot p(Y)}{p(X)} \tag{A.7}$$

Suppose that an experiment can have only two possible outcomes. For example, when a coin is flipped, the possible outcomes are heads and tails. Each performance of an experiment with two possible outcomes is called a **Bernoulli trial**, after the Swiss mathematician James Bernoulli (1654-1705). In general, a possible outcome of a Bernoulli trial is called a *success* or a *failure*. If p is the probability of a success and q is the probability of a failure, it follows that $p + q = 1$.

Many problems can be solved by determining the probability of k successes when an experiment consists of n mutually independent Bernoulli trials. (Bernoulli trials are **mutually independent** if the conditional probability of success on any given trial is p, given any information whatsoever about the outcomes of other trials.)

The probability of exactly k successes in n independent Bernoulli trials, with probability of success p and probability of failure $q = 1 - p$, is $b(k; n, p) = C(n, k) \cdot p^k \cdot q^{n-k}$. When $b(k; n, p)$ is considered as a function of k, we call this function the **binomial distribution**.

Random Processes

A *process* is a naturally occurring or designed sequence of operations or events, possibly taking up time, space, expertise or other resource, which produces some outcome. A process may be identified by the changes it creates in the properties of one or more objects under its influence.

A function may be thought of as a computer program or mechanical device that takes the characteristics of its input and produces output with its own characteristics. Every process may be defined functionally and every process may be defined as one or more functions.

An example random process that will appear later in the text is Poisson process. It is usually employed to model arrivals of people or physical events as occurring at random points in time. *Poisson process* is a counting process for which the times between successive events are *independent and identically distributed* (IID) exponential random variables. For a Poisson process, the number of arrivals in any interval of length τ is Poisson distributed with a parameter $\lambda \cdot \tau$. That is, for all t, $\tau > 0$,

$$P\{A(t+\tau) - A(t) = n\} = e^{-\lambda\tau} \frac{(\lambda\tau)^n}{n!}, \qquad n = 0,1,... \tag{A.8}$$

The average number of arrivals within an interval of length τ is $\lambda\tau$ (based on the mean of the Poisson distribution). This implies that we can view the parameter λ as an arrival rate (average number of arrivals per unit time). If X represents the time between two arrivals, then $P(X > x)$, that is, the probability that the interarrival time is longer than x, is given by $e^{-x/\lambda}$. An interesting property of this process is that it is *memoryless*: the fact that a certain time has elapsed since the last arrival gives us no indication about how much longer we must wait before the next event arrives. An example of the Poisson distribution is shown in Figure A-6.

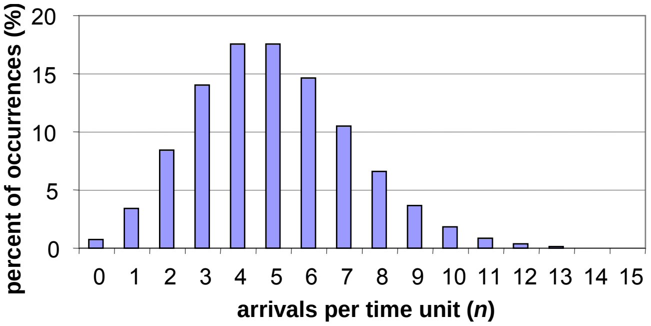


Figure A-6. The histogram of the number of arrivals per unit of time (τ = 1) for a Poisson process with average arrival rate λ = 5.

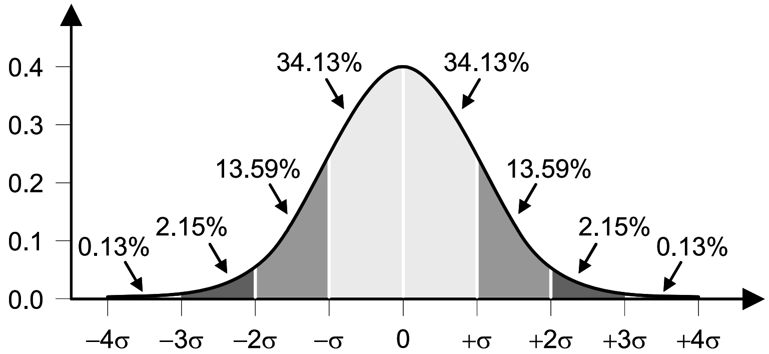


Figure A-7. Areas between selected points under the normal curve.

This model is not entirely realistic for many types of sessions and there is a great amount of literature which shows that it fails particularly at modeling the LAN traffic. However, such simple models provide insight into major tradeoffs involved in network design, and these tradeoffs are often obscured in more realistic and complex models.

Markov process is a random process with the property that the probabilities of occurrence of the various possible outputs depend upon one or more of the preceding outputs.

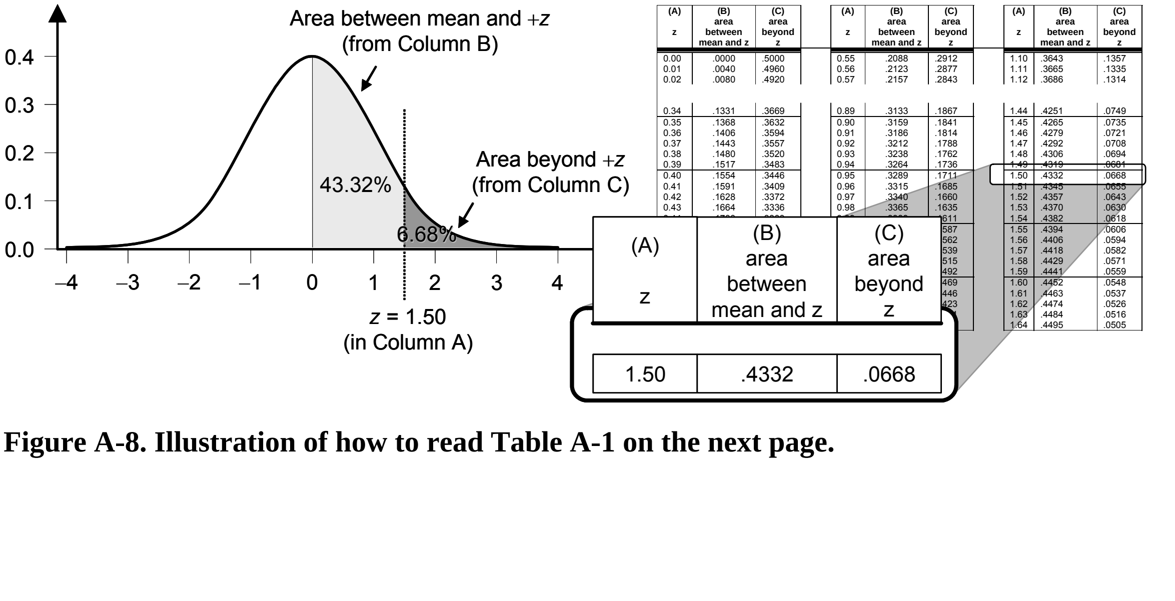
Statistics Review

Proportions of Area Under the Normal Curve

Figure A-7 shows a coarse partition of areas under the normal curve $N(\mu,\sigma)$. Statistical tables are often used to obtain finer partitioning, as shown in Table A-1. To use this table, it is necessary to convert the raw magnitude to a so-called *z*-score. The *z*-score is a standard deviate that allows for using the standard normal distribution $N(0,1)$, the one which has a mean μ = 0.0, a standard deviation σ = 1.0, and a total area under the curve equal to 1.0.

The values in Table A-1 represent the proportion of area under the standard normal curve. The table contains *z*-scores between 0.0 and 4.00 (i.e., four standard deviations, or 4×σ), with 0.01 increments. Because the normal distribution is symmetrical, the table represents *z*-scores ranging between –4.00 and 4.00.

Figure A-8 illustrates how to read Table A-1. Suppose you want to know how big is the area under the normal curve from the mean up to 1.5×σ, i.e., $z = 1.50$ and how much remains beyond. First, we look up Column A and find $z = 1.50$. Second, we read the associated values in Columns B and C, which represent the area between mean and *z*, and the area beyond *z*, respectively.



(A) z	(B) area between mean and z	(C) area beyond z	(A) z	(B) area between mean and z	(C) area beyond z	(A) z	(B) area between mean and z	(C) area beyond z
0.00	.0000	.5000	0.55	.2088	.2912	1.10	.3643	.1357
0.01	.0040	.4960	0.56	.2123	.2877	1.11	.3665	.1335
0.02	.0080	.4920	0.57	.2157	.2843	1.12	.3686	.1314
0.34	.1331	.3669	0.89	.3133	.1867	1.44	.4251	.0749
0.35	.1368	.3632	0.90	.3159	.1841	1.45	.4265	.0735
0.36	.1406	.3594	0.91	.3186	.1814	1.46	.4279	.0721
0.37	.1443	.3557	0.92	.3212	.1788	1.47	.4292	.0708
0.38	.1480	.3520	0.93	.3238	.1762	1.48	.4306	.0694
0.39	.1517	.3483	0.94	.3264	.1736	1.49	.4319	.0681
0.40	.1554	.3446	0.95	.3289	.1711	1.50	.4332	.0668
0.41	.1591	.3409	0.96	.3315	.1685	1.51	.4345	.0655
0.42	.1628	.3372	0.97	.3340	.1660	1.52	.4357	.0643
0.43	.1664	.3336	0.98	.3365	.1635	1.53	.4370	.0630
					1611	1.54	.4382	.0618
					587	1.55	.4394	.0606
					562	1.56	.4406	.0594
					539	1.57	.4418	.0582
					515	1.58	.4429	.0571
					492	1.59	.4441	.0559
					469	1.60	.4452	.0548
					446	1.61	.4463	.0537
					423	1.62	.4474	.0526
						1.63	.4484	.0516
						1.64	.4495	.0505

(A) z	(B) area between mean and z	(C) area beyond z
1.50	.4332	.0668

Figure A-8. Illustration of how to read Table A-1 on the next page.

Table A-1: Proportions of area under the normal curve. (*continued below*)

(A)	(B)	(C)	(A)	(B)	(C)	(A)	(B)	(C)
	area	area		area	area		area	area
z	between	beyond	z	between	beyond	z	between	beyond
	mean and z	z		mean and z	z		mean and z	z
0.00	.0000	.5000	0.55	.2088	.2912	1.10	.3643	.1357
0.01	.0040	.4960	0.56	.2123	.2877	1.11	.3665	.1335
0.02	.0080	.4920	0.57	.2157	.2843	1.12	.3686	.1314
0.03	.0120	.4880	0.58	.2109	.2810	1.13	.3708	.1292
0.04	.0160	.4840	0.59	.2224	.2776	1.14	.3729	.1271
0.05	.0199	.4801	0.60	.2257	.2743	1.15	.3749	.1251
0.06	.0239	.4761	0.61	.2291	.2709	1.16	.3770	.1230
0.07	.0279	.4721	0.62	.2324	.2676	1.17	.3790	.1210
0.08	.0319	.4681	0.63	.2357	.2643	1.18	.3810	.1190
0.09	.0359	.4641	0.64	.2389	.2611	1.19	.3830	.1170
0.10	.0398	.4602	0.65	.2422	.2578	1.20	.3849	.1151
0.11	.0438	.4562	0.66	.2454	.2564	1.21	.3869	.1131
0.12	.0478	.4522	0.67	.2486	.2514	1.22	.3888	.1112
0.13	.0517	.4483	0.68	.2517	.2483	1.23	.3907	.1093
0.14	.0557	.4443	0.69	.2549	.2451	1.24	.3925	.1075
0.15	.0596	.4404	0.70	.2580	.2420	1.25	.3944	.1056
0.16	.0636	.4364	0.71	.2611	.2389	1.26	.3962	.1038
0.17	.0675	.4325	0.72	.2642	.2358	1.27	.3980	.1020
0.18	.0714	.4286	0.73	.2673	.2327	1.28	.3997	.1003
0.19	.0753	.4247	0.74	.2704	.2296	1.29	.4015	.0985
0.20	.0793	.4207	0.75	.2734	.2266	1.30	.4032	.0968
0.21	.0832	.4168	0.76	.2764	.2236	1.31	.4049	.0951
0.22	.0871	.4129	0.77	.2794	.2206	1.32	.4066	.0934
0.23	.0910	.4090	0.78	.2823	.2177	1.33	.4082	.0918
0.24	.0948	.4052	0.79	.2852	.2148	1.34	.4099	.0901
0.25	.0987	.4013	0.80	.2881	.2119	1.35	.4115	.0885
0.26	.1026	.3974	0.81	.2910	.2090	1.36	.4131	.0869
0.27	.1064	.3936	0.82	.2939	.2061	1.37	.4147	.0853
0.28	.1103	.3897	0.83	.2967	.2033	1.38	.4162	.0838
0.29	.1141	.3859	0.84	.2995	.2005	1.39	.4177	.0823
0.30	.1179	.3821	0.85	.3023	.1977	1.40	.4192	.0808
0.31	.1217	.3783	0.86	.3051	.1949	1.41	.4207	.0793
0.32	.1255	.3745	0.87	.3078	.1922	1.42	.4222	.0778
0.33	.1293	.3707	0.88	.3106	.1894	1.43	.4236	.0764
0.34	.1331	.3669	0.89	.3133	.1867	1.44	.4251	.0749
0.35	.1368	.3632	0.90	.3159	.1841	1.45	.4265	.0735
0.36	.1406	.3594	0.91	.3186	.1814	1.46	.4279	.0721
0.37	.1443	.3557	0.92	.3212	.1788	1.47	.4292	.0708
0.38	.1480	.3520	0.93	.3238	.1762	1.48	.4306	.0694
0.39	.1517	.3483	0.94	.3264	.1736	1.49	.4319	.0681
0.40	.1554	.3446	0.95	.3289	.1711	1.50	.4332	.0668
0.41	.1591	.3409	0.96	.3315	.1685	1.51	.4345	.0655
0.42	.1628	.3372	0.97	.3340	.1660	1.52	.4357	.0643
0.43	.1664	.3336	0.98	.3365	.1635	1.53	.4370	.0630
0.44	.1700	.3300	0.99	.3389	.1611	1.54	.4382	.0618
0.45	.1736	.3264	1.00	.3413	.1587	1.55	.4394	.0606
0.46	.1772	.3228	1.01	.3438	.1562	1.56	.4406	.0594
0.47	.1808	.3192	1.02	.3461	.1539	1.57	.4418	.0582
0.48	.1844	.3156	1.03	.3485	.1515	1.58	.4429	.0571
0.49	.1879	.3121	1.04	.3508	.1492	1.59	.4441	.0559
0.50	.1915	.3085	1.05	.3531	.1469	1.60	.4452	.0548
0.51	.1950	.3050	1.06	.3554	.1446	1.61	.4463	.0537
0.52	.1985	.3015	1.07	.3577	.1423	1.62	.4474	.0526
0.53	.2019	.2981	1.08	.3599	.1401	1.63	.4484	.0516
0.54	.2054	.2946	1.09	.3621	.1379	1.64	.4495	.0505

Table A-1 (*continued*)

(A) z	(B) area between mean and z	(C) area beyond z
1.65	.4505	.0495
1.66	.4515	.0485
1.67	.4525	.0475
1.68	.4535	.0465
1.69	.4545	.0455
1.70	.4554	.0446
1.71	.4564	.0436
1.72	.4573	.0427
1.73	.4582	.0418
1.74	.4591	.0409
1.75	.4599	.0401
1.76	.4608	.0392
1.77	.4616	.0384
1.78	.4625	.0375
1.79	.4633	.0367
1.80	.4641	.0359
1.81	.4649	.0351
1.82	.4656	.0344
1.83	.4664	.0336
1.84	.4671	.0329
1.85	.4678	.0322
1.86	.4686	.0314
1.87	.4693	.0307
1.88	.4699	.0301
1.89	.4706	.0294
1.90	.4713	.0287
1.91	.4719	.0281
1.92	.4726	.0274
1.93	.4732	.0268
1.94	.4738	.0262
1.95	.4744	.0256
1.96	.4750	.0250
1.97	.4756	.0244
1.98	.4761	.0239
1.99	.4767	.0233
2.00	.4772	.0228
2.01	.4778	.0222
2.02	.4783	.0217
2.03	.4788	.0212
2.04	.4793	.0207
2.05	.4798	.0202
2.06	.4803	.0197
2.07	.4808	.0192
2.08	.4812	.0188
2.09	.4817	.0183
2.10	.4821	.0179
2.11	.4826	.0174
2.12	.4830	.0170
2.13	.4834	.0166
2.14	.4838	.0162
2.15	.4842	.0158
2.16	.4846	.0154
2.17	.4850	.0150
2.18	.4854	.0146
2.19	.4857	.0143
2.20	.4861	.0139
2.21	.4864	.0136
2.22	.4868	.0132
2.23	.4871	.0129
2.24	.4875	.0125
2.25	.4878	.0122
2.26	.4881	.0119
2.27	.4884	.0116
2.28	.4887	.0113
2.29	.4890	.0110
2.30	.4893	.0107
2.31	.4896	.0104
2.32	.4898	.0102
2.33	.4901	.0099
2.34	.4904	.0096
2.35	.4906	.0094
2.36	.4909	.0091
2.37	.4911	.0089
2.38	.4913	.0087
2.39	.4916	.0084
2.40	.4918	.0082
2.41	.4920	.0080
2.42	.4922	.0078
2.43	.4925	.0075
2.44	.4927	.0073
2.45	.4929	.0071
2.46	.4931	.0069
2.47	.4932	.0068
2.48	.4934	.0066
2.49	.4936	.0064
2.50	.4938	.0062
2.51	.4940	.0060
2.52	.4941	.0059
2.53	.4943	.0057
2.54	.4945	.0055
2.55	.4946	.0054
2.56	.4948	.0052
2.57	.4949	.0051
2.58	.4951	.0049
2.59	.4952	.0048
2.60	.4953	.0047
2.61	.4955	.0045
2.62	.4956	.0044
2.63	.4957	.0043
2.64	.4959	.0041
2.65	.4960	.0040
2.66	.4961	.0039
2.67	.4962	.0038
2.68	.4963	.0037
2.69	.4964	.0036
2.70	.4965	.0035
2.71	.4966	.0034
2.72	.4967	.0033
2.73	.4968	.0032
2.74	.4969	.0031
2.75	.4970	.0030
2.76	.4971	.0029
2.77	.4972	.0028
2.78	.4973	.0027
2.79	.4974	.0026
2.80	.4974	.0026
2.81	.4975	.0025
2.82	.4976	.0024
2.83	.4977	.0023
2.84	.4977	.0023
2.85	.4978	.0022
2.86	.4979	.0021
2.87	.4979	.0021
2.88	.4980	.0020
2.89	.4981	.0019
2.90	.4981	.0019
2.91	.4982	.0018
2.92	.4982	.0018
2.93	.4983	.0017
2.94	.4984	.0016
2.95	.4984	.0016
2.96	.4985	.0015
2.97	.4985	.0015
2.98	.4986	.0014
2.99	.4986	.0014
3.00	.4987	.0013
3.01	.4987	.0013
3.02	.4987	.0013
3.03	.4988	.0012
3.04	.4988	.0012
3.05	.4989	.0011
3.06	.4989	.0011
3.07	.4989	.0011
3.08	.4990	.0010
3.09	.4990	.0010
3.10	.4990	.0010
3.11	.4991	.0009
3.12	.4991	.0009
3.13	.4991	.0009
3.14	.4992	.0008
3.15	.4992	.0008
3.16	.4992	.0008
3.17	.4992	.0008
3.18	.4993	.0007
3.19	.4993	.0007
3.20	.4993	.0007
3.21	.4993	.0007
3.22	.4994	.0006
3.23	.4994	.0006
3.24	.4994	.0006
3.25	.4994	.0006
3.30	.4995	.0005
3.35	.4996	.0004
3.40	.4997	.0003
3.45	.4997	.0003
3.50	.4998	.0002
3.60	.4998	.0002
3.70	.4999	.0001
3.80	.4999	.0001
3.90	.49995	.00005
4.00	.49997	.00003

References

1. I. Aad and C. Castelluccia, “Differentiation mechanisms for IEEE 802.11,” *Proceedings of the IEEE Infocom 2001*, April 2001.
2. M. Allman, V. Paxson, and E. Blanton, “TCP congestion control,” IETF Request for Comments RFC-5681, September 2009. Online at: http://tools.ietf.org/html/rfc5681
3. M. Allman, V. Paxson, and W.R. Stevens, “TCP congestion control,” IETF Request for Comments RFC-2581, April 1999. Online at: http://www.apps.ietf.org/rfc/rfc2581.html
4. G. Almes, S. Kalidindi, and M. Zekauskas, “A one-way delay metric for IPPM,” IETF Request for Comments RFC-2679, September 1999 (a). Online at: http://www.apps.ietf.org/rfc/rfc2679.html
5. G. Almes, S. Kalidindi, and M. Zekauskas, “A round-trip delay metric for IPPM,” IETF Request for Comments RFC-2681, September 1999 (b). Online at: http://www.apps.ietf.org/rfc/rfc2681.html
6. G. Anastasi and L. Lenzini, “QoS provided by the IEEE 802.11 wireless LAN to advanced data applications: A simulation analysis,” *ACM Wireless Networks*, vol. 6, no. 2, pp. 99-108, 2000.
7. D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan, “System support for bandwidth management and content adaptation in Internet applications,” *Proceedings of the USENIX OSDI Conference*, San Diego, CA, October 2000.
8. D. Anick, D. Mitra, and M.M. Sondhi, “Stochastic theory of data-handling system with multiple sources,” *Bell System Technical Journal*, vol. 61, no. 8, pp. 1871-1894, 1982.
9. C. Aras, J.F. Kurose, D. Reeves, and H. Schulzrinne, “Real-time communication in packet-switched networks,” *Proceedings of the IEEE*, vol. 82, no. 1, pp. 122-139, January 1994.
10. B. Badrinath, A. Fox, L. Kleinrock, G. Popek, P. Reiher, and M. Satyanarayanan, “A conceptual framework for network and client adaptation,” *Mobile Networks and Applications (MONET)*, ACM / Kluwer Academic Publishers, vol. 5, pp. 221-231, 2000.
11. H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz, “A comparison of mechanisms for improving TCP performance over wireless links,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp.756-769, December 1997.
12. A.J. Ballardie, P.F. Francis, and J. Crowcroft, “Core based trees,” *ACM SIGCOMM Computer Communication Review*, vol. 23, no. 4, pp. 85-95, August 1993.
13. P. Baran, “Introduction to Distributed Communications Network,” RAND Corporation, Memorandum RM-3420-PR, August 1964. Online at: http://www.rand.org/publications/RM/baran.list.html
14. R. van der Berg, “How the ‘Net works: An introduction to peering and transit,” Social Science Electronic Publishing, Inc., September 2, 2008. Online at: http://ssrn.com/abstract=1443245
15. D. Bertsekas and R. Gallager. *Data Networks*. 2nd edition, Prentice Hall, Upper Saddle River, NJ, 1992.
16. V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, “MACAW: Media access protocol for wireless LANs,” *Proceedings of the ACM SIGCOMM'94 Conference: Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 212-225, August 1994.

17. S.N. Bhatti and J. Crowcroft, "QoS-sensitive flows: Issues in IP packet handling," *IEEE Internet Computing*, vol. 4, no. 4, pp. 48-57, July 2000.

18. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," IETF Request for Comments RFC-2475, December 1998. Online at: http://www.apps.ietf.org/rfc/rfc2475.html

19. M.S. Blumenthal and D.D. Clark, "Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world," *ACM Transactions on Internet Technology*, vol. 1, no. 1, pp. 70-109, August 2001.

20. J.C. Bolot, "End-to-end packet delay and loss behavior in the Internet," *Proceedings of the ACM SIGCOMM '93 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 1993.

21. B. Braden, D.D. Clark, J. Crowcroft, B.S. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L.L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on queue management and congestion avoidance in the Internet," IETF Request for Comments RFC-2309, April 1998. Online at: http://www.apps.ietf.org/rfc/rfc2309.html

22. L.S. Brakmo and L.L. Peterson, "TCP Vegas: End-to-end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465-1480, October 1995.

23. L. Breslau, E.W. Knightly, S. Shenker, I. Stoica, and H. Zhang, "Endpoint Admission Control: Architectural Issues and Performance," *Proceedings of the ACM SIGCOMM 2000 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2000.

24. M. Buettner and D. Wetherall, "An empirical study of UHF RFID performance," *Proceedings of the 14th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2008)*, pp. 223-234, San Francisco, CA, September 2008.

25. Z. Cao and E. Zegura, "Utility max-min: An application-oriented bandwidth allocation scheme," *Proc. IEEE InfoCom*, vol. 2, pp. 793-801, 1999.

26. R.L. Carter and M.E. Crovella, "Measuring bottleneck link speed in packet-switched networks," Technical Report TR-96-006, Department of Computer Science, Boston University, March 1996. Online at: http://www.cs.bu.edu/faculty/crovella/papers.html

27. D. Chalmers and M. Sloman, "A survey of quality of service in mobile computing environments," *IEEE Communications Surveys*, vol. 2, no. 2, 1999.

28. H.S. Chhaya and S. Gupta, "Performance of asynchronous data transfer methods of IEEE 802.11 MAC protocol," *IEEE Personal Communications*, vol. 3, no. 5, October 1996.

29. J.I. Choi, M. Jain, K. Srinivasan, P. Levis, and S. Katti, "Achieving single channel, full duplex wireless communication," *Proceedings of the 16th ACM Annual International Conference on Mobile Computing and Networking (MobiCom 2010)*, pp. 1-12, Chicago, IL, September 2010. Online at: http://sing.stanford.edu/fullduplex/

30. M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith, "Tuning RED for Web traffic," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 249-264, June 2001.

31. Cisco Systems Inc., "Interface Queue Management," Cisco white paper, posted August 1995. Online at: http://www.cisco.com/warp/public/614/16.html

32. Cisco Systems Inc., "Performance Measurements of Advanced Queuing Techniques in the Cisco IOS," Cisco white paper, posted July 1999. Online at: http://www.cisco.com/warp/public/614/15.html

33. Cisco Systems Inc., "Advanced QoS Services for the Intelligent Internet," Cisco white paper, posted June 2006. Online at: http://www.cisco.com/warp/public/cc/pd/iosw/ioft/ioqo/tech/qos_wp.htm

34. Cisco Systems Inc., "Understanding Rapid Spanning Tree Protocol (802.1w)," Cisco white paper, posted October 2006. Online at: http://www.cisco.com/en/US/tech/tk389/tk621/technologies_white_paper09186a0080094cfa.shtml

35. D.D. Clark, “Design philosophy of the DARPA Internet protocols,” *Proceedings of the ACM SIGCOMM*, pp. 106-114, Stanford, CA, August 1988.

36. D.D. Clark, “The structuring of systems using upcalls,” *Proceedings of the 10th ACM Symposium on Operating Systems*, pp. 171-180, December 1985.

37. D.D. Clark and W. Fang, “Explicit allocation of best-effort packet delivery service,” *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 362-373, August 1998.

38. D.D. Clark, S. Shenker, and L. Zhang, “Supporting real-time applications in an integrated services packet network: Architecture and mechanisms,” *Proc. SIGCOMM '92*, Baltimore, MD, August 1992.

39. D.D. Clark and D.L. Tennenhouse, “Architectural considerations for a new generation of protocols,” *ACM SIGCOMM Computer Communications Review*, vol. 20, no. 4, pp. 200-208, August 1990.

40. D.E. Comer, *Internetworking With TCP/IP, Volume I: Principles, Protocols, and Architecture*, 5th Edition, Pearson Prentice Hall, Upper Saddle River, NJ, 2006.

41. B.P. Crow, I. Widjaja, L.G. Kim, and P.T. Sakai, “IEEE 802.11 wireless local area networks,” *IEEE Communications Magazine*, vol. 35, no. 9, pp. 116-126, September 1997.

42. W. Dapeng and R. Negi, “Effective capacity: A wireless link model for support of quality of service,” *IEEE Transactions on Wireless Communications*, vol. 2, no. 4, pp. 630-643, July 2003.

43. B.S. Davie and Y. Rekhter, *MPLS: Technology and Applications*, Morgan Kaufmann Publishers (Academic Press), San Francisco, CA, 2000.

44. J.D. Day and H. Zimmermann, “The OSI reference model,” *Proceedings of the IEEE*, vol.71, no.12, pp. 1334-1340, December 1983.

45. S. Deering and R. Hinden, “Internet Protocol, version 6 (IPv6) specification,” IETF Request for Comments RFC-2460, December 1998. Online at: http://www.apps.ietf.org/rfc/rfc2460.html

46. L. De Ghein, *MPLS Fundamentals: A Comprehensive Introduction to MPLS Theory and Practice*, Cisco Press, Indianapolis, IN, 2007.

47. C. Demichelis and P. Chimento, “IP packet delay variation metric for IP performance metrics (IPPM),” IETF Request for Comments RFC-3393, November 2002. Online at: http://www.apps.ietf.org/rfc/rfc3393.html

48. D.-J. Deng and R.-S. Chang, “A priority scheme for IEEE 802.11 DCF access method,” *IEICE Transactions on Communications*, E82-B-(1), January 1999.

49. J. van Duuren, “Fault-free digital radio communication and Hendrik C. A. van Duuren,” *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1540-1542, October 2001.

50. M. DuVal and T. Siep, “High Rate WPAN for Video,” IEEE document: IEEE 802.15-00/029, Submission date: 6 March 2000. Online at: http://grouper.ieee.org/groups/802/15/pub/2000/Mar00/00029r0P802-15_CFA-Response-High-Rate-WPAN-for-Video-r2.ppt

51. I. Elhanany, M. Kahane, and D. Sadot, “Packet scheduling in next-generation multiterabit networks,” *IEEE Computer*, vol. 34, no. 4, pp. 104-106, April 2001.

52. G. Fairhurst and L. Wood, “Advice to link designers on link Automatic Repeat reQuest (ARQ),” IETF Request for Comments RFC-3366, August 2002. Online at: http://www.apps.ietf.org/rfc/rfc3366.html

53. K. Fall and S. Floyd, “Simulation-based comparisons of Tahoe, Reno and SACK TCP,” *ACM Computer Communication Review*, vol. 26, no. 3, pp. 5-21, July 1996.

54. C. Ferran and S. Watts, “Videoconferencing in the field: A heuristic processing model,” *Management Science*, vol. 54, no. 9, pp. 1565-1578, September 2008.

55. V. Firou, J. Le Boudec, D. Towsley, and Z. Zhang, “Theories and models for Internet quality of service,” *Proceedings of the IEEE (Special Issue on Internet Technology)*, August 2002.

56. S. Floyd, "Congestion control principles," IETF Request for Comments RFC-2914, September 2000. Online at: http://www.apps.ietf.org/rfc/rfc2914.html

57. S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," IETF Request for Comments RFC-3782, April 2004. Online at: http://www.rfc-editor.org/rfc/rfc3782.txt

58. S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397-413, August 1993.

59. Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP performance," *IEEE Transactions on Mobile Computing*, vol. 4, no. 2, pp. 209-221, March 2005.

60. C.L. Fullmer and J.J. Garcia-Luna-Aceves, "Solutions to hidden terminal problems in wireless networks," *Proceedings of ACM SIGCOMM'97 Conference: Applications, Technologies, Architectures, and Protocols for Computer Communication*, Cannes, France, 1997.

61. F.C. Gärtner, "A Survey of Self-Stabilizing Spanning-Tree Construction Algorithms," Swiss Federal Institute of Technology (EPFL), Technical Report IC/2003/38, June 10, 2003. Online at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.3874

62. P. Goyal, S.S. Lam, and H.M. Vin, "Determining end-to-end delay bounds in heterogeneous networks," *ACM Multimedia Systems*, vol. 5, no. 3, pp. 157-163, 1997.
[An earlier version of this paper appeared in *Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '95)*, Durham, NH, pp. 287-298, April 1995.]

63. J.P. Gray, "Line control procedures," *Proceedings of the IEEE*, vol. 60, no. 11, pp. 1301-1312, November 1972.

64. C. Greenhalgh, S. Benford, and G. Reynard, "A QoS architecture for collaborative virtual environments," *Proceedings of the ACM Multimedia Conference*, pp.121-130, 1999.

65. S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review – Research and developments in the Linux kernel*, vol. 42, no. 5, pp. 64-74, July 2008.

66. S. Haykin, *Communication Systems*, 5th Edition, John Wiley & Sons, Inc., Hoboken, NJ, 2009.

67. L. He and J. Walrand, "Pricing and revenue sharing strategies for Internet service providers," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 5, pp. 942-951, May 2006.

68. J. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," *Proceedings of the ACM SIGCOMM 1996 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, August 1996.

69. G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," *Proceedings of the 5th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1999)*, pp. 219-230, Seattle, WA, August 1999.

70. N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal on Selected Areas in Communications*, August 2003.

71. C. Huitema, *IPv6: The New Internet Protocol*, 2nd Edition, Prentice-Hall, Inc., Upper Saddle River, NJ, 1998.

72. N.C. Hutchinson and L.L. Peterson, "The *x*-kernel: An architecture for implementing network protocols," *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64-76, January 1991.

73. D.S. Isenberg, "The dawn of the stupid network," *ACM netWorker*, vol. 2, no. 1, pp. 24-31, February/March 1998. Online at: http://www.isen.com/papers/Dawnstupid.html
An older version of this paper, "Rise of the stupid network," is available online here: http://www.rageboy.com/stupidnet.html

74. V. Jacobson, “Congestion avoidance and control,” *ACM Computer Communication Review*, vol. 18, no. 4, pp. 314-329, August 1988. Online at: ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z

75. M. Jain and C. Dovrolis, “Pathload: A measurement tool for end-to-end available bandwidth,” *Proceedings of the 3rd Passive and Active Measurements Workshop*, Fort Collins, CO, March 2002.

76. R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, New York, NY, 1991.

77. C. Jin, D. Wei, S.H. Low, G. Buhrmaster, J. Bunn, D.H. Choe, R.L.A. Cottrell, J.C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh, “FAST TCP: From Theory to Experiments,” Submitted to *IEEE Communications Magazine*, April 1, 2003. Online at: http://netlab.caltech.edu/FAST/publications.html

78. R. Johari and J.N. Tsitsiklis, “Routing and peering in a competitive Internet,” *Proceedings of the 2004 IEEE Conference on Decision and Control*, Bahamas, December 2004.
Extended version: MIT technical report LIDS-P-2570, January 2003. Online at:
http://citeseer.ist.psu.edu/572286.html

79. D. Kahneman, *Attention and Effort*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.

80. S. Kandula et al. “Walking the tightrope: Responsive yet stable traffic engineering,” *Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*.

81. V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly, “Distributed priority scheduling and medium access in ad hoc networks,” *Wireless Networks*, vol. 8, no. 5, pp. 455-466, 2002.

82. P. Karn, “MACA - A new channel access method for packet radio,” *Proceedings of the 9th ARRL Computer Networking Conference*, pp. 134-140, London, Ontario, Canada, 1990.

83. D. Katabi, M. Handley, and C. Rohrs, “Congestion control for future high bandwidth-delay product networks,” *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, August 2002.

84. R. Katz, “Adaptation and mobility in wireless information systems,” *IEEE Personal Communications*, vol. 1, no. 1, pp. 6-17, 1994.

85. S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley Publ. Co., Reading, MA, 1997.

86. D. Kiwior, J. Kingston, and A. Spratt, “PathMon, a methodology for determining available bandwidth over an unknown network,” *Proceedings of the 2004 IEEE/Sarnoff Symposium on Advances in Wired and Wireless Communication*, pp. 27-30, Princeton, NJ, April 2004.

87. S. Keshav and R. Sharma, “Issues and trends in router design,” *IEEE Communications Magazine*, vol. 36, no. 5, pp. 144-151, May 1998.

88. L. Kleinrock and F. Tobagi, “Packet switching in radio channels: Part I—Carrier sense multiple-access modes and their throughput-delay characteristics,” *IEEE Transactions on Communications*, vol. 23, no. no.12, pp. 1400-1416, December 1975.

89. A. Kumar, “Comparative performance analysis of versions of TCP in a local network with a lossy link,” *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 485-498, August 1998.

90. V.P. Kumar, T.V. Lakshman, and D. Stiliadis, “Beyond best effort: Router architectures for the differentiated services of tomorrow’s Internet,” *IEEE Communications Magazine*, vol. 36, no. 5, pp. 152-164, May 1998.

91. J.F. Kurose and K.W. Ross, *Computer Networking: A Top-Down Approach*. 5th edition. Pearson Education, Inc. (Addison-Wesley), Boston, MA, 2010.

92. K. Lai and M. Baker, “Measuring bandwidth,” *Proceedings of the Conference on Computer Communications (IEEE INFOCOM '99)*, pp. 235-245, New York, NY, March 1999.

93. K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," *Proceedings of the ACM SIGCOMM 2000 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Stockholm, Sweden, August 2000.

94. E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984. Online at: http://www.cs.washington.edu/homes/lazowska/qsp/

95. S. Leffler, M. McKusick, M. Karels, J. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, Reading, MA, 1989.

96. R.R.-F. Liao and A.T. Campbell, "A utility-based approach for quantitative adaptation in wireless packet networks," *Wireless Networks*, vol. 7, no. 5, pp. 541-557, 2001.

97. W. Liu, "Focusing on desirability: The effect of decision interruption and suspension on preferences," *Journal of Consumer Research*. (Forthcoming), December 2008. Online at: http://www.anderson.ucla.edu/x15549.xml

98. Y. Mao and L.K. Saul, "Modeling distances in large-scale networks by matrix factorization," *Proceedings of the Second Internet Measurement Conference (IMC-04)*, pp. 278-287, Sicily, Italy, 2004. Online at: https://wiki.planet-lab.org/bin/view/Planetlab/AnnotatedBibliography

99. S. Mascolo, C. Casetti, M. Gerla, M.Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MobiCom 2001)*, Rome, Italy, pp. 287-296, August 2001.

100. M.L. Massie, B.N. Chun, and D.E. Culler, "The Ganglia distributed monitoring system: Design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, July 2004.

101. J.M. McQuillan, I. Richer, and E.C. Rosen, "The new routing algorithm for the ARPANet," IEEE *IEEE Transactions on Communications*, vol. 28, no. 5, pp. 711-719, 1980.

102. C.S.R. Murthy and B.S. Manoj, *Ad Hoc Wireless Networks: Architectures and Protocols*. Prentice Hall PTR, Upper Saddle River, NJ, 2004.

103. J. Nagle, "Congestion control in IP/TCP internetworks," IETF Request for Comments RFC-896, January 1984. Online at: http://www.rfc-editor.org/rfc/rfc896.txt

104. J. Nagle, "On packet switches with infinite storage," *IEEE Transactions on Communications*, vol. 35, no. 4, pp. 435-438, April 1987.

105. K. Nahrstedt and J.M. Smith, "The QoS broker," *IEEE Multimedia*, vol. 2, no. 1, pp. 53-67, 1995.

106. K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, "QoS-aware middleware for ubiquitous and heterogeneous environments," *IEEE Communications Magazine*, vol. 39, no. 11, pp. 140-148, 2001.

107. B.D. Noble and M. Satyanarayanan, "Experience with adaptive mobile applications in Odyssey," *Mobile Networks and Applications (MONET)* (ACM / Kluwer Academic Publishers), vol. 4, pp. 245–254, 1999.

108. E. Osborne and A. Simha, *Traffic Engineering with MPLS*. Cisco Press, Indianapolis, IN, 2002. Online at: http://cisco-press-traffic-engineering.org.ua/1587050315/main.html

109. J. Padhye, V. Firoiu, D.F. Towsley, and J.F. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133-145, April 2000.

110. J. Padhye, V. Firoiu, D.F. Towsley, and J.F. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '98)*, Vancouver, British Columbia, Canada, pp. 303-314, August/September 1998.

111. V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP performance metrics," IETF Request for Comments RFC-2330, May 1998. Online at: http://www.apps.ietf.org/rfc/rfc2330.html

112. A. Papoulis and S.U. Pillai, *Probability, Random Variables and Stochastic Processes*. 4th edition, McGraw-Hill, New York, NY, 2001.

113. A.K. Parekh and R.G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344-357, June 1993.

114. A.K. Parekh and R.G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 137-150, April 1994.

115. V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," Ph.D. Thesis, University of California, Berkeley, April 1997. (a)

116. V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 601-615, October 1997. (b)

117. V. Paxson, "Strategies for sound Internet measurement," *Proceedings of the ACM IMC*, October 2004.

118. V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," IETF Request for Comments RFC-2988, November 2000. Online at: http://www.rfc-editor.org/rfc/rfc2988.txt

119. E. Perahia and R. Stacey, *Next Generation Wireless LANs: Throughput, Robustness, and Reliability in 802.11n*. Cambridge University Press, Cambridge, UK, 2008.

120. C. Perkins, *RTP: Audio and Video for the Internet*. Pearson Education, Inc., Boston, MA, 2003.

121. C.E. Perkins (Editor), *Ad Hoc Networks*. Addison-Wesley, Upper Saddle River, NJ, 2001.

122. L.L. Peterson and B.S. Davie, *Computer Networks: A Systems Approach*. 4th edition. Morgan Kaufmann Publishers (Elsevier, Inc.), San Francisco, CA, 2007.

123. R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for QoS management," *Proceedings of the IEEE Real-Time Systems Symposium*, pp.298-307, December 1997.

124. D.P. Reed, J.H. Saltzer, and D.D. Clark, "Comment on active networking and end-to-end arguments," *IEEE Network*, vol. 12, no. 3, pp. 69-71, May/June 1998.

125. Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," IETF Request for Comments RFC-4271, January 2006. Online at: http://www.apps.ietf.org/rfc/rfc4271.html

126. J.H. Saltzer, D.P. Reed, D.D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277-288, November 1984.

127. C.U. Saraydar, N.B. Mandayam, and D.J. Goodman, "Efficient power control via pricing in wireless data networks," *IEEE Transactions on Communications*, vol. 50, no. 2, pp. 291-303, February 2002.

128. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF Request for Comments RFC-3550, July 2003. Online at: http://www.ietf.org/rfc/rfc3550.txt

129. A. Sears and J.A. Jacko, "Understanding the relationship between network quality of service and the usability of distributed multimedia documents," *Human-Computer Interaction*, vol. 15, pp. 43-68, 2000.

130. S. Sen, R.R. Choudhury, and S. Nelakuditi, "CSMA/CN: Carrier sense multiple access with collision notification," *Proceedings of the 16th ACM Annual International Conference on Mobile Computing and Networking (MobiCom 2010)*, pp. 25-36, Chicago, IL, September 2010.

131. S. Shakkottai and R. Srikant, "Economics of network pricing with multiple ISPs," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1233-1245, December 2006.

132. C.E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, IL, 1949.

133. S. Shenker, “Fundamental design issues for the future Internet,” *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1176-1188, September 1995.

134. S. Shenker, “Making greed work in networks: A game-theoretic analysis of switch service disciplines,” *Proceedings of the ACM SIGCOMM 1994 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 47-57, 1994.

135. S. Shenker, C. Partridge, and R. Guerin, “Specification of Guaranteed Quality of Service,” IETF Request for Comments RFC-2212, September 1997. Online at: http://www.apps.ietf.org/rfc/rfc2212.html

136. G. Shrimali and S. Kumar, “Paid peering among Internet service providers,” *Proceeding from the 2006 Workshop on Game Theory for Communications and Networks (GameNets '06)*, ACM International Conference Proceeding Series, vol. 199, article no. 11, Pisa, Italy, October 2006.

137. W. Simpson, “The Point-to-Point Protocol (PPP),” IETF Request for Comments RFC-1661, July 1994. Online at: http://www.apps.ietf.org/rfc/rfc1661.html

138. R. Srikant, *The Mathematics of Internet Congestion Control*. Birkhäuser, Boston, MA, 2004.

139. P. Srisuresh and K. Egevang, “Traditional IP Network Address Translator (Traditional NAT),” IETF Request for Comments RFC-3022, January 2001. Online at: http://www.apps.ietf.org/rfc/rfc3022.html

140. P. Srisuresh and M. Holdrege, “IP Network Address Translator (NAT) terminology and considerations,” IETF Request for Comments RFC-2663, August 1999. Online at: http://www.apps.ietf.org/rfc/rfc2663.html

141. W.R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley Publ. Co., Reading, MA, 1994.

142. W.R. Stevens, “TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms,” IETF Request for Comments RFC-2001, January 1997. Online at: http://www.apps.ietf.org/rfc/rfc2001.html

143. J.W. Stewart III, *BGP: Inter-Domain Routing in the Internet*. Addison-Wesley Publ. Co., Reading, MA, 1999.

144. L. Subramanian, V.N. Padmanabhan, R.H. Katz, “Geographic properties of Internet routing,” *Proceedings of the USENIX Annual Technical Conference (ATEC '02)*, pp. 243-259, Monterey, CA, June 2002.

145. D.E. Taylor, “Survey and taxonomy of packet classification techniques,” *ACM Computing Surveys*, vol. 37, no. 3, pp. 238-275, September 2005.

146. C.A. Thekkath, T.D. Nguyen, E. Moy, and E.D. Lazowska, “Implementing network protocols at user level,” *Proceedings of the ACM SIGCOMM*, San Francisco, CA, September 1993.

147. K. Thomson, G.J. Miller, and R. Wilder, “Wide-area traffic patterns and characteristics,” *IEEE Network*, December 1997.

148. P. Thornycroft, “Designed for Speed: Network Infrastructure in an 802.11n World,” white paper, Aruba Networks, Inc. 2009. Online at: http://www.arubanetworks.com/pdf/technology/whitepapers/wp_Designed_Speed_802.11n.pdf

149. J.S. Turner, “New directions in communications (or which way to the information age?).” *IEEE Communications Magazine*, vol. 24, no. 10, pp. 8-15, 1986.

150. C.-Y. Wang and H.-Y. Wei, “IEEE 802.11n MAC enhancement and performance evaluation,” *ACM Mobile Networks and Applications (MONET)*, vol. 14, no. 6, pp. 760-771, December 2009.

151. Z. Wang, *Internet QoS: Architectures and Mechanisms for Quality of Service*, Morgan Kaufmann Publishers (Academic Press), San Francisco, CA, 2001.

152. S. Weinstein, “The mobile Internet: Wireless LAN vs. 3G cellular mobile,” *IEEE Communications Magazine*, pp.26-28, February 2002.

153. A. Wolisz and F.H.P. Fitzek, “QoS support in wireless networks using simultaneous MAC packet transmission (SMPT),” in ATS, April 1999.

154. X.-P. Xiao, *Technical, Commercial and Regulatory Challenges of QoS: An Internet Service Model Perspective*, Morgan Kaufmann Publishers (Elsevier Inc.), San Francisco, CA, 2008.

155. Y. Xiao, “IEEE 802.11n: Enhancements for higher throughput in WLANs,” *IEEE Wireless Communications Magazine*, vol. 12, no. 6, pp. 82-91, December 2005.

156. Y. Xiao and J. Rosdahl, “Performance analysis and enhancement for the current and future IEEE 802.11 MAC protocols,” *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, special issue on Wireless Home Networks, vol. 7, no. 2, pp. 6-19, April 2003.

157. Y. Xiao and J. Rosdahl, “Throughput and delay limits of IEEE 802.11,” *IEEE Communications Letters*, vol. 6, no. 8, pp. 355-357, August 2002.

158. R.D. Yates and D.J. Goodman, *Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers*. 2nd edition, John Wiley & Sons, Inc., New York, NY, 2004.

159. Q. Zhang, W. Zhu, and Y.-Q. Zhang, “Resource allocation for multimedia streaming over the Internet,” *IEEE Transactions on Multimedia*, vol. 3, no. 3, pp. 339-355, September 2001.

160. M. Ziegelmann, *Constrained Shortest Paths and Related Problems: Constrained Network Optimization*, VDM Verlag Dr. Müller, Saarbrücken, Germany, 2007.

Acronyms and Abbreviations

3G — Third Generation (wireless networks)

4G — Fourth Generation (wireless networks)

ABR — Available Bit-Rate

ACK — Acknowledgement

ADDBA — Add Block Acknowledgment

AIMD — Additive Increase/Multiplicative Decrease

ALF — Application Level Framing

AODV — Ad Hoc On-Demand Distance-Vector

AQM — Active Queue Management

AP — Access Point

AF — Assured Forwarding

API — Application Programming Interface

ARP — Address Resolution Protocol

ARQ — Automatic Repeat Request

ASCII — American Standard Code for Information Interchange

ASIC — Application Specific Integrated Circuit

ASN — Autonomous System Number

ATM — Asynchronous Transfer Mode

AWGN — Additive White Gaussian Noise

BACK — Block Acknowledgment

BAN — Body Area Network

BDPDR — Bounded Delay Packet Delivery Ratio

BER — Bit Error Rate

BGP — Border Gateway Protocol

BPDU — Bridge Protocol Data Unit

bps — bits per second

BS — Base Station

BSS — Basic Service Set

CBR — Constant Bit-Rate;
also: Constraint-Based Routing (in MPLS)

CBT — Core Based Tree

CCA — Clear Channel Assessment

CDMA — Code Division Multiple Access

CDN — Content Distribution Network

CIDR — Classless Interdomain Routing

CNAME — Canonical name (in RTCP)

COA — Care-Of Address

CORBA — Common Object Request Broker Architecture

CoS — Class of Service

CPU — Central Processing Unit

CQS — Classify, Queue, and Schedule

CRC — Cyclic Redundancy Check

CSMA — Carrier-Sense Multiple Access
CSMA/CA — CSMA / Collision Avoidance
CSMA/CD — CSMA / Collision Detection

CSPF — Constrained Shortest Path First

CTS — Clear To Send

DBS — Direct Broadcast Satellite

DCF — Distributed Coordination Function

DELBA — Delete Block Acknowledgment

DHCP — Dynamic Host Configuration Protocol

DiffServ — Differentiated Services (alternative: DS)

DIFS — DCF (or Distributed) Inter Frame Space

DNS — Domain Name System

DPI — Deep Packet Inspection

DSR — Dynamic Source Routing

DTN — Disruption-Tolerant Networking

dupACK — Duplicate Acknowledgement

DV — Distance Vector

DVMRP — Distance Vector Multicast Routing Protocol

ECN — Explicit Congestion Notification

EF — Expedited Forwarding

EGP — Exterior Gateway Protocol

EIFS — Extended Inter Frame Space

ESS — Extended Service Set

EV-DO — EVolution – Data Optimized

EWMA — Exponential Weighted Moving Average

FCFS — First Come First Served

FDM — Frequency Division Multiplexing

FDMA — Frequency Division Multiple Access

FEC — Forward Error Correction;
also: Forwarding Equivalence Class (in MPLS)

FIB — Forwarding Information Base

FIFO — First In First Out

FIRO — First In Random Out

FPGA — Field-Programmable Gate Array

FQ — Fair Queuing

FSM — Finite State Machine

FTP — File Transfer Protocol

GBN — Go-Back-N

GPS — Generalized Processor Sharing

GUI — Graphical User Interface

HAA — Home Address Agent

HDLC — High-level Data Link Control

HOL — Head Of Line

HSPA — High Speed Packet Access

HT — High Throughput

HTML — HyperText Markup Language

HTTP — HyperText Transport Protocol

IANA — Internet Assigned Numbers Authority

IBSS — Independent Basic Service Set

ICANN — Internet Corporation for Assigned Names and Numbers

ICMP — Internet Control Message Protocol

IEEE — Institute of Electrical and Electronics Engineers

IETF — Internet Engineering Task Force

IFS — Inter Frame Spacing

IGP — Interior Gateway Protocol

IntServ — Integrated Services

IP — Internet Protocol
- **IPv4** — Internet Protocol version 4
- **IPv6** — Internet Protocol version 6

IPPM — IP Performance Metrics

ISO — International Standards Organization

ISP — Internet Service Provider

j.n.d. — just noticeable difference

Kbps — Kilo bits per second

LAN — Local Area Network

LCFS — Last Come First Served

LCP — Link Control Protocol

LFIB — Label Forwarding Information Base

LIB — Label Information Base

LLC — Logical Link Control

LS — Link State

LSA — Link State Advertisement

LSDB — Link State Database

L-SIG — Legacy Signal (non-high-throughput Signal field of 802.11n physical-layer frame header)

LSP — Label Switched Path (in MPLS);
also: Link State Packet (in LS routing and OSPF)

LSR — Label Switching Router

LTE — Long Term Evolution (also known as 4G)

MAC — Medium Access Control

MACA — Multiple Access with Collision Avoidance

MACAW — Multiple Access with Collision Avoidance for Wireless

MANET — Mobile Ad-hoc Network

Mbps — Mega bits per second

MCS — Modulation and Coding Scheme

MIB — Management Information Base

MIMO — Multiple-Input Multiple-Output

MPDU — MAC Protocol Data Unit

MPEG — Moving Picture Experts Group

MPLS — MultiProtocol Label Switching

MSDU — MAC Service Data Unit

MSS — Maximum Segment Size

MTU — Maximum Transmission Unit

NAK — Negative Acknowledgement

NAT — Network Address Translation

NAV — Network Allocation Vector

NCP — Network Control Protocol

NDP — Neighbor Discovery Protocol

NFE — Network Front-End (Processor)

NIC — Network Interface Card

NLRI — Network Layer Reachability Information

NMS — Network Management System

OLSR — Optimized Link State Routing

OSI — Open Systems Interconnection

OSPF — Open Shortest Path First

P2P — Peer-to-Peer (some would say Pier-to-Pier ☺)

PAN — Personal Area Network

PC — Personal Computer

PCF — Point Coordination Function

PCM — Pulse Code Modulation

PCO — Phased Coexistence Operation

PDA — Personal Digital Assistant

PDU — Protocol Data Unit

pdf — probability distribution function

pmf — probability mass function

PER — Packet Error Rate

PHB — Per-Hop Behavior

PHY — Physical Layer

PIFS — PCF (or Priority) Inter Frame Space

PIM — Protocol Independent Multicast

PLCP — Physical Layer Convergence Procedure

PoP — Point-of-Presence

PPDU — PLCP Protocol Data Unit

PPP — Point-to-Point Protocol

PSDU — PLCP Service Data Unit

PSTN — Public Switched Telephone Network

PtMP — Point-to-Multipoint

PtP — Point-to-Point

QoE — Quality of Experience

QoS — Quality of Service

RED — Random Early Detection

RFC — Request For Comments

RFID — Radio Frequency Identification

RIB — Routing Information Base

RIFS — Reduced Inter Frame Space

RIP — Routing Information Protocol

RMON — Remote Monitoring

RPC — Remote Procedure Call

RPF — Reverse Path Forwarding

RR — Receiver Report (in RTCP)

RSSI — Receive(r) Signal Strength Index/Indication

RSTP — Rapid Spanning Tree Protocol

RSVP — Resource ReSerVation Protocol

RTCP — RTP Control Protocol

RTO — Retransmission Time Out

RTP — Real-time Transport Protocol

RTS — Request To Send

RTSP — Real-Time Streaming Protocol

RTT — Round-Trip Time

SACK — Selective Acknowledgement

SDES — Source Description (in RTCP)

SDM — Spatial Division Multiplexing

SDP — Session Description Protocol

SFD — Start Frame Delimiter

SIFS — Short Inter Frame Space

SIP — Session Initiation Protocol

SLA — Service Level Agreement

SMTP — Simple Mail Transfer Protocol

SN — Sequence Number

SNMP — Simple Network Management Protocol

SNR — Signal-to-Noise Ratio

SONET — Synchronous Optical Network

SR — Selective Repeat;
also: Sender Report (in RTCP)

SSTresh — Slow-Start Threshold

STP — Spanning Tree Protocol

TC — Traffic Category

TCP — Transmission Control Protocol

TDM — Time Division Multiplexing

TDMA — Time Division Multiple Access

TE — Traffic Engineering

TID — Traffic Identifier

TS — Traffic Stream

TTL — Time To Live

TXOP — Transmit Opportunity

UBR — Unspecified Bit Rate

UDP — User Datagram Protocol

URL — Uniform Resource Locator

VBR — Variable Bit Rate

VLAN — Virtual Local Area Network

VLSI — Very Large Scale Integration

VoIP — Voice over IP

VoWiFi — Voice over Wi-Fi

VPN — Virtual Private Network

W3C — World Wide Web Consortium

WAN — Wide Area Network

WAP — Wireless Access Protocol

WEP — Wired Equivalent Privacy

WFQ — Weighted Fair Queuing

Wi-Fi — Wireless Fidelity (synonym for IEEE 802.11)

WiMAX — Worldwide Interoperability for Microwave Access (synonym for IEEE 802.16)

W-LAN — Wireless Local Area Network

WWW — World Wide Web

ZigBee — *See* http://en.wikipedia.org/wiki/ZigBee for the origins of this name

Index