

Exploring VoD in P2P Swarming Systems

Siddhartha Annapureddy
Stanford University
Stanford, CA, USA
reddy@scs.stanford.edu

Saikat Guha
Cornell University
Ithaca, NY, USA
saikat@cs.cornell.edu

Christos Gkantsidis, Dinan Gunawardena
Microsoft Research
Cambridge, UK
{chrisgk,dinang}@microsoft.com

Pablo Rodriguez
Telefonica
Barcelona, Spain
pablorr@tid.es

Abstract—Digital media companies have recently started embracing P2P networks as an alternative distribution channel. However, with current P2P swarming systems users need to download the full video, and hence, wait a long time before they can start watching it. While a lot of effort has gone into optimizing the distribution of large files, little research has been done on how to enable Video-on-Demand (VoD) functionality with P2P swarming systems. The main challenges reside in ensuring that users can start watching a movie at any point in time, with small start-up times and sustainable playback rates.

In this work, we address the issues of providing VoD using P2P mesh-based networks. We investigate scheduling techniques and network coding, in particular. Using both simulations and real experimental results, we show that high-quality VoD is feasible, and give guidelines to build play-as-you-download P2P swarming systems with high playback rates and low start-up delays.

I. INTRODUCTION

Peer-to-peer (P2P) systems have been immensely successful in large scale content distribution. In particular, while current P2P systems are heavily used to distribute video files ([1]), the users need to completely download the file (and hence, suffer long delays) before they can watch the video. Recently, systems such as CoolStreaming and others [2], [3], [4] have shown that it is feasible to use P2P systems to distribute live media content to a large number of users. However, it has been an open question whether similar P2P technologies could be used to provide a VoD service. A P2P VoD service is more challenging to design than a P2P live streaming system because, in addition to providing low start up delays, the system should also allow users arriving at arbitrary times to watch the video. The lack of synchronization among users reduces the block sharing opportunities and increases the complexity of the block transmission algorithms.

Video distribution over the Internet has been a prolific area of research [5], [6], [7], [8]. The particular problem of designing a VoD service has also received extensive attention in the past [7], [9], [10], [11], [12]. An important requirement of a VoD service is *scalability*, i.e., to be able to support a large number of users, as a typical video stream imposes a heavy burden both on the network and the system resources (e.g. disk I/O) of the server. The multicasting paradigm has been proposed to address the scalability issues [10], [9], [13]. However, these systems require a multicast-enabled infrastructure, which unfortunately has never materialized [14].

Peer-to-peer networks promise to provide scalable distribution solutions without infrastructure support. There are two fundamental approaches to building P2P systems: (a) tree-based (push) where trees (or, forests of trees) are usually constructed for dissemination of data [15], [16], [17], and (b)

mesh-based (pull) where peers exchange random blocks [18], [19]. Mesh-based systems do not enforce a structure on the overlay topology and, instead, promise high (swarming) efficiency by allowing peers to exchange random blocks with each other. As a result mesh-based systems have lower protocol overhead, are much easier to design, are more resilient to high rates of churn, and hence are more popular. However, while mesh P2P systems have proved to be efficient for bulk file dissemination, it is still an open question how efficient they can be in providing VoD. The difficulty lies in the fact that users need to receive blocks “sequentially” (and not in random order) in order to watch the movie while downloading, and, unlike streaming systems, the users may be interested in different parts of the movie, and may compete for system resources. The goal then is to design a P2P system which meets the VoD requirements, while maintaining a high utilization of the system resources.

In this paper, we study algorithms that provide the users with a high-quality VoD service while ensuring a high utilization of the system resources. We evaluate our algorithms using both extensive simulations and real experiments. Under different user arrival/departure patterns (heterogeneous user capacities etc. are not addressed in detail due to space constraints). The main results of this paper can be summarized as follows:

- (a) Naive, greedy scheduling algorithms provide bad VoD swarming throughputs. Applying Network Coding [20], [21], [22] over small time-windows of the video (e.g. a *segment* with a few seconds worth of video frames) reduces the risks of uploading duplicate content and minimizes the variance in the performance of each node, thus, improving the overall efficiency of the system.
- (b) While Network Coding solves the scheduling problem within a segment, scheduling across segments (spanning the entire video file) requires algorithms that avoid under-represented video portions. Such algorithms are feasible and can provide good system *throughput* while downloading blocks “pseudo-sequentially”.
- (c) We show that by combining network coding and segment scheduling we can design P2P systems that can provide a “play as you download” experience. We show that with the proposed system the playback rate that the system can support is close to the peer’s maximum bandwidth with some small start-up delay (i.e. initial buffering).

II. MODEL

We assume a large number of users (referred to also as clients, nodes, or peers) interested in some video content,

which initially exists on a special peer that we call the *server*. Users arrive at random points in time, and want to watch the video sequentially from the beginning (fast-forward functionality is not discussed for limitations of space). The resources (especially network bandwidth) of the server are limited, and hence, users should contribute their own resources to the system. The upload and download capacities of the users are also limited and typically asymmetric (i.e. the upload rate is smaller than the download).

A client joins the system by contacting a central tracker (whose address is obtained through an independent bootstrap mechanism). This tracker gives the client a small subset of active nodes (typically 6-8). The client then contacts each of these nodes and joins the network. At any point in time, a node is connected to a small subset of the active nodes, and can exchange content and control messages only with them. We call this subset the *neighborhood* of the node. The neighborhood changes as a result of node arrivals and departures, and because nodes periodically try to find new neighbors to increase their download rates. We assume cooperative (i.e., non-malicious) nodes.

The file is divided into a number of *segments*, which are further divided into *blocks*. The system is media codec agnostic, hence, nodes need to download all blocks; if a block is not available when needed, the video pauses and this is undesirable. Clients have enough storage to keep all the blocks they have downloaded.

III. DESIGN

We have used extensive simulations and measurements using a prototype to understand the factors that affect the performance of VoD over P2P networks, and to evaluate the performance of our algorithms. The simulator models important performance factors, such as access capacities, block scheduling algorithms, and allows us to experiment with large networks; it is described in Sec. III-A. The implementation gives us a more detailed insight into the operation of the system; it is described in Sec. III-B. In Sec. III-C, we describe the performance metrics we have used for our study.

A. Simulator

The simulator takes as input the size of the video file in units of *blocks* (typically 250 in our simulations), the number of nodes (typically 500), their capacities, and the times at which nodes join/depart the system. The simulator operates in discrete intervals of time called *rounds*. A client's upload/download capacity is given as the number of blocks that the client can transmit/receive in one round (typically 1). Each node connects to a small number of neighbors (typically 6-8). The topology changes during the simulation as a result of node arrivals and departures, and as the nodes try to find new neighbors to increase their download rates.

At every round, each node contacts its neighbors to identify those that have useful blocks. Then, there is a random matching of peers that can exchange content. All block transfers, both between peers and from the server, happen simultaneously, and then the system moves to the next round.

Note that while our simulator does not model realistic P2P networks in all their details (e.g. does not model network

delays, locality properties etc.), it does capture some of the important properties of mesh-based P2P networks. Hence, we feel that many of our results are applicable to the design of real mesh-based systems.

B. Implementation

We have developed a prototype to validate our results in a realistic setting. The system resembles typical P2P systems [21] and consists of three types of participants: peers, a tracker, and a logger. Content is seeded into the system by a special peer called server. The tracker enables peer discovery and peer matching. The active peers periodically report to the tracker (e.g. information about their content, rates etc.), and the tracker provides a subset of the active peers to nodes that have too few neighbors. The logger is an aggregation point for peer and tracker trace messages. Every peer in the system reports detailed statistics to the logger; using those statistics we are able to perform an in-depth evaluation of the various system parameters. We rate-limit the upload and download capacities of the peers using a token bucket based algorithm.

Each peer maintains 6-8 connections to other peers. Peers periodically connect to other peers at random and drop connections in an attempt to find better neighbors and increase their download rates. When testing network encoded transfers, we perform the encoding and decoding operations over a Galois Field $GF(2^{16})$; we also experiment with unencoded transfers. In most of our experiments, the file is divided into 100 original blocks (we have also experimented with larger number of blocks obtaining similar results).

In this paper, we will use our implementation to study small scale scenarios, which will highlight the design principles and interactions that need to govern an efficient VoD P2P-swarming system.

C. Methodology

The goal of our system is to ensure a *low setup time* (or initial buffering), and a high sustainable *playback rate* for all users, regardless of their arrival time. To evaluate the performance experienced by the user we do the following: For each user we plot the number of consecutive blocks from the beginning that the user has downloaded as a function of time (or rounds, in the case of simulations) (see Fig. 1). These blocks can be played without interruption. For a given setup time (i.e. amount of initial buffering), we calculate the sustainable playback rate as the maximum slope of a line that does not exceed the y-coordinate at any time. We call that rate the *goodput*. We typically report the median or average goodputs over all nodes and over multiple runs; when appropriate we also report the minimum and maximum values.

We are also interested in the total number of blocks exchanged per round, which we call *throughput*. This metric relates to the utilization of the system resources. Respectively we define the node throughput as the amount of information downloaded by a node in a unit of time. Observe that not all transfers increase the goodput. Hence, our objective is to **maximize throughput** for high system efficiency, while providing high *goodput* to ensure good playback rates for all nodes. In this paper, we show that this task, while non-trivial, is indeed feasible.

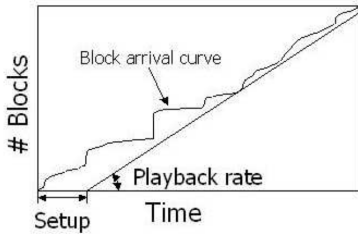


Fig. 1: This hypothetical graph shows the calculation of sustainable playback rate, given the setup time. The y-axis shows the number of consecutive blocks, while the x-axis shows the time.

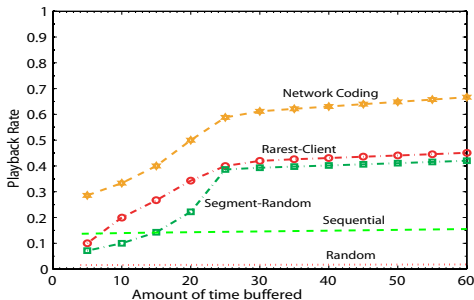


Fig. 2: Comparison of block scheduling

IV. NAIVE APPROACHES

In this section, we experiment with simple algorithms using a simulated network of 500 nodes all arriving at the same time (flash crowd scenario). We shall see that the naïve algorithms do not perform well.

Our first algorithm is inspired by current swarming systems that distribute the blocks of the file in *random* order. This strategy results in high block diversity and good system throughput. However, nodes receive blocks in random order that may not be useful to sustain a high goodput. In Fig. 2 we plot the average sustainable playback rate (i.e. goodput) as a function of the initial buffering. Observe that the rate is given as a fraction of the access link capacity, which is a natural upper limit on the maximum sustainable playback rate. Indeed, the average rate is less than 1% of the access link capacity, even though the system throughput is quite high with an average of 332.44 block exchanges per round (out of 500 maximum). Hence, despite the high throughput, the *random* method results in low goodputs and bad performance for video distribution.

Since nodes consume the blocks of the video sequentially, a natural algorithm could be to download the blocks in the playout order, i.e. *sequentially*. Indeed, Fig. 2 suggests that this policy performs better than *random* and is able to sustain playback rates of roughly 13.2%. Observe, however, that the peers have very similar blocks and hence there are fewer chances to find and exchange innovative blocks. Indeed, the throughput of the system reduced to an average of 65.97 block exchanges per round (15% of the total capacity), which in turn decreased the playback rate.

The *segment-random* policy attempts to combine the high swarming rate of *random* and the good playback rate of *sequential*. The method divides the file into *segments* which are groups of consecutive blocks; for example a file of 250 blocks is divided into 25 segments of 10 blocks each. The peers request blocks at random within a segment, but request segments in order. Fig. 2 suggests that *segment random* has a reasonable mean progress per round (170.21) and better playback rates (35%) than the other algorithms. Still however the performance of the segment-random policy is quite low.

V. NETWORK CODING

In this section, we study network coding techniques to optimize the goodput of the nodes, and the system's progress. Network coding has been proposed for improving the throughput of a network for bulk data transfer [22], [23], [20]. Network coding makes optimal use of bandwidth resources, and bypasses the block-scheduling problem by allowing all nodes to produce encoded data blocks. A good overview of network coding can be found in [24].

With network coding, any received block is useful with high probability. On the other hand, the node has to wait to download the complete file before it can start decoding. This is not acceptable in the context of VoD systems where a node wants to play the blocks soon after the download begins. We avoid this problem by restricting network coding to segments. A node only needs to wait until it downloads a complete segment before it can start decoding. This limits the benefits of coding since an encoded block is only useful to other nodes interested in a particular segment (rather than all the nodes). Moreover, this imposes an initial buffering time which is at least one segment size. (Note that non-uniform segment sizes can be used to minimize this start-up delay.) However, coding prevents the occurrence of rare blocks within a segment, and ensures that bandwidth is not wasted in distributing the same block multiple times. In essence, coding minimizes the risk of making the wrong upload decision.

We have evaluated the efficacy of network coding with our simulator and our prototype. Fig. 2 compares network coding against non-coding heuristics. (Please refer to [25] for the terminology used.) We note that network coding achieves a goodput of 62%, while the best rate without using network coding is 42% (with a setup time of 30 rounds). Also, the average progress of the system is 293.52 blocks with network coding, as compared to 209.57 without coding.

We now present the results from our implementation and evaluate the benefits of network coding. Consider a flash-crowd where 20 clients B_n join the network. The server has the entire file (100 blocks) divided into 10 segments; network coding is applied over all the blocks in a segment. A segment is decoded on-the-fly as soon as 10 linearly independent blocks are received for each segment.

We compare this to a *global-rarest* policy which does not use network coding. In the global-rarest scheme, a client requests the global rarest block in the target segment of its interest, either from the server or from its neighborhood. Note that this scheme requires global information which is not available in such a system, and is considered only for

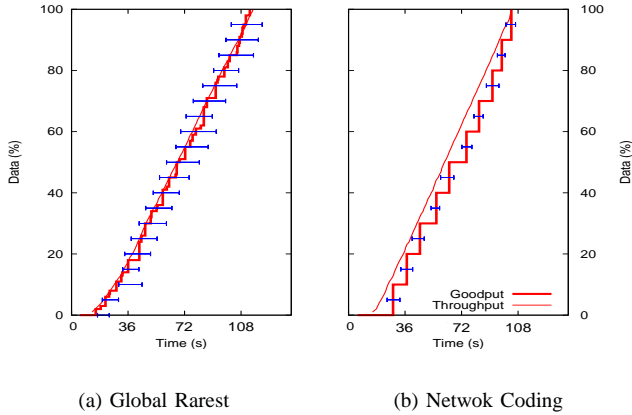


Fig. 3: Average throughput under different policies that do not coding (global rarest) and coding over a segment.

comparison purposes. Note also that this scheme performs the best amongst non-coding policies.

Fig. 3(a) and 3(b) show the throughput and the goodput of the nodes in the system with the global-rarest and network coding policies. The bars mark the maximum and minimum value. Given that global-rarest uses global information about the system, we would expect that it performs optimally. However, this is not the case. Network coding provides a greater throughput than the global-rarest scheme (about 14% better), and, more importantly, it results in significantly less variance and more predictable download times. We have also observed that network coding provides greater benefits in other scenarios that include dynamic arrivals and departures, heterogeneous network capacities, and limited peer network visibility. Due to limitations of space, we do not present those results.

In summary, network coding minimizes the risk of uploading duplicate blocks within a segment. To further improve the performance of the system, next section considers better algorithms for scheduling across segments.

VI. SEGMENT SCHEDULING POLICIES

We now take a deeper look at how segment policies can impact the performance of a VoD P2P system. To this end, we will use our implementation to get a better understanding of all the interactions in a realistic setting.

Segment policies form the analogue of the block scheduling problem ([25]) at the segment granularity. As with naïve block scheduling, we show that a naïve segment policy where clients greedily request blocks from their earliest incomplete segments adversely affects the system throughput. While block scheduling inside a segment is amenable to coding, coding cannot be used *across segments* since the entanglement it creates prevents streaming VoD. Instead, we propose a heuristic-based solution that schedules segments according to how poorly they are seeded in the network. This approach is similar in spirit to traditional rarest-first approaches (though the caveats are not discussed due to space constraints).

Segment policy affects the overall throughput of the system when not all segments are equally represented in the network.

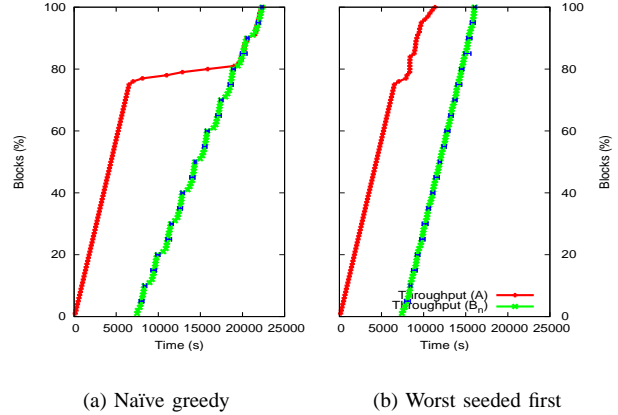


Fig. 4: Average throughput in a flash-crowd ($B_{1..20}$ join when A is 75% done) under different segment policies.

As an example consider the case of a bandwidth-constrained node that contains blocks from both under-represented and popular segments; to increase the system efficiency this node should upload blocks from the under-represented segment. This situation is most prominent when a flash-crowd arrives in the middle of an ongoing download. Consider a server that has the entire file, which is divided into 10 segments containing 10 blocks each. The block policy used within a segment is network coding as described in Section V. One client A has downloaded 75% of the file, when a flash-crowd of 20 clients B_n join the network. For simplicity, we consider nodes having equal upload and download capacities.

Under naïve segment scheduling (as above), the server's upload capacity is shared between client A requesting blocks from segments near the end of the file, and multiple clients, B_n (number depending on the outbound degree of the server), requesting blocks from the first segment(s). Since only the server has the end of the file, and the flash-crowd causes the server's available upload bandwidth to be used in sending blocks from earlier segments, the overall network throughput is reduced.

Figure 4(a) shows the throughput experienced by A and the average throughput of B_n as a function of time. Error bars mark the maximum and minimum values. From the figure, A initially enjoys good throughput (rate-limited by the server's upload bandwidth) until the flash-crowd joins. After this point, A 's throughput is severely reduced as the server re-uploads the initial parts of the file to some B_n s. The server's upload bandwidth is wasted in uploading these segments already represented in the network (at A).

The overall throughput improves if all the nodes seek to improve the diversity of segments in the network. If the segment policy is to upload a block from a lesser represented segment whenever possible (*worst-seeded-policy*), throughput improves significantly for both existing and new nodes as seen in Figure 4(b). The figure plots the throughput for A and B_n under our worst-seeded-first policy (fully described below). Note that A 's throughput near the end of the file is noticeably increased, because the server continues to serve blocks from

later segments to B_n , and A subsequently retrieves these blocks from B_n .

Algorithm 1 SELECTSEGMENT(S,D)

Require: S is source node

Require: D is destination node

```

1:  $A_S \leftarrow \text{AVAILABLESEGMENTS}(S)$ 
2:  $C_D \leftarrow \text{COMPLETEDSEGMENTS}(D)$ 
3:  $P \leftarrow \text{SORTSEGMENTSWORSTSEEDEDFIRST}(A_S \setminus C_D)$ 
4:  $C_S \leftarrow \text{COMPLETEDSEGMENTS}(S)$ 
5:  $I_D \leftarrow \text{EARLIERSTINCOMPLETESEGMENT}(D)$ 
6: for  $i = 0 \dots \text{COUNT}(P)$  do
7:   if  $P_i = I_D$  or  $P_i \in C_S$  then
8:     return  $P_i$ 
9:   end if
10: end for

```

Algorithm 1 describes our worst-seeded-first segment scheduling policy in pseudo-code. We assume that the source node has knowledge of the rarity of segments aggregated across the other nodes in the network; the aggregation can be performed either centrally at the tracker, or can be approximated in a distributed fashion by gossiping between neighboring nodes. Our implementation performs this aggregation at a central tracker.

Our segment policy heuristically increases the diversity of segments in the network. Amongst the candidate segments available at the source node and not available in full at the destination node (lines 1–3), our implementation picks the segment that is least well-represented (lines 3,6) subject to the conditions on line 7. If the poorly seeded segment is immediately of interest to the destination then it is uploaded (clause 1, line 7); otherwise, the source uploads blocks from segments it has completed downloading (clause 2, line 7), to ensure that the block is “new” (across the network; such a block is called globally innovative) with high probability.

Our algorithm hinges on having a good estimate of how well-represented a segment is. This estimate should include nodes that have the complete segment, and those that have partially downloaded the segment. In our implementation, the tracker monitors the rarity of segments in the network. Clients in our system report the fraction of blocks they have received from each segment. Those fractions are used to estimate the popularity of the segments; for example, a segment is considered under-represented if the vast majority of nodes have very few blocks from that segment.

VII. SUMMARY

In this paper we have examined the problem of designing a video on demand service using mesh-based P2P networks. Mesh-based P2P systems are relatively simple to engineer and result in high utilization of system resources, and as a result they have been very successful for large scale bulk file distribution. Unfortunately, however, those systems give very bad performance when used for VoD. We have proposed to combine network coding, that further improves the performance of distributing segments, and segment scheduling, that results in high system throughput while downloading content “pseudo sequentially”, to provide efficient VoD with

small setup delays. Our simulations and experiments suggest that indeed the combination of network coding and segment scheduling provides a significantly performance improvement compared to other algorithms (even compared to algorithms that use global knowledge). Even though further work is required to get a better understanding of the potential benefit of the proposed algorithms in more realistic scenarios, we believe that the guidelines proposed in this paper can be used to build high-performant P2P VoD systems.

REFERENCES

- [1] Andrew Parker, “P2P in 2005,” <http://www.cachelogic.com>, 2005.
- [2] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, “CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming,” in *IEEE Infocom*. 2005, IEEE Press.
- [3] “Pplive,” <http://www.pplive.com/>.
- [4] “Feidian,” <http://tv.net9.org/>.
- [5] Shanwei Cen, Calton Pu, Richard Staehli, Crispin Cowan, and Jonathan Walpole, “A distributed real time MPEG video audio player,” in *NOSSDAV*, 1995.
- [6] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang, and Jon M. Peha, “Streaming video over the internet: Approaches and directions,” *IEEE Tran. on circuits and systems for video technology*, vol. 11, no. 3, pp. 282–300, Mar 2001.
- [7] Ailan Hu, “Video-on-demand broadcasting protocols: A comprehensive study,” in *IEEE Infocom*. Apr 2001, pp. 508–571, IEEE Press.
- [8] John G. Apostolopoulos, Wai tian Tan, and Susie J. Wee, “Video streaming: Concepts, algorithms, and systems,” <http://www.hpl.hp.com/techreports/2002/HPL-2002-260.pdf>, Sep 2002.
- [9] S. Viswanathan and T. Imiechski, “Pyramid broadcasting for video on demand service,” in *IEEE Multimedia Computing and Networking Conference*. 1995, IEEE Press.
- [10] Kevin C. Almeroth and Mostafa H. Ammar, “On the use of multicast delivery to provide a scalable and interactive Video-on-Demand service,” *Journal of Selected Areas in Communications*, vol. 14, no. 6, pp. 1110–1122, 1996.
- [11] Simon Sheu, Kien Hua, and Wallapak Tavanapog, “Chaining: A generalized batching technique for video-on-demand systems,” in *International Conference on Multimedia Computing and Systems (ICMCS’97)*. 1997, IEEE Press.
- [12] Martin Reisslein Despina Aparilla, Keith Ross, “Periodic broadcasting with vbr-encoded video,” in *IEEE Infocom*. 1999, IEEE Press.
- [13] Kien A. Hua and Simon Sheu, “Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems,” in *ACM SIGCOMM*. 1997, pp. 89–100, ACM Press.
- [14] Mostafa Ammar, “Why johnny can’t multicast: Lessons about the evolution of the internet,” *NOSSDAV Keynote Speech*, June 2003.
- [15] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh, “Splitstream: High-bandwidth multicast in a cooperative environment,” in *19th ACM Symposium on Operating Systems Principles (SOSP’03)*, Oct. 2003.
- [16] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang, “A case for end system multicast,” in *Measurement and Modeling of Computer Systems*, 2000, pp. 1–12.
- [17] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O’Toole, Jr., “Overcast: Reliable multicasting with an overlay network,” in *4th ACM Operating Systems Design and Implementation (OSDI’00)*, 2000, pp. 197–212.
- [18] “Gnutella,” <http://gnutella.wego.com/>.
- [19] Dejan Kostic, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat, “Bullet: high bandwidth data dissemination using an overlay mesh,” in *Proceedings of the 19th ACM symposium on Operating systems principles (SOSP)*, Bolton Landing, NY, USA, October 2003, pp. 282–297.
- [20] Christos Gkantsidis and Pablo Rodriguez, “Network coding for large scale content distribution,” in *IEEE Infocom*, 2005.
- [21] Christos Gkantsidis, John Miller, and Pablo Rodriguez, “Anatomy of a p2p content distribution system with network coding,” in *IPTPS*, 2006.
- [22] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” *IEEE Trans. on Information Theory*, vol. 46, pp. 1204–1216, 2000.
- [23] P. A. Chou, Y. Wu, and K. Jain, “Practical network coding,” in *Allerton Conference on Communication, Control, and Computing*, Oct 2003.

- [24] M. Mard, R. Koetter, and P. A. Chou, "Network coding: A new network design paradigm," in *IEEE International Symposium on Information Theory*, Adelaide, Sep 2005.
- [25] Siddhartha Annapureddy, Christos Gkantsidis, and Pablo Rodriguez, "Providing video-on-demand using peer-to-peer networks," in *Internet Protocol Television (IPTV) Workshop, WWW '06*, Edinburgh, Scotland, May 2006.