Codes for a Distributed Caching based Video-On-Demand System

Sameer Pawar, Salim El Rouayheb, Hao Zhang, Kangwook Lee, Kannan Ramchandran
Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley
{spawar, salim, zhanghao, kangwooklee, kannanr}@eecs.berkeley.edu

***Abstract*—We study the role of codes in the optimization and design of a large-scale Video-on-Demand (VoD) system based on distributed caching, that we have architected and built at Berkeley. We show how network codes can convert a combinatorial problem into a tractable one, and enable a fully distributed algorithm that *jointly* optimizes the three-fold problem of cache content placement, cache-to-users topology selection, and cache-to-users rate-allocation. While a description of the general VoD system optimization and design can be found in [1], this paper focuses on the critical role of codes in enabling our VoD system. Specifically, we motivate and describe a specific class of network codes, called DRESS codes, that offer desirable tradeoffs between cache-to-user and cache-to-cache communication aspects of the problem needed to sustain a scalable VoD system.**

I. Introduction

The increasing popularity of Video-on-Demand (VoD) services motivates the need for a scalable and robust VoD content delivery solution. We have recently proposed, designed and built at Berkeley, an optimized VoD system based on distributed caching to address this problem [1]. A brief summary follows.

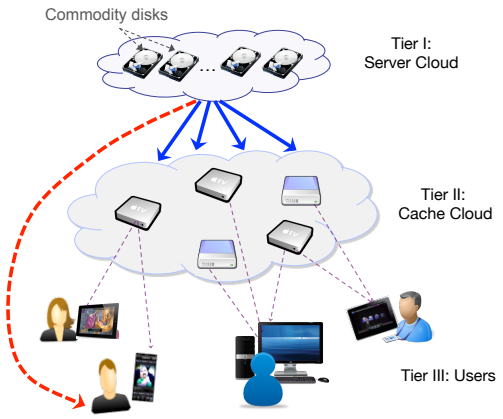

Fig. 1. The three-tier system architecture.

Our system architecture is three-tiered, as shown in Figure 1. The top tier is a server cloud which serves as a vault for a potentially massive catalog of content (movies, to be concrete) of demand to a large user base having an unknown demand distribution (bottom tier). We address the scalability requirement through a critical middle tier of distributed caches that sit between the server cloud and the users, and help distribute the content in a highly scalable and adaptive manner. These distributed cache nodes are inexpensive to deploy but are individually limited in storage, bandwidth, and the (degree bound of) number of I/O connections they can open up to the system users. That is, no single cache node can store more than a tiny fraction of the catalog content, or have the bandwidth to satisfy the streaming demands of more than a small subset of all users, or have the ability to open up connections to more than a small subset of the users in the system.

Given this architecture, our system goal is to effectively use the collective resources of the cache network in minimizing the load on the central server, which has to bear the burden of covering the deficit between the supply (by the collective cache network) and the demand (by the users). Concretely, our problem is to minimize the server load while satisfying the users' streaming demands for their respective content by optimizing (a) what content should be stored on each cache node (while respecting the storage constraints)? (b) which users should each cache connect to (while respecting the I/O degree bounds)? and (c) how should each cache node allocate its bandwidth among the users it connects to (while respecting the bandwidth constraints)? Further, in order to scale efficiently and have practical impact, we would like to solve this optimization problem in a distributed manner using a simple distributed algorithm.

While a general solution to this distributed optimization problem was provided in [1], we would like to emphasize the distributed optimization problem has two components of a combinatorially difficult nature. (i) A first difficulty is the content placement problem of deciding which packets of which movies should each cache node store to be of optimal system help. (ii) A second difficulty is the graph topology selection problem of deciding which users each cache should connect to (within the degree bound constraints). While details of how we overcome the second difficulty (ii) can be found in [2], [1], this paper provides the details of how we address the first difficulty (i).

Specifically, the content placement problem (i) is of a combinatorially explosive nature when there are a lot of choices of movies and cache nodes. We describe in this paper how we resolve this combinatorial explosion problem through the use of appropriately designed network codes, which convert this combinatorial non-multicast VoD problem into a tractable one, and enable a fully distributed algorithm that *jointly* optimizes the three-fold problem of cache content placement, cache-to-users topology selection, and cache-to-users rate-allocation.

At a high level, joint design of appropriate network codes, together with topology selection and rate-allocation strategies, allow not only for a tractable solution to this hard combinatorial problem, but even admit a fully distributed algorithmic solution (which we have deployed in our Berkeley VoD testbed). We describe here the critical enabling role of network codes in our VoD system due to their ability to provide a fluid model approximation to the content placement problem, thereby converting the combinatorially hard problem of packet

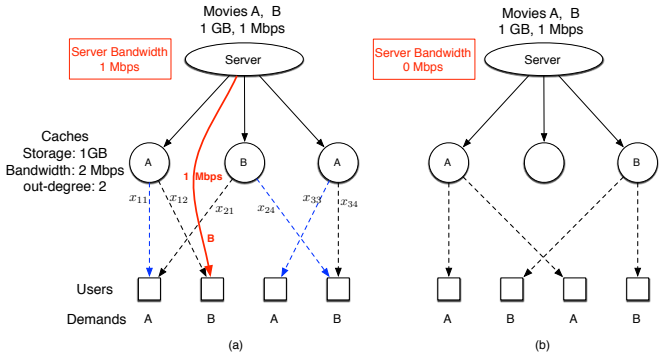

Fig. 2. A simple example of the problem of caching for a VoD system. The server has two movies of size 1GB and rate 1Mbps and there are 2 users requesting each movie. The system employs 3 identical cache nodes with constraints on storage (1GB), bandwidth (2Mbps) and out-degree equal to 2. The problem is to decide, for each cache, which movies to store, which users to connect to, and how much bandwidth to allocate for each user. These questions are in fact coupled. The connections between the cache nodes and users in (a) form a bad topology where the server has to provide at least one movie to a user. The topology in (b) is a good topology where the server does not have to intervene at all.

identity to the easy problem of content quantity. In the sequel, we motivate and describe a specific class of network codes called DRESS codes [4], [5] that are well-suited to our VoD system due to their ability to offer desirable tradeoffs between cache-to-user and cache-to-cache communication aspects of the problem needed to sustain a scalable VoD system.

Related work: Content distribution over networks have been studied in the literature, in terms of file distribution, live streaming and video-on-demand, both from theoretical and system perspectives [15], [16], [17], [18], also see [1] and references therein. The importance of optimal content placement in VoD systems have been explored in [18]. Recently, in [13], the problem of replication based caching (no coding) in wireless setting was studied, where the authors show that it is NP-hard and propose an approximation algorithm. The important role that codes play in caching for *multicast demand* has also been studied in the network coding literature [14].

II. Role of Codes

To minimize the server's total upload, we have to optimally leverage the resources of the cache nodes in order to maximize their contribution to the system. To illustrate the problem, let us consider a simple example depicted in the Fig. 2. Server has two movies, A and B, each of size 1GB and streaming rate of 1Mbps. There are 4 users in the system, two requesting movie A and the other two movie B. The cache cloud consists of 3 identical nodes each having 1 GB of storage capacity, 2 Mbps of upload bandwidth and can connect to any 2 users simultaneously. For each cache node, we have to decide which movie it should store, which users it should connect to, and how it should allocate its upload bandwidth among these users. Figure 2 shows that these problems are interdependent. If we choose a bad topology such as the one in Fig. 2(a), the server has to upload at least one movie to a user that is not served by the caches. However, Fig. 2(b) shows a good topology where the server does not have to intervene. Therefore, the content placement and topology problems are coupled, making the problem harder to solve for general large-scale settings. Later, we will show that even if the optimal topology is known (given by a genie), the problem remains hard.

In general, a cache node does not have to store a full copy of a movie. This may not be possible due to the storage limitation on the node. But more significantly, by having the freedom to store different movies in part rather than in full, a cache node has the potential to provide more help to the system. A user can then get the full movie he wants by contacting a few number of caches. Under this setting, to maximize the aggregate help provided by the cache cloud, the following questions have to be addressed:

(Q1) Topology: *Which users* among its neighbors should each cache connect to, while respecting the connectivity constraint?

(Q2) Content placement: *What packets of which movies* should each cache store, while respecting the storage constraint?

(Q3) Scheduling: *Which packets and to which users* should each cache node serve its content, while respecting the link rate constraints?

Each of these questions involves an exponential number of choices. Moreover, these questions cannot be solved independently, as hinted by the example in Fig. 2. The result is a hard combinatorial problem that is in general intractable.

The problem, as formulated above, remains hard even if we restrict it to the special case of multicast demands, *i.e.*, when all the users want to watch the same movie. However, it is known that the multicast problem is tractable and the key idea for solving it is to use *codes*. In this case, each cache stores coded packets of the movie. The intuition for the multicast case, as explained in the Fountain coding [6] and network coding literature [9], [10], [11], is that codes transform the content into a fluid where coded packets can be thought of as "drops". A user has to collect enough drops (per time unit), and does not need to keep track of the identity of the packets, to be able to decode and play the content. Therefore, solving the problem reduces to ensuring that the network can sustain a flow equal to the movie rate to each user[1].

In this work, we address a more general problem than classic multicast over a known network. First, we do not have a fixed network, rather we have to find the optimal topology within the design constraints of maximum connectivity degree. Secondly, we have a content-placement problem of deciding what content to place where in the cache cloud. Thirdly, we have a non-multicast setup where different users demand different content. We show how to solve our problem through the *joint design of codes, content placement strategies, and topology selection*.

In the non-multicast setting, one question that arises is, whether there is any benefit due to inter-movie coding, *i.e.*, generating coded packets by mixing packets from different movies? The example of Fig. 3 (b) answers this question affirmatively. While inter-movie coding can lead to further reduction in the server bandwidth, finding the right coding scheme is a non-trivial task especially that it is related to the general network coding problem which is known to be notoriously hard. To avoid this problem, in this work, we restrict coding to packets belonging to the same movie[2]. Thus, our VoD problem is effectively transformed into multiple multicast sessions that are *coupled together* since they have

[1]Practical codes can come close to this fluid abstraction.

[2]In many scenarios, this restriction may be a requirement imposed by the system to guarantee the privacy of the data since, with inter-movie coding, a user may have partial access to content that he is not authorized to receive.

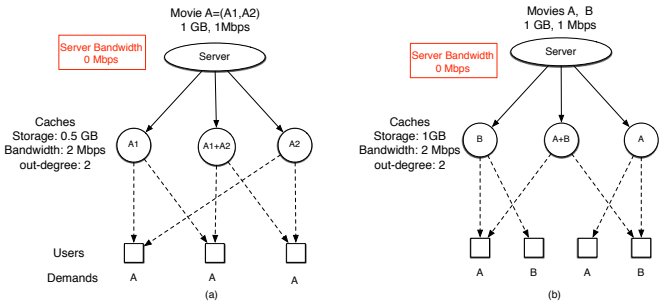

Fig. 3. (a) An example with VoD system with multicast demands where all the users want to watch the same movie A. By storing coded packets of the movie the bandwidth at the server is reduced to 0. Any other content placement at the caches that does not use codes will incur an bandwidth cost at the server of at least 1 Mbps. (b) An example of a VoD system where inter-movie coding (as done on the middle cache node) is needed to minimize the server load. Any other choice that does not involve inter-movie coding, for example storing movie A on the middle cache instead of $A + B$, will result in the server having an upload bandwidth of at least 1 Mbps.

to share the cache resources (storage, bandwidth, topology). The question now is *how much resources should be allocated to each multicast session?* To solve this problem, let us revisit our previously posed questions Q2 and Q3, always keeping the assumption that the answer to Q1 concerning the optimal topology is given by a genie. The key observation is that due to the use of coding which enables a fluid model of the content, the solution space to the remaining two questions gets exponentially reduced. Thanks to the use of codes, Q2 and Q3 now become:

(Q'2) *How many*, instead of "which", (coded) packets should each cache node store of each movie?

(Q'3) *How many*, instead of "what", packets should each cache node serve to each of the users he is connected to?

In summary, codes transform the problem from one of worrying about the combinatorics of individual packet "identities" to the easier one of only "quantities" of content. Answering the above two questions can be now formulated as an LP program[3]. Let us consider again the scenario of Fig. 2(a) and let us index the cache nodes from left to right as 1 to 3. Similarly, we index the users from 1 to 4. Also, let x_{ij} denote the rate on the link that connects cache i to user j, and f_{iA}, f_{iB} denote the fractions of movie A and movie B stored on cache i. Then the problem of maximizing the contribution of cache nodes, for example in Fig. 2 (a), can be formulated as below:

$$\begin{aligned}
&\text{maximize} \quad x_{11} + x_{12} + x_{21} + x_{24} + x_{33} + x_{34} \\
&\text{subject to} \\
&\qquad 1 \cdot f_{iA} + 1 \cdot f_{iB} \le 1 \;\; \} \text{Cache storage constraints} \\
&\qquad \textstyle\sum_j x_{ij} \le 2 \quad \forall i \;\; \} \text{ Cache bandwidth constraints} \\
&\qquad \textstyle\sum_i x_{ij} \le 1 \quad \forall j \;\; \} \text{User streaming constraints} \\
&\qquad \left.\begin{array}{ll} x_{11} \le 1 \cdot f_{1A}, & x_{12} \le 1 \cdot f_{1B} \\ x_{21} \le 1 \cdot f_{2A}, & x_{24} \le 1 \cdot f_{2B} \\ x_{33} \le 1 \cdot f_{3A}, & x_{34} \le 1 \cdot f_{3B} \end{array}\right\} \text{Availability constraints}
\end{aligned}$$

The cache storage and bandwidth constraints make sure that the total content stored on each cache and the sum upload rate of each cache are within its storage and bandwidth capacity respectively. The streaming constraints ensure that the total download rate of each user does not exceed the streaming rate of the movie. The last set of constraints, which we call feasibility constraints, essentially say that a cache node cannot serve a user more than what it has stored. This may sound innocuous but turns out to be an important constraint that "couples" storage and bandwidth. A general formulation of this LP, program along with a fully decentralized primal-dual algorithm for solving it, can be found in [1]. Moreover, the algorithm optimizes the topology selection by iteratively exploring the various topologies using a "Soft-Worst-Neighbor-Choking" algorithm with provable convergence guarantees based on a Markov approximation relaxation concept. The VoD system we built at Berkeley uses this algorithm. More details about the VoD system and the performance of the algorithm are discussed in the section V.

[3]Of course, since we are interested in a decentralized architecture, this LP needs to be solved using a distributed algorithm.

III. Code Design

In this section, we will elaborate on the choice of the practical codes for our VoD system. We start by describing general properties that such codes should satisfy. Then, we propose a solution based on a family of codes called DRESS (Distributed Replication-based Simple Storage) codes that offer desirable tradeoffs among these properties.

A. Desirable Code Properties

Due to the streaming nature of VoD systems, a movie is encoded in the following way. First, it is split into small *chunks* of length R packets, where R depends on the allowed play-buffer size, *i.e.*, the amount of initial buffering needed before playback starts. Each chunk is then encoded separately.

We identify the following properties that codes should satisfy when used for VoD systems:

1) MDS or Quasi-MDS property to approximate the fluid model. The code should allow a user to recover a video chunk formed of R packets by downloading *any* R, or a little more then R, of its coded packets. This corresponds to encoding the chunk using a Maximum Distance Separable (MDS) code, such as a Reed-Solomon (RS) code [19] or Quasi-MDS codes like Fountain codes [6].

2) Small block length to satisfy the buffering constraint. The streaming nature of the VoD problem imposes stringent restrictions on the the number R of the packets that can be coded together. For example, for a video of rate 1Mbps, a 10s play-buffer and packets of size 1 KB, $R = 1250$ packets.

3) Decentralized growth Our problem formulation in section II, precluded cache-to-cache communication. Since the formulation assumes a static scenario where users' demands do not change over time and cache nodes never leave the system, the cost of populating the caches from the server is amortized over time. However, in the scenarios where:

1) Cache nodes are volatile peers that stochastically join and leave the system e.g., in p2p system.
2) The server upload bandwidth and the number of simultaneous connections it can sustain is restrictive.
3) The users demands change abruptly, for example when the system experiences a "flash crowd" requesting a video that is becoming viral.

it may be more beneficial for the system, that a cache downloads its content from other caches instead of downloading from the server i.e., decentralized growth.

4) Security to protect the system from malicious nodes. When a cache node or a user downloads a packet from the cache network, it needs to verify its integrity and detect any malicious nodes in order to prevent it from polluting the whole network. When new coded packets are generated at the caches, verifying the integrity of these newly generated packets pose some security challenges on the system. Hence we require that all packets be created at and authenticated at the server.

Different codes in the literature, such as *random linear network codes* [10], *Fountain codes* [6] or *Regenerating codes* [7], [8] provide different tradeoffs of the code properties discussed in this subsection. In the VoD system we built, we chose a new class of network code called "DRESS codes", due to their desirable tradeoffs in terms of security, disk reads, computations and delay cost.

IV. Distributed Replication based Simple Storage(DRESS) code

A class of Distributed Replication based Simple Storage (DRESS) codes were introduced in [4], [5] in the context of a distributed storage application. In this section we show how these codes can be efficiently used for a VoD system. As explained in subsection III-A, encoding of video is performed by first splitting the video file into small chunks and then encoding each chunk separately. Let R denote the number of data packets in a chunk. For expository purpose, in this section, we treat a chunk of video as a file of interest.

A. DRESS Codes for VoD system

DRESS code first encodes the R data packets, of every chunk, using any outer (θ, R) (θ to be chosen later) MDS code at the server layer. A cache node that is required to store a certain fraction, say f, of a movie, stores the fraction f of each chunk of that movie and the DRESS code dictates how to store this fraction as explained next. Let the atomic unit of storage be α coded packets. A DRESS code consists of successively stored sets of α distinct coded packets, picked uniformly at random from the possible $\binom{\theta}{\alpha}$ subsets of the θ coded packets, till we store fR number of coded packets from every chunk of the movie. Each set of α coded packets is drawn independently and with replacement[4].

B. User Performance for DRESS codes

For the DRESS codes to satisfy the MDS or Quasi-MDS property (fluid model) specified in subsection III-A, we need $\theta \to \infty$. Due to practical limitations and as will be clear later for the 'decentralized growth' of DRESS codes, θ cannot be too large. We show in this subsection that, due to the way DRESS codes populate the cache nodes, even if θ is not too large, there is minimal loss in the system performance. To see this, consider the following example of DRESS code,

Example 1: Consider a DRESS code with parameters $R = 20, \theta = 40, \alpha = 5$, that uses RS-code as an outer code.
Table I shows the user performance for $\theta = 40$ (as compared to $\theta \to \infty$ in case of the true fluid model). Almost all the users get the file by accessing 6 to 7 atomic storage units instead of 4 as implied by the fluid model. From the system view point this translates into merely a control plane overhead[5].

More generally, let $F(k)$ denote a random number of coded packets observed by a user that accesses k number of atomic storage units. Then, using the *Azuma-Hoeffding* concentration result, we get the following bound on users performance,

$$\mathbb{P}\left(F(k) < R\right) \quad \leq \quad \exp\left(-\frac{(F_{ave}(k) - R)^2}{2k\alpha}\right). \qquad (1)$$

where $F_{ave}(k) = \theta(1 - (1 - 1/\theta)^{k\alpha})$ (see [4] for details).

C. Decentralized Growth for DRESS codes

Decentralized growth for DRESS codes occurs as follows: A cache node that desires to store a fraction f, of a movie, successively chooses the sets of indices of α distinct coded packets, picked uniformly at random, to sum up to fR. Each set is chosen independently and with replacement. After the cache node determines the total set, it randomly contacts some G other cache nodes in the system, and requests the coded packets corresponding to this set. This procedure amounts to the server providing the coded packets in the form of sets of uniformly random α packets, thus resulting in a seamless decentralized growth. As a result, the user performance guarantees remain as given in (1). Table I shows the expected behavior of G for the example 1.

The decentralized growth corresponds to cache node attempting to collect a *specific* set of fR coupons out of θ coupons. This coupon collection process can be slow, as also seen in the Table I. Also, it turns out that for DRESS codes there is tradeoff between the user performance and the growth performance [3]. To avoid the coupon collection process involved in the decentralized growth, one may try other alternatives like *Random sampling* or *rarest first*, but these algorithms suffer from well known issues like *the rarest packet syndrome* [12]. Although, some system level tweaks alleviate this issue and these algorithms are heuristically known to perform well for the delay insensitive applications like file download, they are not fundamentally well-suited to delay-sensitive applications like our VoD system under study.

V. Experimental results

We have built a VoD test-bed at Berkeley to validate the theoretical analysis and algorithm in a real system.

A. Video-on-Demand Test-bed

The system consists of a server, a tracker, and a number of user nodes and cache nodes interconnected by a wired TCP/IP network.

Server and Tracker: A server maintains all the available videos. Each video is divided into smaller *chunks* of duration $10s$ each. Each chunk is further divided into $R = 20$ data packets and encoded using an outer $(40, 20)$ Reed-Solomon code. DRESS code then populates the cache nodes with these coded packets.

Cache nodes: initially connect to randomly chosen users, respecting individual connectivity constraints. Each cache node

[4]One can also think of the coded packets as distinct coupons and DRESS codes as populating the cache nodes by successively drawing α distinct coupons uniformly at random from a bag containing a total of θ coupons.

[5]A user has to exchange control information to establish a connection with a cache node. Accessing slightly more, than minimal, number of cache nodes thus translates into additional control overhead. Once connected to the cache nodes, the user first exchanges the map of the coded packets and then requests *any* R distinct packets from its neighboring cache nodes, thus optimally utilizing the network bandwidth resource.

User Performance	Number of atomic storage units accessed	4	5	6	7	8	9	10
	Expected percentage of File missing	20.5375	7.2145	1.2211	0.1123	0.0068	0.0003	0
Decentralized Growth Performance	Number of atomic storage units accessed	10	15	20	25	30	35	40
	Expected percentage of packets missing	28.1988	14.9743	7.9517	4.2226	2.2423	1.1907	0.6323

TABLE I

TABLE SHOWING THE AVERAGE PERFORMANCE OF USER AND DECENTRALIZED GROWTH, FOR THE DREES CODE OF THE EXAMPLE 1, IN TERMS OF THE PERCENTAGES OF FILE AND PACKETS MISSING RESPECTIVELY, AS A FUNCTION OF NUMBER OF THE ATOMIC STORAGE UNITS ACCESSED.

then executes the primal-dual algorithm for the LP program [1] to distributively figure out the content placement and rate allocation. The topology is then updated every $30s$ using the "Soft-Worst-Neighbor-Choking" algorithm of [2], [1].

Users: maintain a play-back buffer equivalent to 4 chunks. As a particular user decodes and plays the chunk right ahead of its playback time, it also receives packets of the next unfulfilled chunk from the cache nodes. If the number of chunks in the play-back buffer falls below 2, the user immediately fetches the coded packets from the server, thus guaranteeing an interruption free streaming service.

B. Experimental setup and Results

Consider a setup with 3 videos that are $10, 11$ and 8 minutes long in duration, have streaming rate of $593, 450$ and 600 Kbps and of size $31, 36,$ and 46 MBytes respectively. There are 30 users and 20 cache nodes in the system. Each user independently picks a video from the video popularity distribution $\{0.6, 0.3, 0.1\}$. Each cache node has total storage capacity of $30MB$, a sum upload capacity of $750Kbps$ and a connectivity constraint of 6, *i.e.*, at any given time a cache node can serve maximum of 6 users. The objective is to maximally use the distributed cache network resources so that all users have an interruption-free streaming experience while minimizing the amount of data served by the server. The system performance plot, for this setup, is shown in Fig. 4. From the performance plots in Fig. 4, we note that the server

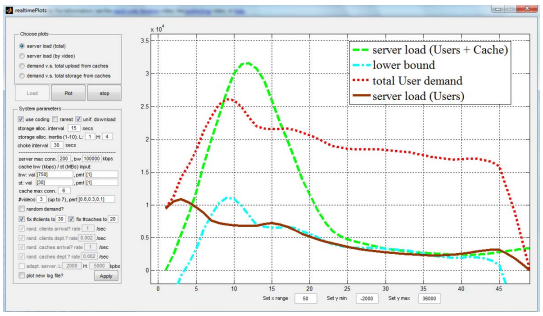

Fig. 4. System performance plots: 1) Brown plot: the server upload rate to users in Kbps versus time (1 unit is 10 sec) for the test-setup, 2) Green plot: total server upload rate to users plus cache (populating load) in Kbps versus time, 3) Red plot: Total user demand in Kbps versus time. 4) Blue plot: Lower bound on the server load computed using the system resources.

load, in terms of the amount of data uploaded by the server, to users is very close to the lower bound and is almost optimal. We have made similar observations for multiple test setups with varied configuration of resources.

VI. CONCLUSION

We have shown how appropriately designed network codes convert a combinatorially hard non-multicast VoD problem into a tractable one. In addition enabling a fully distributed algorithm that *jointly* optimizes the three-fold problem of cache content placement, cache-to-users topology selection, and cache-to-users rate-allocation. Based on this simple, distributed and provably optimal algorithm, we have built a VoD test-bed at Berkeley. Inspired by the practical aspects of the VoD system, we characterized the desirable properties of network codes that are well suited for VoD application and also proposed a new class of network codes called "DRESS codes" that provide a very desirable tradeoff of these properties.

REFERENCES

[1] H. Zhang, M. Chen, A. Parekh, K. Ramchandran, "An Adaptive Multi-channel P2P Video-on-Demand System using Plug-and-Play Helpers," *arXiv:1011.5469*, Nov 2010.

[2] H. Zhang, Z. Shao, M. Chen, and K. Ramchandran, "Optimal Neighbor Selection in BitTorrent-like Peer-to-Peer Networks," *in Proceedings of ACM SIGMETRICS*, San Jose, CA, US, June 7-11, 2011.

[3] S. Pawar, S. El Rouayheb, H. Zhang, K. Lee and K. Ramchandran, "Codes for a Distributed Caching based Video-On-Demand System," *Extended Version http://basics.eecs.berkeley.edu/BasicsGroup/ScalableVideo/VoDExtended.pdf* 2011.

[4] S. Pawar, N. Noorshams, S. El Rouayheb and K. Ramchandran, "DRESS Codes for the Storage Cloud: Simple Randomized Constructions," *Proceedings of IEEE International Symposium on Information Theory*, 2011.

[5] S. El Rouayheb and K. Ramchandran, "Fractional Repetition Codes for Repair in Distributed Storage Systems,"*in Proc. Annual Allerton Conf. On Comm. Control and Compute*, 2010.

[6] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data,". *Proc. ACM SIGCOMM*, September 1998.

[7] A. Dimakis, P. Godfrey, Y. Wu, M. Wainright and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inform. Theory*, 2005.

[8] K. V. Rashmi, N. B. Shah and P. V. Kumar, "Enabling Node Repair in Any Erasure Code for Distributed Storage," *Proceedings of IEEE International Symposium on Information Theory*, 2011.

[9] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, Jul. 2000.

[10] T. Ho, R. Koetter, M. Medard, M. Effros, J. Shi, and D. Karger, "A random linear network coding approach to multicast," *IEEE Transactions on Information Theory*, 2006.

[11] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, L. Tolhuizen, "Polynomial Time Algorithms for Multicast Network Code Construction," *IEEE Transactions on Information Theory*, June 2005.

[12] B. Hajek and J. Zhu, "The Missing Piece Syndrome in Peer-to-Peer Communication," *Proceedings of IEEE International Symposium on Information Theory*, 2010.

[13] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, G. Caire, "FemtoCaching: Wireless Video Content Delivery through Distributed Caching Helpers," *arXiv:1109.4179*, Sep. 2011.

[14] S. Huang, A. Ramamoorthy, M. Medard, "Minimum cost mirror sites using network coding: Replication vs. coding at the source nodes," *IEEE Transactions on Information Theory*, February 2011.

[15] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," *in IEEE INFOCOM*, 2005.

[16] C. Gkantsidis, J. Miller and P. Rodriguez, "Comprehensive view of live network coding P2P system," *in Proc. ACM SIGGCOMM/USENIX*, 2006.

[17] S. Annapureddy, C. Gkantsidis and L. Massoulie, "Proviondg video-on-demand using P2P networks," *in Proc. Internet Protocol TeleVision (IPTV)*, 2006.

[18] B. Tan and L. Massoulie, "Optimal content placement for P2P VoD systems," *IEEE INFOCOM*, 2011.

[19] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. SIAM*, vol. 8, June 1960.