Providing Video-on-Demand using Peer-to-Peer Networks

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

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

Pablo Rodriguez
Microsoft Research
Cambridge, UK
pablo@microsoft.com

ABSTRACT

Digital media companies have recently started embracing peer-assisted distribution networks as an alternative to traditional client-server architectures [5]. Such Peer-to-Peer (P2P) architectures ensure a fast and scalable delivery of media content. However, their drawback is that users need to often wait for the full video to be downloaded before they can start watching it. While a lot of effort has gone into optimizing the distribution of large video files using *P2P swarming* techniques, little research has been done on how to ensure a small start-up time and a sustainable playback rate to enable a play-as-you-download experience.

In this work, we address the challenges underlying the problem of near Video-on-Demand (nVoD) using P2P swarming systems, and provide evidence that high-quality nVoD is feasible. In particular, we investigate the scheduling problem of efficiently disseminating the blocks of a video file in a P2P mesh-based system, and show that pre-fetching and network coding techniques can provide significant benefits. Our results show that, for videos which are 120 minutes in duration, 10 minutes ($\approx 8\%$ of the video's length) of buffering at start-up can enable playback rates that are close (up to $80 - 90\%$) to the access link capacity, even under dynamic changes of the user population.

1. INTRODUCTION

Peer-to-peer (P2P) systemshave been immensely successful in distributing content to a large number of users. Currently, most of the traffic in P2P networks is for video files [19]. Unfortunately, however, the users need to download a large fraction of the file before they can watch the video, which entails that the users wait for a long time. Recently, systems such as Coolstreaming [24] and its derivates ([11, 20]) have shown that it is feasible to distribute live media content to a large number of users using P2P networks. However, it has been an open question whether P2P technology could be used to provide a nVoD service to the users. A nVoD capability would enable users to start watching a video after waiting for a small start-up time, while downloading the video in parallel. A nVoD service also entails a number of challenges. The users should be able to watch the video at an arbitrary time, unlike in live-streaming where they need to synchronize their viewing times. The users should also be able to perform control operations like rewind on the video. In this paper, we demonstrate via simulations, the feasibility of using P2P networks for providing a nVoD service, and extract design principles for building efficient systems.

Video distribution over the Internet has been a prolific area of research [4, 7, 13, 23]. The particular problem of designing a nVoD service has also received extensive attention in the past [2, 10, 13, 21, 22]. An important requirement of a nVoD 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 [2, 14, 22]. However, these systems require a multicast-enabled infrastructure, which unfortunately has never materialized [3].

Peer-to-peer networks, on the other hand, are forged from end-systems, and do not require infratructure support. Such systems have become very popular for bulk file transfers in today's Internet. There are two fundamental P2P approaches – structured (tree-based) and unstructured (mesh-based). In the structured approach, trees (or forests of trees) are usually constructed for dissemination of data [6, 9, 15]. These approaches require a higher control overhead, are sensitive to massive departures of nodes, and are in general, very complex to maintain, when compared with unstructured approaches [16]. This is reflected in the fact that most of the successful P2P systems today are unstructured. Hence, we chose to study the feasibility of using an unstructured mesh-based approach for nVoD.

However, current P2P systems have inherent limitations that do not allow them to support a play-as-you-download experience. In these systems, the peers have partial content which they exchange with each other in order to download the complete file. The system achieves high throughput when the peers can exchange content with each other, and this happens only when they have non-overlapping pieces of the file. Hence, the peers download pieces of the file in random order to minimize the overlap. However, in order to support a play-as-you-download experience, the peers require blocks in sequential order from the beginning. However, if all the peers were to download content in sequential order, they would have overlapping pieces of the file, and the utilization (throughput) of the system would be low. The goal then is to design a P2P system which meets the nVoD requirements, while maintaining a high utilization of the system resources.

An important question towards enabling such nVoD services using P2P networks is the design of algorithms for *scheduling* the propagation of content. As pointed above, simple algorithms like sequential or random distribution of data do not achieve this goal. The algorithms should take into account the dynamics of the peers (peers may join and leave at arbitrary times), the fact that different peers might be watching different parts of the movie, and the heterogeneity of the network capacities of the peers. A real system also needs to deal with non-collaborative and malicious peers; we do not, however, address this last issue here.

In this paper, we will present algorithms for content propagation in unstructured P2P networks, which provide the users with a high-quality nVoD service while ensuring a high utilization of the system resources. We have evolved these algorithms by doing extensive simulations (described later) of mesh-based P2P networks under different user arrival/departure patterns, heterogeneous user capacities etc. We study pre-fetching where a user gets pieces of

IPTV Workshop, International World Wide Web Conference, May 23, 2006, Edinburgh, Scotland, United Kingdom.

the file from the near future, which creates adequate randomness in the system while ensuring a sequential delivery. Another technique that significantly improves the efficiency of the system is network coding, i.e., each user in the network generates a linear combination of blocks that he already has when it needs to send a block to a peer. We have adapted network coding to our system to yield good playback rates and high network utilization. Thus, our contributions are two-fold – to investigate the feasibility of P2P networks for providing nVoD service, and to design algorithms that deliver close-to-optimal performance.

The rest of the paper is organized as follows. In Section 2 we present a short overview of unstructured peer-to-peer networks which form the basis of our study. In Section 3 we present the metrics for evaluating our system and the salient features of our simulations. In Section 4 we present some naive approaches to address the challenges in providing Near-VoD services. In Sections 5 and 6, we introduce two techniques, namely pre-fetching and network coding that significantly improve the metrics of interest. We conclude in Section 7.

2. MODEL

In this section, we present a model of the components that make up the system. 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. The resources (especially network bandwidth) of the server are limited, and hence users contribute their own resources to the system. To this end, the users form an 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 random subset of nodes already in the system. The client then contacts each of these nodes, and incorporates itself into the overlay mesh. Note that we could use other techniques to find random subsets of nodes ([18]).

Thus, each node is oblivious of the other nodes in the system except for a small subset, which we designate as its *neighbourhood*. Each node can exchange content, as well as control messages, only with its immediate neighbours. Thus, each node only maintains *local* state information. This local knowledge, while crucial for the scalability of the system, makes it challenging to obtain system-wide optimal scheduling policies.

When a node loses a neighbours (for example, a neighbour crashes) or wishes to increase its download rate, it can request additional neighbours. Note that we assume fail-stop behaviour 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.

Finally, like the server, 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 do consider scenarios where clients have asymmetric links, i.e., their upload and download capacities are different.

3. DESIGN

In this section, we will start with naive approaches, identify the bottlenecks, and refine the design of the algorithm. We take a simulator-based approach to study some of the key parameters. We have used a simulator since it gives us the flexibility to explore the whole design space and easily scale to many nodes, which would be difficult otherwise. We believe that our simulator captures the important properties of a real system. We will present our simulator in Section 3.1, discuss a few naive approaches, and some refinements in Section 4. Finally, we will present two techniques which significantly improve the system performance in a variety of scenarios – Prefetching (Section 5.2.2) and Network Coding (Section 6).

3.1 Simulator

In this section, we will present the salient features of our simulator. The simulator takes as input the size of the video file, the number of nodes, 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 bandwidth is given in multiples of a standard unit. A standard client has a bandwidth of 1 unit. The amount of data that a standard client consumes in a round is considered a *block*. The size of the video is given as the number of blocks.

When a node joins the system, it obtains a few neighbours at random from the nodes already in the system (provided they have not exceeded their quota of neighbours). The simulator supports dynamic arrivals/departures of nodes, and topology reconfiguration. In fact, at the end of each round, if a node detects that its utilization of the download capacity falls below a certain threshold (10% in our experiments), the node tries to discover and connect to new neighbours. Also, if a node has exceeded the number of neighbours, it will drop one of them at random.

At the beginning of each round, each node contacts its neighbours to determine if useful blocks of the file are available. Then, the node determines the blocks to download based on a scheduling policy (which we call the *client policy*), and its download capacity. Similarly, the node could upload blocks of the file to its own neighbours. The simulator gives preference to exchanges of blocks between two nodes rather than free-loading. Note that the number of nodes that could be satisfied by the server is limited by the server's capacity. The block transfers, either between peers or from the server, all occur in the same round, and the system then moves to the next round.

In all our simulations, we consider networks of 500 clients. Accordingly, we reduce the number of neighbours that each node has to a small range $4 - 6$ (rather than having 50 neighbours, say). We also used a video size of 250 blocks. We have also experimented with higher number of blocks, but did not observe significant changes.

We note that while our simulator is not intended to model realistic P2P networks in all its details (for example, we do not consider 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.

3.2 Methodology

We will now describe our methodology in studying the nVoD problem. We remind the reader that our twin goals are to ensure a low start-up time and a sustainable playback rate. For each simulation, we plot the number of consecutive blocks from the beginning that the node has (this is the amount of video that can be played without interruption) on the y-axis, and the time (in rounds) on the x-axis, for every node as shown in Figure 3.2. For a given setup time, we calculate the sustainable playback rate as the maximum slope of a line which does not exceed the y-coordinate (the number of consecutive blocks) at any time. We make a list of the playback rates for various setup times for each node. For a given setup time, we then plot the 5^{th} percentile, the median, and the 95^{th} percentile of the playback rates of all the participating nodes. We consider a setup time of 20 to 30 rounds (10-15 min) reasonable in our experiments.

Another parameter of interest is the progress per round of the system, also called the *throughput*, which is the total number of block transfers amongst all the nodes per round. The progress of the system indicates the network utilization of the system. Hence, we want the progress of the system to be high. We also want to ensure that the block transfers are useful to the nodes for sustaining

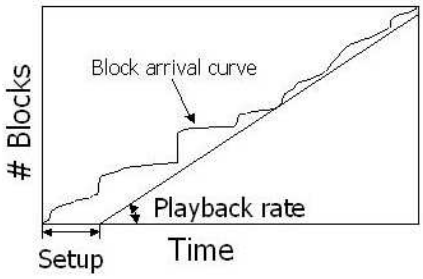

Figure 1: This hypothetical graph depicts the calculation of sustainable playback rate, given the setup time. The y-axis shows the number of consecutive blocks at the node while the x-axis shows the time (in rounds).

a good playback rate.

In studying the nVoD problem, we restrict ourselves to finding optimal scheduling policies for a basic end-system architecture. In particular, we study the scheduling policies at the server and the client that work well together to yield good performance. Whenever a client contacts the server, the server determines, according to an algorithm called the *server policy*, which block(s) to send to the client. Similarly, the client determines which block(s) to download from which neighbour according to an algorithm called the *client policy*. Specifically, we do not investigate alternative techniques for improving the performance of the system, like changing the topology, optimal neighbour selection algorithms etc. These techniques are complementary to our work, and could be integrated into our design.

4. NAIVE APPROACHES

In this section, we will first present some naïve approaches, in order to highlight their shortcomings. In these experiments, we consider a network of 500 nodes that all arrive at the same time to watch a video file (flash crowd scenario). We will study the dynamic arrival and departure of nodes later.

Existing mesh-based file-distribution mechanisms are not geared for streaming videos to a large number of users. These systems essentially distribute the blocks of a file in *random* order. This strategy diversifies the cooperative cache (formed by the nodes together) quickly, and ensures that the progress of the system is high. The problem with this approach is that nodes do not get the blocks of the file in order, and hence the playback rate is very poor. The efficiency of the system is about 1% of the access link capacity (see Figure 2). The mean progress per round of the system is 332.44 blocks. (If all these blocks were useful for playback, the mean rate should have been $\frac{332.44}{500} = 66\%$.) This indicates that while the throughput of the system is high, the playback rate is unacceptably low due to the out-of-order delivery of the video blocks.

Another naïve approach is for each client to greedily fetch the very next block that it needs, which we call a *sequential policy*. In the sequential policy, clients only download blocks in order from the other nodes (or the server). With this policy, nodes get the blocks they need for sustaining playback, resulting in slightly better playback rates (to about 13.2%). However, the mean progress per round is reduced to 65.97 (15%) , as all the nodes fetch the same blocks, thereby diminishing the benefits of cooperation.

An intermediate approach is to retain the high throughput of the random policy with the better playback rate of the sequential policy. To this end, we divide the file into *segments* which are extents of the file comprising consecutive blocks. For instance, a file of 250 blocks can be divided into 25 segments, each consisting of 10 consecutive blocks. In the *segment-random* policy, a client requests a random block from the segment it's interested in. We see that

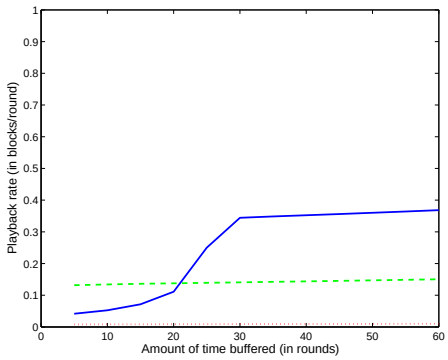

Figure 2: Comparison of random, sequential and segment-random policies

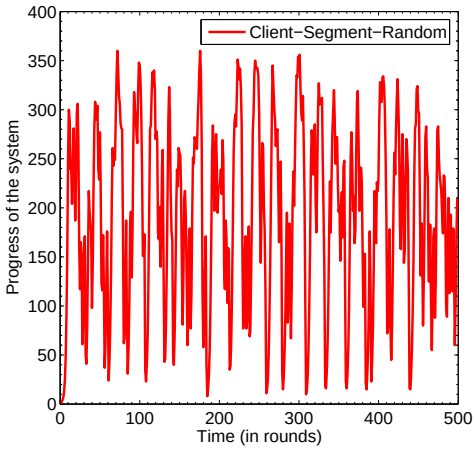

Figure 3: This progress graph depicts the valleys and mountains.

this policy achieves high playback rates (35%, see Figure 2) and a reasonable mean progress per round (170.21). However, note that although improved, the performance of the segment-random policy is still quite low.

4.1 Analysis of poor performance of naïve policies

In this section, we investigate why these simple policies perform poorly. We explain this behaviour by considering the progress of the system over time, measured as the total number of blocks exchanged per round (Figure 3). In these experiments there are 500 nodes, and one of them has all the blocks initially.

From the above graph, we can see that the progress happens in bursts. Progress follows a regular pattern of mountains and valleys, where each "mountain" corresponds to the dissemination of one segment of the file. Valleys are the source of inefficiencies, and they arise for two independent reasons: a) the degradation during the dissemination of the last blocks of a segment (e.g. last block problems in each segment), and b) the slow rise in the initial dissemination of the following segment. The first problem can be solved by preventing the occurrence of rare blocks – either by better scheduling or by using coding techniques (Section 6). The second problem relates to the fact that the first blocks in a segment take several rounds to propagate to all the nodes. To avoid this delay, we look at pre-fetching (Section 5.2.2) policies that pre-populate a few nodes with blocks from the future so that blocks of upcoming segments are within reach for any node.

5. BLOCK SCHEDULING FOR EFFICIENT VOD

In this section, we study the impact of scheduling policies on the playback rate. We first investigate the impact of different policies when the content only resides at a few nodes (e.g. during the initial phases of a flash crowd), and then we study the case where the content is well represented in the network. We did our simulations with flash crowd of 500 nodes.

5.1 The Importance of Server Scheduling during Flash Crowds

During the initial delivery of the first copy of a video file, the server plays a critical role in ensuring high playback rates. The server is the only node to start with that holds a copy of all the blocks, and the system performance is determined by the rate at which new information is injected into the system by the server. We call this initial period, the *ramp-up phase*. We have observed that during this phase, client exchange decisions have little impact in comparison with the download decisions being made when talking to the server.

We will now study the impact of different policies at the server in more detail. To this end, we will consider a) greedy policies where clients tell the server which blocks they require, and b) policies where the server can overrule the client's choice of content and serve them with blocks that are not of immediate interest to the client, but could be useful for the system as a whole. The latter is considered only for comparison purposes since it requires that the server keeps and collects estate information about all peers, moving further away from a distributed architecture. During the next discussion, we have fixed the client policy to *local-rarest*, where the client downloads the rarest block within its target segment. Local rarest is defined as the least represented block in a segment within the peer neighbourhood. We have used a segment size of 10 blocks.

We consider the following server policies: a) local-rarest, b) global-rarest, c) client-neighbourhood-rarest, and d) sequential. In the first two policies, the server gives those blocks that are of immediate interest to the client which contacts it. In the local-rarest policy, clients request a block from the server that is the neighborhood rarest in its current target segment. In the global-rarest policy, clients reaching the server are given a block within their target segment which is the system-wide rarest (note that this requires global knowledge and is considered only for comparison purposes.) In the client-neighbourhood-rarest policy, the client requests the block that is rarest amongst all blocks in its neighborhood (regardless of which segment it belongs to). Note that this policy may serve a node with a block that falls in a future segment, and thus, is not of immediate interest to the client. Similarly, in the sequential policy, the server considers all the blocks of the file and gives the earliest-possible system-wide rarest block to the node (This again requires global knowledge and is considered only for comparison.)

The results for the different server policies are shown in Figure 5. To understand the performance of the different policies, we also analyze the usage of the server bandwidth resources. In Figure 4, we present, for each policy, a histogram showing the number of times each block is injected by the server into the system. We note that policies where the client is given blocks within its target segment only perform very poorly. On the other hand, policies where the client's immediate interests are not always satisfied, and the server gives blocks of wider interest for the whole system, perform quite well. This highlights the fact that even if some greedy policies can lead to very poor performance. Instead, clients should consider downloading blocks from other non-immediate but highly under-represented segments. In doing so, the performance of the overall system is significantly improved. For instance, the sequential policy performs best because the server gives most of the blocks only once, and hence does not waste the server bandwidth resources in sending well represented blocks. Similarly, with the client-neighbourhood

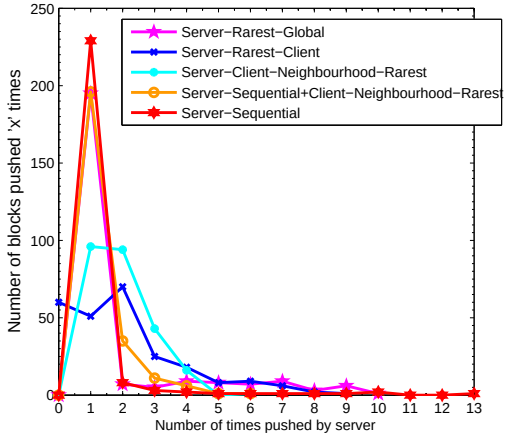

Figure 4: The above graph shows the number of blocks which are pushed with a given frequency (given on the x-axis). Note that the sequential policy performs the best.

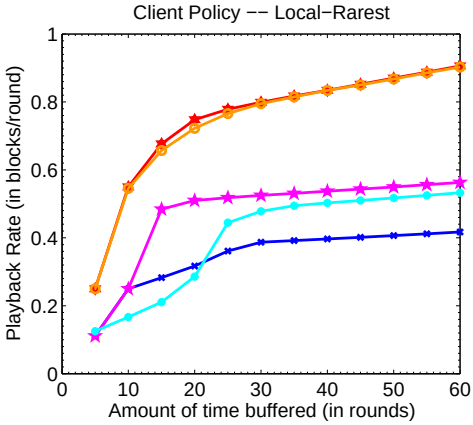

Figure 5: Comparison of different server policies after setting the client policy to local-rarest. The server policy *Sequential + Client-Neighbourhood-Rarest* performs the best. (Figure legends as in Fig. 4)

policy, each client fetches blocks which are useful to the entire neighbourhood and not just for itself. This reduces the frequency of any given block being pushed by the server.

We need to approximate the sequential policy, (which requires global knowledge), using state scalable in the number of nodes. One approach is for nodes to periodically report their progress to a central tracker. Since nodes play blocks in order, and we assume that they hold all the previously played blocks, this gives the server a very good estimate of the blocks needed the most in the system.

An alternative approach to ease the block scheduling problem is to use coding techniques that ensure that the probability of serving a duplicate block within a given segment is negligible. We will consider such coding techniques in Section 6.

In summary, server policies are crucial during the *ramp-up phase*, and systems that are designed to prioritize the client's needs perform badly. Instead, clients should accept content that may not be of immediate use to them, but that is widely needed. Hence, it is important to devise simple techniques that provide the server with a reasonable system-wide view of the content needed the most in the system.

5.2 Client Policies after Flash-crowd Effect

In this section, we try to understand the impact of client policies in helping content propagate across the network once the content is placed in the system. To this extend, we study different client policies after the server injects at least one copy of each block into

the system and departs. This ensures at least one copy of the file is randomly and evenly spread amongst all the nodes present from the beginning. We remove the server from the system and let the clients interact among each other.

From Figure 3, we show the progress of the system and show that it follows a series of valleys and mountains that degrade the performance of the system. We will now discuss two client techniques that alleviate this condition: a) Local-Rarest per segment which improves the peaks in the progress and reduce the period of degradation (during the dissemination of the last blocks of a segment), and b) Pre-fetching across segments which prevents the long periods of delay, waiting for the initial blocks of a segment to propagate.

5.2.1 *Improved Scheduling: Local Rarest*

In this section, we study the segment-random and segment-local-rarest policies. In the segment-random policy, we consider the first segment of blocks that the node requires, and fetch one of them at random. In the segment-local-rarest policy, we again consider the first segment the node requires and fetch the block in that segment that is rarest in the node's neighbourhood.

In Figure 6, we see that a local-rarest policy increases both the height of the peaks and reduces the duration of the valleys. We note that the mean progress per round improves from 191.21 with segment-random to 203.04 with local-rarest. The local-rarest policy improves performance by ensuring that all blocks are well-represented in the network (as opposed to favouring some blocks over the others). While such policies are known to perform well in current file swarming systems, their performance is still quite poor in VoD systems. This is due to the many idle phases produced when some blocks take very long to propagate from the nodes holding the blocks to the areas where they are needed. To counteract this problem, we will next study the impact of pre-fetching policies.

5.2.2 *Pre-fetching*

We now study the effect of pre-fetching, i.e. fetching a block that is needed later than the segment of interest, on improving the slow initial rise. 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 of immediate interest and could be considered as "wasted" downloads from a client's point-of-view, they still hold an overall benefit for the system. In fact, as we will see next, nodes doing pre-fetching, 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 the blocks, thus minimizing the overhead of block propagation from remote locations.

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 *probabilistic-first-range-random-second-range-rarest*. In this policy, we consider the first two segments of blocks that the client needs. We choose between the segments using a biased coin, typically we pick the first segment with 90% probability and the second segment with 10% probability. Within each segment, we pick the rarest block in the neighbourhood.

In the probabilistic-first-range-random-second-range-random policy, we pick the first or the second range with a certain probability (typically $90-10$ as above). In each of these ranges, we pick a block at random. In Figure 7, we plot the progress of a client local-rarest policy that does not do pre-fetching with a policy that does pre-fetching. We note that the mean progress per round improves from 203.04 with local-rarest 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, it also improves the height of the valleys; however, the advantages of pre-fetching far outweigh its disadvantages.

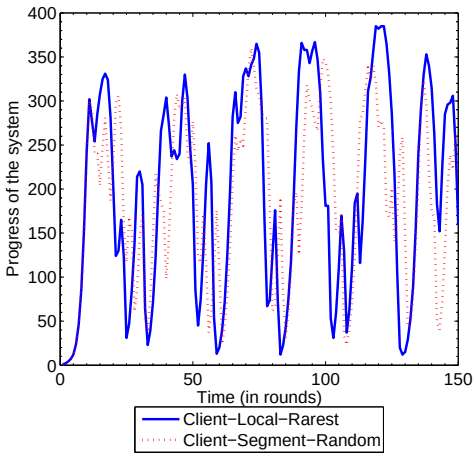

Figure 6: We see that the local-rarest policy both improves the height of the peaks and reduces the duration of degradation, when compared with a segment-random policy.

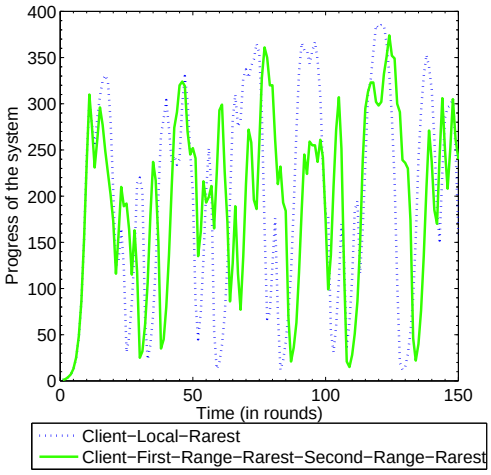

Figure 7: The above graph demonstrates the benefits of prefetching vis-a-vis the local-rarest policy. We note that the valleys are not as deep with pre-fetching.

6. NETWORK CODING

We now consider network coding for effective block scheduling to increase the system's progress and the playback rates of the nodes. We provide a brief description of network coding and analyze its pros and cons. We then present preliminary results.

Network coding has been proposed by [1,8,12] et. al. for improving the throughput of a network for bulk data transfer. The main idea is to allow all the nodes in the system (not just the server as with erasure coding) to encode data blocks. Network coding makes optimal use of bandwidth resources, and the scheduling problem becomes trivial. A good overview of network coding can be found in [17].

Next, we illustrate the benefit of network coding with a simple example (Figure 8). Assume that node A has already received blocks 1 and 2. Without 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. But

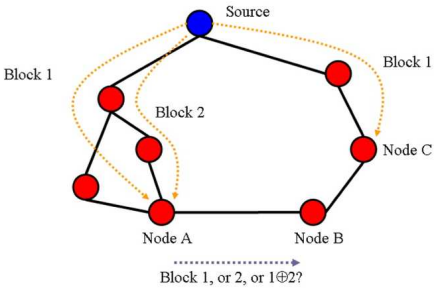

Figure 8: This example shows the benefits of network coding when nodes only have local knowledge.

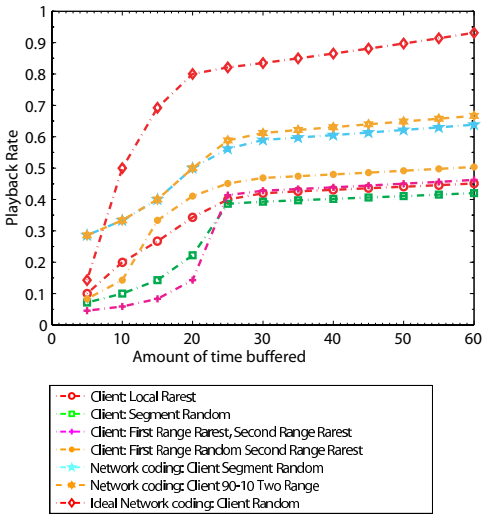

Figure 9: Playback rate vs. setup time graph, when one node holds all the 500 blocks initially.

with network coding, 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 pick the right block to download. But with network coding, this task becomes trivial.

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. This is unacceptable in the context of VoD systems where a node wants to play the blocks as soon as it starts downloading. To this end, we restrict network coding to a given segment only. Thus, a node only needs to wait till it downloads 10 blocks (e.g. the size of a 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. However, coding prevents the occurrence of rare blocks, allows the same encoded block to be used by multiple nodes requiring distinct blocks, and blocks requested from a node are unique (with high probability) ensuring that bandwidth is not wasted uploading the same block multiple times.

We have done a preliminary evaluation of network coding. We have found that a hybrid strategy that retains the benefits of network coding and pre-fetching, *Netcoding-Client-90-10-Two-Ranges*, performs remarkably well in all these scenarios. With this polcy, a client fetches a block from the segment of interest with 90% probability and the second segment with 10% probability. Once a segment is selected, a block is chosen uniformly at random from that segment. We see (from Figure 9) that the playback rate using netcoding is 62%, while the best rate without using netcoding is 47% (setup time 30). Also, the progress of the system is 293.52 as compared to 209.57 without netcoding. We also found that netcoding performs very well in various scenarios like dynamic arrivals, departures, and heterogeneous network capacities.

7. SUMMARY

In this paper we examine the problem of designing a near Video-on-Demand service using mesh-based peer-to-peer networks. We argue that scheduling the propagation of blocks is a very important factor in determining the playback rates at the nodes. We show that naive strategies (like random and sequential policies) have very bad performance. We identify the bottlenecks and propose two new techniques, pre-fetching and network coding, that overcome these problems, making efficient nVoD possible with small setup times. We are currently building RedCarpet, a real nVoD system that incorporates the design principles discussed in this paper.

8. 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] 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*, 14(6):1110–1122, 1996.

[3] Mostafa Ammar. Why johnny can't multicast: Lessons about the evolution of the internet. NOSSDAV Keynote Speech, June 2003.

[4] 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.

[5] Bbc imp. `http://www.bbc.co.uk/imp/`, 2006.

[6] 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)*, October 2003.

[7] Shanwei Cen, Calton Pu, Richard Staehli, Crispin Cowan, and Jonathan Walpole. A distributed real time MPEG video audio player. In *NOSSDAV*, 1995.

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

[9] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Measurement and Modeling of Computer Systems*, pages 1–12, 2000.

[10] Martin Reisslein Despina Saparilla, Keith Ross. Periodic broadcasting with vbr-encoded video. In *IEEE Infocom*. IEEE Press, 1999.

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

[12] Christos Gkantsidis and Pablo Rodriguez. Network coding for large scale content distribution. In *IEEE Infocom*, 2005.

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

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

[15] 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)*, pages 197–212, 2000.

[16] 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)*, pages 282–297, Bolton Landing, NY, USA, October 2003.

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

[18] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proc. FOCS*, 2001.

[19] Andrew Parker. P2P in 2005. `http://www.cachelogic.com`, 2005.

[20] Pplive. `http://www.pplive.com/`.

[21] 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)*. IEEE Press, 1997.

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

[23] 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*, 11(3):282–300, Mar 2001.

[24] 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.