Chapter 1
Introduction to Computer Networks

Contents

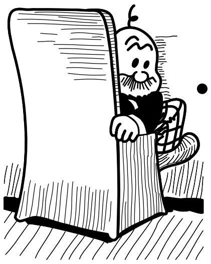
1.1 Introduction

A network is a set of devices (often referred to as nodes) connected by communication links 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. Examples 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

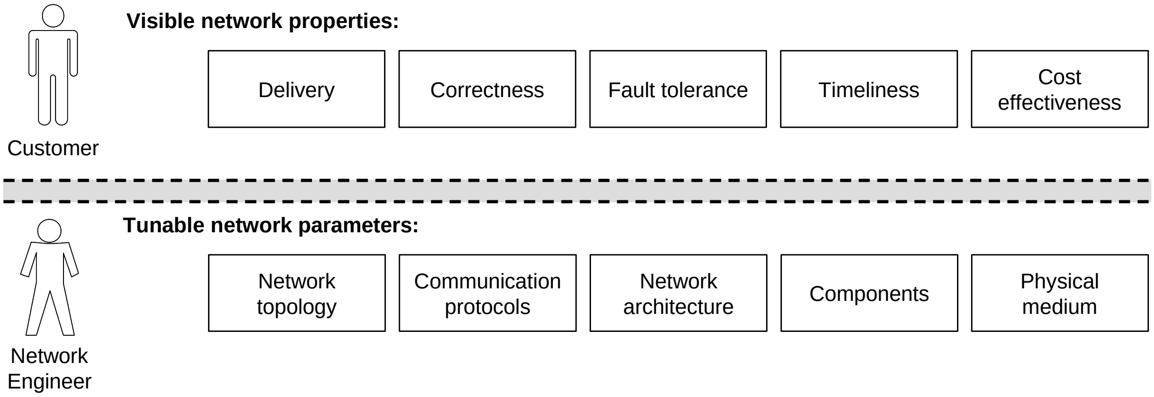


Figure 1-1: The customer cares about the visible network properties that can be controlled by the adjusting the network parameters.

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, whether they are acceptable 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.

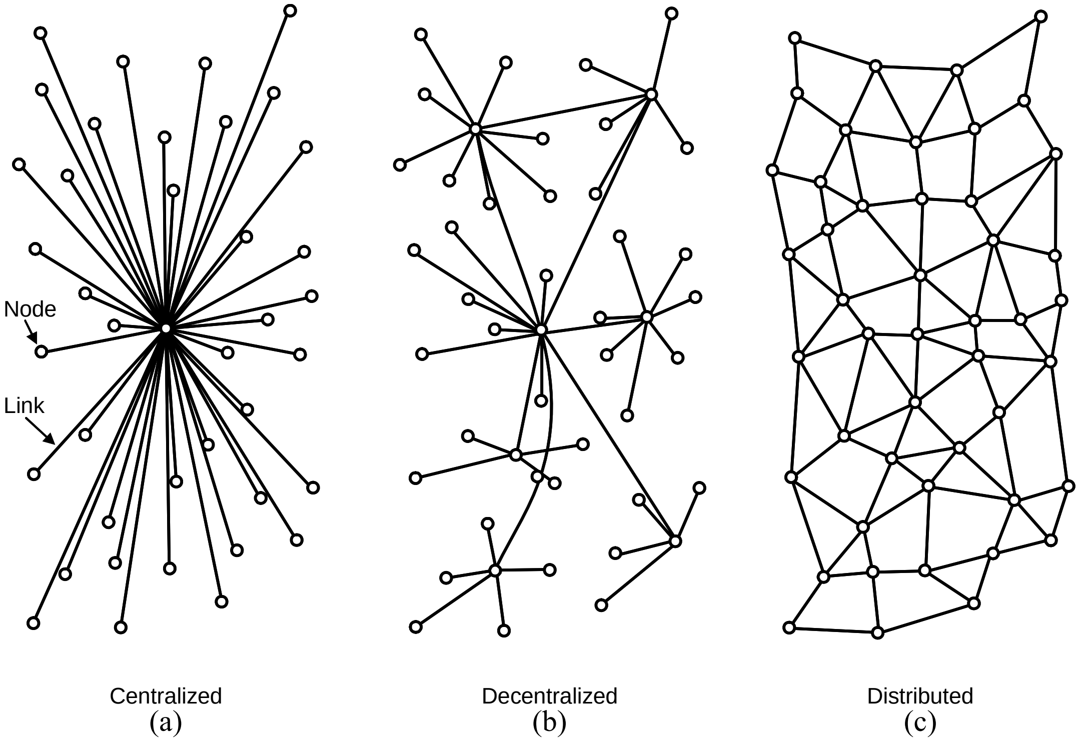
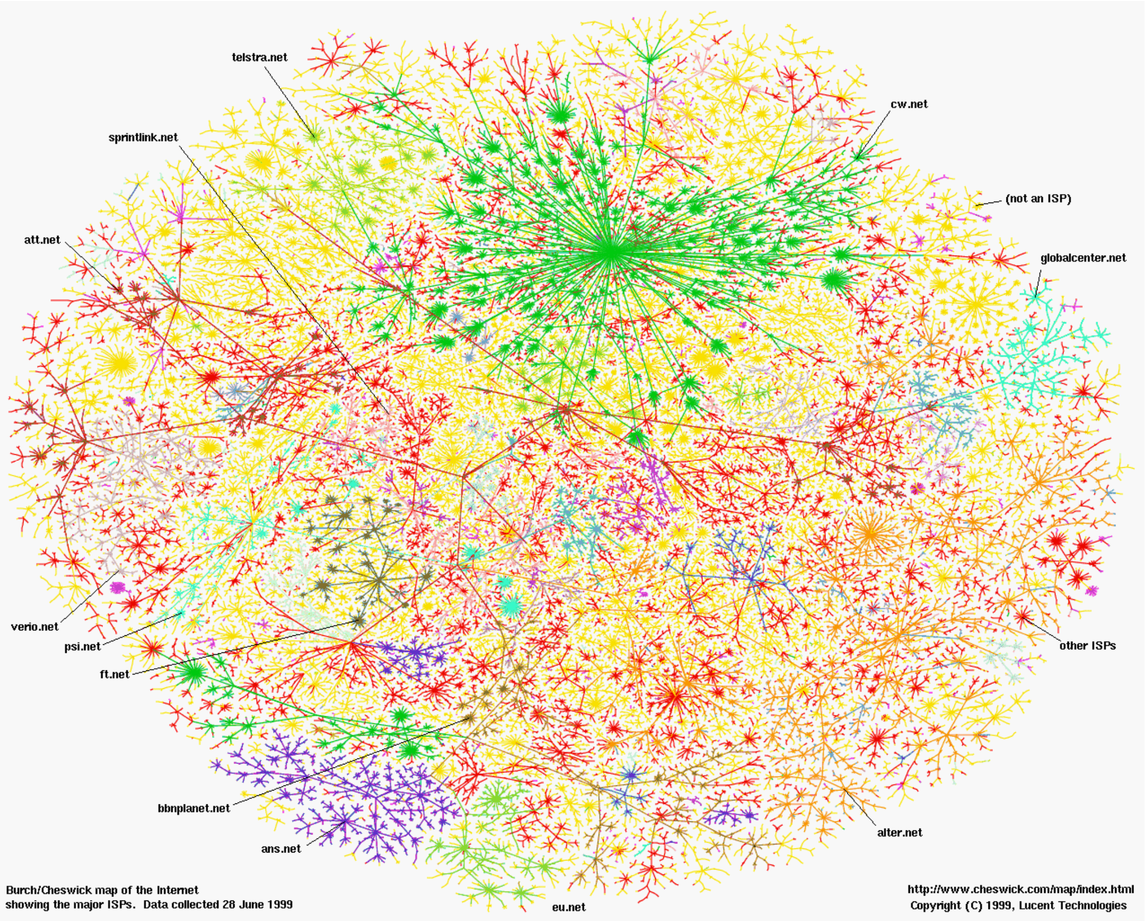


Figure 1-2: Different network topologies have different robustness characteristics relative to the failures of network elements.

- Connection topology: completely connected graph vs. 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).

[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.



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: how much of the network is a fixed infrastructure vs. ad hoc built for a temporary need

- Component characteristics: reliability and performance of individual components (nodes and links). Faster and more reliable components are also more costly. When a network node (known as 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 the "waiting room," so loss occurs when messages arrive at a full queue.

- Performance metric: success rate of transmitted packets + average delay + delay variability (jitter)

- Different applications (data/voice/multimedia) have different requirements: sensitive to loss vs. sensitive to delay/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:

- *Heterogeneity*: Diverse software and hardware of network components need to coexist and interoperate. The diversity results from different user needs, as well as because installed

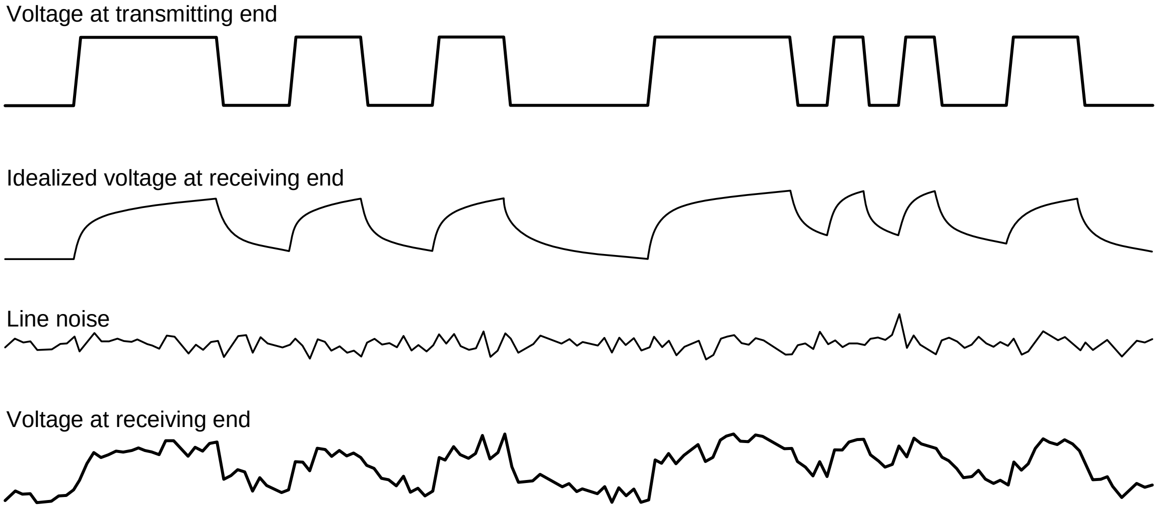


Figure 1-4: Digital signal distortion in transmission due to noise and time constants associated with physical lines.

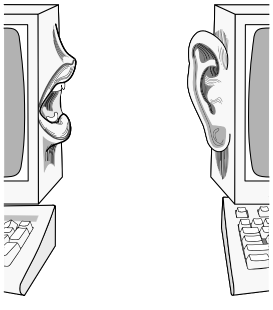
infrastructure tends to live long enough to become mixed with several new generations of technologies.

- *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 their 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. 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 time constants 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 that is shown in the bottom row 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 separated and easier to recognize in the distorted signal. The minimum tolerable separation depends on the physical characteristics of a transmission line. If each pulse corresponds to a single bit of information, then the



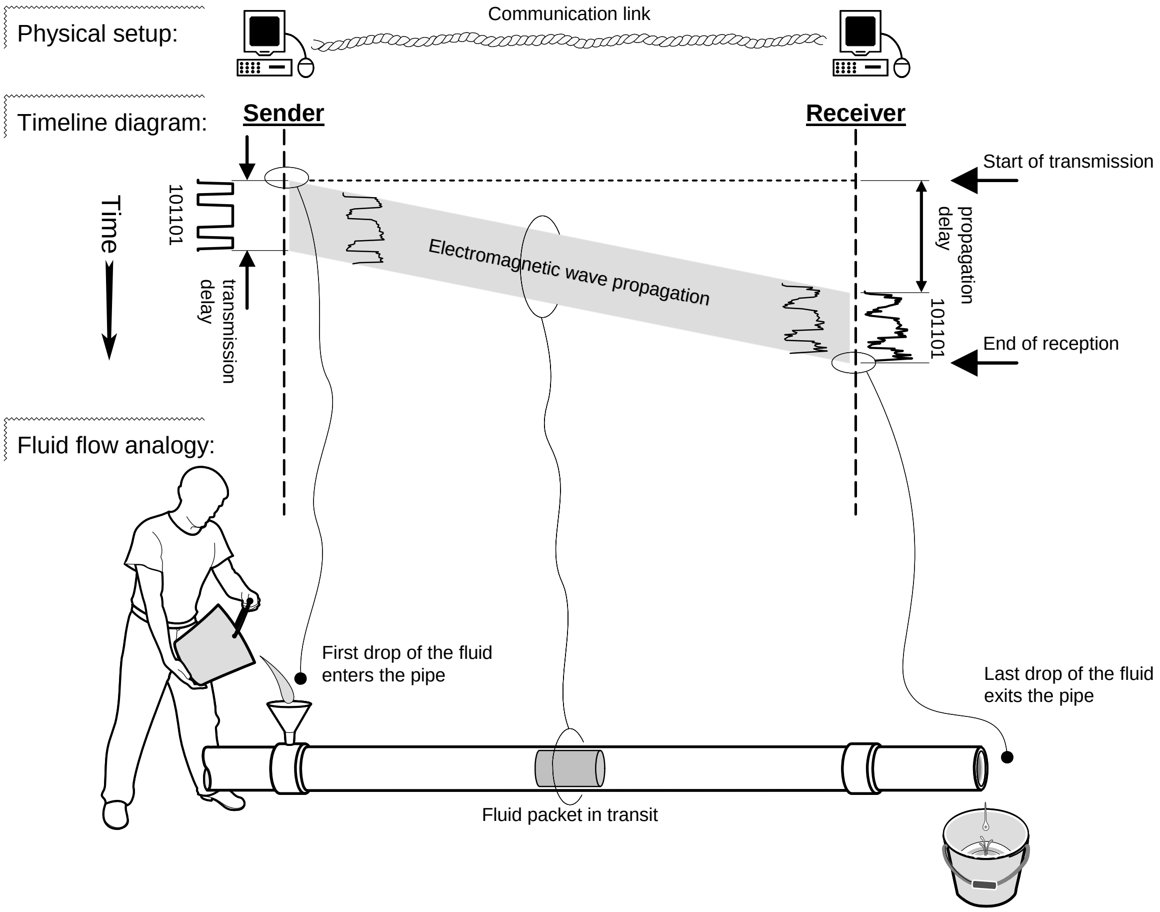


Figure 1-5: Timeline diagram for data transmission from sender to receiver.

minimum tolerable separation of pulses determines the maximum number of bits that can be transmitted over a particular transmission line.

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 line are conceptually equivalent, as represented in Figure 1-6, because the transmission capacity for every line is expressed in bits/sec or bps. In this text, 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, 2006].

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 data collisions. 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).

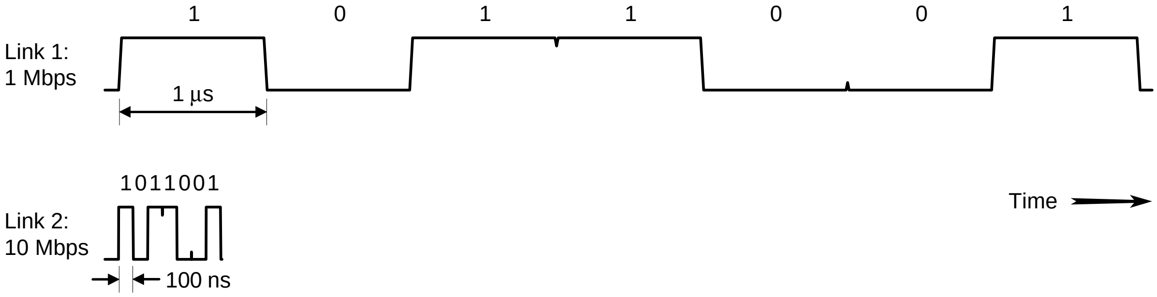


Figure 1-6: Transmission link capacity determines the speed at which the link can transmit data. In this example, Link 2 can transmit ten times more data than Link 1 in the same period. Each bit on Link 1 is 1 μs wide, while on Link 2 each bit is 100 ns wide.

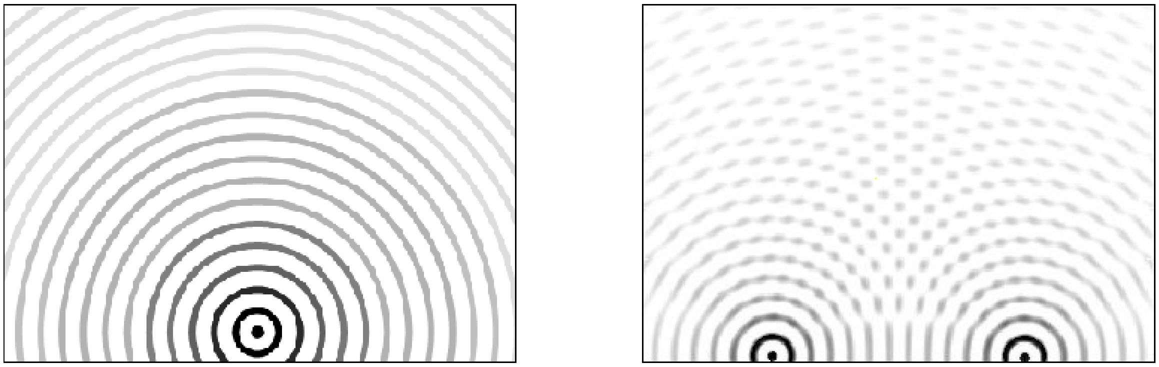


Figure 1-7: Wireless transmission. Left: single point source. Right: interference of two point sources.

Wireless Link

Consider a simple case of a point source radiating electromagnetic waves in all directions (Figure 1-7, left). The received signal strength decreases exponentially with the sender-receiver distance. As with 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 to discern signal from noise reliably. 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-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**. The key insight is that *collisions occur at the receiver*—the sender is not disturbed by concurrent transmissions, but receiver cannot correctly decode the 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. Notice, however, that the interference of simultaneously transmitting sources never disappears—it only is reduced exponentially with an increasing mutual distance (Figure 1-8). The distance (relative to the receiver) at which the 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, if nodes *C* and *E* are transmitting simultaneously their combined interference at *B* may be sufficiently high to cause as great or greater number of errors as a single transmitting node that is within the interference range.

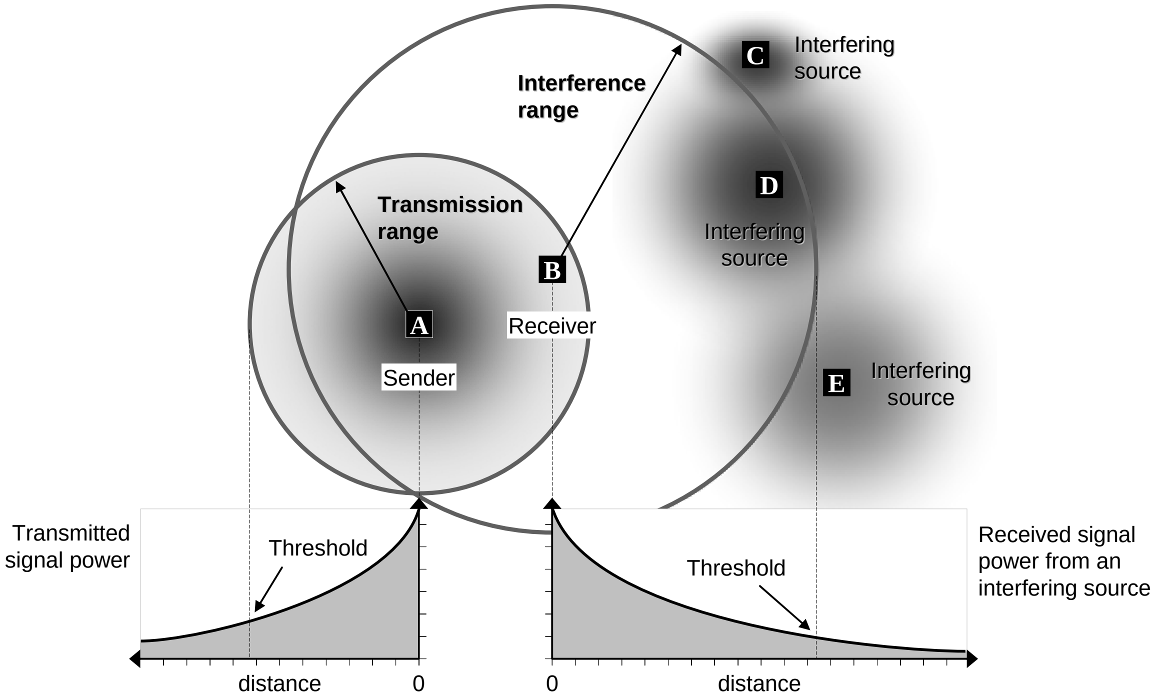


Figure 1-8: Transmission range and interference range for wireless links.

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 notice 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 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. 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. 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.

Different network technologies impose different limits on the size of blocks they can handle, which is known as the **maximum transmission unit** (MTU). For example, a regular Ethernet

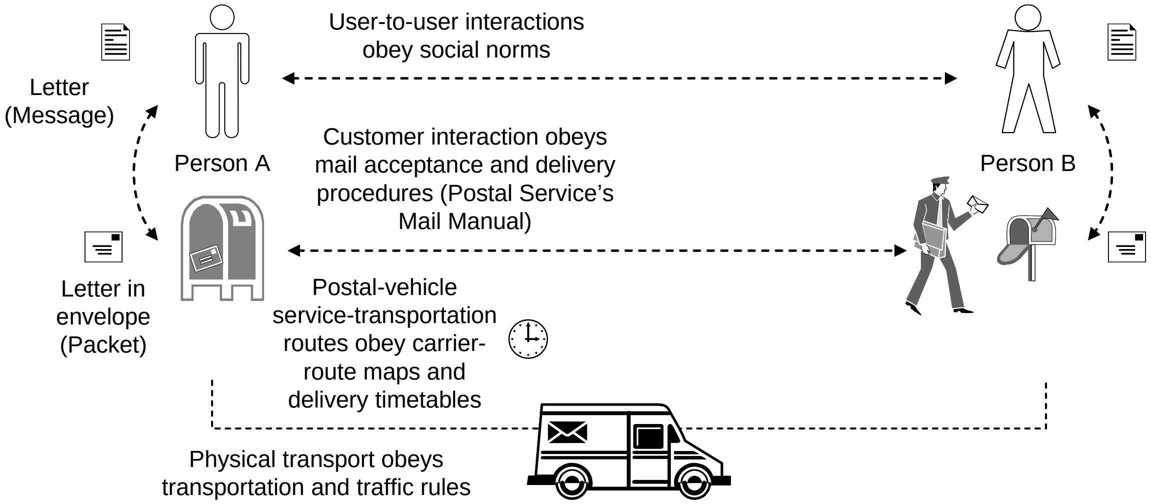

User-to-user interactions obey social norms
Letter (Message)
Person A
Person B
Customer interaction obeys mail acceptance and delivery procedures (Postal Service's Mail Manual)
Letter in envelope (Packet)
Postal-vehicle service-transportation routes obey carrier-route maps and delivery timetables
Physical transport obeys transportation and traffic rules

Figure 1-9: Protocol layers for conventional mail transport.

frame uses a frame format that limits the size of the payload it sends to 1,500 bytes. Notice 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

Link sharing 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. It is hard to imagine accomplishing any task involving multiple entities without relying on a protocol. For example, one could codify the rules governing a purchase of goods by a customer (C) 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

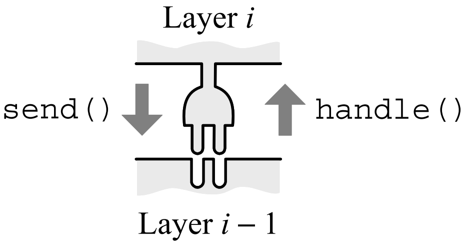
8. C←M Issue confirmation

The customer and merchant may be located remote from each other and using other entities to accomplish the above task.

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. 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 and 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.

Communication in computer networks is very complex. One effective means of dealing with complexity, known as *modular design* with *separation of concerns*. In this approach, the system is split into modules and each module is assigned separate 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 that 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. The layering approach forbids the modules from using services from (or providing to) non-adjacent layers.

Each layer of the layered architecture 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()`. The upper layer also defines a `handle()` callback operation for the lower layer to call it to handle an incoming packet.
2. *Peer interface* to the counterpart protocol on another 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. 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. However, 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.

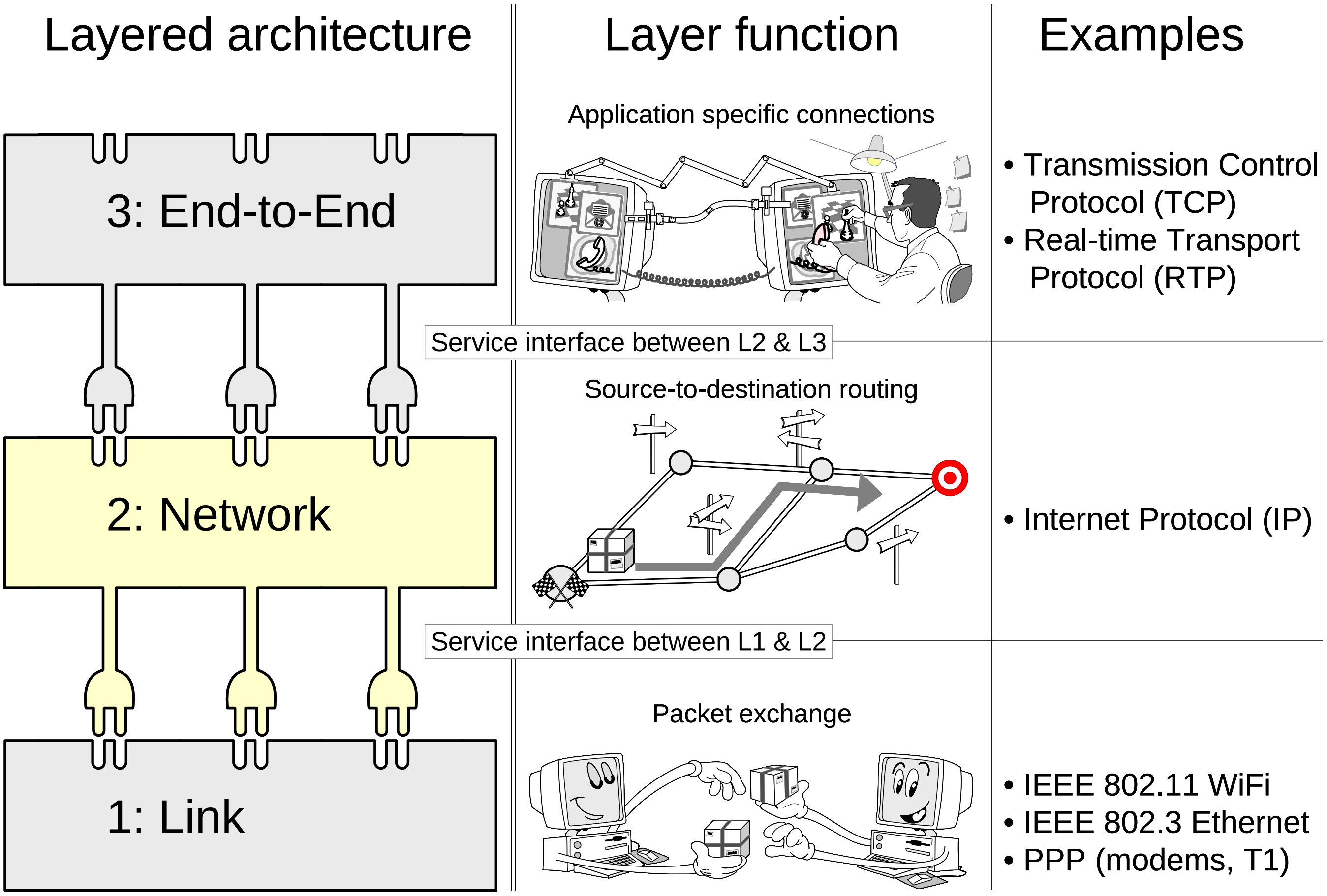
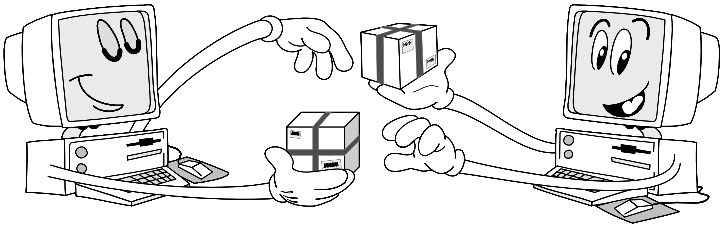

Layered architecture
Layer function
Examples
3: End-to-End
Application specific connections
• Transmission Control Protocol (TCP)
• Real-time Transport Protocol (RTP)
Service interface between L2 & L3
2: Network
Source-to-destination routing
• Internet Protocol (IP)
Service interface between L1 & L2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

Figure 1-10: Three-layer reference architecture for communication protocol layering.

Three-Layer Model

In recent years, the three-layer model (Figure 1-10) has emerged as reference architecture of computer networking protocols.

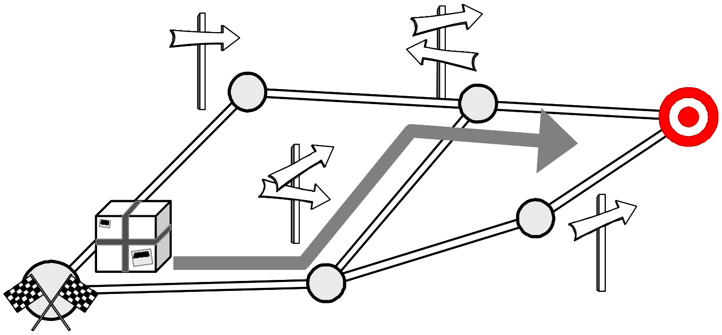


LAYER-1 – Link layer: it implements a packet delivery service between nodes that are attached to the same physical link (or, physical medium). The physical link may be point-to-point from one transmitter to a receiver, or it may be shared by a number of transmitters and receivers (known as "broadcast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

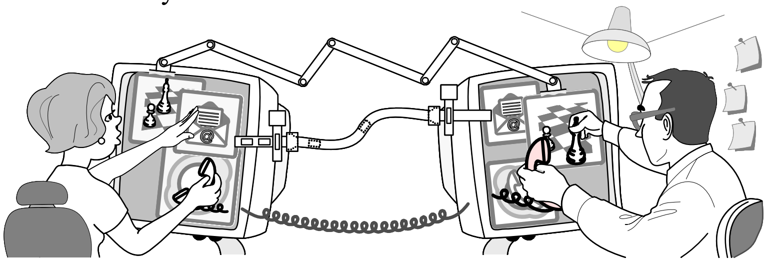
There is the "physical layer," which implements a digital communication link that delivers bits. But, you would not know it because it is usually tightly coupled with the link layer by the link technology standard. Link and physical layer are usually standardized together and technology vendors package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

In wireless networks, the 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. However, because this book is mainly about protocols and not about physical communication, I will keep both together in a single, Link layer.

The link layer is not concerned with bridging multiple end hosts; this is why we need the network layer.



LAYER-2 – Network layer: Combination of links to connect arbitrary end hosts. 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. For example, manipulation-based applications (such as video games) require an equivalent of mechanical links to convey the user's action.

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 be defined to 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 1-11 shows the payers 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. Every layer accepts the payload that is handed to it and processes it to add its characteristic information in the form of an additional header. 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 the protocol stack. Because intermediate nodes are not the final destination (or, end point), they do not have the complete protocol stack, but rather only the two bottommost layers: link and network layers (see Figure 1-11).

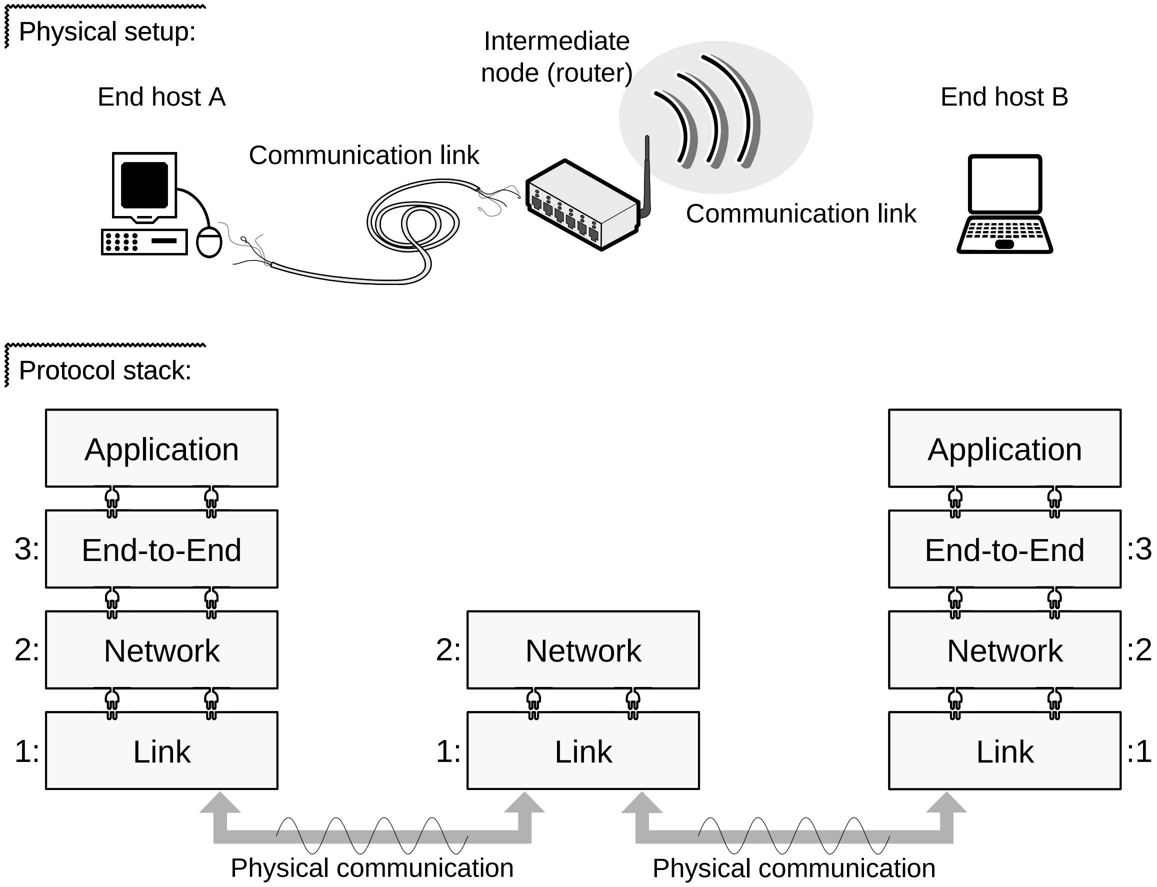


Figure 1-11: Protocol layering in end hosts and intermediate nodes (switches or routers).

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 persistent sender)
    public static final int N;

    // 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 addresses based on final destination addresses
    //         (this object is shared with the routing protocol)
    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, 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 receiver 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) {
            // ...determine which upper layer protocol should 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());
        }
    }

    public void setHandler(
        String receivingProtocol, ProtocolLayer_iUP upperProtocol
    ) {
        // add a <key, value> entry into the routing look-up table
        upperLayerProtocols.put(receivingProtocol, upperProtocol);
    }
```

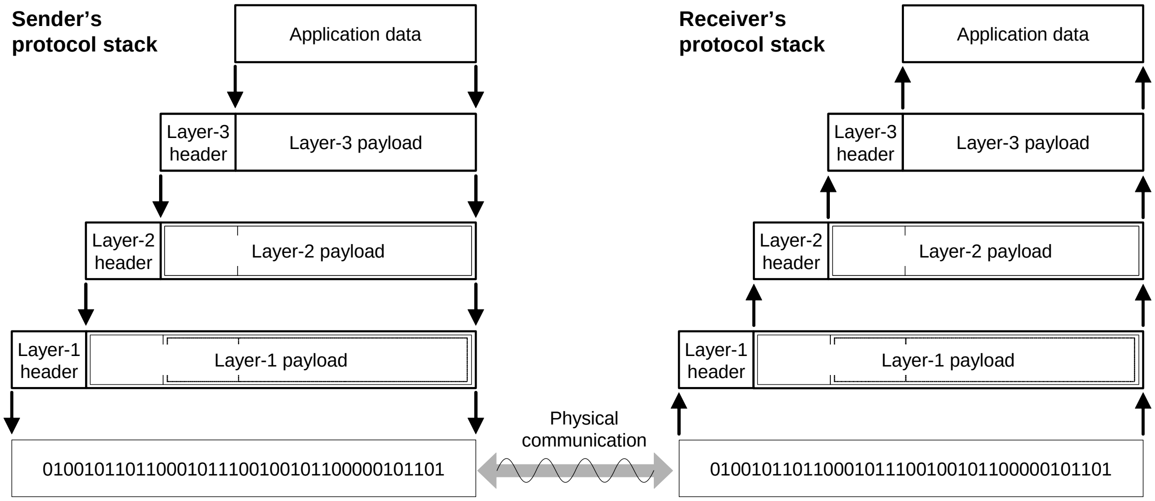


Figure 1-12: Packet nesting across the protocol stack: an entire packet of an upper layer becomes the data payload of a lower layer.

```
55      // Method called by the routing protocol (running in a different thread or process)
56      public void setReceiver(
56a         String destinationAddr, String receiverAddr
56b     ) {
57          // add a <key, value> entry into the forwarding look-up table
58          synchronized (forwardingTable) { // critical region
59              forwardingTable.put(destinationAddr, receiverAddr);
60          } // end of the critical region
61      }
62  }
```

Here I provide only a brief description of the above pseudocode. We will encounter and explain the details later, 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 demultiplexing* and allows the protocol at a lower-layer layer to serve different upper-layer protocols.

To keep the pseudo code manageable, some functionality is not shown in Listing 1-1. 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. 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 8.3 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 1-12). The protocol at a lower layer is *not* aware of any structure in the data passed down

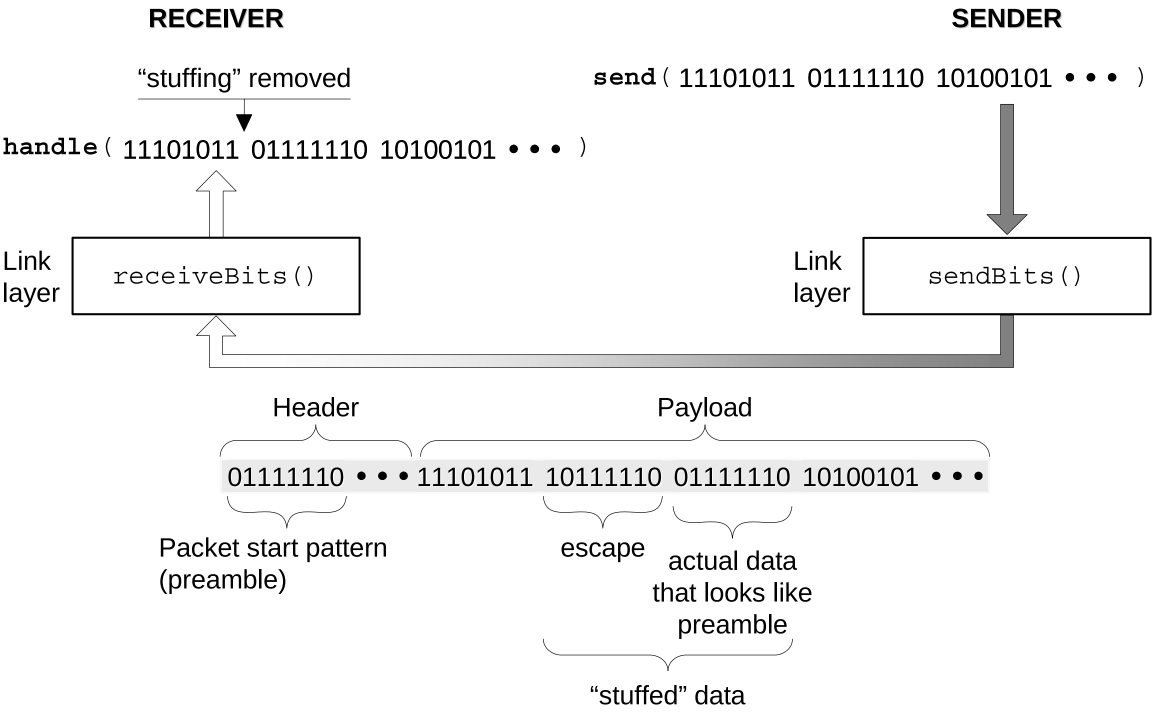


Figure 1-13: Bit stuffing to escape special control patterns in the frame data.

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.

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 the Link layer's `handle()`, which in turn calls the upper-layer (Network) method `handle()`. The Link layer's `send()` method, instead of using a lower-layer service, itself does the sending by calling this layer's own method `sendBits()`.

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 (described at the start of this section). Data transparency means that the link layer must not forbid the upper-layer protocol from sending 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 units used to measure the payload is). Bit stuffing defines a special control escape bit pattern, call it *ESC* (Figure 1-13). 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()`.

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 2.1.1.

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 1-14). 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), 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). Notice that this layer is distinct from the application itself, which provides business logic and user interface.

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 information exchanged between the network nodes. This layer performs *translation* of data representations and formats to support interoperability between different encoding systems (ASCII vs. Unicode) or hardware architectures. 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 exchanges between two hosts (called *session*), to keep track of the progress of their communication. This layer establishes, 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, or error recovery.

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

Layer 2 – Link: Its function is to organize bits into packets or frames, and to provide packet exchange between adjacent nodes.

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

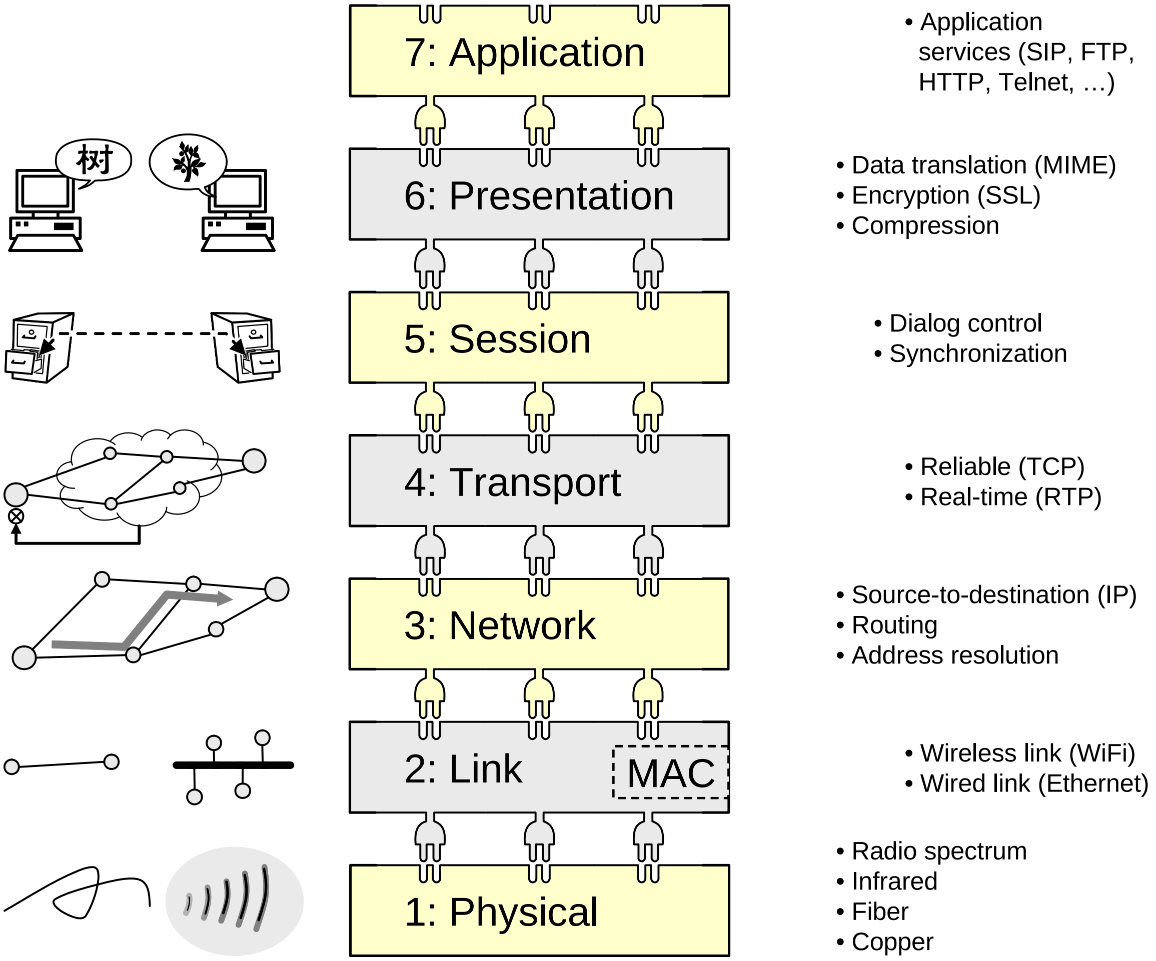


Figure 1-14: OSI reference architecture for communication protocol layering.

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

The seven layers can be conceptually organized to three subgroups. 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. Layers 5, 6, and 7—session, presentation, and application—can be thought of as *user support layers*. They allow interoperability among unrelated software systems. Layer 4, the transport layer, ensures end-to-end reliable data transmission, while layer 2 ensures reliable data transmission on a single link.

When compared to the three-layer model, 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 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, I will mainly use the three-layer model in the rest of this text.

1.2 Reliable Transmission via Redundancy

To counter the line noise, a common technique is to add redundancy or context to the message. For example, assume that the transmitted word is "information" and the received word is "inrtormation." A human receiver would quickly figure out that the original message is "information," because this is the closest meaningful word to the one that is received. Similarly, assume you are tossing a coin and want to transmit the sequence of outcomes (head/tail) to your friend. Instead of transmitting H or T, for every H you can transmit HHH and for every T you can transmit TTT. The advantage of sending two redundant letters is that if one of the original letters flip, say TTT is sent and TTH is received, the receiver can easily determine that the original message is TTT, which corresponds to "tail." Of course, if two letters become flipped, catastrophically, so TTT turns to THH, 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 letters, we can have ten: for every H you can transmit HHHHHHHHHH and for every T you can transmit TTTTTTTTTT. 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 the line can transmit only a limited number of bits per unit of time. 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).

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.

If damage/loss can be detected, then an option is to request retransmission but, request + retransmission takes time ⇒ large 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. Notice 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. The noise amplitude is distributed according to the normal distribution, as shown in [Figure X]. What are the best choices for the codebook?

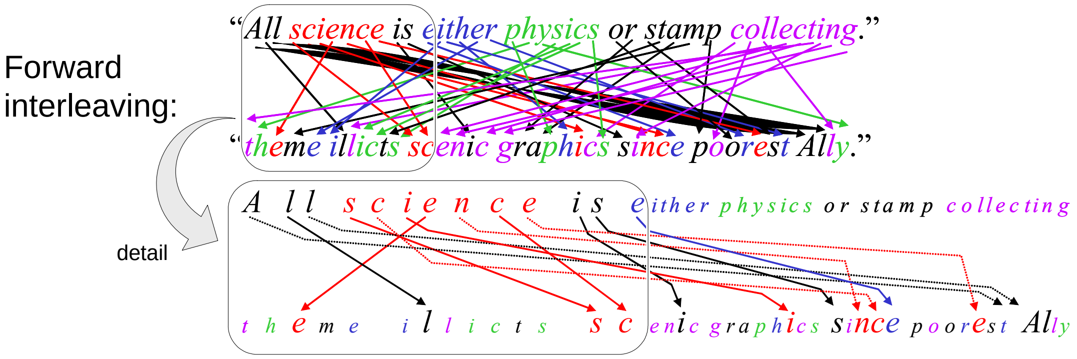
Note: this really represents a *continuous* case, not digital, because numbers are not binary and errors are not binary. But just for the sake of simplicity…

1.2.2 Interleaving

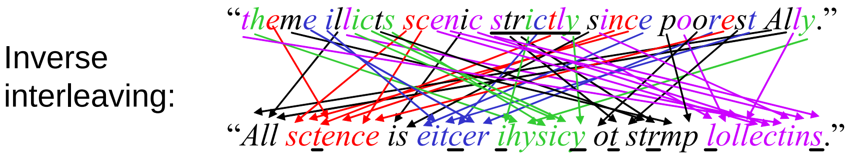
Redundancy and error-correcting codes are useful when errors are randomly distributed. If errors are clustered, they are not effective. Consider the following example. Say you want to send the following message to a friend: "*All science is either physics or stamp collecting*."[2] A random noise in the 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, which operates in the same frequency range as Wi-Fi wireless networking technology.

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. But, 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 infinitely 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 in two ways:

- Packet error: Receiver receives the packet and discovers error via error control
- Packet loss: Receiver never receives the packet

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

Common requirements for a reliable protocol are that: (1) it delivers at most one copy of a given packet to the receiver; 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 is delivered, by sending repeatedly until it receives an acknowledgment. To make retransmission possible, a copy is kept in the transmit buffer (temporary local storage) until it is successfully received by the receiver and the sender received the acknowledgement. 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 is relatively slow.

During network transit, 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?

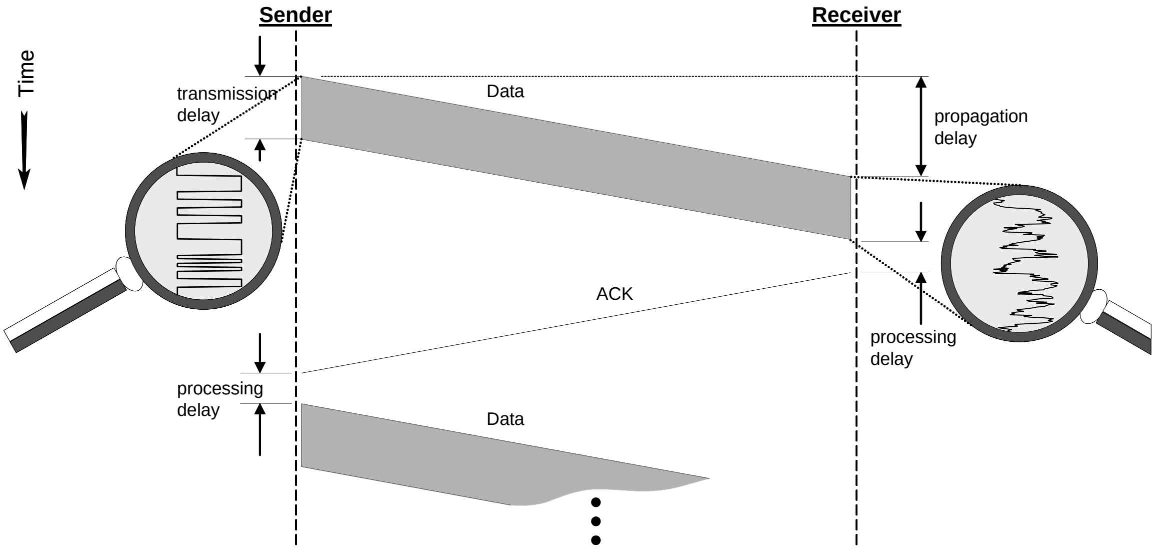


Figure 1-15: Timeline diagram for reliable data transmission with acknowledgements.

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

The ***sender utilization*** of an ARQ connection is defined as the fraction of time that the sender is busy sending data.

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 1-15. 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 the teller's window ("processing delay" or "service delay"), and drive to 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 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. It also depends on the length L of the packet (in bits). Hence, the transmission delay is:

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

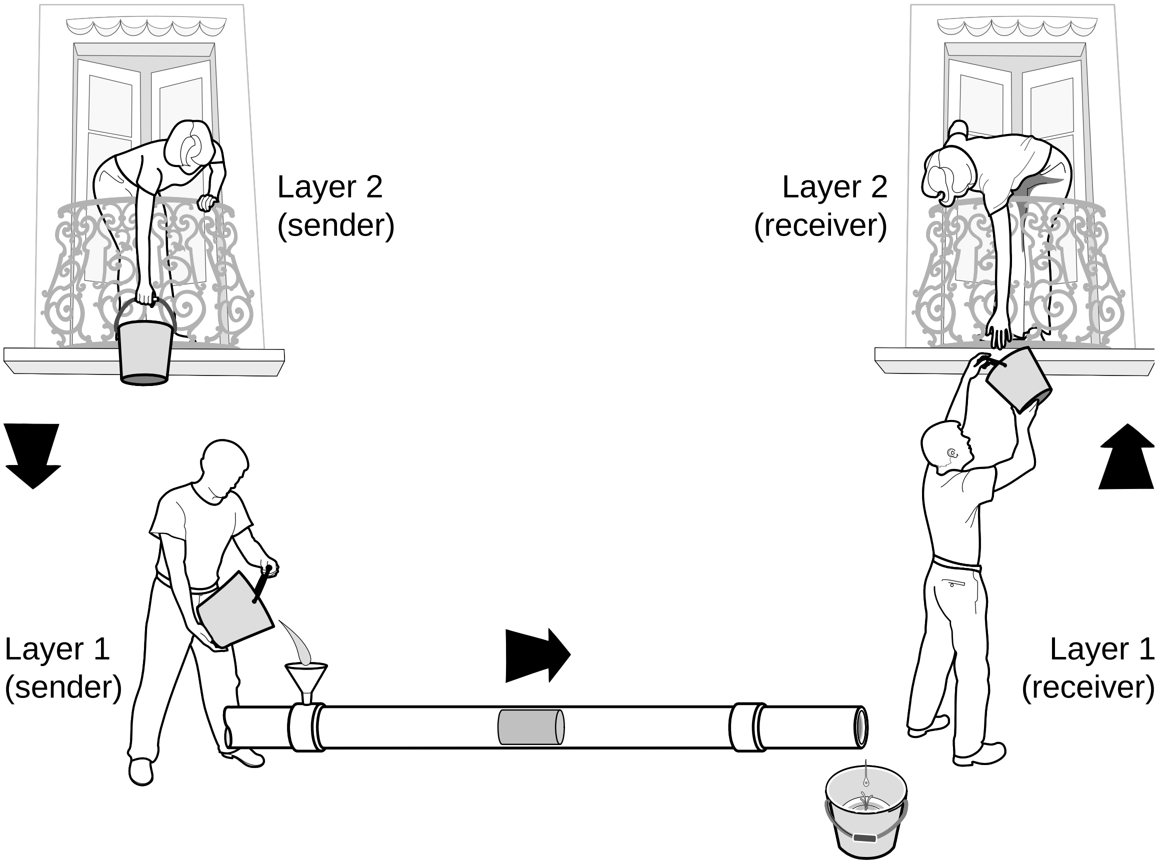


Figure 1-16: Fluid flow analogy for delays in packet delivery between the protocol layers.

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 ($c \approx 3\times10^8$ 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 at the sender if it is immediately bounced back at the receiver. This time on a single transmission link is often assumed to equal RTT =

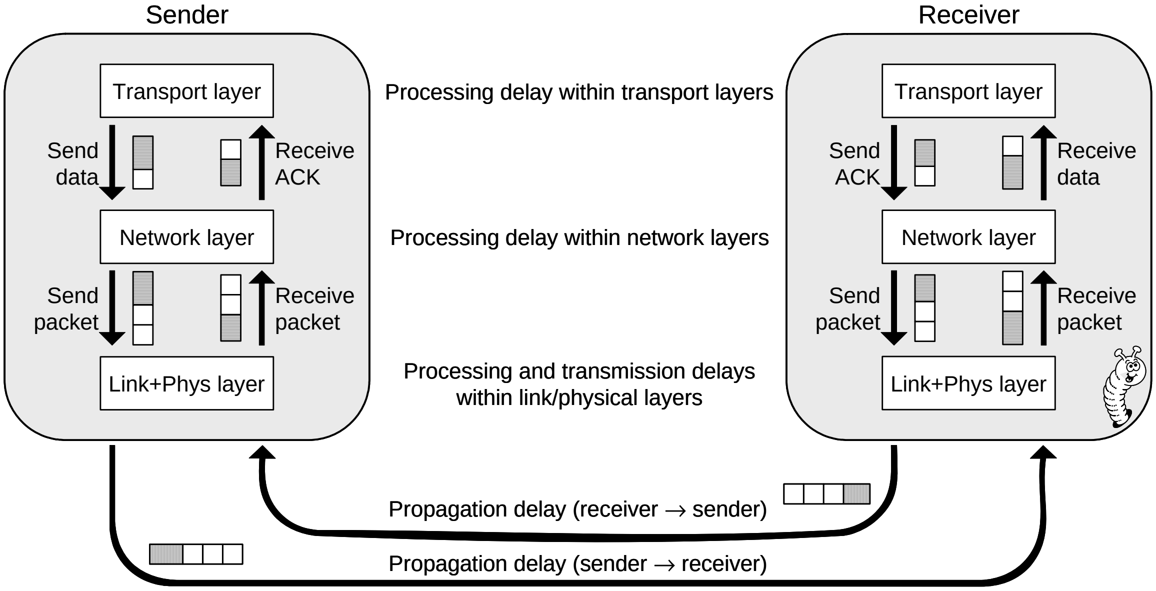

Sender
Receiver
Transport layer
Send data
Receive ACK
Network layer
Send packet
Receive packet
Link+Phys layer
Transport layer
Send ACK
Receive data
Network layer
Send packet
Receive packet
Link+Phys layer
Processing delay within transport layers
Processing delay within network layers
Processing and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

Figure 1-17: Delay components that contribute to round-trip time (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 2.1.2. 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 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 acknowledgement is received, and subtracting these two values:

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

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

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 1-16 how delays are introduced between the protocol layers. The physical-layer (layer 1) receiver waits until the 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 1-17. The 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 the sender's protocol stack may incur significant *processing delays*. Suppose that the sender is at the transport layer and it measures the RTT to receive the acknowledgement from the receiver, which is also at the transport layer. 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 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 a retry-limit is reached. 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 acknowledgement or negative acknowledgement
- Retransmission of a failed packet, which requires storing the packet at the sender until the sender obtains a positive acknowledgement 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 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*. This protocol buffers only a single packet at the sender and does not deal with the next packet before ensuring that the current packet is correctly received (Figure 1-15). A packet loss is detected by the expiration of a timer, which is set when the packet is transmitted.

When the sender receives a corrupted ACK/NAK, it could send back to the receiver a NAK (negative acknowledgement). For pragmatic reasons (to keep the sender software simple), 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 acknowledgement 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 is known as the **expected number of transmissions**. 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 acknowledgement. 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)st attempt will succeed equals:

$$Q(0, k+1) = \binom{k}{0} \cdot (1 - p_{\text{succ}})^k \cdot p_{\text{succ}}^1 = p_{\text{fail}}^k \cdot p_{\text{succ}} \tag{1.7}$$

where k = 1, 2, 3, … . The round in which a packet is successfully transmitted is a random variable N, with the probability distribution function given by (1.7). Its expected value is

$$E\{N\} = \sum_{k=0}^{\infty} (k+1) \cdot Q(0, k+1) = \sum_{k=0}^{\infty} (k+1) \cdot p_{\text{fail}}^k \cdot p_{\text{succ}} = p_{\text{succ}} \cdot \left(\sum_{k=0}^{\infty} p_{\text{fail}}^k + \sum_{k-0}^{\infty} k \cdot p_{\text{fail}}^k \right)$$

Recall that the well-known summation formula for the geometric series is

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

Therefore we obtain (recall that $p_{\text{fail}} = 1 - p_{\text{succ}}$):

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

We can also determine the **average 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 acknowledgement packets can be ignored. A single failed packet transmission takes a total of $t_{\text{fail}} = t_x + t_{\text{out}}$, where t_{out} is the retransmission timer's countdown time. If a packet is successfully transmitted after k failed

[3] This 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 a wireless channel.

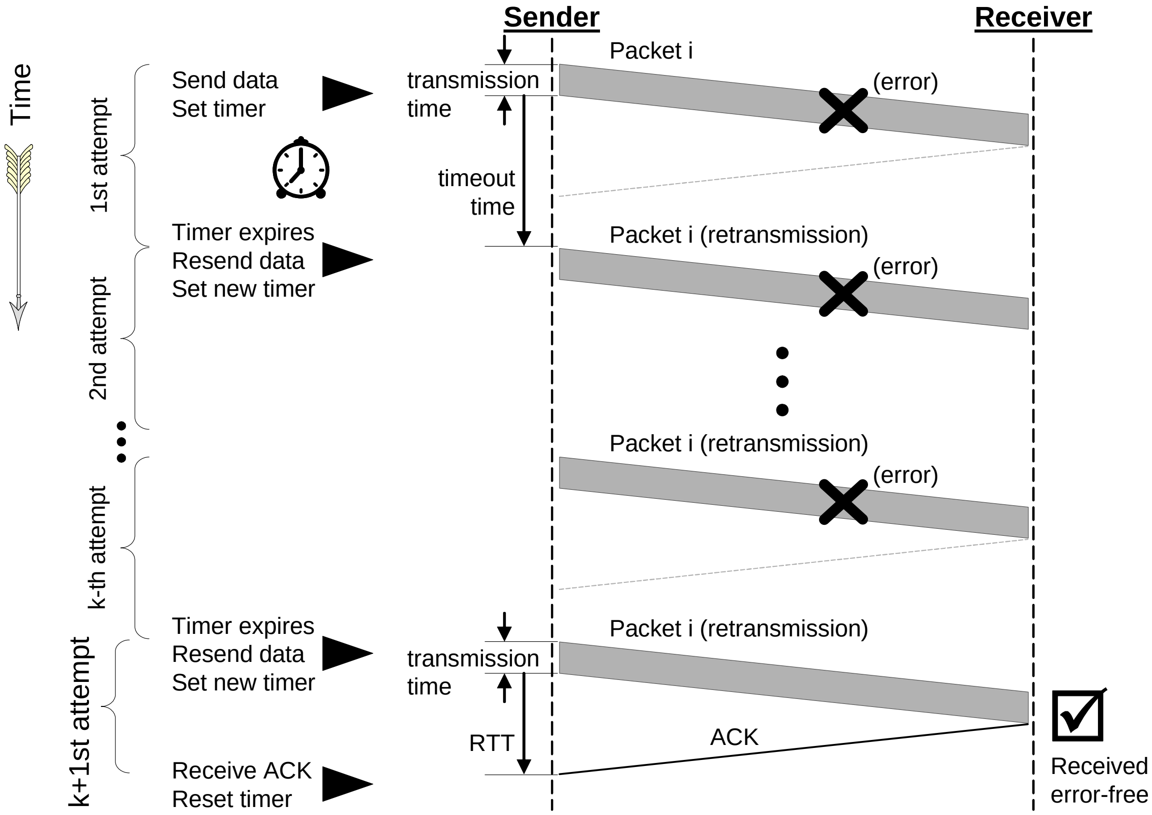


Figure 1-18: Stop-and-Wait with errors. The transmission succeeds after *k* failed attempts.

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 1-18). 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

$$E\{T^{\text{total}}\} = \sum_{k=0}^{\infty} \left(k \cdot t_{\text{fail}} + t_{\text{succ}}\right) \cdot p_{\text{fail}}^{k} \cdot p_{\text{succ}} = p_{\text{succ}} \cdot \left(t_{\text{succ}} \sum_{k=0}^{\infty} p_{\text{fail}}^{k} + t_{\text{fail}} \sum_{k=0}^{\infty} k \cdot p_{\text{fail}}^{k} \right)$$

Following a derivation similar as for Eq. (1.8), we obtain

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

The expected sender utilization in case of a noisy link is

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

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

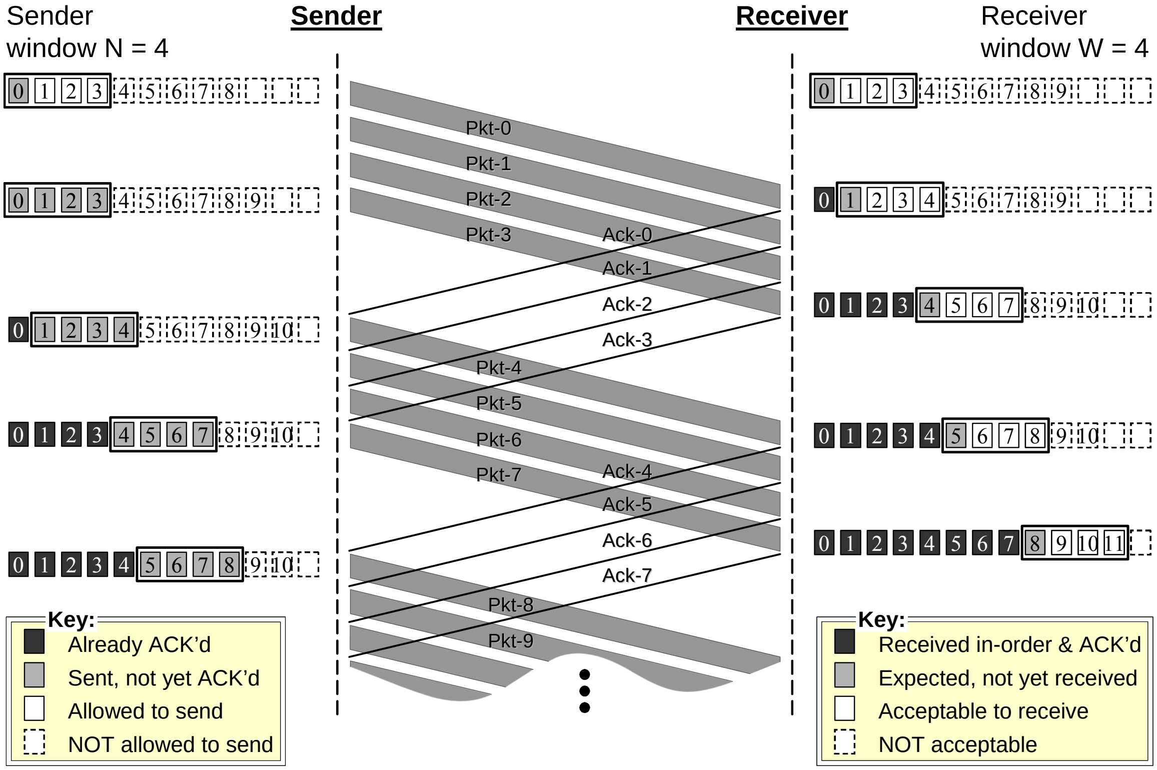


Figure 1-19: Sliding window protocol in operation over an error-free link.

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 acknowledgement. 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*.

The **sender window size** N is a measure of the maximum number of outstanding (i.e., unacknowledged) packets in the network. Figure 1-19 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. In the case shown in Figure 1-19, both sender and receiver have the same window size. In general case it is required that $N \leq W$.

The sliding window sender should store in local memory (buffer) all outstanding packets for which the acknowledgement has not yet been received. 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:

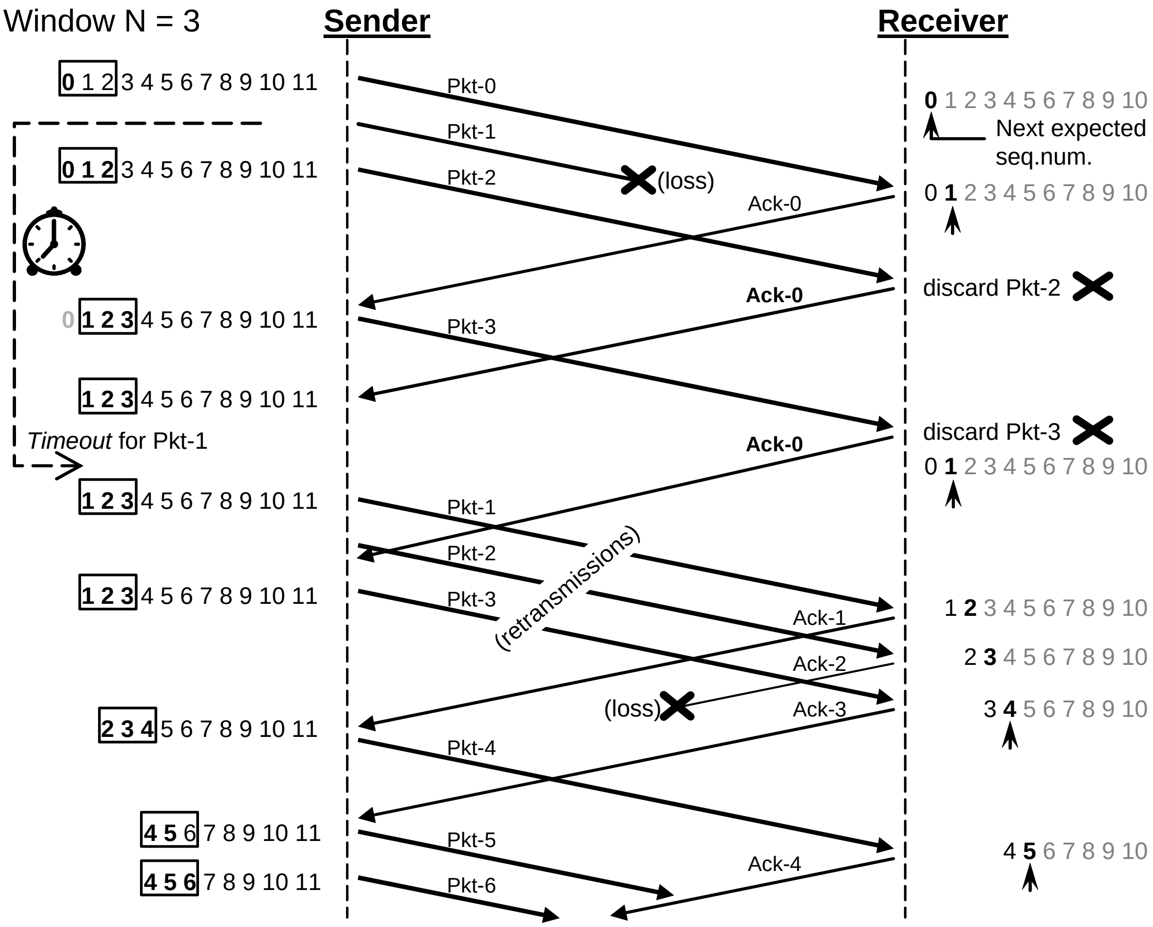


Figure 1-20: Go-back-N protocol in operation under communication errors.

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 deal with communication errors.

Go-back-*N*

The key idea of the Go-back-*N* protocol is to have the receiver as simple as possible. 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 what is the next expected packet (single variable), and does not need any memory to buffer the out-of-order packets.

As with other sliding-window protocols, the Go-back-*N* sender should be able to buffer up to *N* outstanding packets.

The operation of Go-back-*N* is illustrated in Figure 1-20. The sender sends sender-window-size (*N* = 3) packets and stops, waiting for acknowledgements 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 acknowledgement.

Because Pkt-1 is lost, it will never be acknowledged and its retransmission timer will expire. When a timeout occurs, the Go-back-*N* sender resends *all* packets that have been previously sent

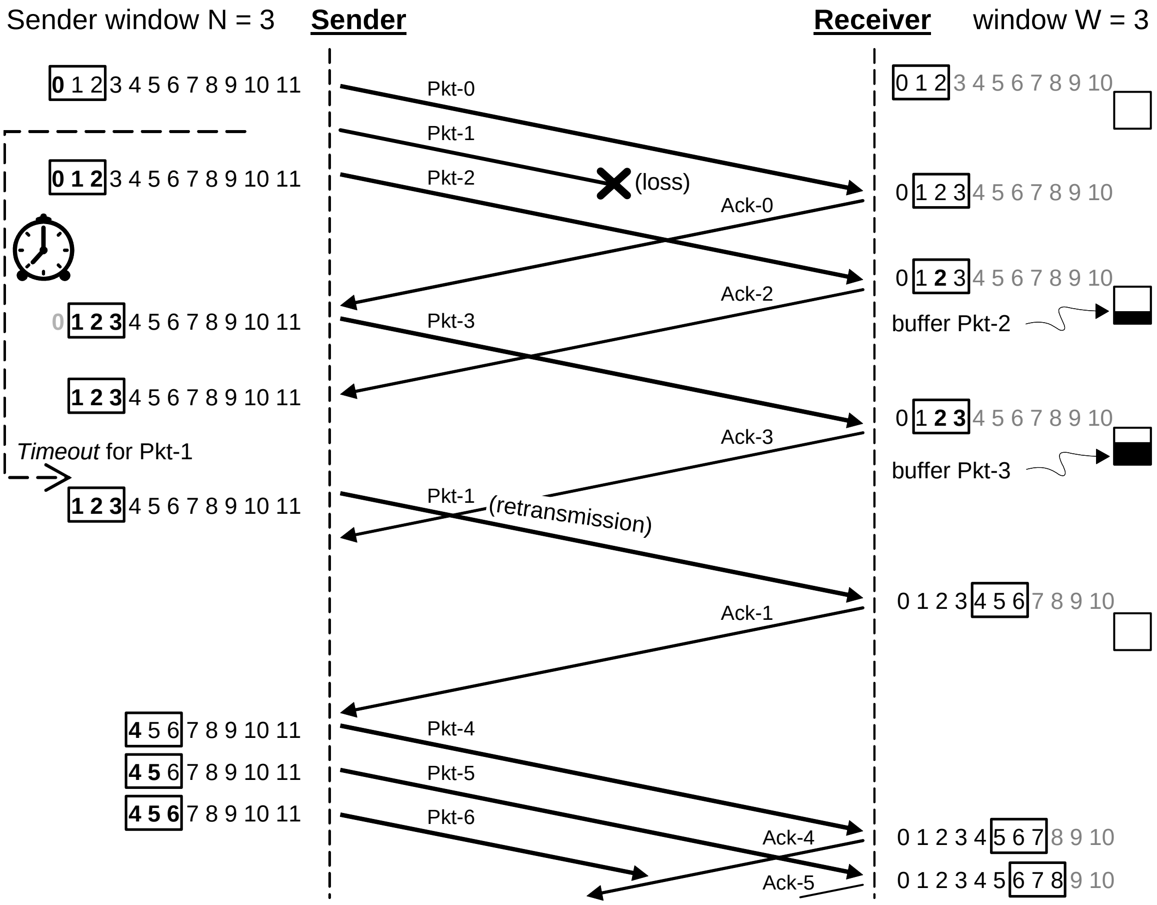


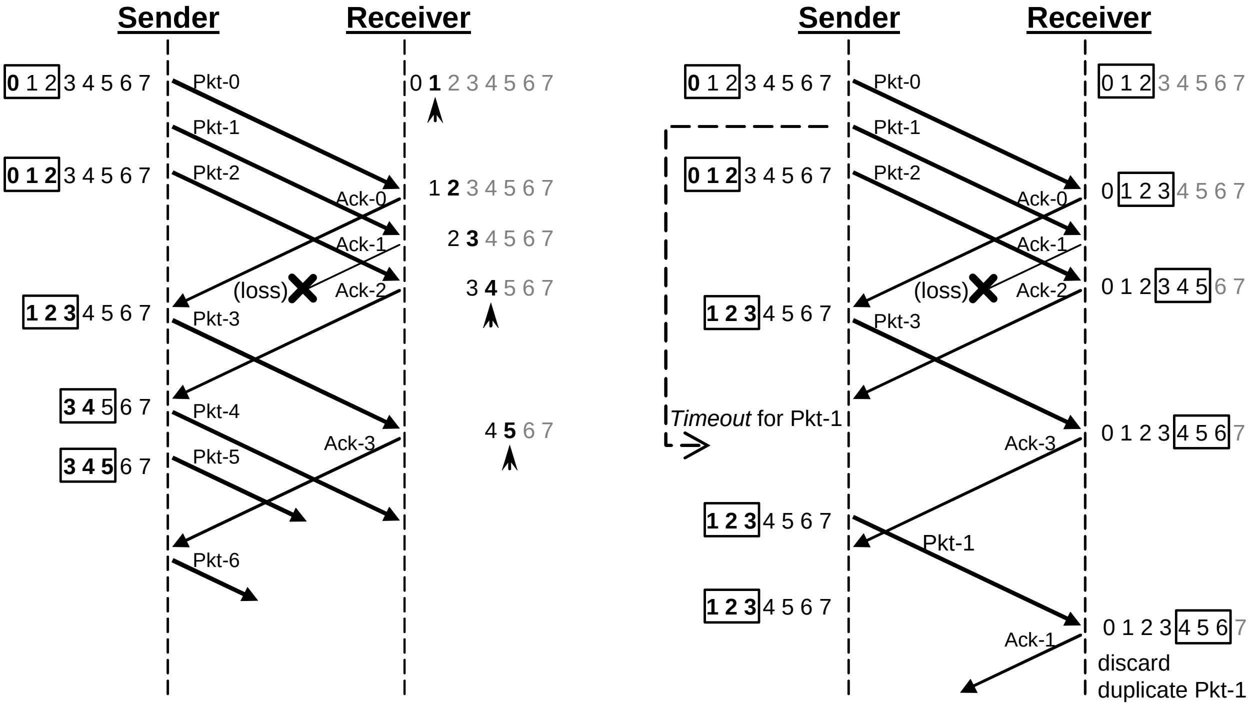
Figure 1-21: Selective Repeat protocol in operation under communication errors.

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 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 receiver memorizes a single variable, which is the sequence number of the next expected packet. The Go-back-*N* 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 this example, 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. One salient feature of Go-back-*N* is that the receiver sends **cumulative acknowledgements**, where an acknowledgement 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 acknowledgement even for incorrectly received packets, but in this case, the previously correctly received packet is being acknowledged. In Figure 1-20, the receipt of Pkt-2 generates a duplicate acknowledgement Ack-0. Notice also that when Ack-2 is lost, the sender takes Ack-3 to



(a) Go-back-N

(b) Selective Repeat

Figure 1-22: Comparison of Go-back-N and Selective-Repeat acknowledgments.

acknowledge all previous packets, including Pkt-2. Hence, a lost acknowledgement does not need to be retransmitted as long as the acknowledgement acknowledging the following packet arrives before the retransmission timer expires.

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 1-21 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 acknowledgements**, where an acknowledgement 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 until the in-order missing packets are received.

Figure 1-22 illustrates the difference between the behaviors of GBN cumulative acknowledgements and SR individual acknowledgements. Notice 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 acknowledgement has been lost. Without an acknowledgement, the sender window would never move forward and the communication would come to a halt. (Notice also that a packet can be retransmitted when its acknowledgement is delayed, so the timeout occurs before the acknowledgement 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 1-22(a), Ack-1 was lost, but when Ack-2 arrives it acknowledges all packets up to and including Pkt-2. This acknowledgement shifts the sender's window forward by 2 and the sender advances uninterrupted. Unlike this, in Figure 1-22(b) the SR sender needs to retransmit Pkt-1 because it never receives Ack-1 before its timeout expired.

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

SIDEBAR 1.1: The Many Faces of Acknowledgements

♦ The attentive reader may have noticed that acknowledgements are used for multiple 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. Notice that an acknowledgment usually only confirms that the data packet arrived error-free to the receiver, but it does not say anything about whether the receiver acted upon the received data and completed the required processing. This requires additional acknowledgement at the application level. Later, in Chapter 2, we will learn about some additional uses of acknowledgements in the TCP protocol.

1.3.3 Broadcast Links

Broadcast links allow connecting multiple network nodes via the same link. 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 a **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.

There are several techniques for transmission coordination on broadcast links. Collisions could be prevented by designing strictly timed transmissions; avoided by listening before speaking; or, detected after they happen and remedied by retransmission of corrupted information. An example of preventing collisions by design is TDMA (Time Division Multiple Access). It creates unique time slots and assigns a different time slot to each node. A node is allowed to transmit only within its assigned time slot. After all nodes are given opportunity to transmit, the cycle is repeated. The problem with this technique is that if some nodes do not have data ready for transmission, their

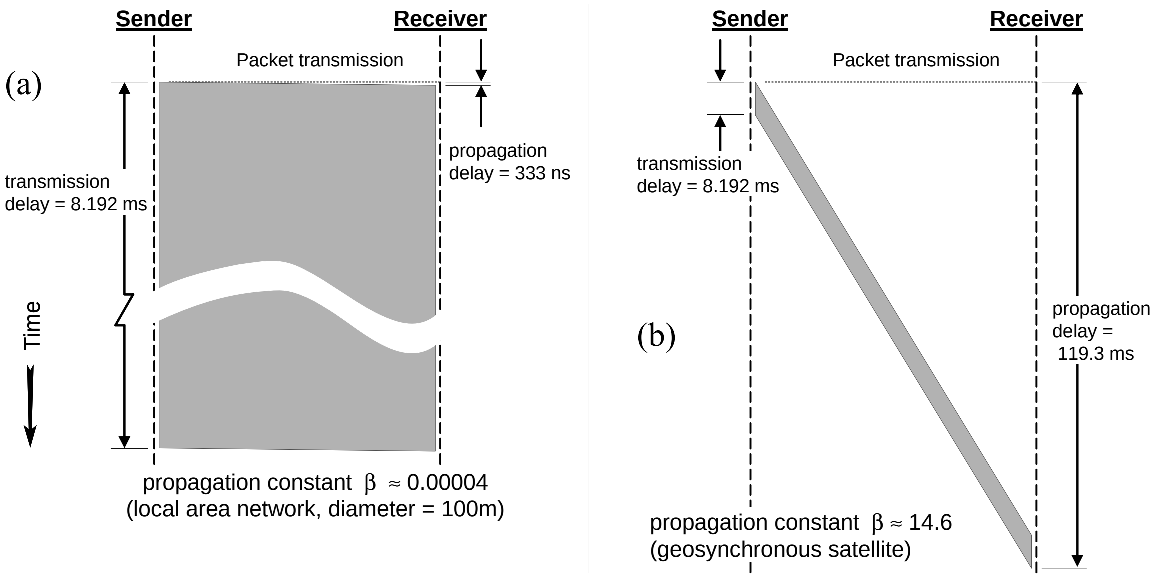
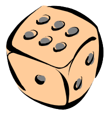

(a)
Sender
Receiver
Packet transmission
transmission delay = 8.192 ms
propagation delay = 333 ns
Time
propagation constant β ≈ 0.00004
(local area network, diameter = 100m)
(b)
Sender
Receiver
Packet transmission
transmission delay = 8.192 ms
propagation delay = 119.3 ms
propagation constant β ≈ 14.6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

Figure 1-23: Propagation constant β for different wireless networks.

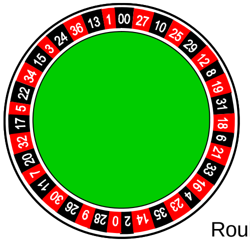
slot goes unused. Any other nodes that may wish to transmit are delayed and have to wait for their predetermined slot even though the link is currently idle.

A popular class of protocols for broadcast links is **random-access protocols**, which are based on the Stop-and-Wait ARQ, with addition of the backoff delay mechanism. **Backoff mechanism** is a key mechanism for coordination of multiple senders on a broadcast link. Stop-and-wait has no reason to backoff because it assumes that the sender is *not* contending for the link against other senders and any loss is due to a transmission error. Conversely, 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 amount of delay (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 for the TCP protocol (Section 2.1.2). The sender usually doubles the range of backoff delays for every failed transmission, which is why this method is also known as **binary exponential backoff**. 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 first deciding how long to wait by tossing a coin (two choices: heads or tails); if both make the same choice and again experience a collision, then they try by rolling a dice (six choices), etc.

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 reason for addressing reliability at the link layer is as follows. A wireless link is significantly more unreliable than a wired one. Noise, interference, and other propagation effects result in the loss of a significant number of frames. Even with error-correction codes, a number of MAC frames may not successfully be received. This situation can be dealt with by reliability mechanisms at a higher layer, such as transport-layer protocol. 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 2.1.2). It is therefore more efficient to deal with errors at the link level and retransmit the corrupted packets.

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 ends is an important parameter. Intuitively, the **parameter β** is the number of bits that a transmitting station can place on the medium before the station furthest away 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

$$\beta = \frac{t_p}{t_x} = \frac{d \cdot R}{v \cdot L} \tag{1.11}$$

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 Kbytes packet equals t_x = 8.192 ms. The relationship is illustrated in Figure 1-23(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 is done with 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 1-23, the respective β parameters for these networks are $\beta_{\text{LAN}} \approx 0.00004$ and $\beta_{\text{GSS}} \approx 14.6$. The time taken by the electronics for detection should also be added to the propagation time when computing β, but it is usually ignored as negligible. We will see later how parameter β plays an important role in network design.

The ALOHA Protocol

Problems related to this section: Problem 1.13 → ?

A simple protocol for broadcast media is called ALOHA. There are two versions of ALOHA: *pure* or *plain ALOHA* transmits packets as soon as they become ready, and *slotted ALOHA* which transmits packets only at regular intervals. The state diagram for the sender side of both variations of the protocol is shown in Figure 1-24. 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. After transmission, the sender stops-and-waits for the acknowledgement. If the acknowledgement arrives, this is the end of the current cycle, and the sender expects the next packet to become available for sending. If the acknowledgement does not arrive, the sender assumes that this is because collision happened and it increases its backoff interval. The **backoff interval** is the

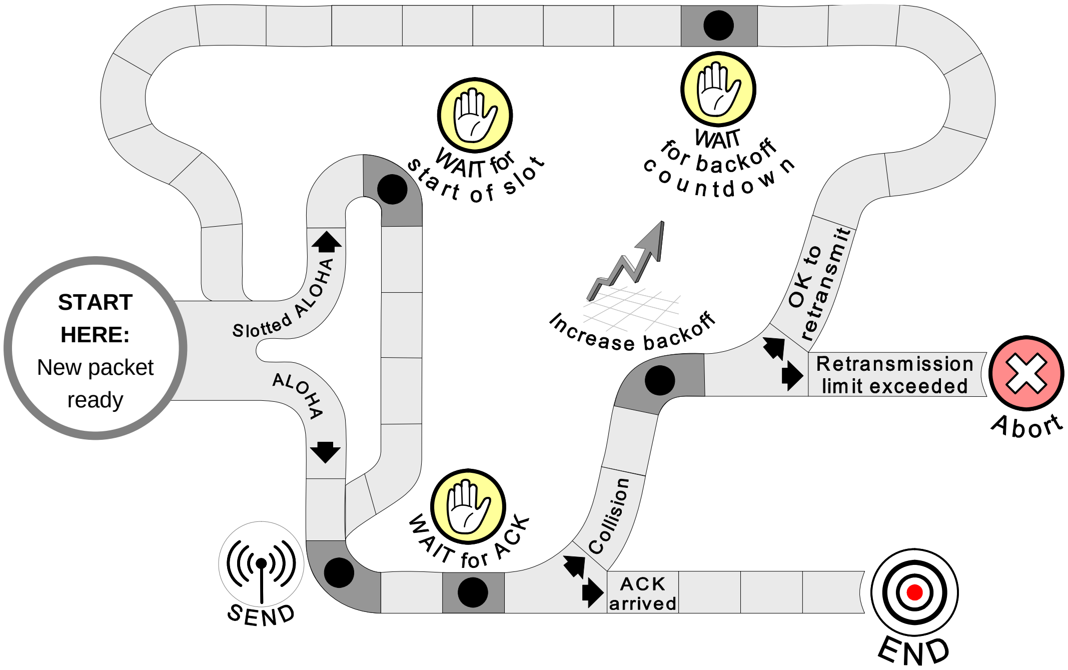


Figure 1-24: The sender's state diagram for ALOHA and Slotted ALOHA protocols.

amount of time the sender waits to reduce the probability of collision with another sender that also has a packet ready for transmission. After waiting for backoff countdown, the sender repeats the cycle and retransmits the packet. As we know from the earlier discussion, the sender does not persist forever in resending the packet, and if it 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, except for the backoff interval. ALOHA also does not initiate transmission of the next packet before ensuring that the current packet is correctly received. Let us first consider a pure ALOHA protocol. To derive its throughput, we make the following assumptions:

- There are a total of m wireless nodes and each node generates new packets for transmission according to a Poisson process (see Appendix) with rate λ/m.
- Each node can hold only a single packet at a time, and the packet is stored until the node receives a positive acknowledgement that the packet is successfully transmitted. While storing the packet, the node is said to be *backlogged*.
- When a new packet is generated, what happens to it depends on whether or not the node is already backlogged. If the node it is backlogged, the newly generated packet is discarded; if the node is *not* backlogged, the newly generated packet is immediately transmitted (in pure ALOHA) and stored until acknowledgement is received.
- A backlogged node can retransmit at any moment with a certain probability.
- All packets have the same length. The time needed for packet transmission (transmission delay) is called *slot* length, and it is normalized to equal 1.
- 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

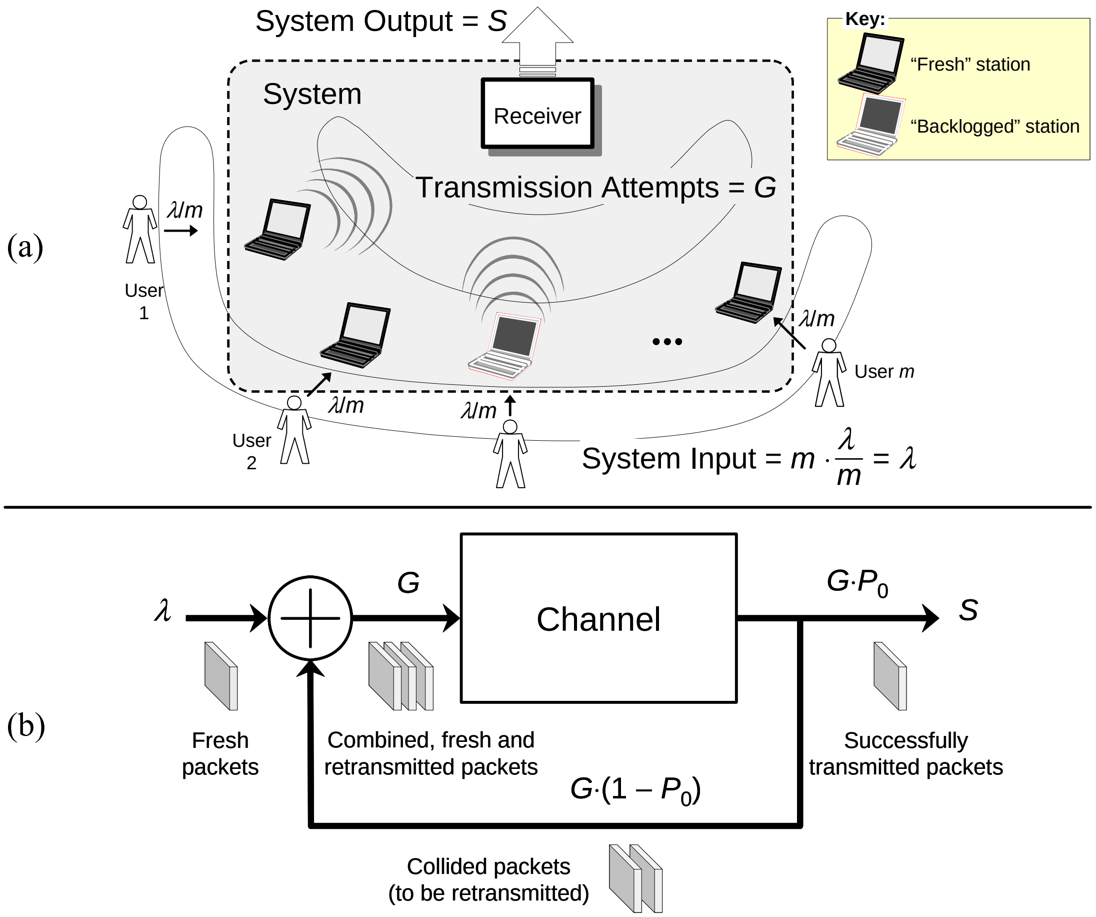


Figure 1-25: (a) ALOHA system representation. (b) Modeled as a feedback system.

packets will be lost; all nodes receive *instantaneous feedback* about the success or failure of the transmission. In other words, the acknowledgement is received *immediately* upon packet transmission, without any propagation delays.

This system can be modeled as in Figure 1-25. For a reasonable throughput, we would expect $0 < \lambda < 1$ because the system can successfully carry at most one packet per slot, i.e., only one node can "talk" (or, transmit) at a time. Also, for the system to function, the departure rate of packets out from the system should equal the arrival rate in equilibrium. 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.

The following simplified derivation yields a reasonable approximation. 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 $G > \lambda$.

The probability of successful transmission (i.e., throughput S) is the probability of an arrival times the probability that the packet does not suffer collision; because these are independent events, the joint probability is the product of their probabilities. The probability of an arrival is

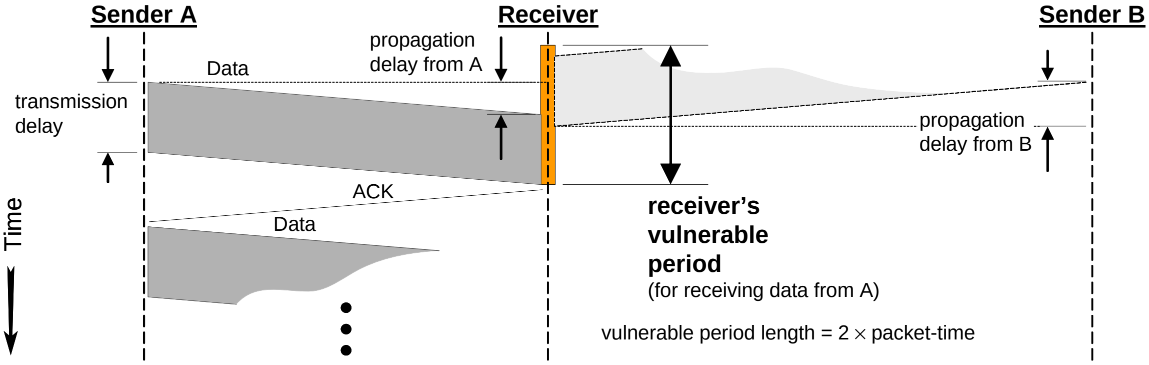


Figure 1-26: The receiver's vulnerable period during which collisions are possible.

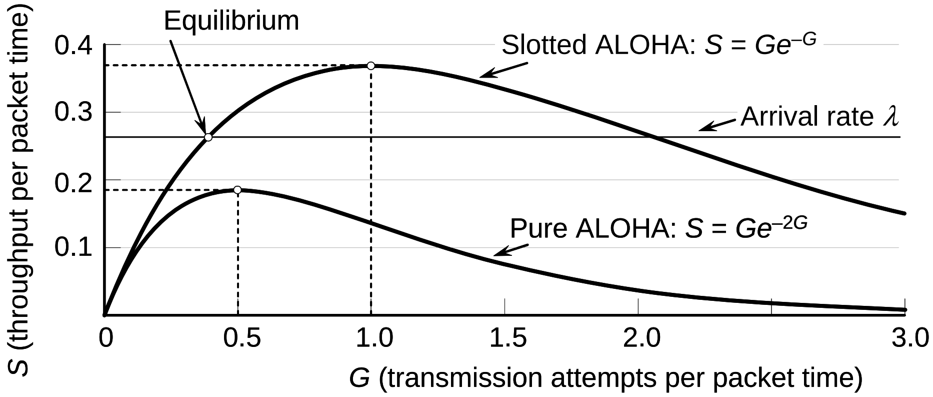


Figure 1-27: Efficiency of the ALOHA MAC protocol. (In the case of Slotted ALOHA, the packet time is equal to the *slot time*.)

$P_a = \tau \cdot G$, where $\tau = 1$ is the slot duration and G is the total arrival rate on the channel (new and backlogged packets, combined).

The packet will not suffer collision if no other senders have transmitted their packets during the so-called "vulnerable period" or "window of vulnerability." We define the receiver's **vulnerable period** as the time during which other transmissions may cause collision with the sender's transmission. For pure ALOHA, the vulnerable period is two slots long $[t - 1, t + 1)$, as illustrated in Figure 1-26. Any transmission that started within one packet-time before this transmission or during this transmission will overlap with this transmission and result in a collision. For slotted ALOHA, the vulnerable period lasts one slot $[t, t + 1)$, assuming that all stations are synchronized and can start transmission only at slot intervals. From the Poisson distribution formula (see Appendix A), $P_0 = P\{\mathrm{A}(t + \tau) - A(t) = 0\}$. With $\tau = 1$ for slotted ALOHA, we have

$$S = P_a \cdot P_0 = (1 \cdot G) \cdot P\{A(t+1) - A(t) = 0\} = G \cdot e^{-G} \tag{1.12}$$

For pure ALOHA, $\tau = 2$, so $S = G{\cdot}e^{-2G}$. In equilibrium, the arrival rate (system input), λ, to the system should be the same as the departure rate (system output), $S = G{\cdot}e^{-G}$. The reader should recall Figure 1-25(a), and this relationship is illustrated in Figure 1-27.

We see that for slotted ALOHA, the maximum possible throughput of $1/e \approx 0.368$ occurs at $G = 1$. This is reasonable, because if $G < 1$, too many idle slots are generated, and if $G > 1$, too many collisions are generated. At $G = 1$, the packet departure rate is one packet per packet time (or, per

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*.

slot), the fraction $1/e$ of which are newly arrived packets and $1-\frac{1}{e}$ are the successfully retransmitted backlogged packets.

Carrier Sense Multiple Access Protocols (CSMA)

Problems related to this section: Problem 1.15 → ?

The key problem with the ALOHA protocol is 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 retransmits with a retransmission probability.

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 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 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 keep listening until it becomes idle or stop listening and try later

(2) Whether to transmit immediately upon finding the channel idle or slightly delay the transmission

Upon finding the channel busy, the node might listen persistently until the end of the ongoing transmission. Another option is to listen periodically. 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 hold 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 remove it, because both nodes might hold their transmissions for the same amount of

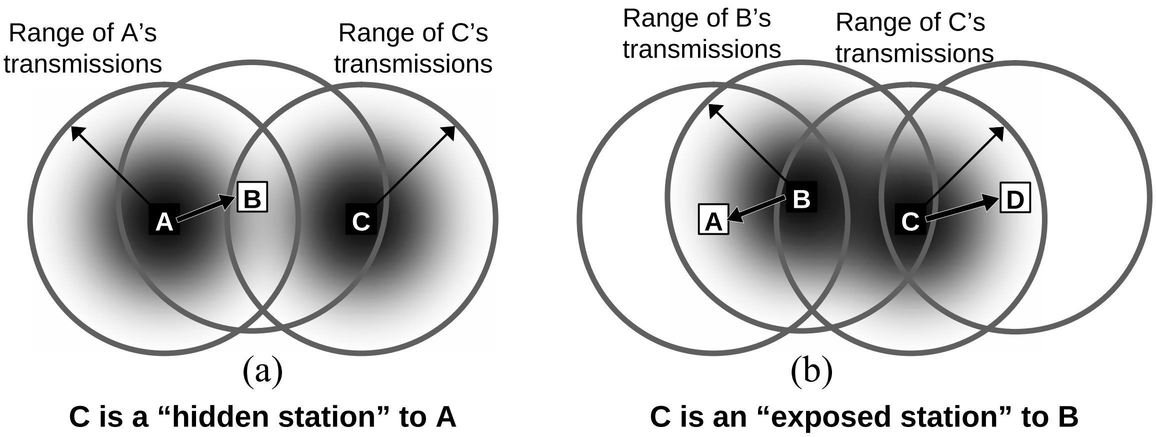

Range of A's transmissions
Range of C's transmissions
A
B
C
(a)
C is a "hidden station" to A
Range of B's transmissions
Range of C's transmissions
A
B
C
D
(b)
C is an "exposed station" to B

Figure 1-28: (a) Hidden station problem: *C* cannot hear *A*'s transmissions. (b) Exposed station problem: *C* defers transmission to *D* because it hears *B*'s transmission.

time. Several CSMA protocols that make different choices regarding the listening and transmission start are shown in Table 1-1. For each of the 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) and repeats the listening-and-transmission procedure.

The efficiency of CSMA is better than that of ALOHA because of the CSMA's shorter vulnerable period: The stations will not initiate transmission if they sense a transmission already in progress. Notice 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 delays than 1-persistent CSMA.

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 is simply due to the exponential propagation loss of the radio signal, as discussed earlier in Section 1.1.2. 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.

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 1-28(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*, wiping out the frame 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 can hear at most one transmitter at a time. In the **exposed station problem**, station *C* defers transmission to *D* because it hears *B*'s transmission, as illustrated in Figure 1-28(b). If *C* senses the medium, it will hear an ongoing transmission 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.

Hidden and exposed station problems arise only for CSMA-type protocols. ALOHA, for instance, does not suffer from such problems because it does not perform channel sensing 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 carrier-sensing mechanism essentially becomes useless. With exposed stations it becomes worse because carrier sensing prevents the exposed stations from transmission, where ALOHA would not mind the busy channel.

CSMA/CD

Problems related to this section: ? → ?

Persistent and nonpersistent CSMA protocols are clearly an improvement over ALOHA because they ensure that no station begins to transmit when it senses the channel busy. Another improvement is for stations to abort their transmissions as soon as they detect a collision[4]. Quickly terminating damaged packets saves time and bandwidth. This protocol is known as *CSMA with Collision Detection*, or CSMA/CD, which is a variant of 1-persistent CSMA. It works as follows (Figure 1-29):

[4] In networks with wired media, the station compares the signal that it places on the wire with the one observed on the wire to detect collision. If these signals are not the same, a collision has occurred.

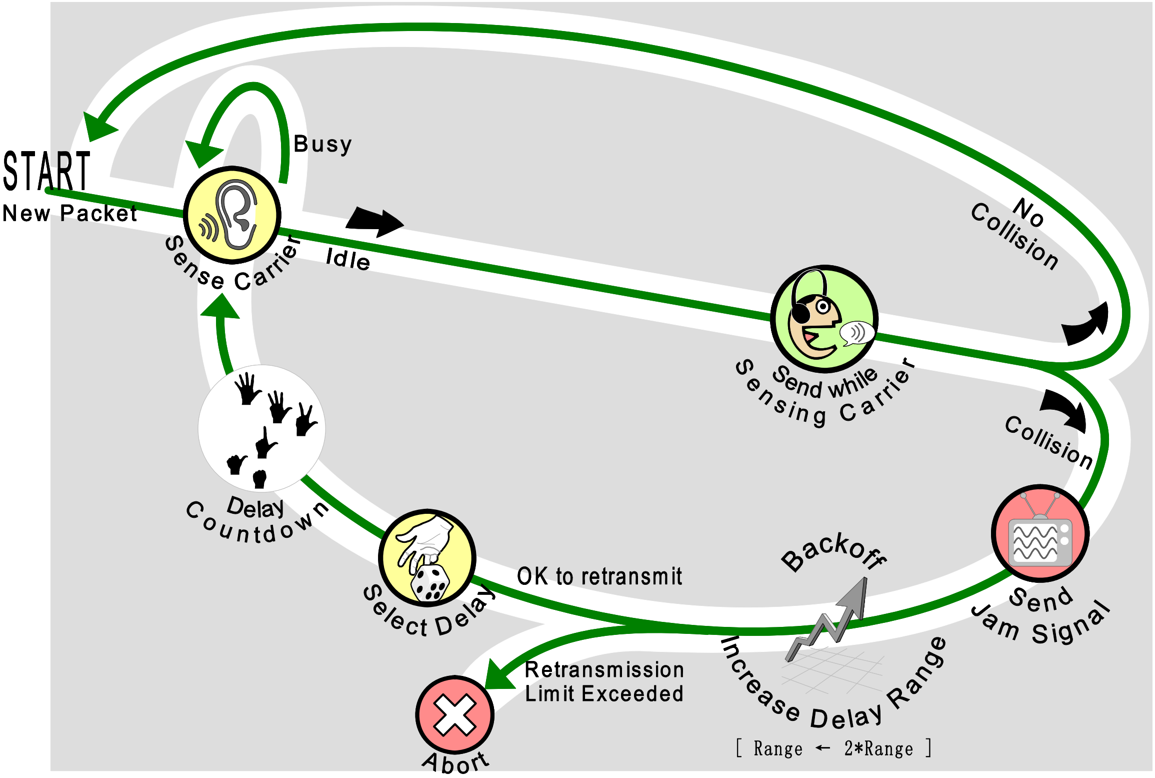


Figure 1-29: The sender's state diagram for CSMA/CD protocol.

1. Wait until the channel is idle.
2. When the channel is idle, transmit immediately *and* sense the carrier during the transmission (or, "listen *while* talking").
3. If you detect collision, abort the ongoing packet 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**) before its transmitted signal has had time to propagate to all stations on the CSMA/CD 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 noticed the signal and to be deferring to it. The time to acquire the medium is thus based on the round-trip propagation time. If the station transmits the complete frame successfully and has additional data to transmit, it will again listen to the channel before attempting a transmission (Figure 1-29).

The collision detection process is illustrated in Figure 1-30. At time t_0 both stations are listening ready to transmit. The CSMA/CD protocol requires that the transmitter detect the collision before it has stopped transmitting its frame. Therefore, the transmission time of the smallest frame must be larger than one round-trip propagation time, i.e., 2β, where β is the propagation constant described in Figure 1-23. The station that detects collision must transmit a **jam signal**, which carries a special binary pattern to inform the other stations that a collision occurred. The jam pattern consists of 32 to 48 bits. The transmission of the jam pattern ensures that the collision lasts long enough to be detected by all stations on the network.

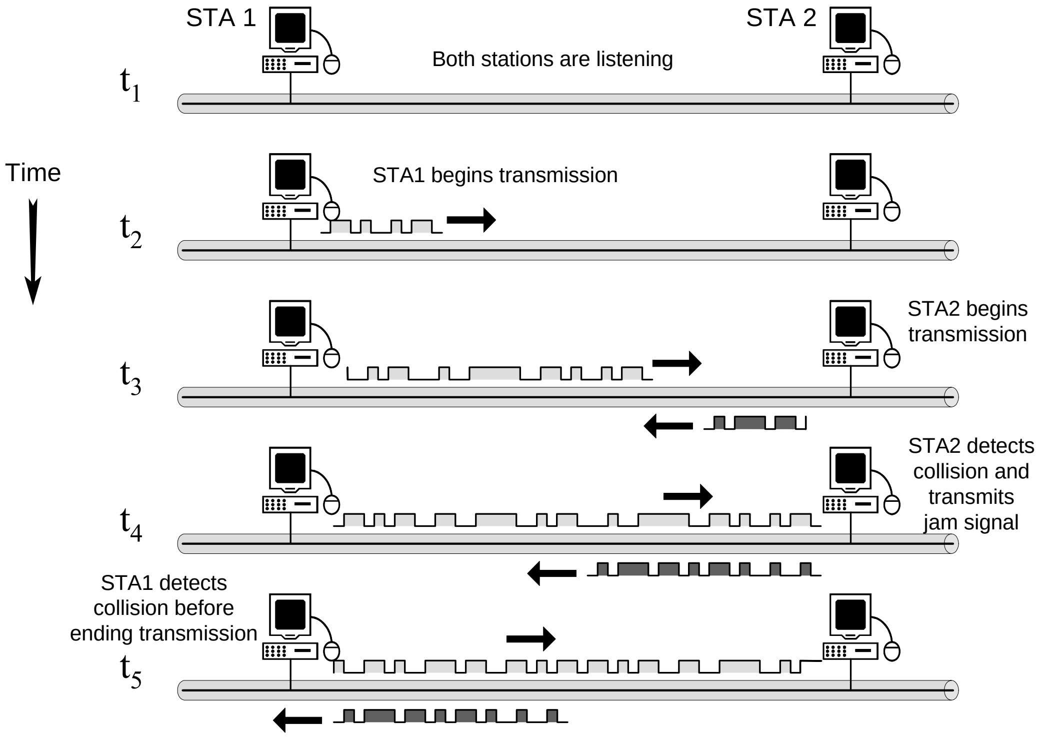

STA 1
STA 2
Both stations are listening
t_1
Time
STA1 begins transmission
t_2
STA2 begins transmission
t_3
STA2 detects collision and transmits jam signal
t_4
STA1 detects collision before ending transmission
t_5

Figure 1-30: Collision detection by CSMA/CD stations.

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 what it reads back is different from what it is putting out, it knows that a collision is occurring.

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. As the number of retransmission attempts increases, the number of possibilities for the choice of delay increases. The backoff range is usually *truncated*, which means that after a certain number of increases, the retransmission timeout reaches a ceiling and the exponential growth stops. For example, if the ceiling is set at k=10, then the maximum delay is 1023 slot times. In addition, as shown in Figure 1-29, 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 frame. The sender resets its backoff parameters and retransmission counters at the end of a successful transmission or if the transmission is aborted.

Notice that CSMA/CD achieves reliable transmission without acknowledgements. If the sender does not detect collision, this means that the sender has not detected any errors during the transmission. Therefore, it simply assumes that the receiver received the same signal (i.e., the frame was received error free), and there is no need for an acknowledgement.

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 1-29. At the end of a previous transmission, station-1 and station-3 each have one frame to transmit, while

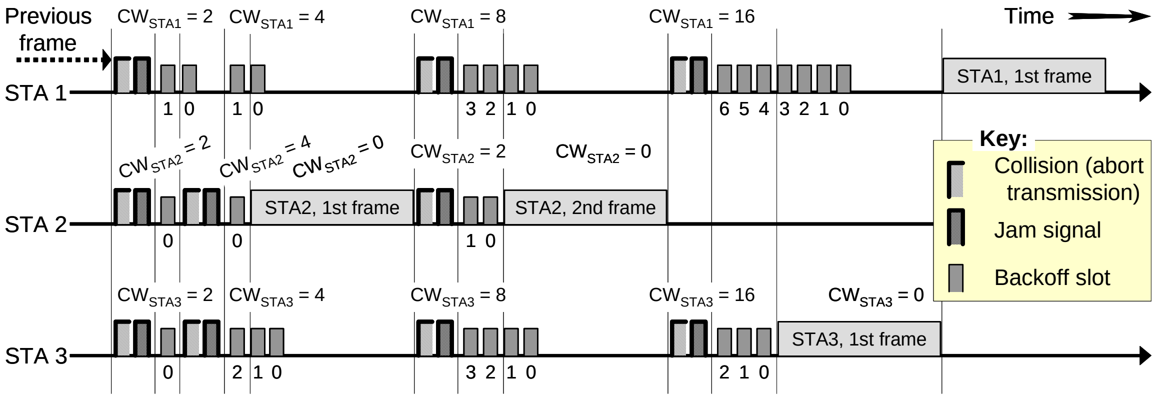


Figure 1-31: Example of three CSMA/CD stations transmission with collision and backoff.

station-2 has two frames. Assume that all frames 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. Show the timing diagram and indicate the contention window (CW) sizes.

The solution is shown in Figure 1-31. 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 frame 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 frames 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 frame.

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 slot. If each station transmits during a contention slot with probability p, the probability A that some station will acquire the channel in that slot is

$$A = m \cdot p \cdot (1 - p)^{m-1}$$

A is maximized when $p = 1/m$, with $A \to 1/e$ 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 the station will suffer collision ($j - 1$) times as succeed on the j^{th} attempt (i.e., that the contention interval has exactly j slots in it) is $A \cdot (1 - A)^{j-1}$. Therefore, the average number of slots per contention is given as the expected value

$$\sum_{j=0}^{\infty} j \cdot A \cdot (1 - A)^{j-1} = \frac{1}{A}$$

Because each slot is $2\cdot\beta$ long, the mean contention interval, w, is $2\cdot\beta/A$. Assuming optimal p, 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 frame takes $t_x = L/R$ seconds to transmit, then the channel efficiency is

$$\eta_{\mathrm{CSMA/CD}} = \frac{L/R}{L/R + 2\cdot\beta/A} = \frac{1}{1+2\cdot\beta\cdot e\cdot R/L} \tag{1.13}$$

where A is substituted with the optimal value $1/e$.

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 1.18 → Problem 1.21

In wireless LANs, it is not practical to do collision detection because of two main reasons:

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 loss, where the signal drops 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 loss we have the following problem. The fact that the transmitting station senses the medium free does not necessarily mean that the medium is free around the receiver area. (This is the known as the *hidden station problem*, as described in Figure 1-28.)

As a result, when a station transmits a frame, it has no idea whether the frame collided with another frame or not until it receives an acknowledgement from the receiver (or times out due to the lack of an acknowledgement). In this situation, collisions have a greater effect on performance than with CSMA/CD, where colliding frames 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 1-32 shows the sender's state diagram. After a frame was 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 at least $2\times\beta$, ensuring that the station is always capable of determining if another station has accessed the medium at the beginning of the previous slot. The time interval between frames (or, packets) needed for the carrier-sense mechanism to determine that the medium is idle and available for transmission is called **backoff slot**.

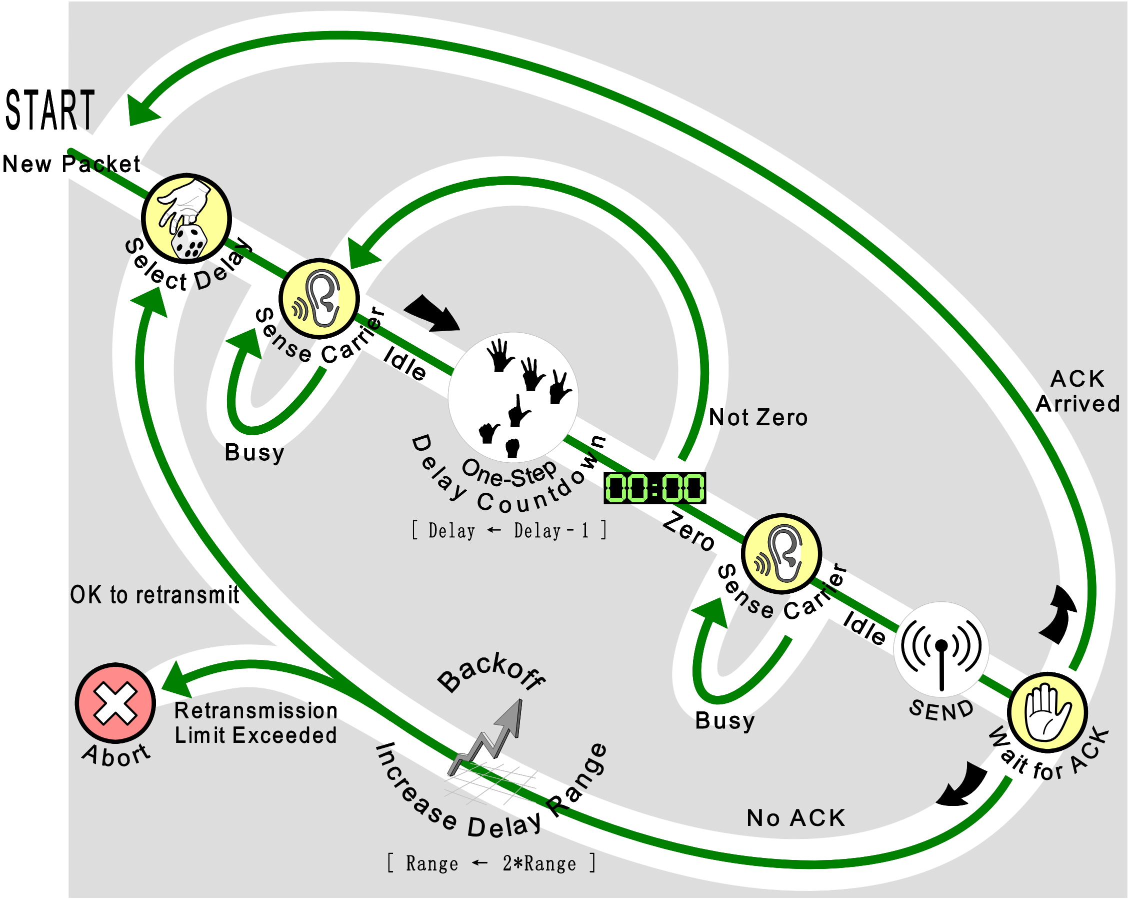


Figure 1-32: The sender's state diagram for CSMA/CA protocol.

When a station wants to transmit data, 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. Notice that unlike CSMA/CD (Figure 1-29), CSMA/CA station performs carrier sensing after every slot counted down, i.e., it is listening during the contention window (Figure 1-32). 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 frame after it counts down to zero.

After transmitting a frame, the station waits for the receiver to send an ACK. If no ACK is received, the frame 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*). 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 p or 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 frame completes transmission. In this way, stations that happen to choose a longer timer value get higher priority in the next round of contention.

As it can be seen, CSMA/CA deliberately introduces delay in transmission in order to avoid collision. Avoiding collisions increases the protocol efficiency in terms of the percentage of frames that get successfully transmitted (useful throughput). Notice that 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

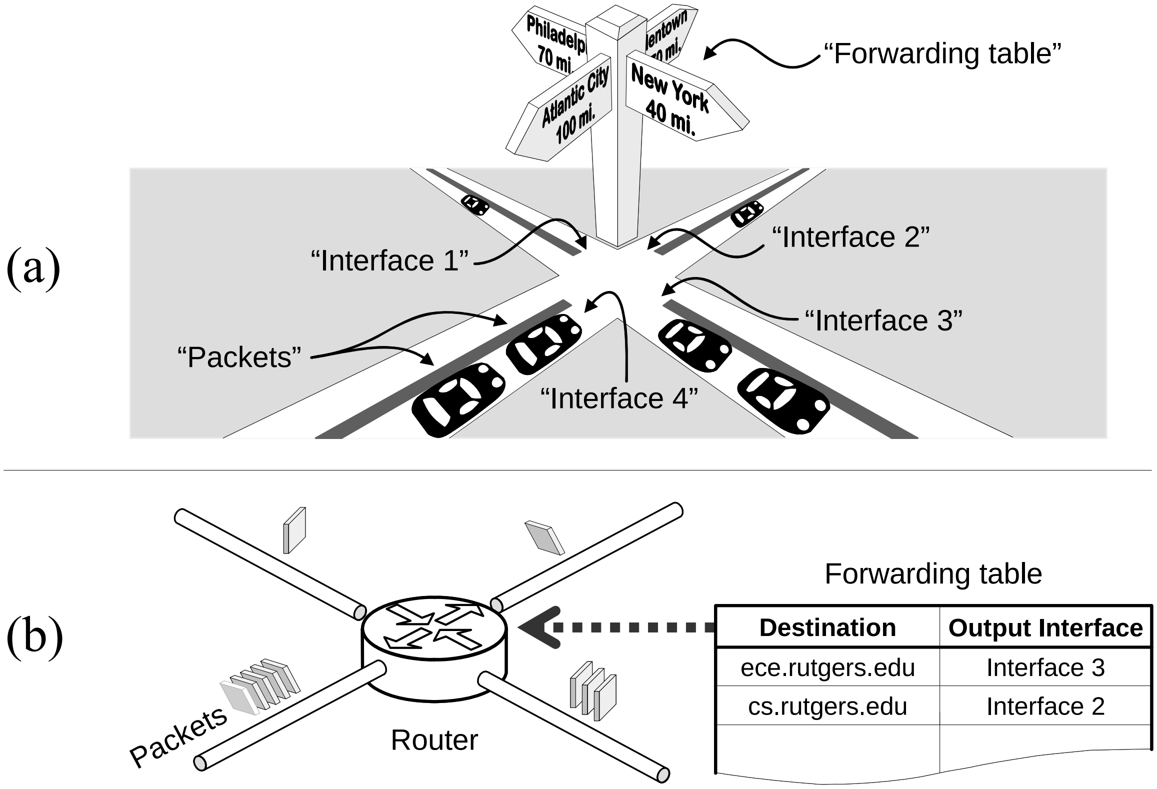
In general networks, arbitrary source-destination node pairs communicate via intermediary network nodes. These intermediary nodes are called switches or routers and their main purpose is to bring packets to their destinations. A good routing protocol will also do it in an efficient way, meaning via the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

Bridges, Switches, and Routers

Two general approaches are used to interconnect multiple networks: *bridges* or *routers*. **Bridges** are simple networking devices that are used for interconnecting local area networks (LANs) that use identical protocols for the 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. Notice that there are more sophisticated bridges, which are capable of mapping from one link-layer format to another.

A **packet switch** is a network device with several incoming and outgoing links that forwards packets from incoming to outgoing links. The appropriate outgoing link is decided based on the packet’s guidance information (contained in the packet header). Routers are general-purpose devices that can interconnect arbitrary networks. A **router** has two important functions: (1) *routing*, which is the process of finding and maintaining optimal paths between source and destination nodes; and, (2) *forwarding* (or *switching*), which is the process of relaying incoming data packets along the routing path. A router is a switch that builds its forwarding table by routing algorithms. Routing often searches for the shortest path, which in abstract graphs is a graph distance between the 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

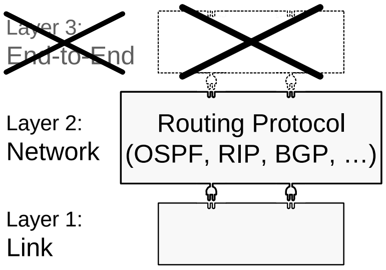
Figure 1-33: A router can be thought of as a crossroads, with connecting points corresponding to the router interfaces. When a car (packet) enters the intersection, it is directed out by looking up the forwarding table.

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

Figure 1-33 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, as the road signs on different road intersections list different information depending on the intersection’s location relative to the roadmap, so the routing tables in different routers list different information depending on the router’s location relative to the rest of the network.

A **router** is a network device that interconnects two or more computer networks, where each network may be using a different link-layer protocol. 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 optimal paths, known as *routing algorithms*.

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 = 100;

    // 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, 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;
            // update my routing table based on the received report
            synchronized (routingTable) { // critical region
            } // 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 …

As will be seen later, routing is not an easy task. 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 is viewed as overhead because it carries control information and reduces the resources available for carrying user information. 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. 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 and/or degraded performance of the applications.

Path MTU is the smallest maximum transmission unit of any link on the current path (also known as route) between two hosts. The concept of MTU is defined in Section 1.1.3.

This section deals mainly with the control functions of routers that include building the routing tables. Later, in Section 4.1 we will 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 packet delivery. Also, due to the limited size of the 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, i.e., with no intermediaries. A network of networks is called *internetwork*.

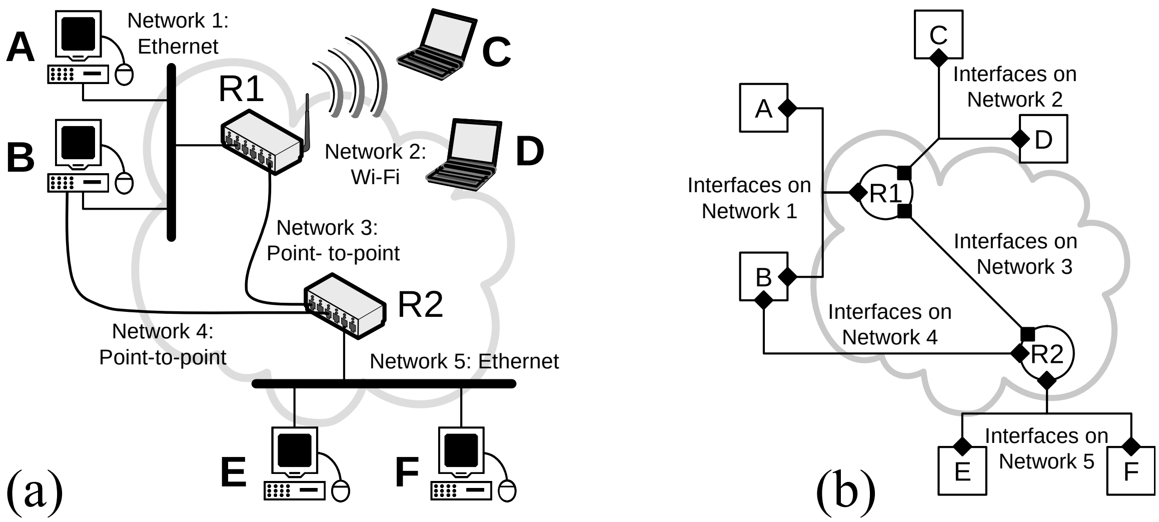


Figure 1-34: Example internetwork: (a) The physical networks include 2 Ethernets, 2 point-to-point links, and 1 Wi-Fi network. (b) Topology of the internetwork and interfaces.

Consider an example internetwork in Figure 1-34(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 frame format, and each format has a limit on how much data can be sent in a single frame (link MTU, Section 1.1.3).

Two types of network nodes are distinguished: 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 1-34(a), it is not intended to be used for transit traffic. Hosts usually do not participate in the routing algorithm. Unlike hosts, *routers* have the primary function of relaying transit traffic from other nodes. Each router has a minimum of two, but usually many more, network interfaces. In Figure 1-34(a), both routers R1 and R2 have three network attachment points (interfaces) each. Each interface on every host and router must have a network address that is globally unique.[5] A node with two or more network interfaces is said to be **multihomed**[6] (or, multiconnected). Notice that multihomed hosts do *not* participate in routing or forwarding of transit traffic. Multihomed hosts act as any other end host, except they may use different interfaces for different destinations, depending on the destination distance.

The whole idea behind a network layer protocol is to implement the concept of a "virtual network" where devices 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 data link layer

[5] This is not necessarily true for interfaces that are behind NATs, as discussed later.

[6] Most notebook computers nowadays come with 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 addresses and they are simultaneously active on their respective physical networks.

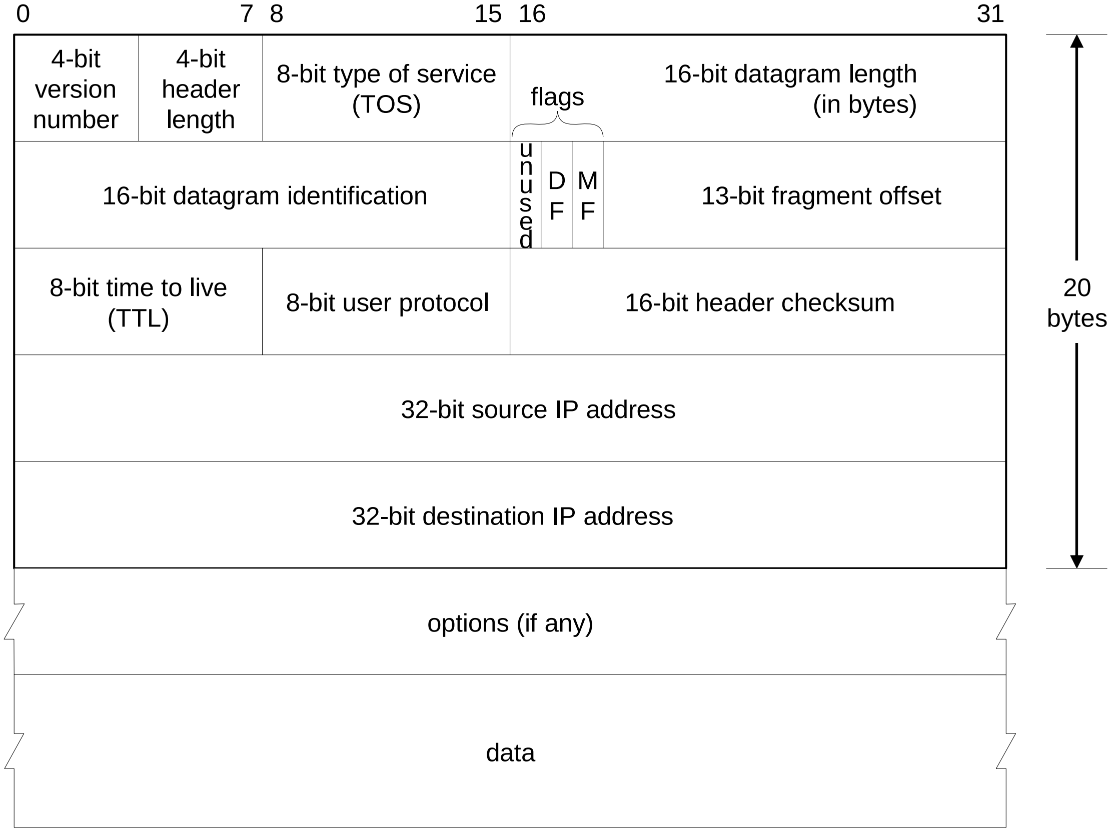
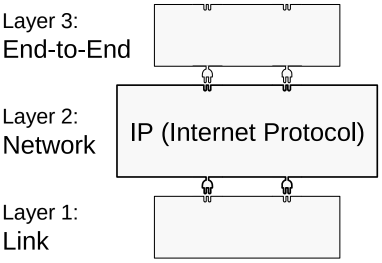


Figure 1-35: The format of IPv4 datagrams.

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),[7] 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 8.1.



IP Header Format

Data transmitted over an internet using IP is carried in packets called IP *datagrams*. Figure 1-35 shows the format of IP version 4 datagrams. Its fields are as follows:

Version number: This field indicates version number, 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, which is also the minimum allowed

[7] 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.

value. 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.

Type of service: 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. It was never widely used as originally defined, and its meaning has been subsequently redefined for use by a technique called Differentiated Services (DS), which will be described later in Section 3.3.5.

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

Identification: This is a *sequence number* that, together with the source address, destination address, and user protocol, is intended to identify a datagram uniquely.

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 prohibits fragmentation when set. 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, measured in 8-byte (64-bit) units. This 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 $65{,}536 \div 8{,}192 = 8$-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 to which the IP protocol at the destination will deliver the payload. In other words, 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 to the header only. 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 formed by taking the ones complement of the 16-bit ones-complement addition of all 16-bit words in the header. Before the computation, the checksum field is itself initialized to a value of zero.

Source IP address: This address identifies the end host that originated the datagram. Described later in Section 1.4.4.

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

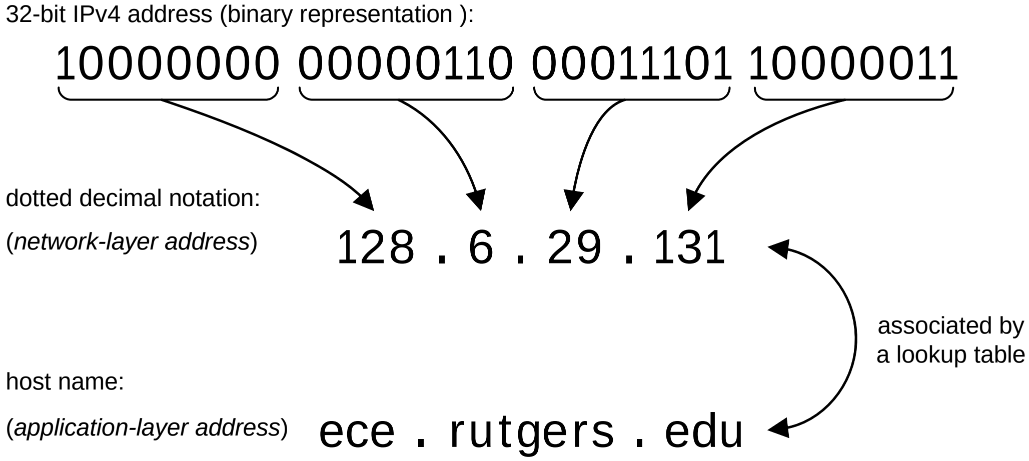

32-bit IPv4 address (binary representation):
10000000 00000110 00011101 10000011
dotted decimal notation:
(*network-layer address*)
128 . 6 . 29 . 131
associated by a lookup table
host name:
(*application-layer address*)
ece . rutgers . edu

Figure 1-36: Dotted decimal notation for IP version 4 addresses.

Options: This field encodes the options requested by the sending user.

In order to send messages using IP we encapsulate the higher-layer data into IP datagrams. These datagrams must then be sent down to the data link layer, 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 data link layer implementation puts the entire IP datagram into the data portion (the payload) of its frame format, just as IP puts transport layer messages, transport headers and all, into its IP Data field.

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[8]. 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

[8] 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.

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 1-36 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*. Notice 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), described in Section 8.4.

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. 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 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 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 protocol. Network layer addresses are standardized by the Internet Engineering Task Force (http://www.ietf.org).

Notice 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 geographic 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 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 to the destination. Section 1.4.4 describes how IPv4 addresses are structured to assist routing.

Datagram Fragmentation and Reassembly

Problems related to this section: Problem 1.22

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 data link layer where they are sent over physical network links.

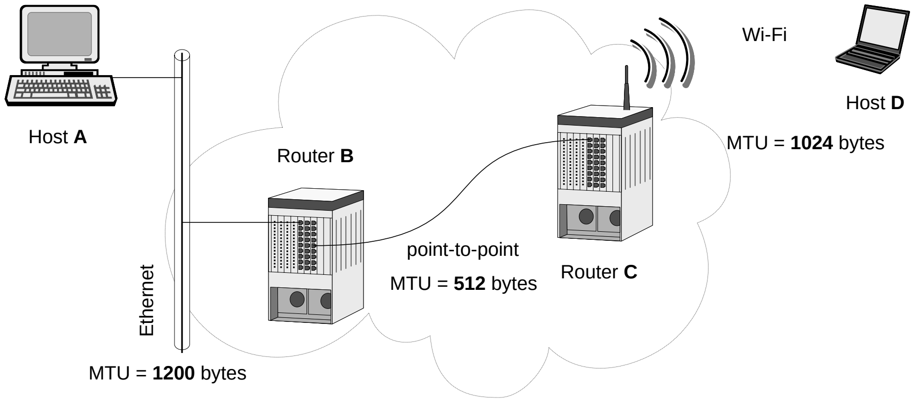


Figure 1-37: Example scenario for IP datagram fragmentation.

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, 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 layers 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 1-37, 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, which in turn uses IP. The first physical network is Ethernet, which for illustration is configured to limit the size of the payload it sends to 1,200 bytes. The second network uses a Point-to-Point protocol that limits the payload size 512 bytes and the third network is Wi-Fi with the payload limit equal to 1024 bytes.

Figure 1-38 illustrates the process by which IP datagrams to be transmitted 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 1-38, the IP layer at router *B* creates three smaller datagrams from the original datagram.

The bottom row in Figure 1-38(b) shows the contents of the fragmentation-related fields of the datagram headers (the second row of the IP header shown in Figure 1-35). 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, the size of the first two datagrams 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. Notice also that the offset value of the second fragment is 61, which means that this fragment starts at $8 \times 61 = 488$ bytes in the original IP datagram from which this fragment is created.

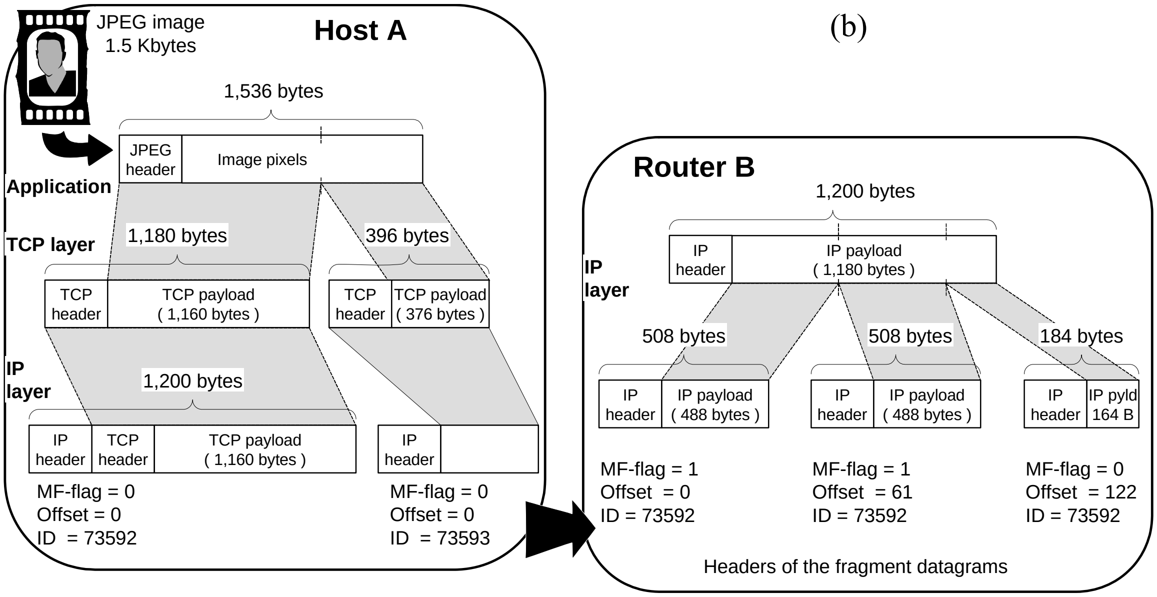


Figure 1-38: IP datagram fragmentation at Router B of the network shown in Figure 1-37. The fragments will be reassembled at the destination (Host D) in an exactly reverse process.

It is important to emphasize that lower layer protocols do not distinguish any structure in the payload passed to them by an upper layer protocol. Therefore, although in the example of Figure 1-38 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, the resulting IP datagram will not exceed 512 bytes in size.

1.4.2 Link State Routing

Problems related to this section: Problem 1.23 → ?

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 constituting 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 presents the path vector routing, which is similar to the distance vector routing.

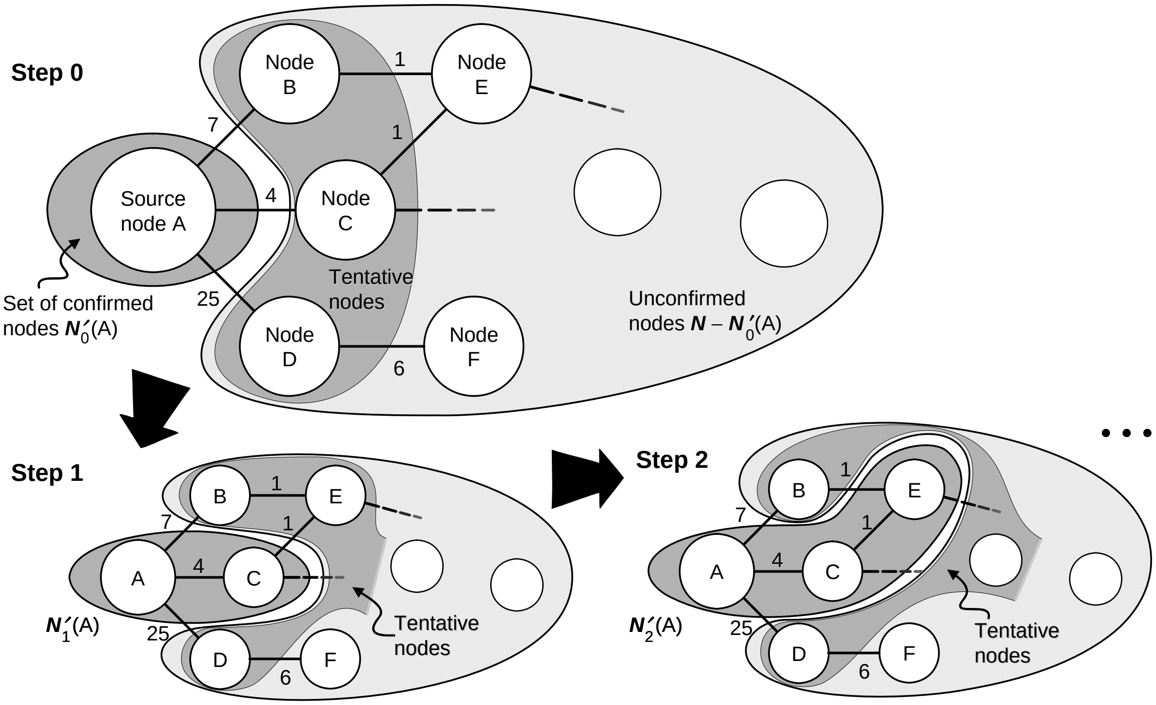


Figure 1-39: Illustration of finding the shortest path using Dijkstra's algorithm.

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 order of the increasing path cost (Figure 1-39). At the k^{th} step we have the set $\mathbf{N}'_k(A)$ of the k closest nodes to node A ("confirmed nodes") as well as the shortest distance D_X from each node X in $\mathbf{N}'_k(A)$ to node A. Of all paths connecting some node not in $\mathbf{N}'_k(A)$ ("unconfirmed nodes") with node A, there is the shortest one that passes exclusively through nodes in $\mathbf{N}'_k(A)$, because $c(X, Y) \geq 0$. Therefore, the $(k + 1)$st closest node should be selected among those unconfirmed nodes that are neighbors of nodes in $\mathbf{N}'_k(A)$. These nodes are marked as "tentative nodes" in Figure 1-39.

When a router (network node A) is initialized, it determines the link cost on each of its network interfaces. For example, in Figure 1-40 the cost of the link connecting node A to node B is labeled as "10" units, that is $c(A, B) = 10$. 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 Packet* (LSP) or *Link-State Advertisement* (LSA), which contains the following information:

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

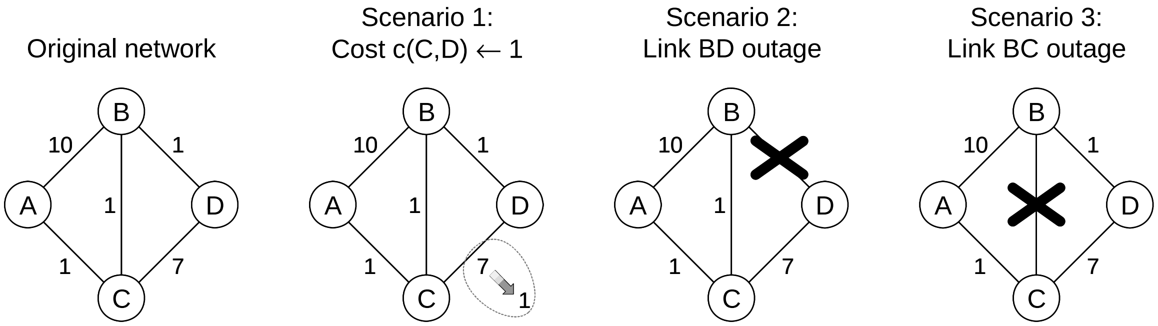


Figure 1-40: Example network used for illustrating the routing algorithms.

In the initial step, all nodes send their LSPs 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 LSPs. The next step is to build a routing table, which is an intermediate step towards building a forwarding table. A **routing table** of a node (source) contains the paths and distances to all other nodes (destinations) in the network. A **forwarding table** of a node pairs different destination nodes with appropriate output interfaces of this node.

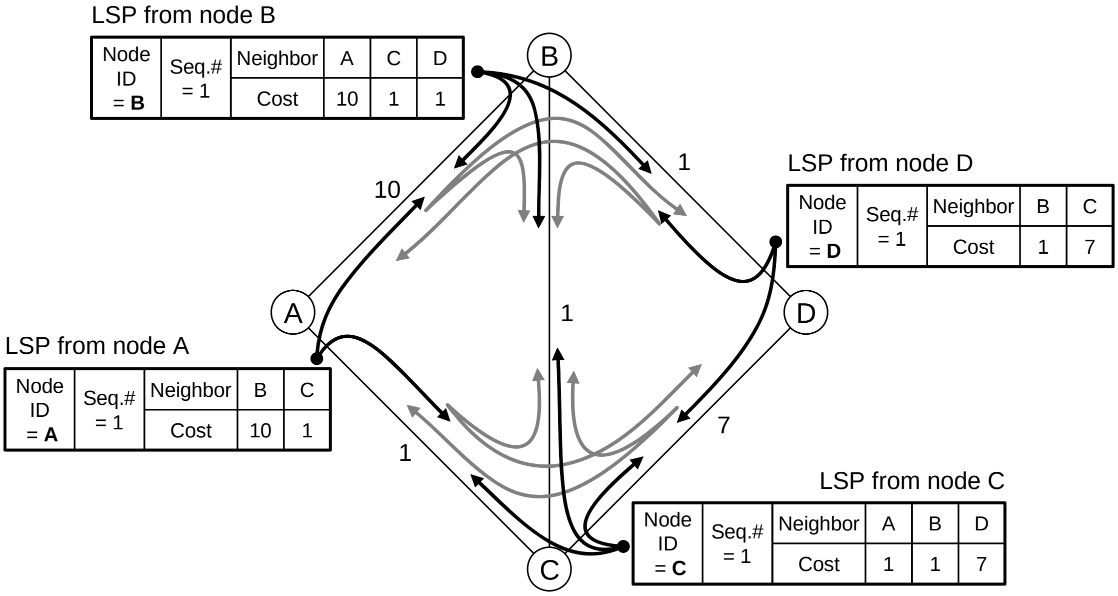
Let ***N*** denote the set of all nodes in a network. In Figure 1-40, ***N*** = {*A*, *B*, *C*, *D*}. The process can be summarized as an iterative execution of the following steps

1. Check the LSPs of all nodes in the confirmed set ***N′***to update the tentative set (recall that tentative nodes are unconfirmed nodes that are neighbors of confirmed nodes)
2. Move the tentative node with the shortest path 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 original network in Figure 1-40 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. Assume that all nodes broadcast their LSPs and each node already received LSPs from all other nodes in the network before it starts the shortest path computation, as shown in this figure:



LSP from node B

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

LSP from node D

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

LSP from node A

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

LSP 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 1-40. Each node is represented with a triplet (*Destination node ID*, *Path length*, *Next hop*). The node *x* maintains two sets: 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 LSP to all other nodes and recomputes its routing table based on the received LSPs.

Table 1-2: Steps for building a routing table at node *A* in Figure 1-40. 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 LSP.
1	(*A*, 0, –)	(*B*, 10, *B*), (*C*, 1, *C*)	*A*'s LSP 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 LSP 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 LSP 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 its LSP.
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.
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.

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.

If not all the nodes are working from *exactly* the same map, 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 the two. 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.

The most popular practical implementation of link-state routing is *Open Shortest Path First* (OSPF) protocol, reviewed in Section 8.2.2.

1.4.3 Distance Vector Routing

Problems related to this section: Problem 1.25 → Problem 1.27

The key idea of the distance vector routing algorithm is that each node assumes that its neighbors 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 1-41 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.

Figure 1-40 shows an example network that will be used to describe the *distance vector routing algorithm*. Let $\boldsymbol{N}$ denote the set of all nodes in a network. In Figure 1-40, $\boldsymbol{N} = \{A, B, C, D\}$. The two types of quantities that this algorithm uses are:

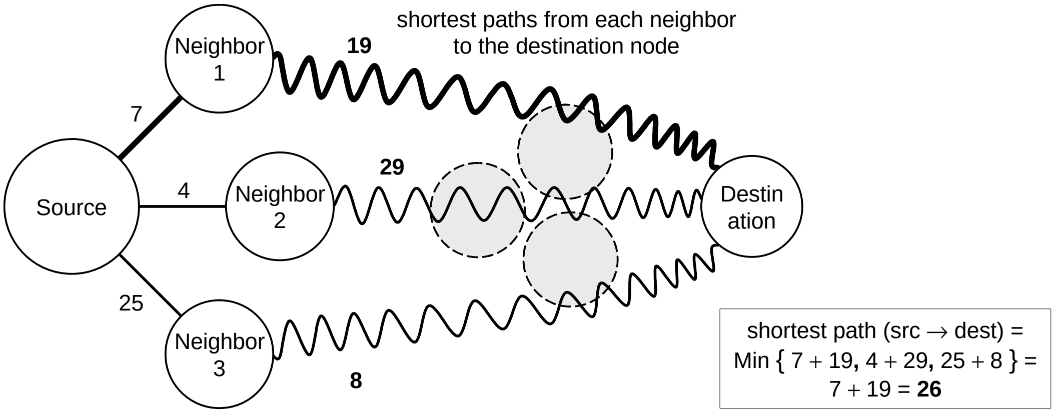

shortest paths from each neighbor to the destination node
Neighbor 1
19
7
Source
4
Neighbor 2
29
Destin ation
25
Neighbor 3
8
shortest path (src → dest) = Min { 7 + 19, 4 + 29, 25 + 8 } = 7 + 19 = 26

Figure 1-41: Illustration of finding the shortest path using Bellman-Ford algorithm. The thick line (crossing Neighbor 1) represents the shortest path from Source to Destination.

(i) *Link cost* assigned to an individual link directly connecting a pair of nodes (routers). These are given to the algorithm either by having the network operator manually enter the cost values or by having an independent program determine these costs. For example, in Figure 1-40 the cost of the link connecting the nodes A and B is labeled as "10" units, that is $c(A, B) = 10$.

(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 from node X to node Y is denoted as $D_X(Y)$. These will be computed by the routing algorithm.

The **distance vector** of node X is the vector of distances from node X to all other nodes in the network, denoted as $DV(X) = \{D_X(Y);\ Y \in \boldsymbol{N}\}$. When determining the minimum cost path, 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.

Let $\eta(X)$ symbolize the set of neighboring nodes of node X. For example, in Figure 1-40 $\eta(A) = \{B, C\}$ because these 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:

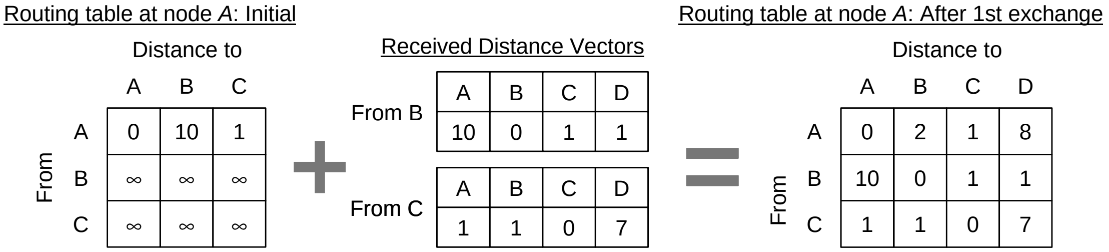
$$D_X(Y) = \min_{V \in \eta(X)} \{c(X, Y) + D_V(Y)\} \tag{1.14}$$

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 vector 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 1-40 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

From B

A	B	C	D
10	0	1	1

From C

A	B	C	D
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

Notice 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 distance vector to its immediate neighbors, and *A* receives distance vectors from *B* and *C*. For the sake of 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 with the new one. As shown earlier, *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:

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

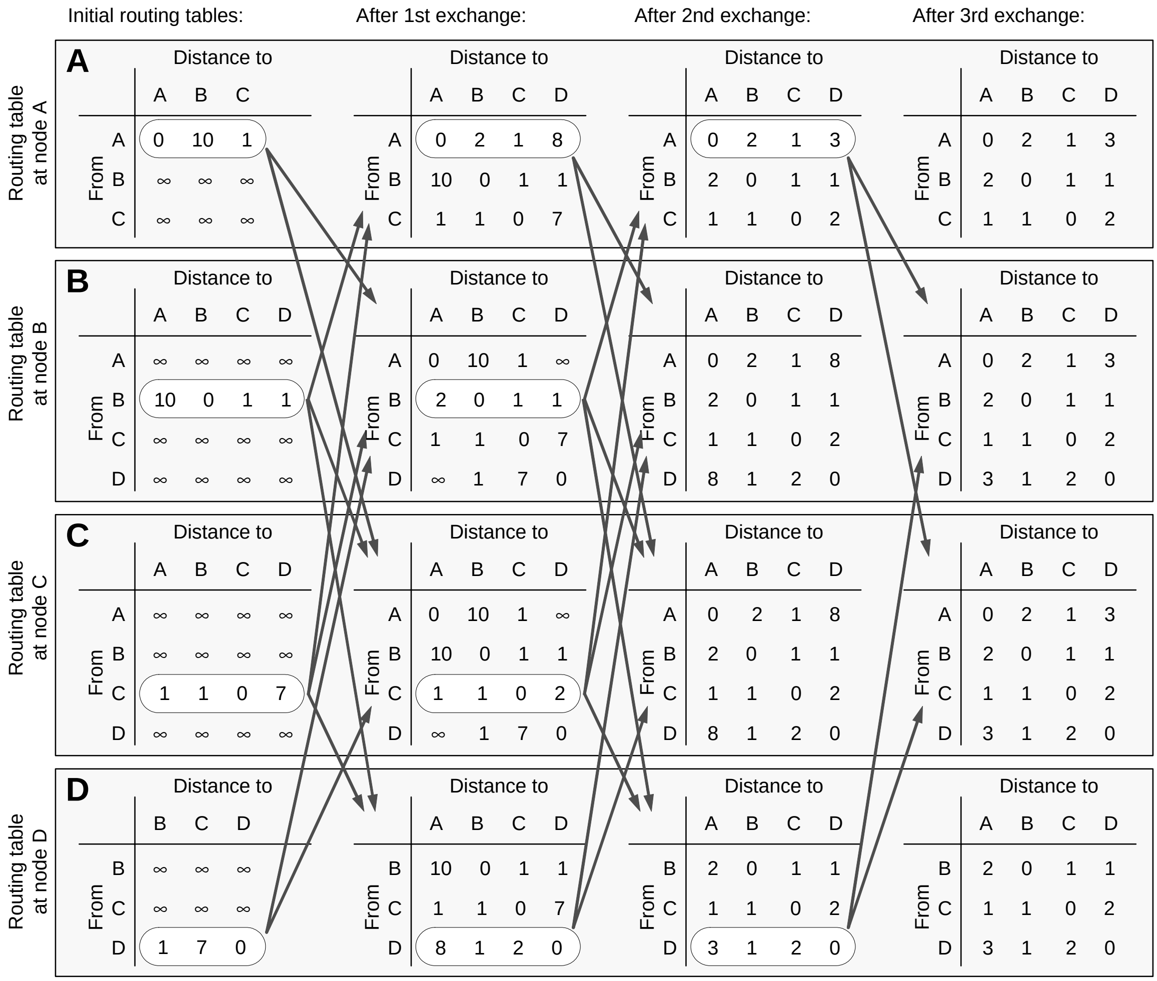
$$D_A(C) = \min\{c(A,B) + D_B(C),\ c(A,C) + D_C(C)\} = \min\{10+1,\ 1+0\} = 1$$

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

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

Similar computations will take place on all other nodes and the whole process is illustrated in Figure 1-42. The end result is as shown in Figure 1-42 as column entitled "After 1st exchange." Because for every node the newly computed distance vector is different from the previous one, each node sends its distance new 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 1-42, 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. 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.



Routing table at node A

	Initial routing tables:			After 1st exchange:				After 2nd exchange:				After 3rd exchange:			
From	A	B	C	A	B	C	D	A	B	C	D	A	B	C	D
A	0	10	1	0	2	1	8	0	2	1	3	0	2	1	3
B	∞	∞	∞	10	0	1	1	2	0	1	1	2	0	1	1
C	∞	∞	∞	1	1	0	7	1	1	0	2	1	1	0	2

Routing table at node B

	Initial routing tables:				After 1st exchange:				After 2nd exchange:				After 3rd exchange:			
From	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
A	∞	∞	∞	∞	0	10	1	∞	0	2	1	8	0	2	1	3
B	10	0	1	1	2	0	1	1	2	0	1	1	2	0	1	1
C	∞	∞	∞	∞	1	1	0	7	1	1	0	2	1	1	0	2
D	∞	∞	∞	∞	∞	1	7	0	8	1	2	0	3	1	2	0

Routing table at node C

	Initial routing tables:				After 1st exchange:				After 2nd exchange:				After 3rd exchange:			
From	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
A	∞	∞	∞	∞	0	10	1	∞	0	2	1	8	0	2	1	3
B	∞	∞	∞	∞	10	0	1	1	2	0	1	1	2	0	1	1
C	1	1	0	7	1	1	0	2	1	1	0	2	1	1	0	2
D	∞	∞	∞	∞	∞	1	7	0	8	1	2	0	3	1	2	0

Routing table at node D

	Initial routing tables:			After 1st exchange:				After 2nd exchange:				After 3rd exchange:			
From	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
B	∞	∞	∞	10	0	1	1	2	0	1	1	2	0	1	1
C	∞	∞	∞	1	1	0	7	1	1	0	2	1	1	0	2
D	1	7	0	8	1	2	0	3	1	2	0	3	1	2	0

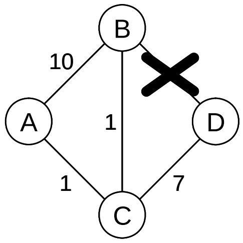
Figure 1-42: Distance vector (DV) algorithm for the original network in Figure 1-40.

Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead (because each node informs only its 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 reason for this is that when a node updates and distributes its distance vector to the neighbors, it does not reveal the information it used to compute the vector. Because of this, remote routers do not have sufficient information to determine whether their choice of the next hop will cause routing loops to form. While 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 1-40(c) reproduced 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 ∞. (Notice that *B* cannot use old distance vectors it obtained earlier from the 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*,[9] *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) 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 *C* is chosen as the next hop. Because we 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 value of the distances, not the paths over which these distances are computed, *C* does *not* know that it lays on *B*'s shortest path to *D*!

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

From	Distance to 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*) = ∞
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

From	Distance to 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 *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 changed, it reports its new distance vector to its neighbors (triggered update). After receiving *B*'s new distance vector, *C* will determine that its new shortest path to *D* measures 4, via *C*. 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 known as **hold-down timers**. When a node detects a link failure, it reports to all neighboring nodes that an attached network has gone down. The neighbors immediately start their hold-down timers to ensure that this route will not be mistakenly reinstated by an advertisement received from another router that has not yet learned about this route being unavailable. Until the timer elapses, the router ignores updates regarding

[9] 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*.

this route. Router accepts and reinstates the invalid route if it receives a new update with a better metric than its own or 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.

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 (i.e., "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 8.2.1.

1.4.4 IPv4 Address Structure and CIDR

Problems related to this section: Problem 1.28 → Problem 1.30

Section 1.4.1 briefly mentions that the structure of network addresses should assist with message routing along the path to the destination. This may not be obvious at first, so let us consider again the analogy between a router and a crossroads (Figure 1-33). Suppose you are driving from Philadelphia to Bloomfield, New Jersey (Figure 1-43). If the sign on the 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." The intersections would be congested by cars looking-up the

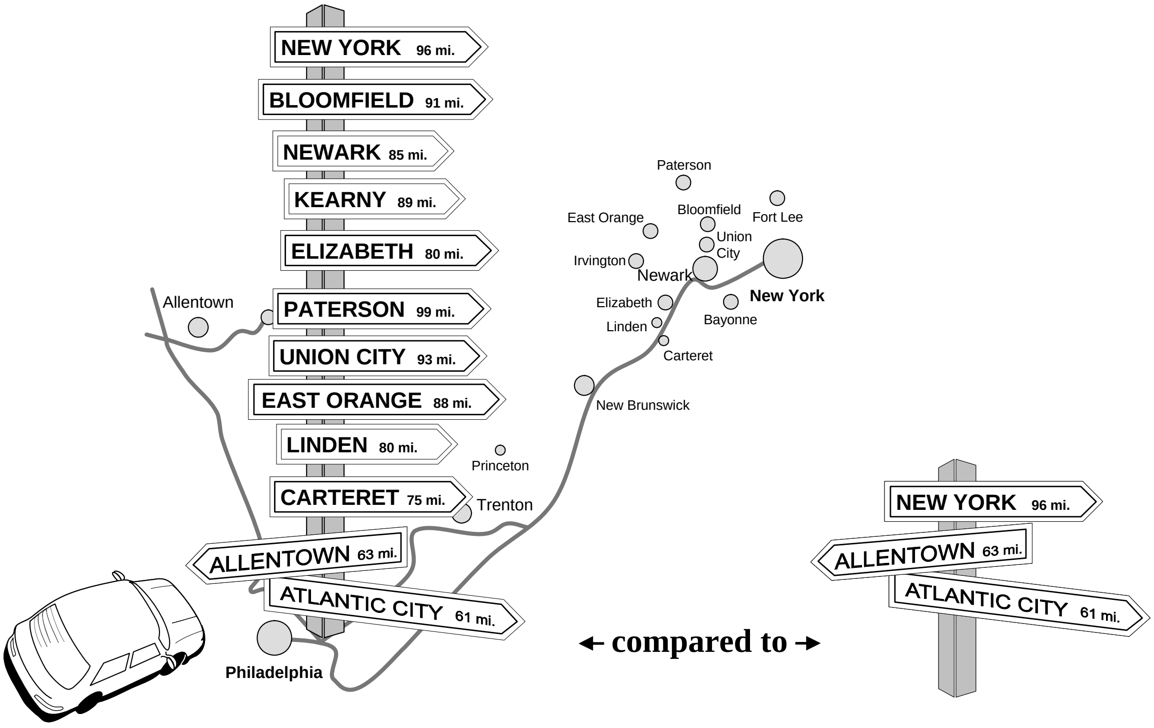


Figure 1-43: Illustration of the problem with the forwarding-table size. Real world road signs contain only a few destinations (lower right corner) to keep them manageable.

long table and trying to figure out which way to exit out of the intersection. The problem is solved by listing only the major city names on the signs. Notice 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 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. 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; 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 destination address and forwards the packet onwards. Conversely, if the destination address *is* in this router's region, it does a lookup on the "intra-region" portion and forwards the packet on. This structuring of network layer addresses dramatically reduces the size of the forwarding tables. The state in the forwarding tables for routes not in the same region is at most equal to the number of regions in the network, typically much smaller than the total number of possible addresses.

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?
- 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 the organization's hierarchy (number of levels and nodes per level) remain 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 this 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 1-44). Class A addresses start with a "0" and have the next 7 bits for network number and the last 24 bits for host number. Class B addresses start with "10", use the next 14 bits for network number, and the last 16 bits for host number (e.g., Rutgers has a Class B network, with addresses in dotted-decimal notation of the form `128.6.*`). Class C addresses start with "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. They start with "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 "1111" are reserved for experiments.

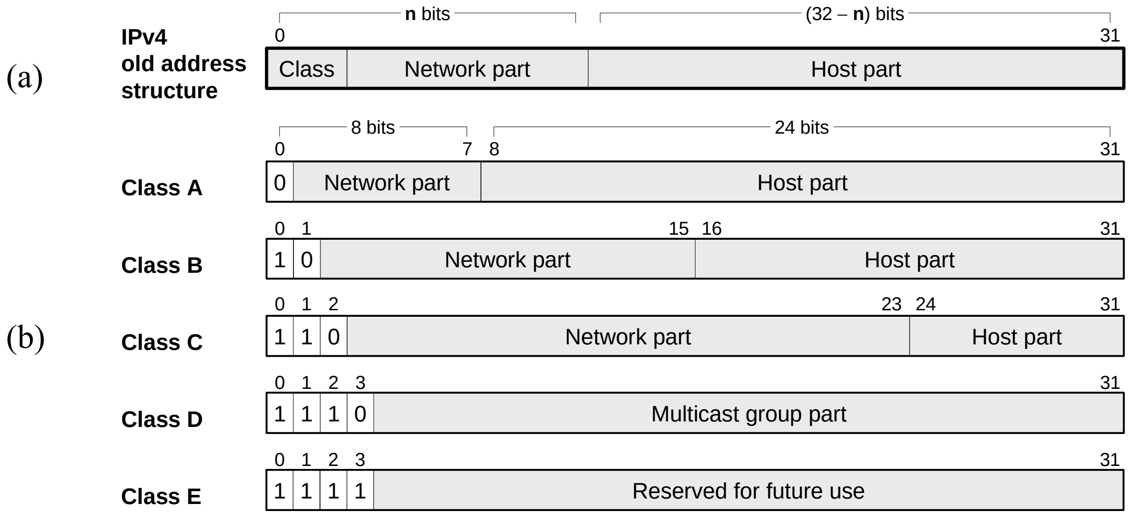


Figure 1-44: (a) Class-based structure of IPv4 addresses (deprecated). (b) The structure of the individual address classes.

The router-forwarding task in IPv4 is a bit more complicated than for an unstructured addressing scheme that requires an exact-match. 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 class.

Thus, in the original design of IPv4, address space was partitioned in 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} = 128 addresses each. The address space has been managed by IETF and organizations requested and obtained a set of addresses belonging to a class. 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 128 hosts and acquired a Class B address, almost 65,400 addresses went unused and could not be assigned to another organization.

Figure 1-45 lists special IPv4 addresses.

CIDR Scheme for Internet Protocol (IPv4) Addresses

In 1991, it became clear that the 2^{14} = 16,384 Class B addresses would soon run out and a different approach was needed. It was observed that addresses from the enormous Class C space were rarely allocated and 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.

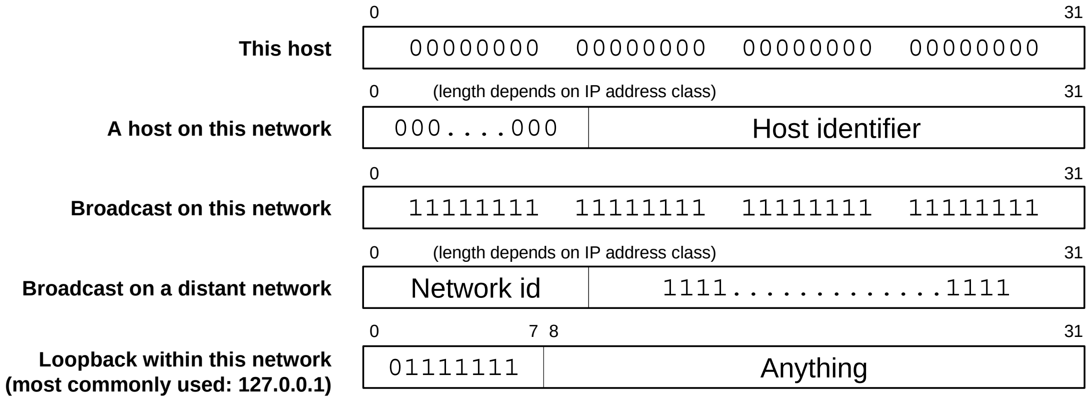


Figure 1-45: Special IP version 4 addresses.

Routing protocols that work with aggregated Class C address sets are said to follow **Classless Interdomain Routing** or **CIDR** (pronounced "cider"). CIDR not only solves the problem of address shortages, but also by aggregating Class C sets into contiguous regions, it reduces the forwarding table sizes because routers aggregate routes based on IP prefixes in a classless manner. Instead of having a forwarding-table entry for every individual address, the router keeps a single entry for a subset of addresses (see analogy in Figure 1-43).

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** and it is a number between 1 and 32. Therefore, when a network is assigned A/m, it means that it gets the $2^{(32-m)}$ addresses, all sharing the first m bits of A. For example, the network "192.206.0.0/21" corresponds to the 2048 addresses in the range from 192.206.0.0 to 192.206.7.255.

SIDEBAR 1.2: 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, each Ethernet adaptor on a subnetwork may be from a different manufacturer, and noncontiguous subnetworks may have adaptors from the same manufacturer. However, this type of hierarchy 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 8.3.1.)

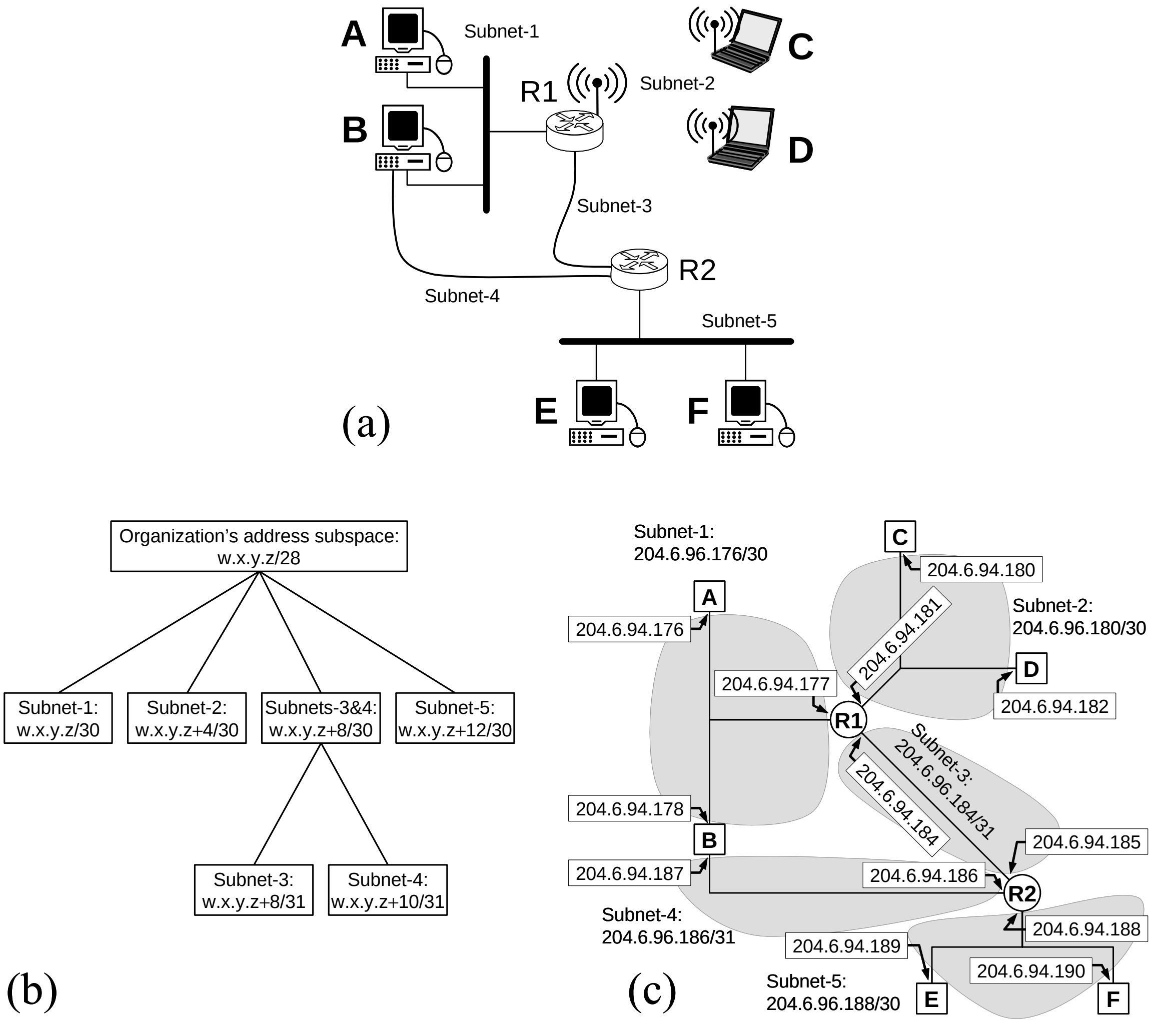


Figure 1-46: (a) Example internetwork with five physical networks reproduced from Figure 1-34 above. (b) Desired hierarchical address assignment under the CIDR scheme. (c) Example of an actual address assignment.

Suppose for the sake of illustration that you are administering your organization's network as shown in Figure 1-34, reproduced here in Figure 1-46(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. 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. 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 1-46(b). Subnets 3 and 4 have only two interfaces each, so they need 2 addresses each. Their assignments will have the mask $m = 31$. You can group these two in a single set with $m = 30$. Subnets 1, 2, and 5 have three interfaces each, so you need at least 2 bits (4 addresses) for each and their masks will equal $m = 30$. Therefore, you need 4 × 4 addresses (of which three will be unused) and your address region will be of the form *w.x.y.z*/28, which gives you $2^{(32-28)} = 2^4 = 16$ addresses. Let us assume that the actual address subspace assignment that you acquired is

Table 1-3: CIDR hierarchical address assignment for the internetwork in Figure 1-46.

Subnet	Subnet mask	Network prefix	Interface addresses
1	204.6.94.176/30	11001100 00000110 01011110 101100--	A: 204.6.94.176
			R1-1: 204.6.94.177
			B-1: 204.6.94.178
2	204.6.94.180/30	11001100 00000110 01011110 101101--	C: 204.6.94.180
			R1-2: 204.6.94.181
			D: 204.6.94.182
3	204.6.94.184/31	11001100 00000110 01011110 1011100-	R1-3: 204.6.94.184
			R2-1: 204.6.94.185
4	204.6.94.186/31	11001100 00000110 01011110 1011101-	R2-2: 204.6.94.186
			B-2: 204.6.94.187
5	204.6.94.188/30	11001100 00000110 01011110 101111--	R2-3: 204.6.94.188
			E: 204.6.94.189
			F: 204.6.94.190

204.6.94.176/28. Then you could assign the individual addresses to the network interfaces as shown in Table 1-3 as well as in Figure 1-46(c).

1.4.5 Autonomous Systems and Path Vector Routing

Problems related to this section: ? → ?

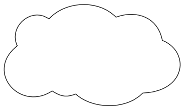
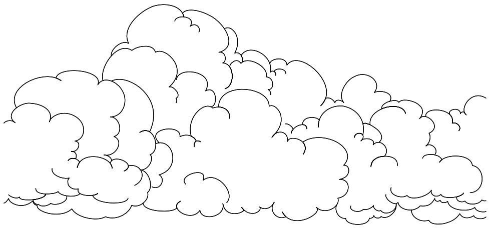


Figure 1-34 presents a naïve view of the Internet, where many hosts are mutually connected via intermediary nodes (routers or switches) that live inside the "network cloud." This would imply that the cloud is managed by a single administrative organization and all nodes cooperate to provide the best service to the consumers' hosts. In reality the Internet is composed of many independent networks (or, "clouds"), each managed by a different organization driven by its own commercial or political interests. (The reader may also wish to refer to Figure 1-3 to get a sense of complexity of the Internet.) Each individual administrative domain is known as an **autonomous system** (AS). Given their divergent commercial interests, these administrative domains are more likely to compete (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 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* that 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 Systems: 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 1-47 illustrates an example of several Autonomous Systems. In order to get traffic from one end-user to another end-user, 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, since 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 economic 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. Since 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 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.

An economic 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 8.2.3 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

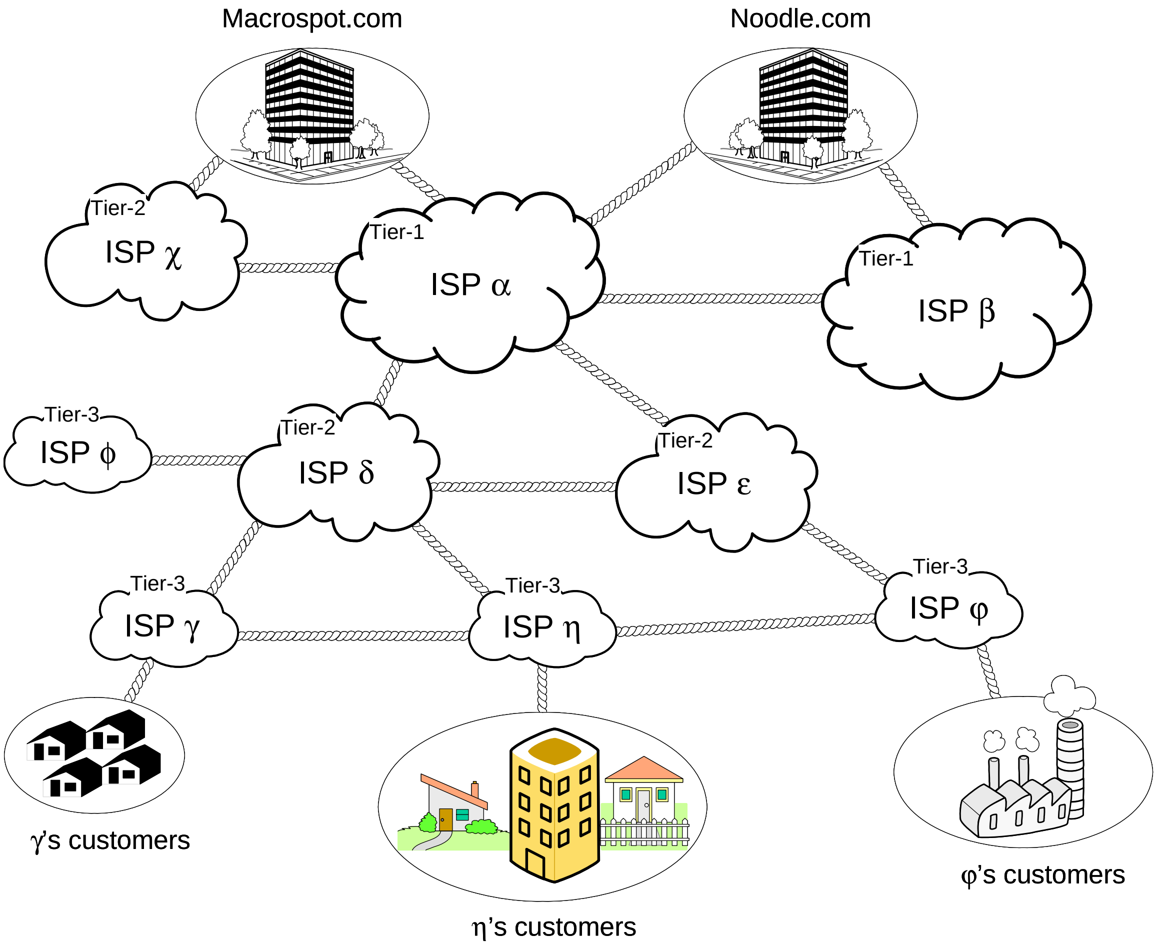


Figure 1-47: An example collection of Autonomous Systems with physical interconnections.

able to reach any other network on the Internet, Autonomous System operators work with each other in following ways:

- *Sell transit* (or Internet access) service to that AS ("transit provider" sells transit service to a "transit customer"),
- *Peer* directly with that AS, or with an AS who sells transit service to that AS, or
- *Pay* another AS for *transit* service, where that "transit provider" must in turn also sell, peer, or pay for access.

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.

Consider the example in Figure 1-47. 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 entries. They are said to be *default-free*. At present (2010) there are about 10 Tier-1 ISPs in the world. 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 would form a peering agreement with other Tier-1 ISPs, and sell transit to lower tiers ISPs. Similarly, a Tier-2 (regional

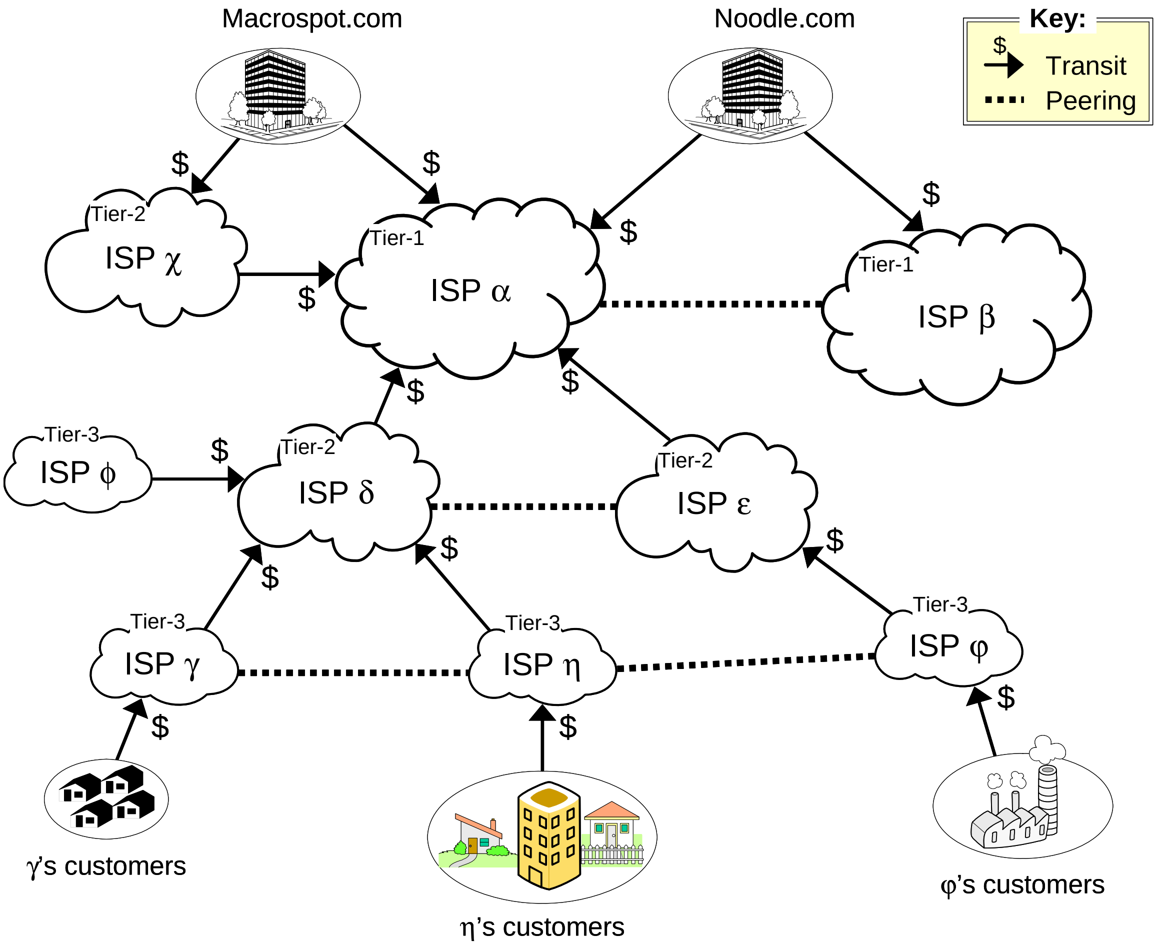


Figure 1-48: Feasible business relationships for the example ASs in Figure 1-47.

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

Figure 1-48 shows reasonable business relationships between the ISPs in Figure 1-47. 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 1-48 each have connections to more than one other AS but they refuse to carry transit traffic; such an AS is called **multihomed AS**. ISPs

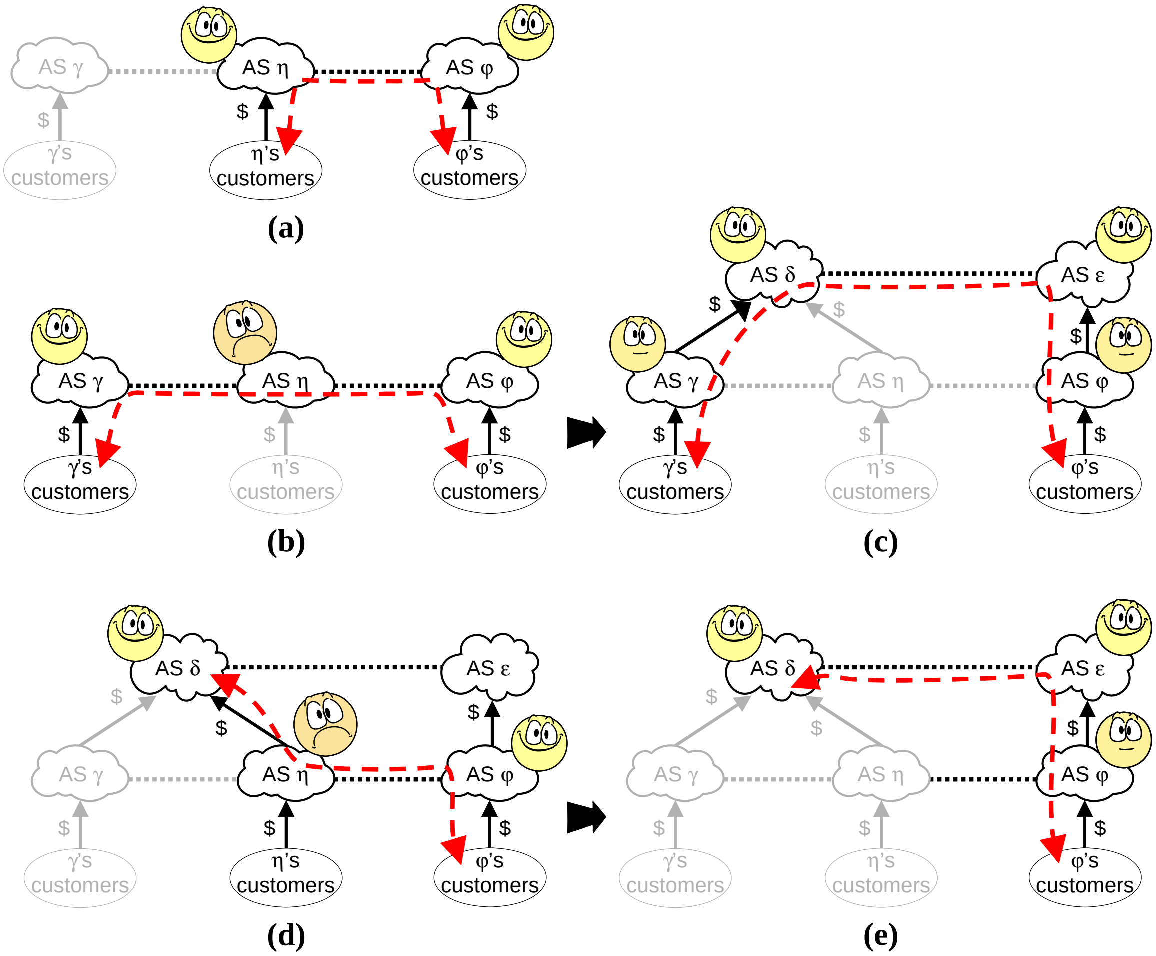


Figure 1-49: Providing selective transit service to make or save money.

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 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, 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 three types of business relationships with the ASs to which is has a direct physical interconnection: transit provider, transit customer, or peer. To its paying customers, the AS wants to provide unlimited transit service. However, to its provider(s) and peers it probably wishes to provide a *selected transit service*. Figure 1-49 gives examples of how conflicting interests of different parties can be resolved. The guiding principle is that ASs will want to avoid highly asymmetrical relationships without reciprocity. In Figure 1-49(a), both ASη and ASφ benefit from peering because it helps them to provide global reachability to their customers. In Figure 1-49(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 will not carry transit traffic between its peers. An appropriate solution is presented in Figure 1-49(c), where ASγ and ASφ use their transit providers (ASδ and ASε, respectively), to carry their mutual transit traffic. ASδ and ASε are peers and are happy to provide transit service to their transit customers (ASγ and ASφ). Figure 1-49(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 1-49(e) (which is essentially the same as Figure 1-49(c)).

To implement these economic decisions and prevent unfavorable arrangements, ASs design and enforce *routing policies*. An AS that wants 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 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 1-48, 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 1-50. 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}. A border router (router *K*) in ASδ receives this message and disseminates it to other routers in ASδ. 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. 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. 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 their routing and forwarding tables based on the received path vector. Finally, when a router in ASφ needs to send a data packet to a destination in ASϕ it sends the packet first to the next hop on the path to ASϕ, which is ASε.

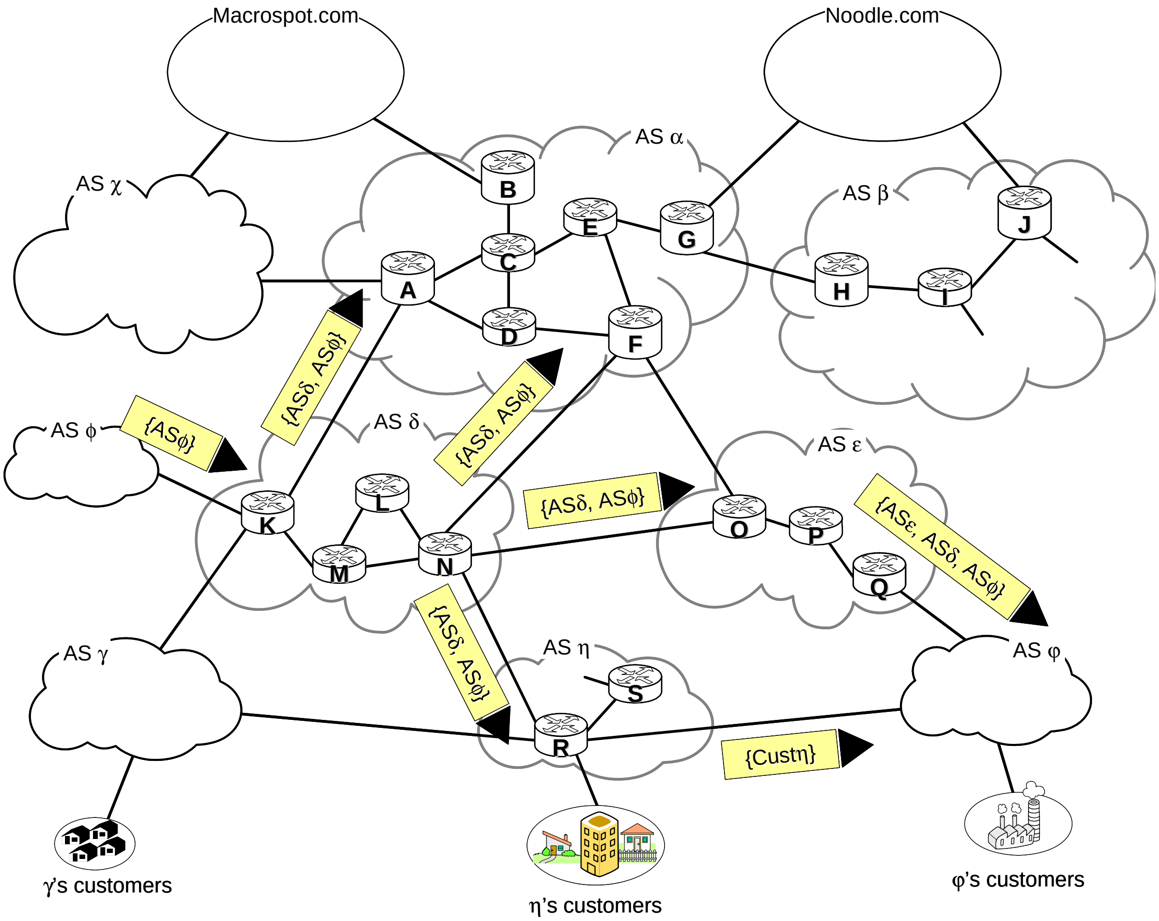


Figure 1-50: Example of routers within the ASs in Figure 1-47. Also shown is how a routing update message from ASϕ propagates to ASφ.

Path Vector Routing

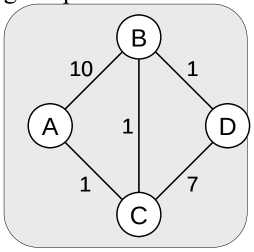
Path vector routing is used for *inter-domain* or *exterior routing* (routing between different Autonomous Systems). The path vector routing algorithm is somewhat similar to the distance vector algorithm. Each border (or edge) router advertises the destinations it can reach to its neighboring router (in a different AS). However, instead of advertising networks in terms of a destination and the distance to that destination, 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 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 information 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 1-50, where a router in ASϕ sends a scheduled update message. After a node 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 original network in Figure 1-40 (reproduced below) and assume that it uses the path vector routing algorithm. For simplicity, assume that the path vector contains router addresses instead of 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 node *A*, the routing table initially looks as follows. The notation ⟨*d* | *X*, *Y*, *Z*⟩ symbolizes that the path from the node under consideration to node *Z* is *d* units long, and *X* and *Y* are the nodes along the path to *Z*. If the path metric simply counts the number of hops, then the path-vector packets do not need to carry the distance *d*, because it can be determined simply by counting the nodes along the path.



Routing table at node *A*: Initial

From \ Path to	A	B	C
A	⟨0 | A⟩	⟨10 | B⟩	⟨C | 1⟩
B	⟨∞ | ⟩	⟨∞ | ⟩	⟨∞ | ⟩
C	⟨∞ | ⟩	⟨∞ | ⟩	⟨∞ | ⟩

+

Received Path Vectors

	A	B	C	D
From B	⟨10 | A⟩	⟨0 | B⟩	⟨1 | C⟩	⟨1 | D⟩

	A	B	C	D
From C	⟨1 | C⟩	⟨1 | B⟩	⟨0 | C⟩	⟨7 | D⟩

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

=

From \ Path to	A	B	C	D
A	⟨0 | A⟩	⟨2 | C, B⟩	⟨1 | C⟩	⟨8 | C, D⟩
B	⟨10 | A⟩	⟨0 | B⟩	⟨1 | C⟩	⟨1 | D⟩
C	⟨1 | A⟩	⟨1 | B⟩	⟨0 | C⟩	⟨7 | D⟩

Again, node *A* only keeps the path 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 advertises its path vector to its immediate neighbors, and *A* receives their path vectors. When a node receives an updated path vector from its neighbor, the node 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:

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

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

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

The new values for *A*'s path vector are shown in the above table. Notice that *A*'s new path to *B* is via *C* and the corresponding table entry is ⟨2 | *C*, *B*⟩.

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 newly computed path vector 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.

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 Autonomous Systems. Such a node is called a **speaker node** or **gateway router**. For example, in Figure 1-50 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, except that only speaker nodes in each Autonomous System can communicate with routers in other Autonomous Systems (i.e., their speaker nodes). The speaker node advertises the path, not the metric of the links, in its Autonomous System or other Autonomous Systems. In other words, there are no weights attached to the links in a path vector.

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 for routing between different Autonomous System is how to route data packets from the origin to the destination in the *most profitable manner*. These protocols are known as **Exterior Gateway Protocols**[10] 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 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 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 speaker routers in the same AS (if there are any), as well as other, non-speaker routers. For example, Figure 1-50 speaker *K* in ASδ needs to exchange its routing information with speaker *N* (and vice versa), as well as with non-speaker routers *L* and *M*. Notice again that only speaker routers run both IGP and Exterior Gateway Protocol (and maintain a 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

[10] 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).

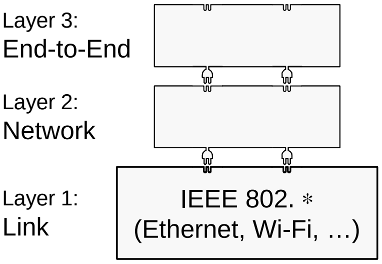
addresses that cannot be matched, and only routers in Tier-1 ISPs are default-free because they know prefixes to all networks in the global Internet.

Consider again the scenario shown in Figure 1-50 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 1-50 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 1-50, 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 lowest router-to-speaker cost among all speakers with a path to the destination. Figure 1-51 summarizes the steps taken at a router for adding the new entry to its forwarding table. In Figure 1-50, 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 8.2.3 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, i.e., these are not 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 distinction is between *point-to-point* and *broadcast* links. 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 the 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* (need to avoid confusion between control codes and data) and requires *data stuffing* (Figure 1-13).

• *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 8.3.1). Point-to-point protocols do not need MAC (Section 1.5.1).

• *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. 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. (See also Section 2.1.3 for more about flow control.)

There are two types of communication links: (1) *point-to-point link* with one sender, one receiver on the link, and no media 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 deal with than a broadcast link because broadcast requires coordination for accessing the medium. 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

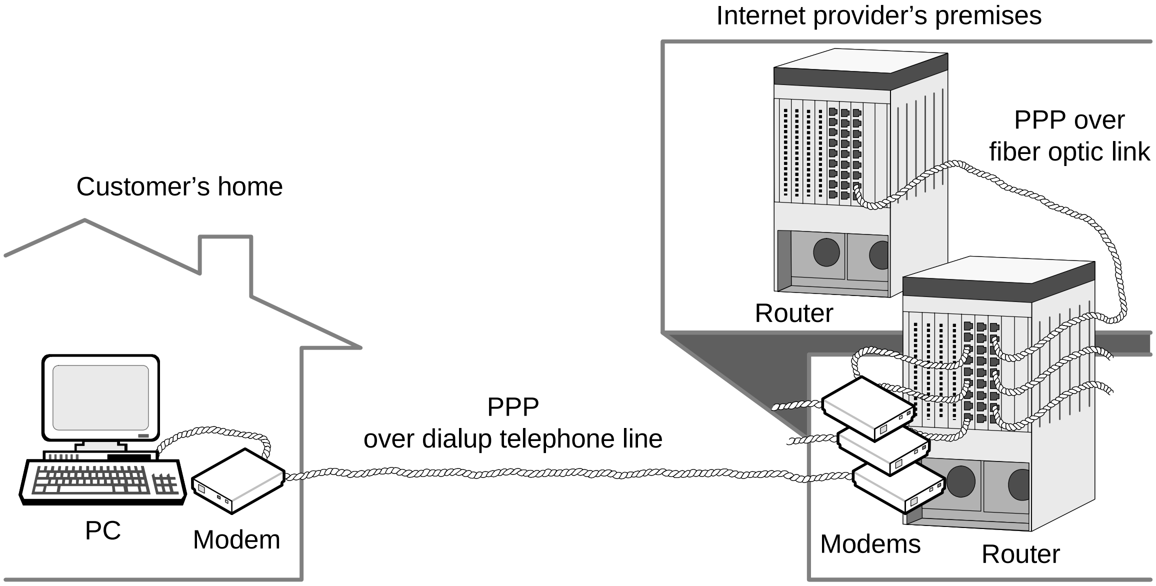


Figure 1-52: Point-to-point protocol (PPP) provides link-layer connectivity between a pair of network nodes over many types of physical networks.

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 build up from point-to-point leased lines.

1.5.1 Point-to-Point Protocol (PPP)

Figure 1-52 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 1-52 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 1-52). 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.

The format for a PPP frame is shown in Figure 1-53. 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. Notice 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. Using the broadcast address avoids the issue of 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.

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

Figure 1-53: Point-to-point protocol (PPP) frame format.

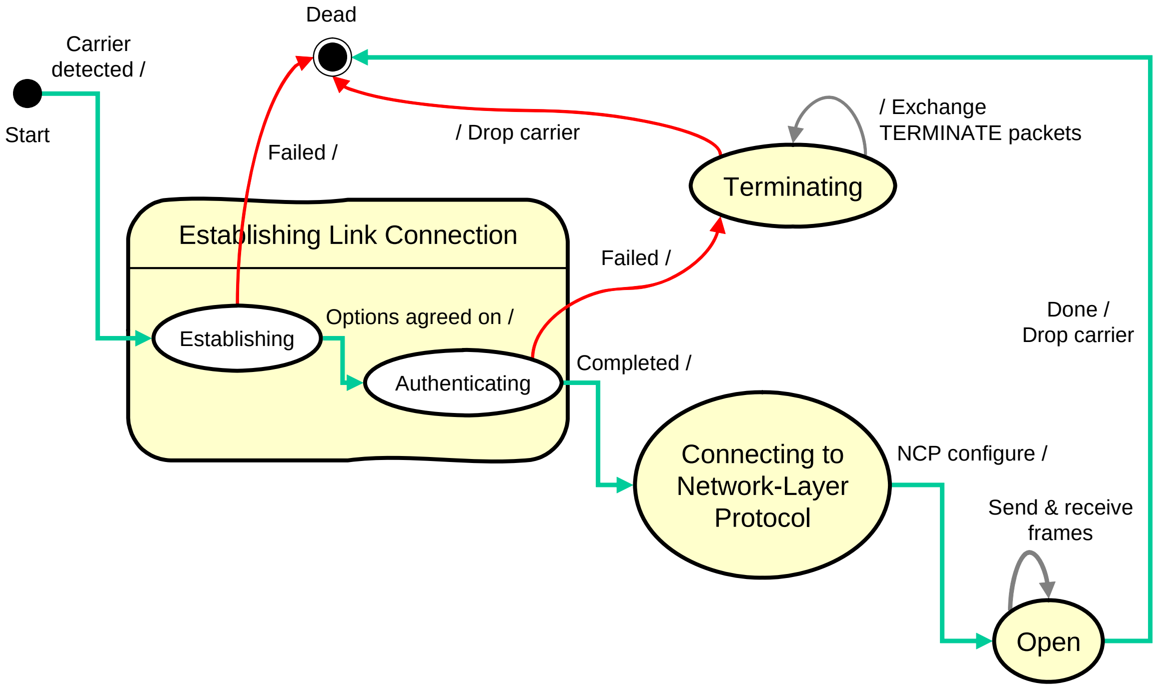


Figure 1-54: State diagram for the point-to-point protocol (PPP).

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.

The Protocol field is used for demultiplexing at the receiver: it identifies the upper-layer protocol (e.g., IP) 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) and see how the method `handle()` calls `upperProtocol.handle()` to handle the received payload.

The Payload field is 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, but has no error correction/recovery.

Figure 1-54 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 (e.g., maximum frame length,

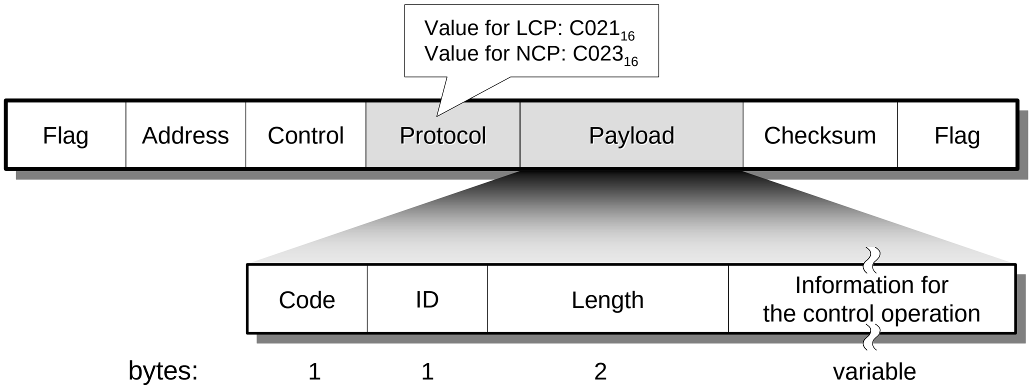

Value for LCP: C021$_{16}$
Value for NCP: C023$_{16}$
Flag
Address
Control
Protocol
Payload
Checksum
Flag
Code
ID
Length
Information for the control operation
bytes:
1
1
2
variable

Figure 1-55: LCP or NCP packet encapsulated in a PPP frame.

authentication, whether to omit the Address and Control fields). 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.

If 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 exchange several PPP control frames.

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 1-55). 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 address.

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

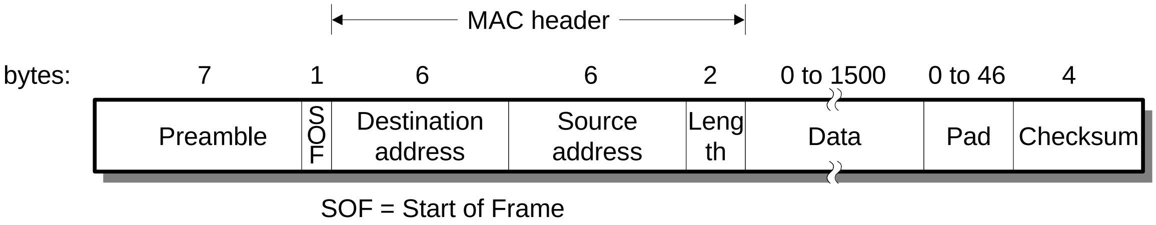


Figure 1-56: IEEE 802.3 (Ethernet) link-layer frame format.

1.5.2 Ethernet (IEEE 802.3)

Based on the CSMA/CD protocol shown in Figure 1-29. The frame format for Ethernet is shown in Figure 1-56.

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. 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.

As already noted for the CSMA/CD protocol (Section 1.3.3), the transmission time of the smallest frame must be larger than one round-trip propagation time, i.e., 2β. This requirement limits the distance between two computers on the wired Ethernet LAN. The smallest frame is 64 bytes. 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, which is larger than the round-trip time across 2500 m (about 18 μs) plus the delays across repeaters and to detect the collision.

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 minimum interframe space 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. In other words, if an Ethernet network adapter senses that there is no signal energy entering the adapter from the channel for 96-bit times, it declares the channel idle and starts to transmit the frame.

The Ethernet specification allows no more than 1,024 hosts and it can span only a geographic area of 2,500 m.

Ethernet Hubs

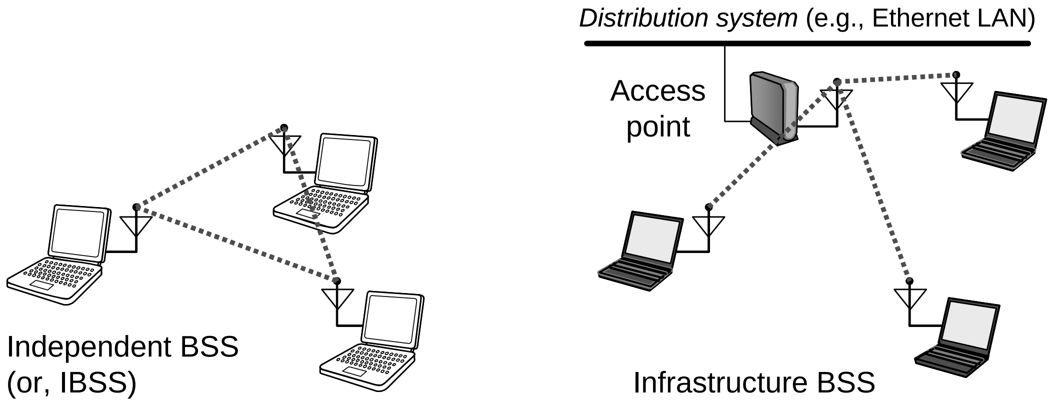

Distribution system (e.g., Ethernet LAN)
Access point
Independent BSS (or, IBSS)
Infrastructure BSS

Figure 1-57: IEEE 802.11 (Wi-Fi) independent and infrastructure basic service sets (BSSs).

1.5.3 Wi-Fi (IEEE 802.11)

Problems related to this section: Problem 1.33 → Problem 1.34

IEEE 802.11, also known as Wi-Fi, …

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 particular area, due to the uncertainties of electromagnetic propagation. There are two types of BSS, as shown in Figure 1-57. When all of the stations in the BSS are mobile stations 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**.

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 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 from the AP to the other mobile 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, when the station enters a power saving mode.

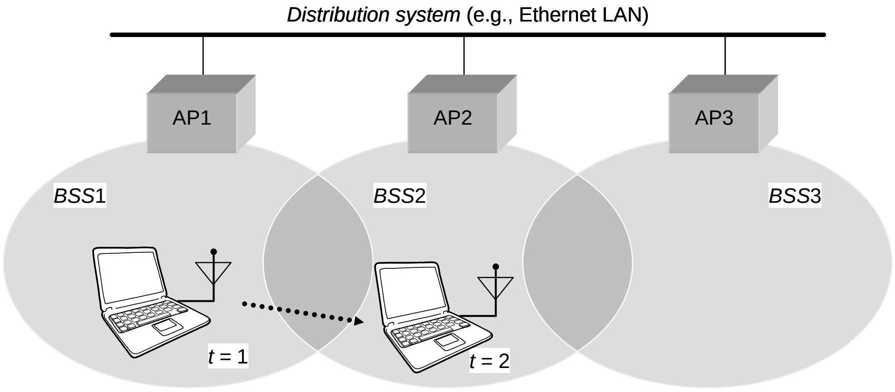


Figure 1-58: IEEE 802.11 (Wi-Fi) extended service set (ESS) and mobility support.

Extended service set (ESS) extends the range of mobility from a single BSS to any arbitrary range (Figure 1-58). An ESS is a set of infrastructure BSSs, where the APs communicate among themselves to forward traffic from one BSS to another and to facilitate the roaming of mobile stations between the BSSs. 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 OSI layer-2 connection. ESS is the highest-level abstraction supported by 802.11 networks. Roaming between different ESSs is not supported by IEEE 802.11 and must be supported by a higher-level protocol, e.g., Mobile IP.

Figure 1-59 shows the 802.11 frame format. The general MAC-layer format (Figure 1-59(a)) is used for all data and control frames, but not all fields are used in all types of frames. There can be up to four address fields in an 802.11 frame. When all four fields are present, the address types include source, destination, transmitting station, and receiving station. The first two represent the end nodes and the last two may be intermediary nodes. 802.11 uses the same MAC-48 address format as Ethernet (Section 1.5.2). One of the fields could also be the BSS identifier, which is used in the probe request and response frames, used when mobile stations scan an area for existing 802.11 networks.

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 this transmission is completed. The deferral period is called **network allocation vector** (NAV), and we will see later in Figure 1-65 how it is used. In some control frames, this field contains a network association, or connection identifier.

The 802.11 physical-layer frame (Figure 1-59(b)) is known as PLCP protocol data unit (PPDU), where PLCP stands for “physical (PHY) layer convergence procedure.” The version shown in Figure 1-59(b) is known as *Long PPDU format*. A **preamble** is a bit sequence that receivers watch for to lock onto the rest of the frame transmission. 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 1-59(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 performance. (More discussion is provided in Chapter 6.)

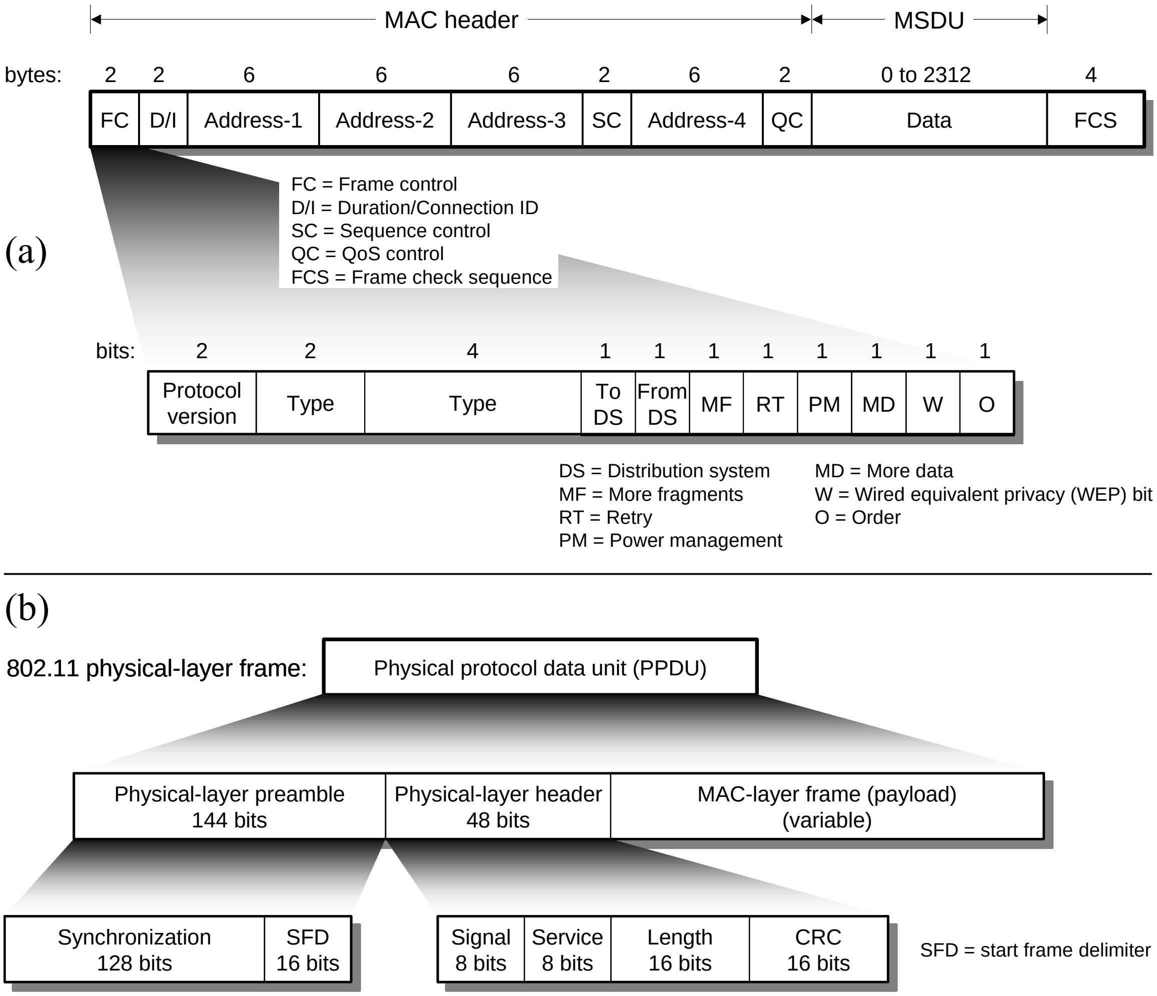


Figure 1-59: IEEE 802.11 (Wi-Fi) frame formats. (a) Link-layer (or, MAC-layer) frame format. (b) Physical-layer frame format (also known as Long PPDU format).

Medium Access Control (MAC) Protocol

The medium access control (MAC) protocol for IEEE 802.11 is a CSMA/CA protocol. As described earlier in 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*. The station sets a **contention timer** to a time interval randomly selected in the range [0, $CW-1$], and counts down to zero while sensing the carrier. If the carrier is idle when the countdown reaches zero, the station transmits.

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

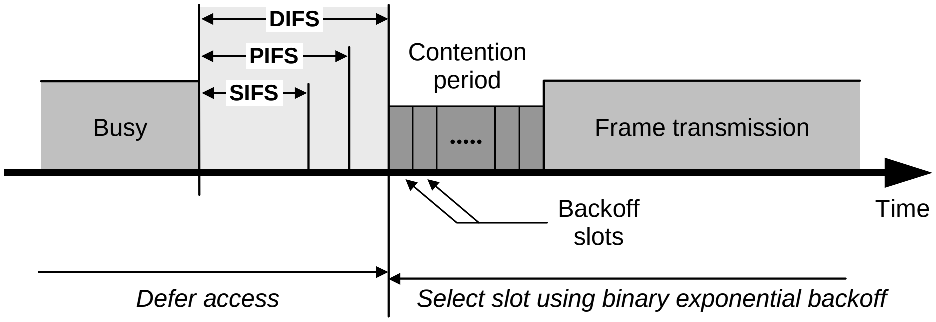


Figure 1-60: IEEE 802.11 interframe spacing relationships. Different length IFSs are used by different priority stations.

different types of traffic. Then, high-priority traffic does not have to wait as long after the medium has become idle. If there is any high-priority traffic, it grabs the medium before lower-priority frames have a chance to try.

Again, when a station wants to transmit data, it first senses whether the medium is busy. Two rules apply here:

1. If the medium has been idle for longer than an IFS corresponding to its priority level, 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 first waits for its assigned IFS, and then enters the access deferral state. The station can transmit the packet if the medium is idle after the contention timer expires.

To assist with interoperability between different data rates, the interframe space is a fixed amount of time, independent of the physical layer bit rate. There are two basic intervals determined by the physical layer (PHY): the *short interframe space* (SIFS), which is equal to the parameter β, 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[11], the *clear channel assessment* (CCA) interval, the air propagation delay on the medium, and the link-layer processing delay.

The four different types of IFSs defined in 802.11 are (see Figure 1-60):

SIFS: Short interframe space is used for the highest priority transmissions, such as control frames, or to separate transmissions belonging to a single dialog (e.g. Frame-fragment–ACK). 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 receive mode and be capable of decoding the incoming packet. For example, for the 802.11 FH PHY this value is set to 28 microseconds.

PIFS: PCF (or priority) interframe space is used by the PCF during contention-free operation. The coordinator uses PIFS when issuing polls and the polled station may transmit after the

[11] *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

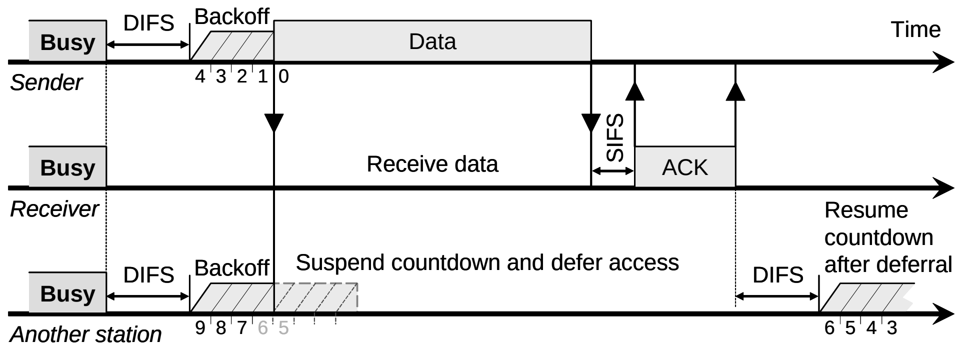


Figure 1-61: IEEE 802.11 basic transmission mode is a based on the stop-and-wait ARQ. Notice the backoff slot countdown during the contention period.

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 medium idle time for asynchronous frames contending for access. Stations may have immediate access to the medium if it has been free for a period longer than the DIFS. DIFS is equal to SIFS plus two slot times.

EIFS: Extended interframe space (not illustrated in Figure 1-60) is much longer than any of the other interframe spaces. It is used by any station that has received a frame containing errors that it could not understand. This station cannot detect the duration information and set its NAV for the Virtual Carrier Sense (defined later). EIFS ensures that the station is prevented from colliding with a future packet belonging to the current dialog. In other words, EIFS allows the ongoing exchanges to complete correctly before this station is allowed to transmit.

The values of some important 802.11b system parameters are shown in Table 1-4. The values shown are for the 1Mbps channel bit rate and some of them are different for other bit rates.

Table 1-4: IEEE 802.11b system parameters. (PHY preamble serves for the receiver to distinguish silence from transmission periods and detect the beginning of a new packet.)

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)
MTU*	Adjustable, up to 2304 bytes for frame body before encryption

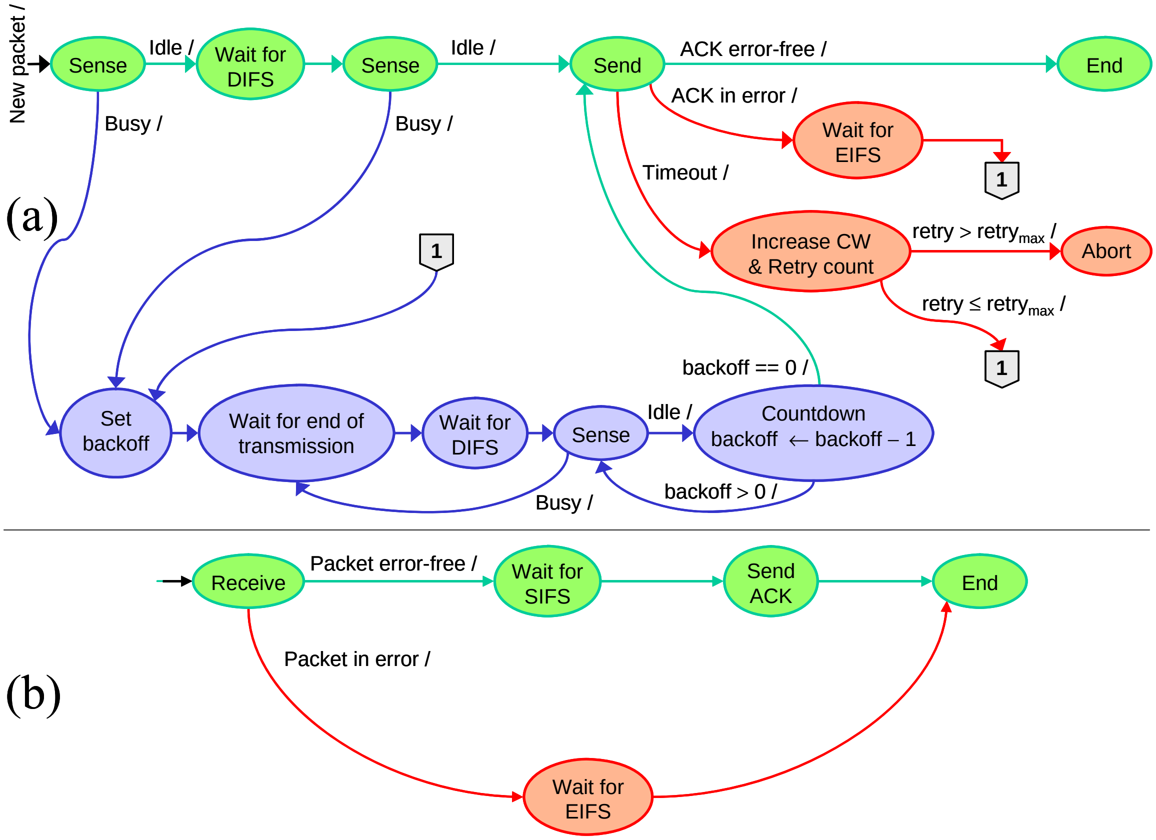


Figure 1-62: (a) Sender's state diagram of basic packet transmission for 802.11 MAC protocol. Compare to Figure 1-32. In "Set backoff," the backoff counter is set randomly to a number ∈ {0, …, *CW*–1}. (b) Receiver's state diagram for 802.11 MAC protocol.

(*) The Maximum Transmission Unit (MTU) size specifies the maximum size of a physical packet created by a transmitting device. The reader may also encounter the number 2312 (as in Figure 1-59(a)), which is the largest WEP encrypted frame payload (also known as MSDU, for *M*AC *S*ervice *D*ata *U*nit). Also, 2346 is the largest frame possible with WEP encryption and every MAC header field in use (including Address 4, see Figure 1-59(a)). In practice, MSDU size seldom exceeds 1508 bytes because of the need to bridge with Ethernet.

An example of a frame transmission from a sender to a receiver is shown in Figure 1-61. Notice that even the units of the atomic transmission (data and acknowledgement) 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) packet.

The state diagrams for 802.11 senders and receivers are shown in Figure 1-62. Notice that the sender's state diagram is based on the CSMA/CA protocol shown in Figure 1-32, with the key difference of introducing the interframe space.

Here is an example:

Example 1.6 Illustration of Timing Diagrams for IEEE 802.11

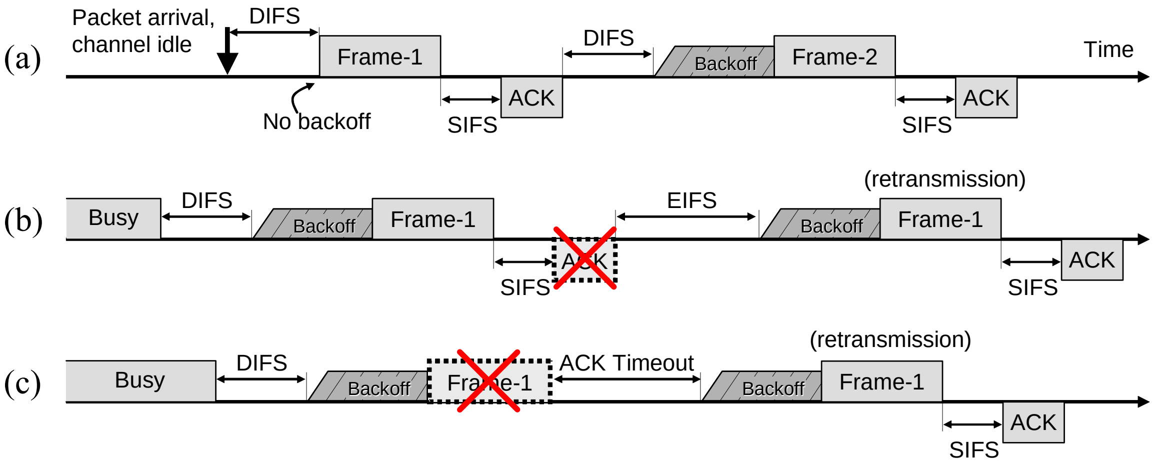


Figure 1-63: Timing diagrams. (a) Timing of successful frame transmissions under the DCF. (b) Frame retransmission due to ACK failure. (c) Frame retransmission due to an erroneous data frame reception.

Consider a local area network (infrastructure BSS) using the IEEE 802.11 protocol shown in Figure 1-62. 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 acknowledgement 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 1-63. The sender's actions are shown above the time axis and the receiver's actions are shown below the time axis. A crossed block represents a loss or erroneous reception of the corresponding frame.

The timing of successful frame transmissions is shown in Figure 1-63(a). If the channel is idle upon the packet arrival, the station transmits immediately, without backoff. However, it has to backoff for its own second transmission.

Figure 1-63(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 1-62.

On the other hand, if no ACK frame is received within a timeout interval, due possibly to an erroneous reception at the receiver of the preceding data frame, as shown in Figure 1-63(c), the transmitter contends again for the medium to retransmit the frame after an ACK timeout. (Notice that the ACK timeout is much shorter than the EIFS interval; in fact, ACK_timeout = $t_{\text{SIFS}} + t_{\text{ACK}} + t_{\text{slot}}$. Check Table 1-4 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

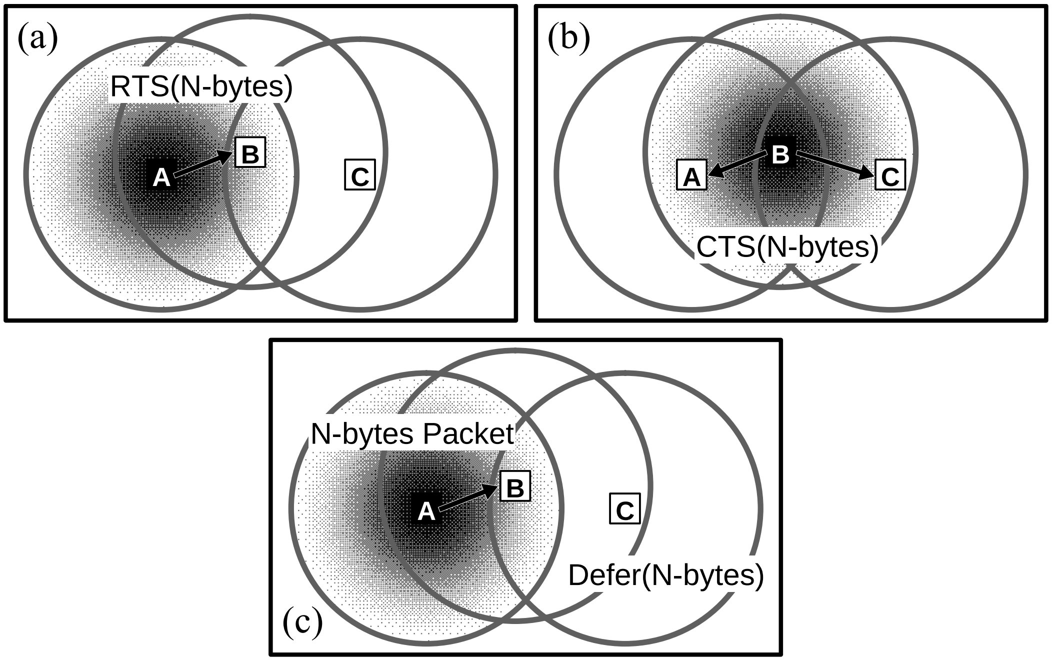


Figure 1-64: IEEE 802.11 protocol is augmented by RTS/CTS frames for hidden stations.

basic access method (Figure 1-61) with two more 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 1-4.) The process is shown in Figure 1-64. The sender first sends the RTS frame to the receiver (Figure 1-64(a)). If the transmission succeeds, the receiver responds by outputting another short frame (CTS). The CTS frame is intended not only for the sender, but also for all other stations in the receiver's range (Figure 1-64(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 (large) 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 1-64(c)). Notice also that the frame length is indicated in each frame, which is the Duration D/I field of the frame header (Figure 1-59(a)).

The 4-way handshake of the RTS/CTS/DATA/ACK exchange of the 802.11 DCF protocol (Figure 1-65) requires that the roles of sender and receiver be interchanged several times between pairs of communicating nodes, so neighbors of both these nodes must remain silent *during the entire exchange*. This is achieved by relying on the virtual carrier sense mechanism of 802.11, i.e., by having the neighboring nodes set their *network allocation vector* (NAV) values from the Duration D/I field specified in either the RTS or CTS frames they overhear (Figure 1-59(a)). By using the NAV, the stations ensure that atomic operations are not interrupted. The NAV time duration is carried in the frame headers on the RTS, CTS, data and ACK frames. (Notice that the NAV vector is set only in the RTS/CTS access mode and *not* in the basic access mode shown in Figure 1-61, because RTS/CTS perform channel reservation for the subsequent data frame.)

The additional RTS/CTS exchange shortens the vulnerable period from the entire data frame in the basic method (Figure 1-61) down to the duration of the RTS/CTS exchange in the RTS/CTS method (Figure 1-65). If a "covered station" transmits simultaneously with the sender, they will

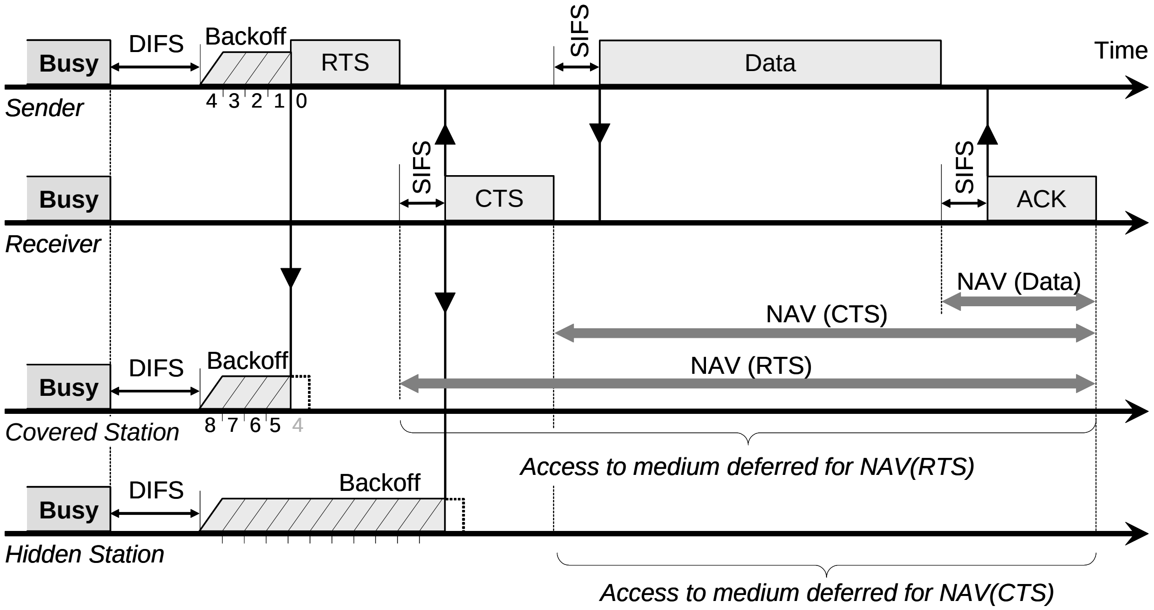


Figure 1-65: The 802.11 protocol atomic unit exchange in RTS/CTS transmission mode consists of four frames: RTS, CTS, Data, and ACK. (Compare to Figure 1-61.)

collide within the RTS frame. If the hidden stations hear the CTS frame, they 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 a collision happens, it will last only a short period because RTS and CTS frames are very short, unlike data frames, which can be very long. This RTS/CTS exchange partially solves the hidden station problem but the exposed node problem remains unaddressed. The hidden station problem is solved only partially, because if a hidden station starts with transmission *simultaneously* with the CTS frame, 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 1-28(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 a 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

This text reviews basic results about quality of service (QoS) in networked systems, particularly highlighting the wireless networks.

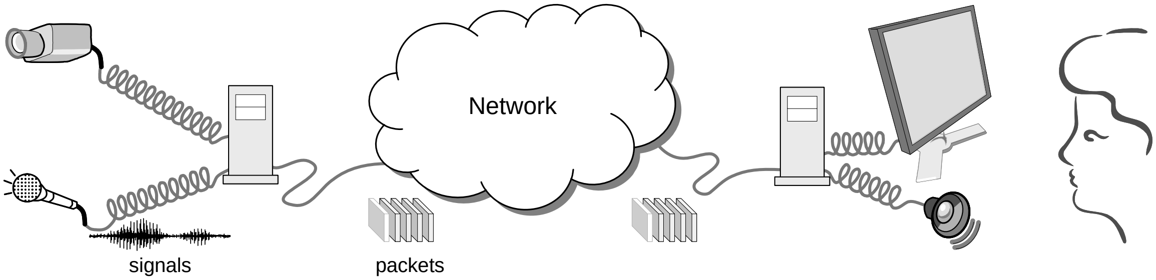


Figure 1-66: Conceptual model of multimedia information delivery over the network.

The recurring theme in this text is the ***delay and its statistical properties*** for a computing system. Delay (also referred to as *latency*) is modeled differently at different abstraction levels, but the key issue remains the same: how to limit the delay so it meets task constraints. A complement of delay is *capacity*, also referred to as *bandwidth*, which was covered in the previous volume. Therein, we have seen that the system capacity is subject to physical (or economic) limitations. Constraints on delay, on the other hand, are imposed subjectively by the task of the information recipient—information often loses value for the recipient if not received within a certain deadline.

Processing and communication of information in a networked system are generally referred to as *servicing* of information. The dual constraints of capacity and delay naturally call for compromises on the quality of service. If the given capacity and delay specifications cannot provide the full service to the customer (in our case information), a compromise in service quality must be made and a sub-optimal service agreed to as a better alternative to unacceptable delays or no service at all. In other words, if the receiver can admit certain degree of information loss, then the latencies can be reduced to an acceptable range.

In order to achieve the optimal tradeoff, the players (source, intermediary, and destination) and their parameters must be considered as shown in Figure 1-67. We first define information qualities and then analyze how they get affected by the system processing.

Latency and information loss are tightly coupled and by adjusting one, we can control the other. Thus, both enter the quality-of-service specification. If all information must be received to meet the receiver's requirements, then the loss must be dealt with within the system, and the user is only aware of latencies.

Time is always an issue in information systems as is generally in life. However, there are different time constraints, such as soft, hard, as well as their statistical properties.

We are interested in assessing the servicing parameters of the intermediate and controlling them to achieve information delivery satisfactory for the receiver.

Because delay is inversely proportional to the packet loss, by adjusting one we can control the other. Some systems are "black box"—they cannot be controlled, e.g., Wi-Fi, where we cannot control the packet loss because the system parameters of the maximum number of retries determine the delay. In this case, we can control the input traffic to obtain the desired output.

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, it could be within the organizational boundaries, concealed for security or privacy reasons.

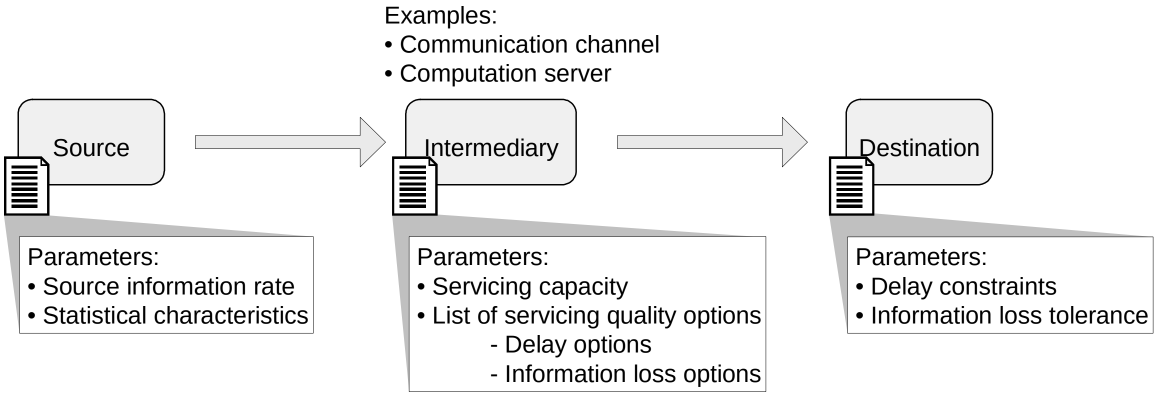


Figure 1-67: Key factors in quality of service assurance.

Figure 1-67 is drawn as if the source and destination are individual computers and (geographically) separated. The reality is not always so simple. Instead of computers, these may be people or organizations using multiple computers, or they could be networks of sensors and/or actuators.

Users exchange information through computing applications. A distributed application at one end accepts information and presents it 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. We call **traffic** the aggregate bitstreams that can be observed at any cut-point in the system.

The information that applications generate can take many forms: text, audio, voice, graphics, pictures, animations, and videos. Moreover, the information transfer may be one-way, two-way, broadcast, or multipoint.

Traffic management is the set of policies and mechanisms that allow a network to satisfy efficiently a diverse range of service requests. The two fundamental aspects of traffic management, *diversity* in user requirements and *efficiency* in satisfying them, act at cross purposes, creating a tension that has led to a rich set of mechanisms. Some of these mechanisms include flow control and scheduling (Chapter 5).

QoS guarantees: hard and soft

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 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.

In this text we first concentrate on the parameters of the network players, traffic characteristics of information sources, information needs of information sinks, and delay and loss introduced by the intermediaries. Then we review the techniques designed to mitigate the delay and loss to meet the sinks information needs in the best possible way.

Performance Bounds

Network performance bounds can be expressed either *deterministically* or *statistically*. A deterministic bound holds for every packet sent on a connection. A statistic bound is a probabilistic bound on 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 is delayed by more than 200 ms is smaller than 0.01.

Quality of Service

Network operator can guarantee performance bounds for a connection only by reserving sufficient network resources, either on-the-fly, during the connection-establishment phase, or in advance.

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 for consistent network data delivery. Some applications are more stringent about their QoS requirements than other applications, and for this reason (among others), we have two basic types of QoS available:

- **Resource reservation** (integrated services): network resources are apportioned according to an application’s QoS request, and subject to bandwidth management policy.
- **Prioritization** (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.

These types of QoS 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).

1.6.1 QoS Outlook

QoS is becoming more 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. 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

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. Many researchers feel that there is higher chance 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 service providers may not always want to have the best network performance. In other words, the two types of end users (service 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!

1.6.2 Network Neutrality vs. Tiered Services

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

Search the Web for net neutrality

topic. On one hand, consumers' rights groups and large Internet companies, such as Google and eBay, have tried to get 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. 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. Some industry analysts speak about a looming crisis as more Internet users send and receive bandwidth intensive content.

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.

See further discussion about net neutrality in Section 9.4.

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. To learn more about the basics of computer networking, the reader should consult another networking book. Perhaps two of the most regarded introductory networking books currently are [Peterson & Davie 2007] and [Kurose & Ross 2010].

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 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] describes upcall architecture. Hutchinson and Peterson [1991] describe 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.

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.

Path MTU Discovery for IP version 4 is described in RFC-1191, and for IPv6 in RFC-1981.

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 8.1.

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.

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 8.4), 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 entities creates the need to reconcile economic forces with engineering solutions. A routing protocol within a single administrative domain (or, 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 (or, Autonomous Systems) must allow them to implement their economic preferences. In 1980s when 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 advertisement 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 8.2.3). 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.

The IETF has defined a serial line protocol, the Point-to-Point Protocol (PPP) 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 1-53). 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].

The IEEE 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 latest 802.11n specification is described in Section 6.3.1

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

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 acknowledgements 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 it will take to transmit the entire message.

(b) If, instead, the message is sent as 2×*N* packets, each *L*/2 bits long, how long it will 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

Suppose host *A* has four packets to send to host *B* using Stop-and-Wait protocol. If the packets are unnumbered (i.e., the packet header does not contain the sequence number), draw a time-sequence diagram to show what packets arrive at the receiver and what ACKs are sent back to host *A* if the ACK for packet 2 is lost.

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 described in Example 2.1 (Section 2.2) 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

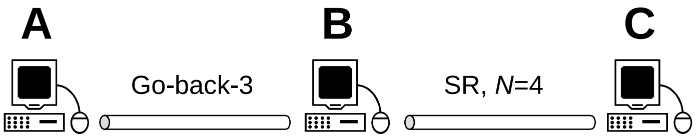
Consider a system of two hosts using a sliding-window protocol sending data simultaneously in both directions. Assume that the maximum frame sizes (MTUs) for both directions are equal and the acknowledgements may be piggybacked on data packets, i.e., an acknowledgement is carried in the header of a data 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). Notice 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.

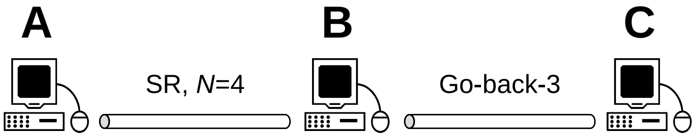


Draw side-by-side the timing diagrams for A→B and B→C transmissions up to the time where the first seven packets from *A* show up on *C*. Assume that the 2nd and 5th packets arrive in error to host *B* on their first transmission, and the 5th packet arrives in error to host *C* on its first transmission.

Discuss whether ACKs should be sent from C to A, i.e., source-to-destination.

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

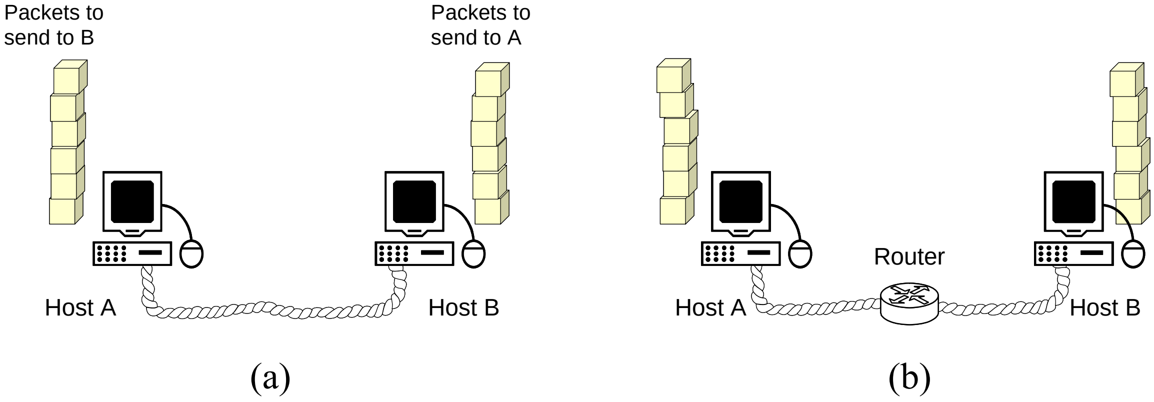
Assume the following system characteristics (see the figure below):

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 acknowledgement 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 acknowledgement. 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 acknowledgements immediately upon a successful packet reception. Rather, the acknowledgements 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) acknowledgement packet is sent to acknowledge all the packets received in this period. Otherwise, no acknowledgement 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. Notice 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?

(b) Again assuming 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

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 1.16

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 1.17

Problem 1.18

Consider a local area network using the CSMA/CA protocol shown in Figure 1-32. 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 1.19

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 1-32 for details of the algorithm.) What is the probability that the station will be in a particular backoff state?

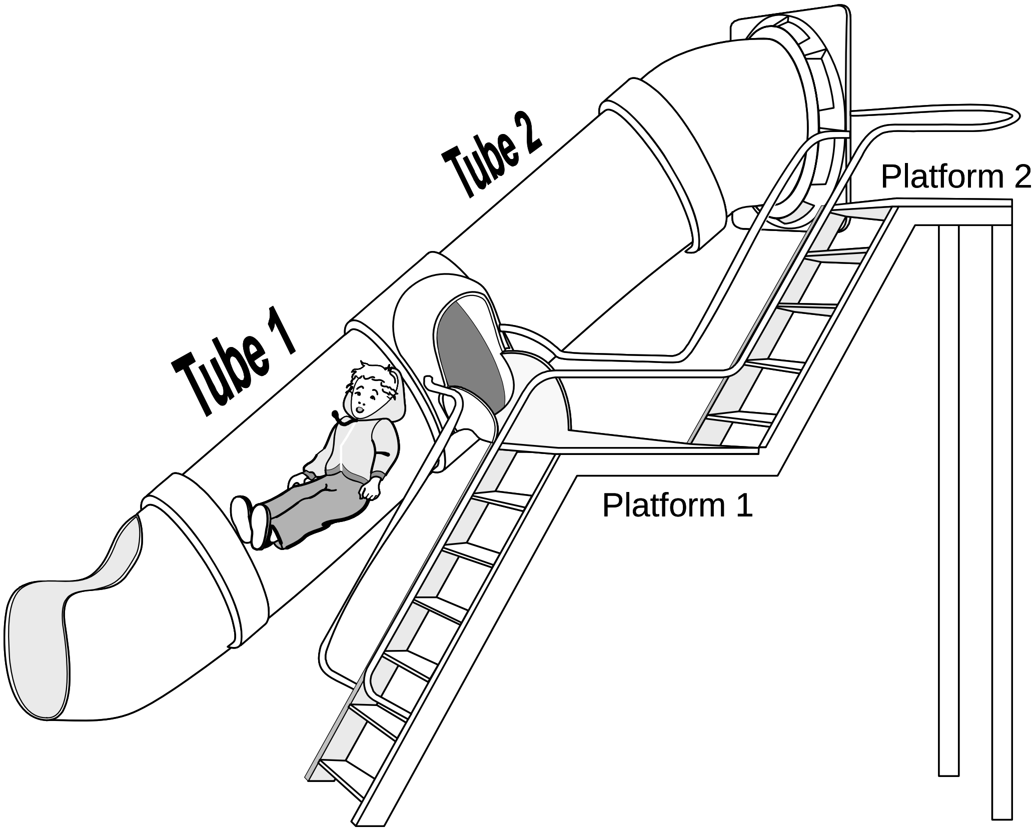


Figure 1-68: A playground-slide analogy to help solve Problem 1.19.

Hint: Figure 1-68 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 this problem as the problem of determining the probabilities that the kid will be found in Tube-1 or in Tube-2. For the sake of simplicity, we will distort the reality and assume that climbing the stairs takes no time, so the kid is always found in one of the tubes.

Problem 1.20

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 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

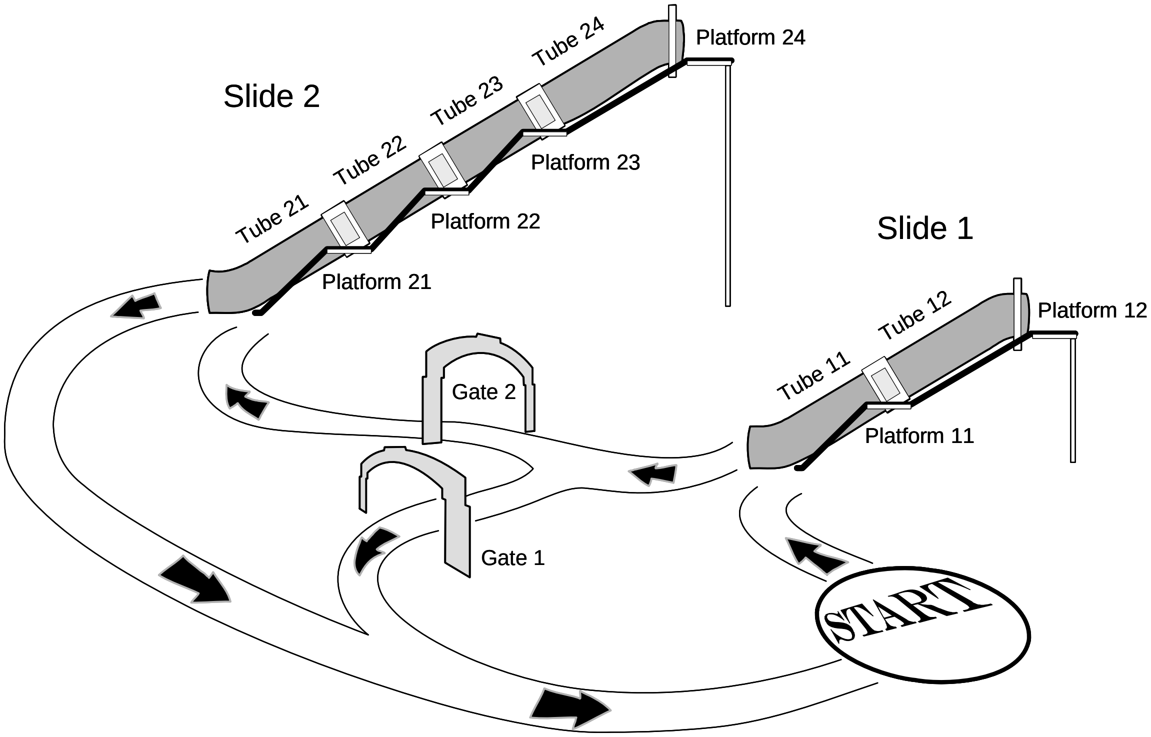


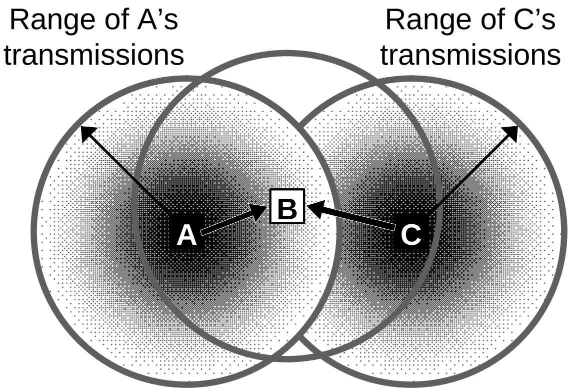
Figure 1-69: An analogy of a playground with two slides to help solve Problem 1.20.

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 1.19, we now imagine an amusement park with two slides, as shown in Figure 1-69. 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.

Problem 1.21

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 acknowledgements 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 acknowledgement for a preceding transmission. Let us denote the time when the acknowledgement transmission finished as $t = 0$. Do the following:

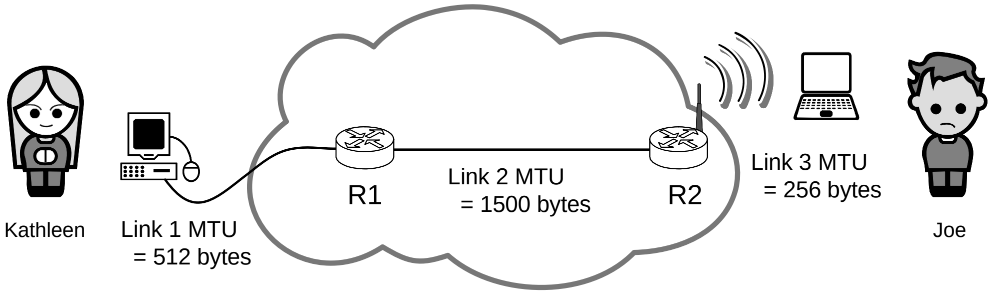
(a) Assuming that station *A* selects the backoff countdown b_{A1} = 12 slots, determine the vulnerable period for reception of the packet for *A* at receiver *B*.

(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), acknowledgement timeout time (t_{ACK}) and the backoff periods selected by *A* and *C* for their first transmission (b_{A1} and b_{C1}, respectively).

Problem 1.22

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.

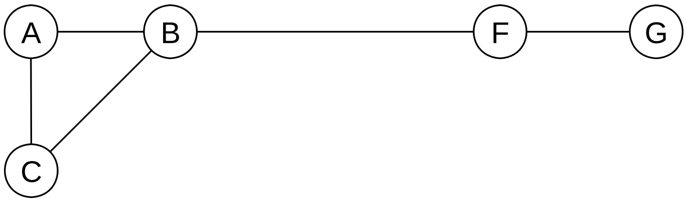
(b) How many fragments Joe receives on the IP level? Show the derivation.

(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 1.23

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

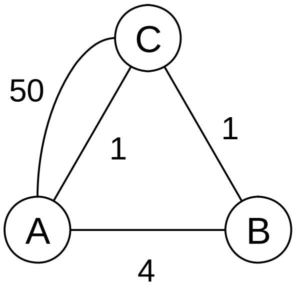
Show what link-state packets (LSPs) will flood back and forth for each step above. Assume that (*i*) the initial LSP sequence number at all nodes is 1, (*ii*) no packets time out, (*iii*) each node increments the sequence number in their LSP by 1 for each step, and (*iv*) both ends of a link use the same sequence number in their LSP for that link, greater than any sequence number either used before.

[You may simplify your answer for steps (b)-(f) by showing only the LSPs which change (not only the sequence number) from the previous step.]

Problem 1.24

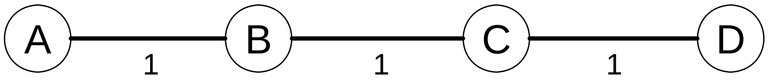
Problem 1.25

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 1.26

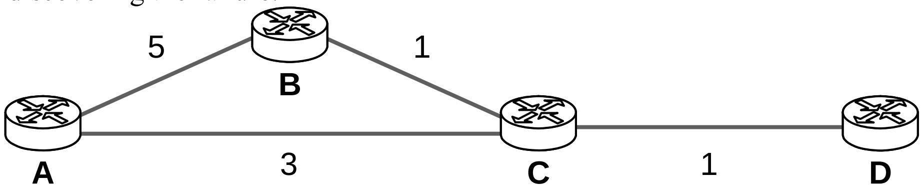
Consider the following network, using distance-vector routing:



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 1.27

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.

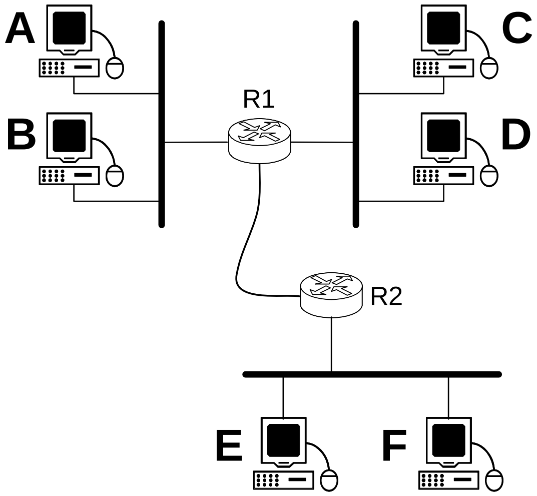
(b) Use the results from part (a) and show the forwarding table at node *A*. [Note: use the notation *AC* to denote the output port in node *A* on the link to node *C*.]

(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 1.28

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.

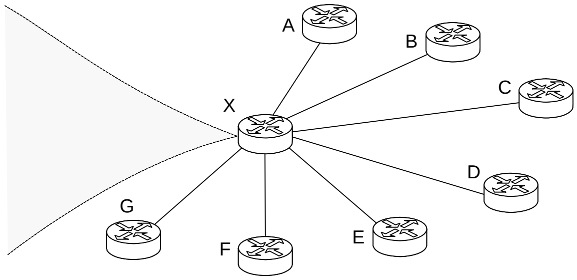


(a) Assign meaningfully the IP addresses to all hosts on the network. Allocate the minimum possible block of addresses for your network, assuming that no new hosts will be added to the current configuration.
(b) Show how routing/forwarding tables at the routers should look *after* the network stabilizes (do not show the process).

Problem 1.29

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 other match is found.)

Problem 1.30

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 1.31

Problem 1.32

Problem 1.33

Consider a local area network using the CSMA/CA protocol shown in Figure 1-32. 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 1.18.

Problem 1.34

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 1-4 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 1.35