Is High-Quality VoD Feasible using P2P Swarming? *

Siddhartha Annapureddy
New York University
New York City, NY, USA
reddy@scs.stanford.edu

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

Christos Gkantsidis
Microsoft Research
Cambridge, UK
chrisgk@microsoft.com

Dinan Gunawardena
Microsoft Research
Cambridge, UK
dinang@microsoft.com

Pablo Rodriguez
Telefonica Research
Barcelona, Spain
pablorr@tid.es

ABSTRACT

Peer-to-peer technologies are increasingly becoming the medium of choice for delivering media content, both professional and home-grown, to large user populations. Indeed, current *P2P swarming* systems have been shown to be very efficient for large-scale content distribution with few server resources. However, such systems have been designed for generic file distribution and provide a limited user experience for viewing media content. For example, users need to wait to download the full video before they can start watching it. In general, the main challenge resides in designing systems that ensure 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 a Video-on-Demand (VoD) using P2P mesh-based networks. We show that providing high quality VoD using P2P is feasible using a combination of techniques including (a) network coding, (b) optimized resource allocation across different parts of the video, and (c) overlay topology management algorithms. Our evaluation also shows that systems that do not optimize in all these dimensions could significantly under-utilize the network resources resulting in poor VoD performance. We present our results based on simulations and a prototype implementation.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; E.4 [**Data**]: Coding and Information Theory; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems, Performance Evaluation (efficiency and effectiveness)*; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Video*

General Terms

Algorithms, Experimentation

Keywords

content distribution, network coding, overlays, peer-to-peer, topology management, video streaming

*An earlier version of this paper appeared in [4].

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

1. INTRODUCTION

Peer-to-peer (P2P) systems have been immensely successful for large scale content distribution. Current peer-to-peer applications generate a large percentage of the traffic over the Internet and, more relevant to this paper, a large fraction of that traffic relates to distributing video content [30]. However, with current systems, the users need to download the complete file, and as a result suffer long delays before they can watch the video. Recently systems such as CoolStreaming and others [15, 32, 38] have been very successful in delivering live media content to a large number of users using mesh peer-to-peer technology. However, it has been an open question whether similar P2P technologies can be used to provide a VoD service. A P2P VoD service is more challenging to design than a P2P live streaming system because the system should allow users arriving at arbitrary times to watch (arbitrary parts of) the video, in addition to providing low start up delays. The fact that different users may be watching different parts of the video at any time can greatly impact the efficiency of a swarming protocol. The lack of synchronization among users reduces the block sharing opportunities and increases the complexity of the block transmission algorithms.

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 [7, 11, 24], and (b) mesh-based (pull) where peers exchange random blocks [18, 26]. 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) Naïve, greedy scheduling algorithms provide bad VoD swarming throughputs. Applying Network Coding [1, 16, 17] 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) The performance of the system, which translates to high utilization of system resources and also good user experience, depends critically on creating proper mesh topologies using efficient peer-matching algorithms. Such peer-matching algorithms should take into account both the content at each peer, as well as their bandwidth.

(d) We show that by combining network coding, segment scheduling, and peer-matching algorithms 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).

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 gives a brief outline of our system. Section 4 describes our evaluation methodology, as well as, the simulator and the prototype we have used to evaluate the system. Section 5 argues that naïve extensions to current P2P swarming systems and shows are inadequate to support VoD. In particular, to support VoD over mesh P2P technology we need efficient block and segment scheduling algorithms; they are described in Section 6 and Section 7 respectively. Topology management is described in Section 8. The adaptation of our algorithms in the presence of clients with heterogeneous capacities is discussed in Section 9.

2. RELATED WORK

Video streaming over the Internet has been on of the most prolific research areas for over a decade, see [5, 9, 21, 36] and the references therein. Most related to this work are the research efforts for designing video distribution systems that can support a large number of users. Multicasting has been proposed to provide a scalable video streaming service, even in the presence of non-homogeneous receivers [25, 27, 31]. Multicasting is a natural paradigm for live video streaming. It has been also extended for supporting near-Video-on-Demand services. The simplest approach is to periodically start a new broadcast scheme [2]. More elaborate schemes propose to divide the video into segments and distribute each segment in different multicast channels [21, 22, 34]. The main disadvantage of such systems is that there is no support for native multicasting in the Internet today.

Even though native multicasting is not available, there have been many proposals to use overlay multicast distribution for live streaming events [8, 14, 19, 23, 33, 37]. Observe that such systems support live media streaming and not near-Video-on-Demand. Moreover, there is extra network overhead and complexity for maintaining the overlay topology. It is an open question whether such overlay multicasting approaches can be extended to support near-VoD, maybe by using similar approaches as in [22, 34]. Inspired by the success of unstructured P2P networks, the authors in [28, 38] propose to use mesh-P2P networks for live video streaming. Similarly to our approach, [28, 38] use mesh-based P2P networks and, hence, have low overhead for topology maintenance, but, unlike our approach, support live video streaming instead of nVoD.

More relevant to our work are the BASS [13] and BiToS [35] systems. BASS extends the current BitTorrent system [12] to provide a near-Video-on-Demand service [13]. BASS assumes that there is a streaming server, that all nodes connect to the streaming server, but, all nodes use the P2P network to help each other and alleviate the load from the server. Even though BASS reduces the load at the server by a significant amount, the design of the system is still server oriented, and, hence, the bandwidth requirements at the server increase linearly with the number of users. In this work we assume that peers rely only on the P2P network to retrieve the content; hence, in our approach it is important to understand the performance of the various block propagation algorithms. Since in our approach we depend on a dynamic and fluctuating P2P network, we use deeper buffering compared to BASS.

The BiToS system is also based on BitTorrent [35]. The main idea is to divide the missing blocks into two sets, "high priority set" and "remaining piece set", and request with higher probability blocks from the high priority set. While the emphasis is on careful scheduling of the video blocks in BiToS, the use of network coding obviates this problem in our system. Also, we note that BiToS does not identify the issues with topology management (and instead uses the standard BitTorrent algorithms).

3. 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 server. Users are free to arrive at any point in time and want to watch the video from the beginning (seek, and fast-forward functionalities are out of the scope of this paper). In other words, we assume *linear viewing*, but we allow the users to join at arbitrary times. The resources (especially network bandwidth) of the server are limited, and hence users contribute their own resources to the system. Users organize themselves in an unstructured overlay mesh which resembles a random graph.

A client joins the system by contacting a central tracker (whose address is obtained by an independent bootstrap mechanism). This tracker gives the client a subset of nodes already in the system. The client then contacts each of these nodes, and incorporates itself into the overlay mesh. Thus, each node is oblivious of the other nodes in the system except for a small subset, which we designate as its *neighborhood*. Each node can exchange content, as well as control messages, only with its immediate neighbors.

When a node loses a neighbor (for example, a neighbor crashes) or wishes to increase its download rate, it can request additional neighbors. Note that we assume fail-stop behavior from the clients, i.e., they either function correctly or they cease to be a part of the system. In particular, they are not actively malicious; this is out of the scope of this paper.

We assume that the clients themselves are resource-constrained (esp. network bandwidth). Thus, the download and the upload rates of a client are limited by its capacity. We consider scenarios where clients have asymmetric links, i.e., their upload and download capacities are different.

The file is divided into a number of *blocks*. A number of consecutive blocks may be grouped into *segments* to improve efficiency. The system is media codec agnostic, hence, we do not rely on media format knowledge to recover from errors and packet losses; as a result, an important constraint of our system is that all blocks needs to be received without errors by the clients. Clients have enough

storage capacity to keep a copy of all the blocks downloaded up to that point in time. Blocks can only be played if they are received in order and they are only available for other peers to download while the user is watching the video content or shortly after he is done.

In our system, we have considered various arrival patterns, but focussed on flash-crowd events where most users arrive close in time after the video is published, and the file initially resides at a single location with limited upload capacity. We note that typically, in steady-state, it is possible to find a few nodes that have downloaded the entire content and can act as servers; the resulting increase in serving capacity in steady state eases the problem of content scheduling. Hence, we have focussed on flash-crowd arrival patterns because this configuration exercises the system the most.

4. 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. 4.1. The implementation gives us a more detailed insight into the operation of the system; it is described in Sec. 4.2. In Sec. 4.3, we describe the performance metrics we have used for our study.

4.1 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.

4.2 Implementation

We have developed a prototype to validate our results in a realistic setting. The system resembles typical P2P systems [16] 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.

4.3 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.

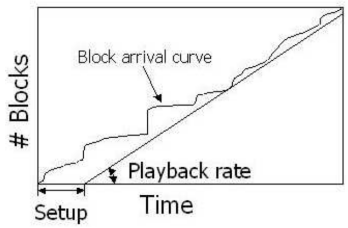


Figure 1: This hypothetical graph shows the calculation of sustainable playback rate, given the setup time. The y-axis shows the number of consecutive blocks, while the x-axis shows the time.

5. 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

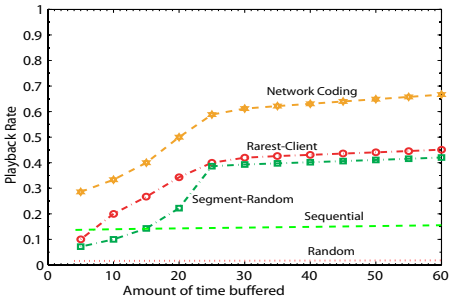


Figure 2: Comparison of block scheduling

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. However, the performance of the segment-random policy is still quite low.

In addition to the naive approaches described above, we experimented with a number of block scheduling policies (some leveraging even global knowledge of the system) to discover good heuristics. A consistent observation was that greedy policies performed quite badly, while algorithms which required clients to download blocks that are not of immediate interest to them, but are from under-represented segments, improved the overall performance of the system.

5.1 Pre-fetching

We now study the effect of *pre-fetching*, i.e. probabilistically fetching a block that is needed later than the immediate segment of interest. The idea with pre-fetching is that nodes download blocks from the future segments with a small probability. Even though these block downloads are not immediately useful, and could be considered as "wasted" downloads from a client's point-of-view, they still hold an overall benefit for the system. The reason is that nodes doing pre-fetching for future segments, act as launching pads for the content that they pre-fetch. In essence, pre-fetching provides easy access to blocks when they are needed by creating additional sources for future blocks and minimizing the overhead of block propagation from remote locations. The end result is a smoother transition across segments.

We have considered a number of policies some of which pre-fetch from all the required segments, some of which only consider a few segments into the future, and policies with different probabilities of pre-fetching. The policy which performed consistently well across various scenarios is the following. Each peer considers the first two segments of blocks that it needs. The peer chooses between the segments using a biased coin, typically it picks the first segment with 90% probability and the second segment with 10% probability. Within each segment, it downloads a particular block following one of the block policies described earlier on.

In Figure 3, we highlight the benefits of pre-fetching. We plot the progress of a client doing a local-rarest policy with and without pre-fetching. We note that the valleys are not as deep with pre-fetching, thus providing smoother segment transitions. In fact, the mean progress per round improves from 203.04 to 226.71 with pre-fetching. Doing pre-fetching, blocks are pre-populated at different parts of the network, and act as additional sources that can speed up block propagation when blocks are needed. The tradeoff is that while pre-fetching hurts the propagation of the current segment at a few nodes, it also improves the height of the valleys for most nodes, and we note that the advantages far outweigh its disadvantages.

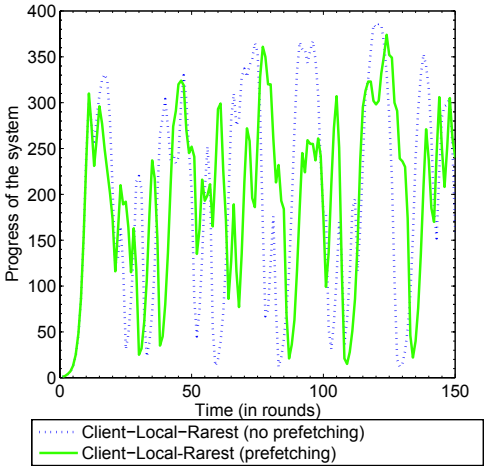
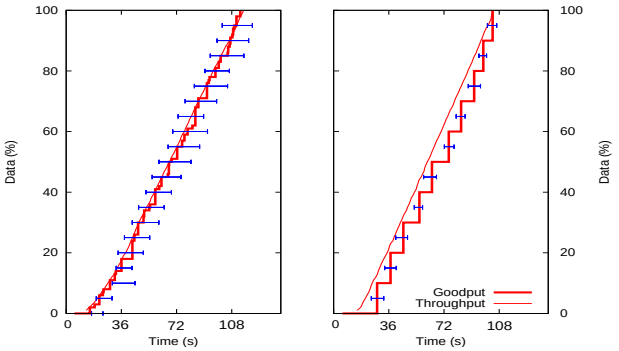


Figure 3: The above graph demonstrates the benefits of prefetching.

6. NETWORK CODING

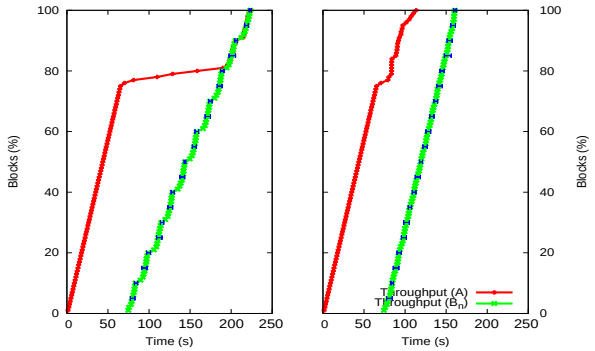
In this section, we study network coding techniques to optimize the goodput of the nodes, and the system's progress. Network coding has been proposed for improving the throughput of a network for bulk data transfer [1, 10, 17]. Network coding makes optimal use of bandwidth resources, and bypasses the block-scheduling problem by allowing all nodes to produce encoded data blocks. A good overview of network coding can be found in [29].

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



(a) Global Rarest (b) Netwok Coding

Figure 4: Average throughput under different policies that do no coding (global rarest) and coding over a segment.



(a) Naïve greedy (b) Worst seeded first

Figure 5: Average throughput in a flash-crowd ($B_{1...20}$ join when A is 75% done) under different segment policies.

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

Fig. 4(a) and 4(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.

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

7. 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 ([3]) 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. Consider a scenario where bandwidth-constrained node that contains blocks from both under-represented and popular segments, uploads blocks from the under-represented segment. This effect 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 6. 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 5(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_ns. 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 5(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.

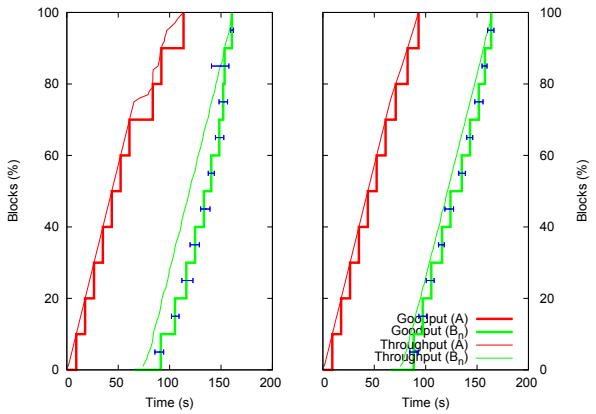
Algorithm 1 SELECTSEGMENT(S,D)

Require: S is source node
Require: D is destination node
1: $A_S \leftarrow$ AVAILABLESEGMENTS(S)
2: $C_D \leftarrow$ COMPLETEDSEGMENTS(D)
3: P $\leftarrow$ SORTSEGMENTSWORSTSEEDEDFIRST($A_S \setminus C_D$)
4: $C_S \leftarrow$ COMPLETEDSEGMENTS(S)
5: $I_D \leftarrow$ EARLIERSTINCOMPLETESEGMENT(D)
6: **for** $i = 0 \ldots$ 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 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.



(a) No topology management (b) With topology management

Figure 6: Average goodput in a flash-crowd ($B_{1\ldots20}$ join when A is 75% done) with and without topology management.

8. TOPOLOGY MANAGEMENT

In this section we look at how topology management can be used to not only improve the throughput of the system, but also the goodput observed by individual clients. We study the system behavior through our implementation as before and use the flash-crowd scenario from Section 7 as our running example.

We will first show how the lack of proper topology management algorithms can significantly impact the goodput of individual clients. For instance, consider Figure 6(a) that plots the average throughput and goodput of an early node, say A, that has downloaded 75% of the file, and a collection of late arrivals, say B_n, when using the worst-seeded-first segment policy (with thresholds) described in Section 7. Observe that A and B_n are interested in downloading different parts of the video and, hence, have competing interests. The steps in the graph reflect when a node completes downloading all the coded blocks from a segment and can decode and watch that segment (goodput). We can see that A has good goodput (the goodput curve regularly intercepts the throughput curve) implying that A always downloads blocks that are immediately useful to A. In contrast, B_n has worse goodput since the server penalizes B_n to download worst-seeded segments (from near the end of the file). Consequently, B_n's goodput is adversely affected leading to increased average buffering times and low playback rates.

The goal of topology management then is to create (overlay) connections that improve the overall goodput of the system, without compromising the throughput benefits achieved through segment scheduling; in effect, this policy implicitly imposes some structure on the mesh network. Our topology management algorithm is described in Algorithm 2. The algorithm essentially tries to retain connections that demonstrate a high goodput. Thus, it encourages peers targeting the same parts of the video to communicate with each other, resulting in a high goodput for the entire length of the download. A node S is allowed to upload to another node D if S has not already saturated its upload bandwidth (line 1). If S's upload capacity is all used, then S will still accept D only if D is currently targeting the *worst-seeded* segment as calculated by S (lines 4–6). If by accepting D, S crosses its configured limit for the number of connections, S drops an existing connection that provides the least goodput to the corresponding neighbour (lines 9–

10). Periodically, nodes that have spare download capacity request a set of random nodes from the tracker and attempt to download from them; of course, subject to that node's acceptance as per the algorithm. These connection attempts cause nodes in the network to re-evaluate their goodput, and opportunistically improve it. This induced connection-churn also encourages neighbor diversity, and prevents the creation of isolated clusters.

The benefits of topology management can be seen in Figure 6(b). The topology management algorithm first restricts B_n from connecting to the server. Instead, B_ns are steered to A and the other B_ns. In this process, the server continues to seed innovative content to A unabated, increasing A's throughput. At the same time, A seeds the beginning of the file to some B_n, which in turn distributes it amongst the other B_ns. Once A finishes downloading, B_ns are allowed to connect to both A and the server. It is clear from Figure 6(b), that the proposed topology management policy both improves the goodput of new nodes (B_n), while retaining the throughput (and goodput) of old nodes (A).

Algorithm 2 SHOULDUPLOADTO(S,D)

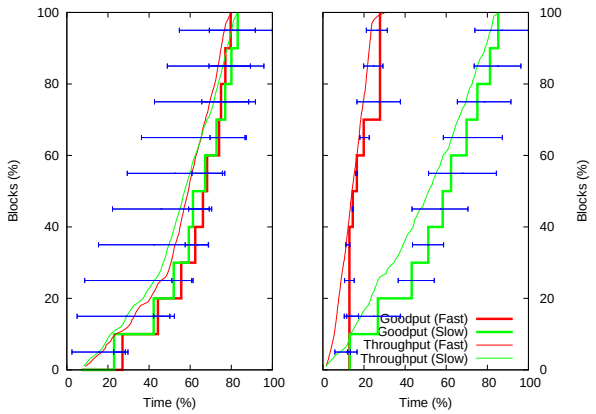
Require: S is the source node
Require: D is the destination node
1: **if** HASSPAREUPLOADCAPACITY(S) **then**
2: **return true**
3: **end if**
4: ID ← EARLIERSTINCOMPLETESEGMENT(D)
5: SD ← SELECTSEGMENT(S,D)
6: **if** ID ≠ SD **then**
7: **return false**
8: **end if**
9: **if** TOOMANYUPLOADCONNECTIONS(S) **then**
10: KICKUPLOADWITHWORSTPEERGOODPUT(S)
11: **end if**
12: **return true**

9. HETEROGENEOUS CAPACITIES

In this section, we revisit our segment scheduling and topology management policies and refine them to handle heterogeneous networks where nodes have asymmetric upload and download capacities (i.e., the capacities are different). As a motivation for our final algorithm, we present a scenario where a capacity-oblivious segment policy falters.

Consider a hypothetical network with a fast server, a slow node A that has downloaded 25% of the file and is connected to the server, and a flash-crowd B_n of mixed capacity (i.e., both fast and slow nodes). The distribution of the capacities of the nodes in B_n is loosely based on [6]. In this scenario, consistent with the foregoing discussion, the server uploads the worst-seeded segments (i.e. those not in A). Since the server has spare capacity, some B_n are allowed to connect to the server (Alg. 2, line 2). However, they are forced to download segments that are of no immediate use to them (e.g. segments near the middle or end of the file), and they only receive blocks for their immediate segment of interest through node A, which happens to be slow.

Figure 7(a) shows that heterogeneity-oblivious algorithms (as presented above) affect not only the goodput, but also the throughput of fast nodes. In fact, we can see that both the throughput and the goodput of the fast nodes is reduced to the throughput and the goodput of the slow nodes (in particular, that of node A). This is because fast nodes are initially forced to depend on a slow node (A), which is the only node giving them 'good' blocks.



(a) Heterogeneity Oblivious (b) Heterogeneity Aware

Figure 7: Average goodput of slow and fast nodes in a flash-crowd with and without heterogeneity awareness.

A small change in the computation of the worst-seeded segment addresses this problem. The change involves assigning a weight to the contribution of each node towards the aggregate seeding of each segment in proportion to that node's upload bandwidth. This is a small but significant change to traditional block rarest-first policies, since it takes into account not only the number of copies of a given segment, but also how efficiently they can be uploaded into the network. The result of this change, as shown in Figure 7(b), is very high throughputs and goodputs for both fast and slow nodes, and very low start-up latencies (e.g. a small percentage of the whole file). In fact, from Figure 7(b), we can see that fast nodes can target playback rates that are very close to their bandwidth limits, and are not throttled by the slow nodes (which also observe good performance).

One final important detail of our implementation is the way the threshold policy in the worst-seeded computation is adapted to include heterogeneous capacities. The problem is that, if a segment exists at a large number of nodes (which happen to be slow), the threshold policy described earlier will consider such segments to be well represented. But in reality, the segment stored at these nodes would not propagate well if the upload capacities of such nodes is very small. To avoid this, our policy needs to dynamically adjust the threshold policy to include the current seeding capacity of each segment. For this purpose, in our implementation, we scale the worst-seeded computation by the peer's upload capacity (represented as a fraction between 0 and 1 of the *fastest active peer's* upload capacity). In doing so, nodes are forced to keep downloading a segment until that segment has an overall upload capacity equal to that of the other segments in the system. The new scheme thus prevents the formation of rare segments.

10. SUMMARY

In this paper we have examined the problem of designing a Video-on-Demand (VoD) 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 in large scale bulk file distribution. Unfortunately, however, those systems give very bad performance when used for VoD. In this paper, we have proposed network coding to improve the throughput of block distribution across the system, segment scheduling to further improve the throughput while delivering con-

tent "pseudo sequentially", and topology management to group together peers interested in the same part of the video; all these techniques work well together to provide efficient VoD with small setup delays. Our simulations and experiments suggest that this combination of network coding, segment scheduling, and topology management provides a significant improvement in performance compared to other algorithms (even compared to algorithms that use global knowledge). Though further work is required towards a better understanding of the efficacy of our algorithms in more realistic scenarios, we believe that the guidelines proposed in this paper can be used to build high-performance P2P VoD systems.

Our system was designed to guarantee that the video starts playing shortly after the beginning of the download, and progresses without interruptions until the end of the movie. While we have made an implicit assumption that users watch the entire video linearly, we believe that the same principles used in our system could be extended to support non-linear viewing, i.e., where users would be able to start watching from arbitrary points in the video and perform fast forward and rewind operations (e.g. VCR-type functionality). We note that our design already supports seek operations that reposition the video stream to a part of the video that has already been downloaded (e.g. rewind, pause); this is due to the fact that we locally store the entire video while the user is connected to the system. However, if the user desires to watch a part of the video that is not available locally, then the user will suffer a (moderate) waiting time as the system searches for peers to download the desired content from. Designing architectures that minimize this seek time is an interesting open research problem.

11. REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. on Information Theory*, 46:1204–1216, 2000.

[2] K. C. Almeroth and M. H. Ammar. On the use of multicast delivery to provide a scalable and interactive Video-on-Demand service. *Journal of Selected Areas in Communications*, 14(6):1110–1122, 1996.

[3] S. Annapureddy, C. Gkantsidis, and P. Rodriguez. Providing video-on-demand using peer-to-peer networks. In *Internet Protocol TeleVision (IPTV) Workshop, WWW '06*, Edinburgh, Scotland, May 2006.

[4] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez. Exploring vod in p2p swarming systems. In *IEEE Infocom*, 2007.

[5] J. G. Apostolopoulos, W.-T. Tan, and S. J. Wee. Video streaming: Concepts, Algorithms, and Systems. `http://www.hpl.hp.com/techreports/2002/HPL-2002-260.pdf`, Sep 2002.

[6] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent network's performance mechanisms. In *Proceedings of IEEE INFOCOM 2006*, Barcelona, Spain, April 2006.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Oct. 2003.

[8] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *ACM SOSP'03*, Lake Bolton, New York, USA, Oct 2003.

[9] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. A distributed real time MPEG video audio player. In *NOSSDAV*, 1995.

[10] P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *Allerton Conference on Communication, Control, and Computing*, Oct 2003.

[11] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Measurement and Modeling of Computer Systems*, pages 1–12, 2000.

[12] B. Cohen. BitTorrent. `http://www.bittorrent.com`.

[13] C. Dana, D. Li, D. Harrison, and C.-N. Chuah. BASS: BitTorrent assisted streaming system for video-on-demand. In *International Workshop on Multimedia Signal Processing (MMSP)*. IEEE Press, 2005.

[14] End system multicast. `http://esm.cs.cmu.edu/`, 2005.

[15] Feidian. `http://tv.net9.org/`.

[16] C. Gkantsidis, J. Miller, and P. Rodriguez. Anatomy of a p2p content distribution system with network coding. In *IPTPS*, 2006.

[17] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *IEEE Infocom*, 2005.

[18] Gnutella. `http://gnutella.wego.com/`.

[19] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. Promise: Peer-to-peer media streaming using collectcast. In *Multimedia*. ACM Press, 2003.

[20] T. Ho, M. Mdard, M. Effros, and D. Karger. On randomized network coding. In *41st Allerton Annual Conference on Communication, Control and Computing*, Oct 2003.

[21] A. Hu. Video-on-demand broadcasting protocols: A comprehensive study. In *IEEE Infocom*, pages 508–571. IEEE Press, Apr 2001.

[22] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *ACM SIGCOMM*, pages 89–100. ACM Press, 1997.

[23] Y. hua Chu, A. Ganjam, T. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an internet broadcast system based on overlay multicast. In *USENIX Annual Technical Conference*. USENIX, 2004.

[24] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *4th ACM Operating Systems Design and Implementation (OSDI'00)*, pages 197–212, 2000.

[25] V. Kompella, J. Pasquale, and G. Polyzos. Multicasting for multimedia applications. In *IEEE Infocom'92*, volume 3, pages 2078–2085. IEEE Press, May 1992.

[26] D. Kostic', A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of the 19th ACM symposium on Operating systems principles (SOSP)*, pages 282–297, Bolton Landing, NY, USA, October 2003.

[27] X. Li, S. Pauly, and M. Ammar. Video multicast over the internet. *IEEE Network*, 1999.

[28] N. Magharei, A. Rasti, D. Stutzbach, and R. Rejaie. Peer-to-peer receiver-driven mesh-based streaming. In *ACM SigComm (poster session)*. ACM Press, 2005.

[29] M. Medard, R. Koetter, and P. A. Chou. Network coding: A new network design paradigm. In *IEEE International Symposium on Information Theory*, Adelaide, Sep 2005.

[30] A. Parker. P2P in 2005. `http://www.cachelogic.com`, 2005.

[31] J. C. Pasquale, G. C. Polyzos, and G. Xylomenos. The multimedia multicasting problem. *ACM Multimedia Systems*, 6:43–59, 1998.

[32] PPLive. `http://www.pplive.com/`.

[33] D. A. Tran, K. A. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE Infocom*. IEEE Press, 2003.

[34] S. Viswanathan and T. Imiehnski. Pyramid broadcasting for video on demand service. In *IEEE Multimedia Computing and Networking Conference*. IEEE Press, 1995.

[35] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for supporting streaming applications. In *IEEE Global Internet*, 2006.

[36] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M. Peha. Streaming video over the internet: Approaches and directions. *IEEE Tran. on circuits and systems for video technology*, 11(3):282–300, Mar 2001.

[37] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*. IEEE Press, 2002.

[38] 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*. IEEE Press, 2005.

APPENDIX

In this section, we give a short description of the benefits and the mechanics of network coding. We illustrate the benefits of network coding with a simple example (Figure 8). Assume that node A has already received blocks 1 and 2. Without a scheduler having global knowledge, node B will download block 1 or 2 with equal probability. Simultaneously, let's say node C independently downloads block 1. If node B were to download block 1, the link between B and C would be rendered useless.

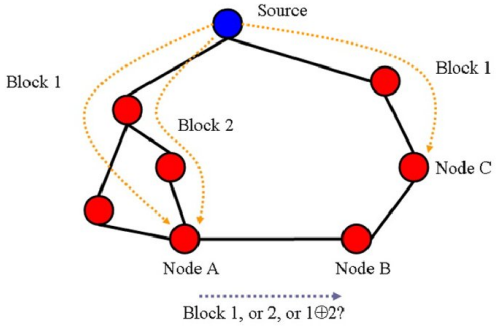


Figure 8: This example shows the benefits of network coding when nodes only have local knowledge.

With network coding, however, node A routinely sends a linear combination of the blocks it has (shown in the figure as $1 \oplus 2$) to node B, which can then be used with node C. Note that without a knowledge of the block transfers in other parts of the network, it's not easy for node B to determine the right block to download. But with network coding, this task becomes trivial.

We now detail how network coding can be used in a P2P system. In Figure 9, the file exists initially only at the server. When node A contacts the server, the server combines all the blocks of the file to create an encoded block $E1$. The server picks random coefficients $c_1, c_2, \cdots, c_n$, and generates $E1 = \sum_{i=1}^{n} c_i.b_i$, where b_i represents a block. The server then sends node A, $E1$ and the *coefficient vector* $\overrightarrow{c} = (c_i)$. Note that all the coefficients are chosen from and the operations done in a finite field [1] When node A sends a block to node B, A similarly combines the already encoded blocks it has (namely $E1$ and $E2$) and sends node B, the encoded block $E3 = c_1^{\text{“}}.E1 + c_2^{\text{“}}.E2$, and the new coefficient vector $c_1^{\text{“}}.\overrightarrow{c} + c_2^{\text{“}}.\overrightarrow{c'}$. With network coding, any block that a node receives is useful with very high probability. The downside of network coding is that a node often has to wait until it downloads the whole file before it can start decoding the blocks.

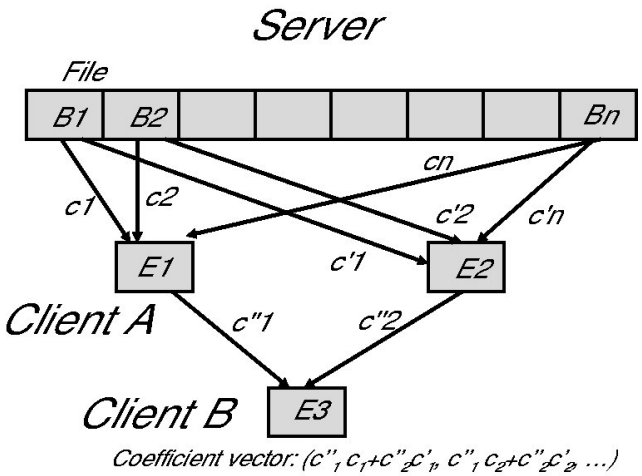


Figure 9: A brief description of network coding in RedCarpet .

[1] If the finite field is small in size, there could be "collisions" where two nodes pick the same set of coefficients, thereby degrading the performance [20]. Typically, field sizes of 2^{16} provide enough diversity.