

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

S. Annapureddy  
New York University

C. Gkantsidis, P. Rodriguez, and L. Massoulie  
Microsoft Research, Cambridge

**Abstract**—Digital media companies have recently started embracing peer-assisted distribution networks as an alternative to traditional client-server architectures. Such P2P architectures ensure a fast and scalable delivery of media content. However, one of the drawbacks of current P2P architectures 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 been put in 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 enabling near Video-on-Demand using P2P swarming systems. We show that high-quality Near-VoD is feasible. In particular, we investigate the challenges involved in efficiently disseminating the blocks of a video file in a distributed mesh-P2P system and show that pre-fetching and coding techniques can provide significant benefits. Our results indicate that a few minutes of start-up buffering can enable playback rates that are close (up to 80 – 90%) to the access link capacity even for unsynchronized user arrivals.

## I. INTRODUCTION

Peer-to-peer (P2P) systems, such as Gnutella [3], KaZaA [4], eDonkey [1], and BitTorrent [15], have been immensely successful in distributing content to large number of users. Currently, most of the traffic in P2P networks is for video files [31]. But unfortunately, the users need to download the complete file before being able to play the video, which entails that the users wait for a long time. Recently, systems such as Coolstreaming [38] and its derivatives ([2], [5]) have shown that it is feasible to distribute live media content to large number of users using the P2P technology. However, it has been an open question whether the P2P technology could be used to provide a near-Video-on-Demand (VoD) service to the users. A Near-VoD capability would enable users to watch a movie while downloading and, moreover, the users will have the flexibility to watch the video at any arbitrary time, i.e., users do not need to synchronize their viewing times as in live media content distribution. Also, the users can watch any part of the video, i.e., they will be able to perform operations like rewind and fast-forward on the

video file. In this paper, we investigate the potential of leveraging P2P networks for a nVoD service and provide guidance, and design principles to efficiently build such systems.

Video distribution over the Internet has been one of the most prolific areas of research [11], [13], [21], [36]. The particular problem of designing a near-VoD service has also received extensive attention in the past [9], [17], [21], [33], [35]. An important requirement of a near-VoD service is to be able to support a large number of users, and, hence, such systems should be *scalable*. The need for scalability becomes clear if we consider that a typical video stream incurs 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 [9], [22], [35]. Indeed, many systems such as Pyramid Broadcasting [35] and SkyScaper [22] can provide scalable Near-VoD service by using elegant techniques for dividing the video into segments and broadcasting each segment in a different multicast channel. However, both these systems require a multicast-enabled infrastructure, which unfortunately has never materialized [10].

Peer-to-peer networks is another architecture that can efficiently support the distribution of content, but, unlike multicasting which requires infrastructure support, p2p networks have been extremely popular for bulk file transfers in today's Internet. However, unstructured peer-to-peer networks have inherent limitations that do not allow them to support a play-as-you-download experience. Recall that in current P2P networks the peers have partial content and that they exchange content with each other in order to download the complete file. The system achieves high throughput when the users can exchange content with each other and this happens if they have non-overlapping pieces of the file. In other words, current P2P systems strive to increase the entropy of the content, and toward this direction the peers download pieces of the file in random order. However, in order to support a play-as-you-download experience, the users would prefer to receive blocks in sequential order from the beginning.

The main disadvantage of downloading content in sequential order is that the peers have very similar pieces of the file, fewer possibilities for exchanging content, and, as a result, the utilization (throughput) of the system is low. The goal then is to design a P2P-based system for distributing media content such that the users can start playing the video soon after the beginning of the download, and, still, achieve a high utilization of the resources of the networks.

One of the fundamental questions toward enabling such nVoD services using P2P networks is the design of algorithms for *scheduling* the propagation of content, such that the throughput of the system remains high and, at the same time, the pieces of the video are delivered almost “in order” for users to enjoy a play-as-you-download experience. As described above, neither sequential nor random distribution of data can achieve this goal. Moreover, the design of content propagation algorithms should take into account the dynamics of the peers (peers may join and leave at arbitrary times), the resulting differences in the playing requirements of each peer (i.e. each peer may be watching a different part of the movie), and the heterogeneity of the peers (typically peers have different access capacities). A real system also needs to deal with non-collaborative and malicious peers; however these issues are beyond the scope of this paper.

One plausible approach for solving the piece propagation problem could be to arrange the nodes in a special topology so that the pieces of the file move in an orderly fashion and at the same time, the resources of the network are used efficiently. The construction of multiple multicast trees, where the nodes interested in later parts of the movie are placed closer to the root of the trees, is a candidate example. This approach has been motivated by various P2P systems that have been proposed for live video distribution, such as End System Multicast [7], SplitStream [6], and others; there have been also proposals for supporting nVoD using specially crafted overlay topologies [33]. The main disadvantage of such approaches is that they require algorithms for building and maintaining the structure of the topology. In comparison, algorithms for unstructured or mesh topologies are almost trivial. Our work can also be seen as an investigation of the feasibility of extending unstructured P2P networks to support a VoD user experience.

In this paper we examine algorithms for content propagation in unstructured P2P networks with the goal of supporting the play-as-you-download functionality. Using extensive simulations, we observe that it is possible to support the targeted service using hybrid algorithms that combine randomness (for efficiency) and, almost, in-order propagation (which is suitable for video streaming). In

particular, the contributions of this paper are as follows:

- a) We show that it is possible to support a scalable play-as-you-download service using unstructured peer-to-peer networks. With a small startup latency, which is less than 10% of the duration of the content, we can support playback rates that are comparable to the access capacities of the users (up to 80-90%); the access capacity is an upper limit of the playback rate.
- b) In order to support a play-as-you-download service using peer-to-peer networks, we propose heuristic algorithms to support efficient media streaming. To this extend, we also evaluate the benefits of prefetching a small number of blocks from the near future to ensure that the system does not get stalled.
- c) We observe that the behavior of the server is very important in maintaining high performance. We propose simple, stateless algorithms for maximizing the benefit from the server.
- d) We show that Network Coding [8], [14], [18], [26], i.e. creating combinations of packets in the middle of the network, can further increase the achievable playback rates. By adapting ideas proposed in [18] we are able to achieve higher content propagation rates, especially in more extreme scenarios, and, hence, higher, playback rates.

Our work shows that it is possible to support a near Video-on-Demand experience by properly extending the currently available peer-to-peer systems and provides guidelines for doing so.

The rest of the paper is organized as follows. In Section II we present a short overview of unstructured peer-to-peer networks which form the basis of our study. In Section III we present the metrics for evaluating our system and the salient features of our simulations. In Section IV we present some naive approaches to address the challenges in providing Near-VoD services. In Sections V and VI, we introduce two techniques, namely pre-fetching and network coding that significantly improve the metrics of interest. In Sections VII, VIII, and IX, we evaluate the effectiveness of our policies under various scenarios like dynamic arrivals, mass departures and in the presence of heterogeneous clients. We provide related work in Section X and conclude in Section XI.

## II. MODEL

We envisage to provide near-Video-on-Demand services to a large number of users. Instead of relying solely on a server for the dissemination of video content, we leverage the resources of the participating clients, making it highly scalable. In this section, we present a detailed model of the structural components involved.

We assume a large number of users interested in some video content (hereafter the users could also be referred to interchangeably as clients, end-systems, nodes or peers). This video content is initially present only on the server. The resources (most importantly, network bandwidth) of the server are limited and hence, users contribute their own resources to the system, in exchange for better playback experience. To this end, the participating users form an overlay mesh which resembles a random graph.

Whenever a client wishes to join the system, it contacts a central tracker (whose address it obtains by an independent bootstrap mechanism). This tracker gives the new node a random subset of nodes already in the system – typically the number ranges from 4-6. The new node then contacts each of these nodes and incorporates itself into the client overlay mesh. Note that instead of using a central tracker, we could use other techniques to find random subsets of nodes, like the ones proposed in [30].

Thus, each node is oblivious of the other nodes in the system except for a small subset which we designate as its *neighbourhood*. Note that this neighbourhood relationship is symmetric, i.e., if  $A$  is a neighbour of  $B$ , then  $B$  is also a neighbour of  $A$ . 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 property, while crucial for the scalability of the system, makes it challenging to obtain system-wide optimal scheduling policies.

When a node loses its 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 in accordance with the protocol or they cease to be a part of the system. In particular, they are not actively malicious – for example, they do not inject spurious data or messages into the system.

Finally, like the server, we assume that the clients themselves are constrained for resources (esp. network bandwidth). Thus, the rate at which a client can receive data from all its neighbours is limited by its download capacity. Similarly, the rate at which a client uploads to its neighbours is also limited. We shall assume that clients have symmetric links, i.e., their upload and download capacities are the same and are independent of each other.

### III. DESIGN

In this section, we will evolve a good design which addresses the scheduling problem in near-Video-on-Demand starting from naive approaches. In the process, we will highlight the factors which affect the performance of such a system. We take a simulator-based approach to study some of the key parameters. We have used a simulator since it provides us with the flexibility to explore the

whole design space and easily scale to many nodes, which would be difficult otherwise. We believe, however, that our simulator represents a real system quite faithfully. We will first present our simulator in Section III-A. We then discuss a few naive approaches and some refinements in Section IV. Finally, we will present two techniques which significantly improve the performance in a variety of scenarios – Prefetching (Section V-B.2) and Network Coding (Section VI).

#### A. Simulator

In this section, We will present the salient features of our simulator. The simulator takes as input the number of nodes, their capacities, the times at which they join the system and the size of the video file.

The simulator operates in discrete intervals of time (called *rounds*), each of which is 30s in duration. We target a streaming rate of 500 kbps and hence, the amount of data that can be transferred in a round is  $30s * 500\text{kbps} \approx 2 \text{ MB}$ . We assume a video to be 120min in duration (approximately the size of a movie) and hence, the entire file consists of  $120 \text{ min} / 30s = 240$  blocks, which we have rounded to 250 blocks in our experiments.

When a node joins the system, it picks 4 neighbours at random from the nodes already in the system (provided they have not exceeded the maximum number of neighbours, which is 6 in our experiments). The simulator supports dynamic arrivals and 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, however, 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 then the system moves to the second round.

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 critical properties of end-system cooperative architectures.

## B. Methodology

We will now describe our methodology in studying the near-Video-on-Demand problem. We emphasize that our goal is to ensure a low start-up time and a sustainable playback rate at the same time. Thus, for each simulation, we plot the number of consecutive blocks from the beginning that the node has, on the y-axis, and the time (in rounds) on the x-axis, for every node as shown in Figure 1. 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 obtain the median of the playback rates of all the participating nodes (In most of our experiments, we found that the 5<sup>th</sup> and the 95<sup>th</sup> percentiles closely follow the median playback rate.) We consider a setup time of 20 to 30 rounds (10-15 min) reasonable in our experiments.

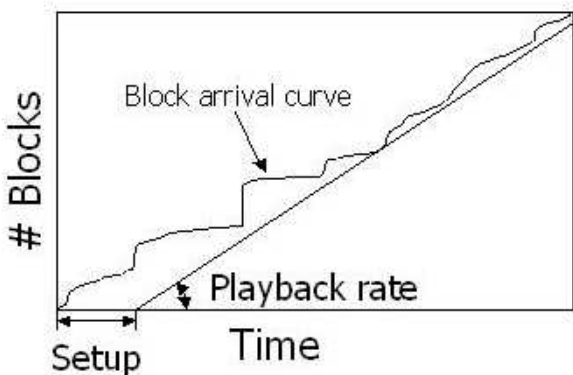


Fig. 1. This graph depicts the calculation of sustainable playback rate given the setup time. The y-axis shows the number of consecutive blocks at the node while the x-axis shows the time (in rounds).

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. While we want the progress of the system to be higher, we also want the block transfers to be useful to the nodes for sustaining a good playback rate, i.e., if a node is playing at the start of the movie, we want to fetch blocks nearer to the point of playback rather than fetching blocks from the end of the file.

In studying the near-Video-on-Demand 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*.

In this paper, 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 can be integrated into our design.

## IV. NAIVE APPROACHES

In this section, we will first present some naive approaches to solve the near-Video-on-Demand problem. To highlight the problems with these approaches, we consider a network of 500 nodes that want to simultaneously watch a video file. We will study dynamic arrival and departure of nodes later.

Existing file-distribution mechanisms like BitTorrent are not geared towards streaming of video files 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 as there are always blocks that could be transferred between two nodes (or of course, from the server). 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. From Figure 2, we can see the efficiency of the system is about 1%, which means that nodes need to download 99% of the movie before they can start watching it without interruptions. However, the total number of rounds required to download the movie is quite high. 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 blocks of the file.

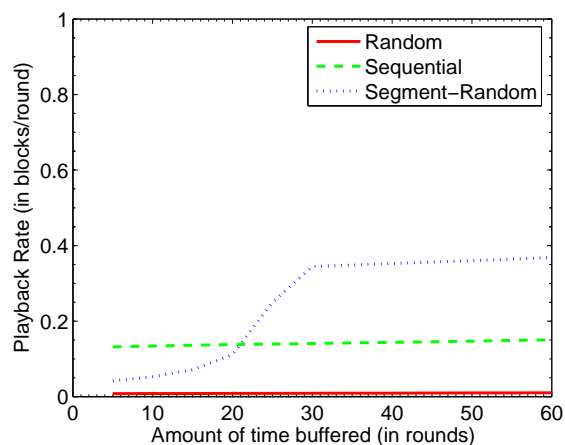


Fig. 2. Comparison of random, sequential and segment-random policies

Another naive 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 both the other nodes and the server. With this policy, nodes get the blocks that they actually want to play in order, resulting in a slightly better playback rate (to about 15% efficiency). However, the throughput of the system is reduced as all the nodes fetch the same blocks, thus, limiting the diversity of the system and the opportunities for cooperation.

An intermediate approach is to combine both solutions to achieve the high throughput of the random policy with the better playback rate of the sequential policy. To this end, one can 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. This segment-random policy is used by P2P live-streaming systems like [38].

With such a *segment-random* policy, a client requests a random block from the segment it's interested in. From Figure 2 we can see that this policy achieves both higher playback rates and low start-up times, thus retaining the best features of the random and sequential policies. However, note that although improved, the performance of the segment-random policy is still quite low. In fact, even with large initial buffering times, the playback rate is only around 35% of the access link rate.

1) *Valleys and Mountains*: We now look deeper into the performance of the system and try to better understand why these simple policies perform very 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.

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 the inefficiencies, and they arise for two independent reasons: a) the prolonged 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 phase of the dissemination of the following segment. The first problem can be solved by ensuring that there is enough diversity in the system. This can be achieved by doing proper scheduling (e.g. preventing rare blocks) or using coding techniques which introduce extra randomization in the system (we will study this in Section VI). The second problem relates to the fact that the first blocks in a segment take several rounds to propagate to all the nodes in the system. To

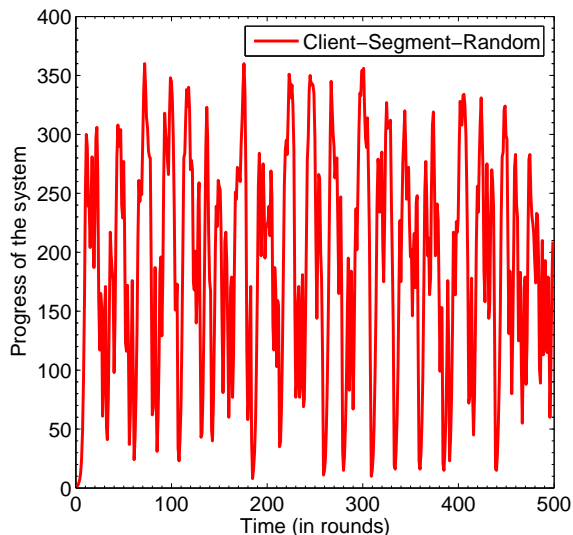


Fig. 3. Progress graph depicts the valleys and mountains

prevent this type of idle times, in Section V-B.2 we will look at pre-fetching policies that pre-populate a few nodes with blocks from the future so that blocks of upcoming segments are within reach for any node.

## V. BLOCK-SCHEDULING FOR EFFICIENT VOD

In this section we study the impact of different scheduling policies on the playback rate. We divide our study into investigating policies at the server (i.e., the node that has the only copy of the file in the system initially), and policies at the clients. We will first consider a large set of nodes (500 in number) that arrive at about the same time. This will help us understand the impact during a flash-crowd VoD delivery. Later, we will consider other arrival patterns.

### A. The Importance of Server Scheduling

During the initial delivery of the first copy of a movie file, the server plays a critical role in ensuring high playback rates. The server is the only node in the system that holds a copy of all the blocks in the file, and the rate of the system is very much 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*. During this phase, client policies have little impact since their experience is pretty much determined by whatever happens at the server. We will next study more in-depth the impact of different server policies. To this end, we will consider a) greedy policies where clients going to the server try to download content useful only for themselves, 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.

To study in detail the impact of server policies, we have fixed the client policy over the next set of experiments. For the rest of our experiments, we use a segment size of 10 blocks.

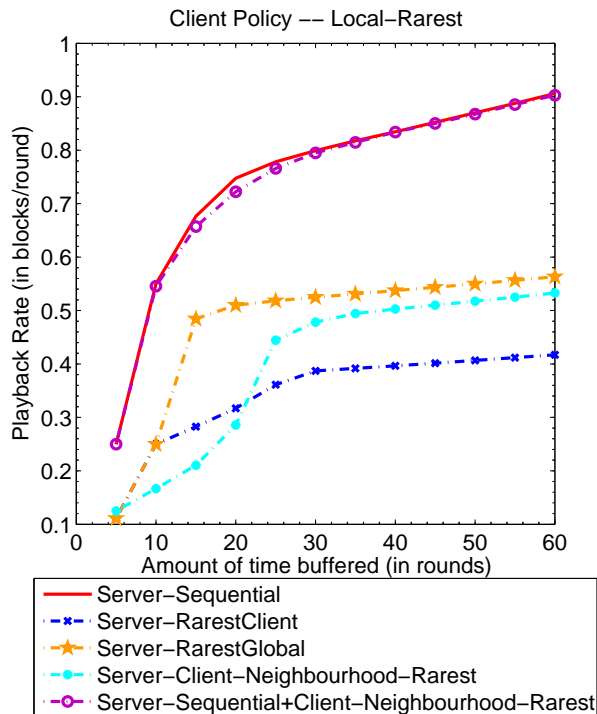


Fig. 4. Comparison of different server policies after setting the client policy to local-rarest. Note that the server policy set to a hybrid of Sequential + Client-Neighbourhood-Rarest performs the best.

We consider the following server policies: a) local-rarest, c) global-rarest, d) client-neighbourhood-rarest, and f) 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 rarest in the client’s neighbourhood. 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 of course requires global knowledge and is considered only for comparison purposes. In the client-neighbourhood-rarest policy, the server considers all the blocks which lie in the target segment not only of the client but also of all its neighbours. The server then gives the block that is rarest amongst all such blocks. 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. Of course, this again requires global knowledge and is considered only for comparison. We will next describe some simple strategies that can approximate this behaviour.

The results for the different server policies are shown in Figure 4. To better understand the performance of the different policies we analyze the usage of the server bandwidth resources. In Figure 5, for each policy, we present a histogram showing the number of times each block is injected by the server into the system.

We note that policies where the server satisfies the client’s requests perform very poorly. On the other hand, policies where the server overrules the client’s requests and serves blocks of no immediate interest to the client per se, but are of wider use for the system as a whole, perform quite well. This can be explained from the fact that the server is the only node having all the content initially, and hence it plays a crucial role in diversifying the cooperative cache quickly. Thus, the bandwidth resources of the server are precious and should not be wasted by injecting multiple copies of the same blocks.

For instance, with the sequential policy, the server gives most of the blocks only once and hence does not waste resources. Similarly, with the client-neighbourhood policy, each client fetches blocks which are useful to the neighbourhood as a whole and not just for itself. This reduces the frequency of any given block being pushed by the server. This consideration for the neighbourhood also makes this policy better than the global-rarest scheme.

While the client-neighbourhood policy performs quite well, it still results in a significant amount of duplicate information being delivered. One possible solution, is to use coding techniques that ensure that the probability of serving a duplicate block are reduced (Section VI). Other possible approaches are for the server to have a good estimate of the globally-rarest blocks in the system and serve them accordingly. One simple solution is for nodes to periodically report to a central tracker their progress. 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. We will discuss more of these policies in Section VIII.

In summary, server policies are crucial 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.

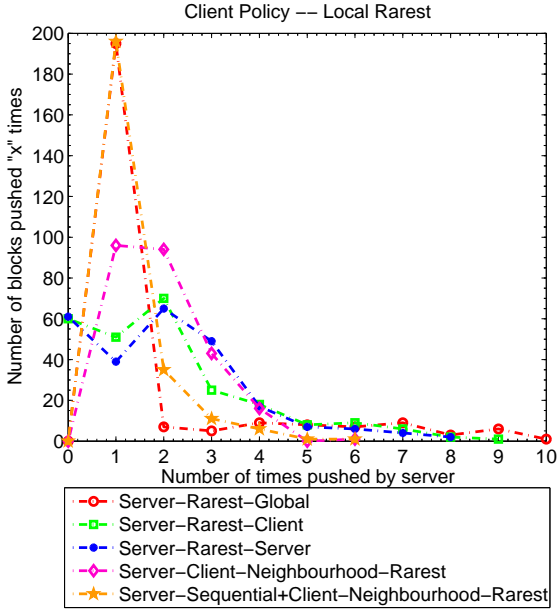


Fig. 5. The above graph shows the number of blocks which are pushed with a given frequency (which is shown on the x-axis). We note that the sequential policy performs the best because it pushes most blocks only once, and thus minimizes the wastage of server bandwidth resources.

### B. Client policies

In this section, we will present an analysis of different client policies. In order to eliminate the impact of the server during the ramp-up phase, we will assume that the server injects at least one copy of each block into the system and departs. Hence, at least one copy of the file is randomly and evenly spread amongst all the nodes present from the beginning.

From Figure 3, we see that the progress of the system follows a series of valleys and peaks that degrades the overall performance of the system. We will now discuss two techniques – 1) Local-Rarest which improves the peaks in the progress and reduce the period of degradation (during the dissemination of the last blocks of a segment), and 2) Pre-fetching which prevents the long periods of waiting for the initial blocks of a segment to propagate.

1) *Improved Scheduling: Local Rarest:* In our preliminary experiments, we noticed a distinct trend where the progress of the system is characterized by valleys and mountains (Figure 5).

We will now study the impact of having more intelligent scheduling decisions for the client side. To this end, 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 but

fetch that block 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. By adopting a local-rarest strategy, the scheduling of blocks improves and content flows more efficiently amongst the nodes. However, while such local-rarest policies are known to perform quite well in file downloading systems such as BitTorrent, 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 a block copy to the areas where they are needed. To counteract this problem, we will next study the impact of pre-fetching policies.

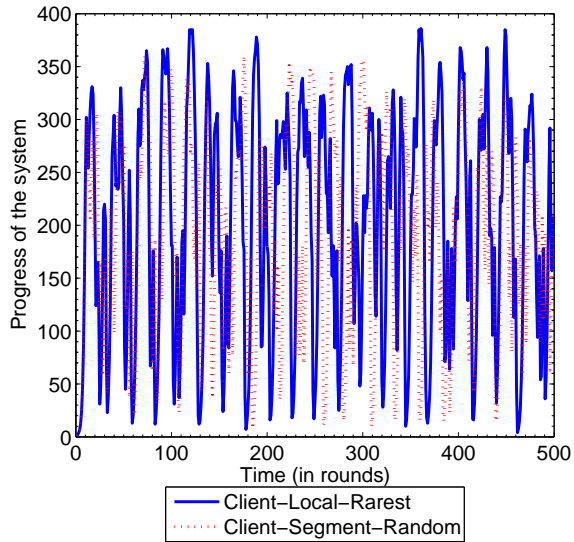


Fig. 6. We see that the local-rarest policy both improves the height of the peaks and reduces the duration of degradation, when compared with a segment-random policy. This happens because the local-rarest policy tries to ensure that all blocks are well-represented in the network as opposed to favouring the propagation of only a few blocks at the expense of others.

2) *Pre-fetching:* We now study the effect of pre-fetching, i.e. fetching a block that is needed later than the first 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 have a beneficial effect on the overall 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 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 with 90% probability, we pick the first segment and with 10% probability, we pick the second segment. Within each segment, we pick a block at random. In the probabilistic-first-range-random-second-range-rarest 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 the rarest block.

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 see that with pre-fetching, the valleys are not as deep.

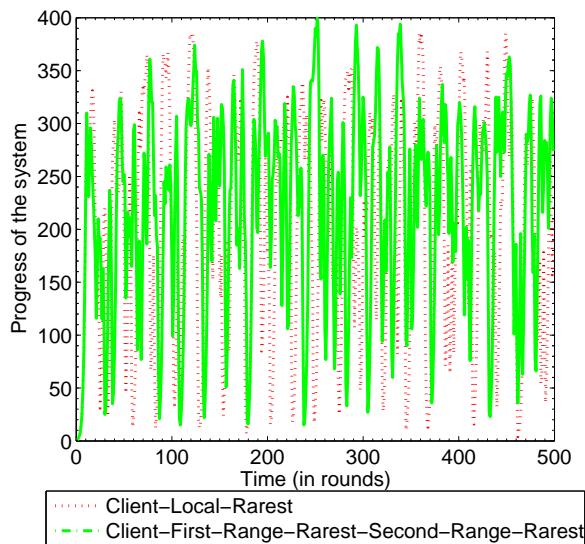


Fig. 7. The above graph demonstrates the benefits of prefetching. We see that the valleys in the progress of the system which uses a prefetching policy are far fewer in number. This happens because each node tries to fetch a few blocks in advance (with a small probability). These nodes then act as additional sources when the rest of the nodes too require those blocks.

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 actually needed. The tradeoff here 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.

## VI. NETWORK CODING

We now consider network coding to facilitate the scheduling of blocks among nodes, consequently, increasing the overall system’s progress the playback rates of the nodes. We first provide a brief description of network coding below and analyze its pros and cons. We then evaluate its benefits.

Network coding has been proposed by [8], [14], [18] 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 is the case with erasure coding) to encode data blocks. Network coding makes optimal use of bandwidth resources and the scheduling problem in this case is very easy. A good overview of network coding can be found in [29].

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

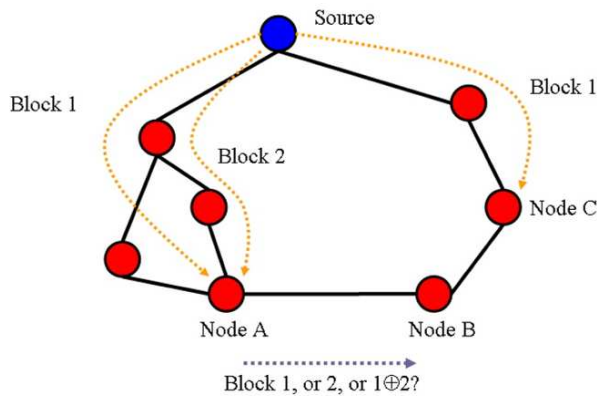


Fig. 8. This example shows the benefits of network coding when nodes only have local knowledge.

We now detail how network coding can be used in a P2P VoD 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  $c_1, c_2, \dots, c_n$ , and generates  $E1 = \sum_{i=1}^n c_i \cdot b_i$ , where  $b_i$  represents a block. The server then sends node A  $E1$  and the coefficient vector  $\vec{c} = (c_i)$ . Note that all the coefficients are chosen from and the operations done in a finite field.<sup>1</sup> 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  $E3 = c_1'' \cdot E1 + c_2'' \cdot E2$  and the new coefficient vector  $c_1'' \cdot \vec{c} + c_2'' \cdot \vec{c}'$ .

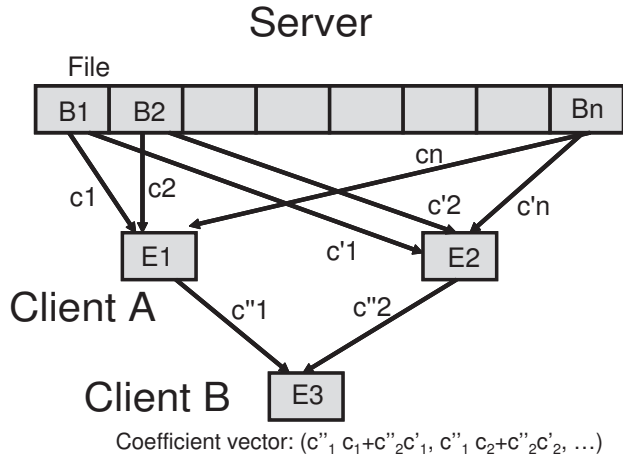


Fig. 9. A brief description of network coding in RedCarpet.

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 also makes sure that all blocks in a given segment can be easily completed and that the same encoded block can be used by multiple nodes in need of different blocks.

Next, we evaluate the performance of such coding scheme. The client policies which we have investigated with network coding are *Netcoding-Client-Segment-Random* where a client tries to fetch a block in the

<sup>1</sup>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  $2^{16}$  are enough to provide enough diversity.

segment of interest, and a pre-fetching policy *Netcoding-Client-90-10-Two-Ranges* in which 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.

In Figure 10(a), we compare the performance of such coding schemes with other non-coding schemes when a copy of the file is randomly distributed among all nodes. In this situation, there is enough diversity in the system to ensure that nodes can easily access any block in the file, consequently, coding strategies do not provide much benefit. In Figure 10(b), we show the situation where there are a large number of empty nodes that arrive simultaneously, and the original file only resides in a given node. This could be the case of a flash-crowd distribution. In such scenario, coding ensures increases the probability that blocks requested from the single node are unique, and thus, the propagation efficiency increases. Without coding, nodes often get stalled waiting for some blocks in a segment. Coding provides on average a 20-25% better playout rates for the same level of buffering.

When looking at the performance of network coding and pre-fetching (i.e. *Netcoding-Client-90-10-Two-Ranges*), in flash-crowd situations, pre-fetching improves the throughput of coding-based systems by an additional 10%. In our Figures, we have also included an ideal strategy called *ideal-netcoding-client-random* in which clients fetch blocks in random order (using network coding) but every block fetched is considered useful for playback (irrespective of the segment the block falls in). Although not practical, such strategy represents the optimal playback rate that can be achieved with the given system configuration and can be used as a reference point for comparison purposes. We observe that the performance of the network coding system is remarkably close to the optimal performance, despite the fact that receivers target arbitrary start play-out times.

## VII. DYNAMIC ARRIVALS

We now study the effectiveness of our policies when nodes join the system in different arrival patterns. In particular, we have considered the following arrival patterns for our simulations all of which were done with 500 nodes:

- 1) We start with 500 nodes but only one node amongst them has all the blocks of the video file. We note that this is a really important scenario as it corresponds to flash crowds in real systems.
- 2) We start with 10 nodes, each with 25 blocks of the file. Then 2 nodes join the system every round until there are 500 nodes.

- 3) We start with 50 nodes, each with 5 blocks of the file. Then, 50 nodes join as a group every 10 rounds until their number reaches 500.

For the above experiments, we assume that once a node downloads the entire video, it continues in the system for the next round with a probability of 0.5, i.e., on average, a node leaves 2 rounds after finishing. We present our results in Figures 10(b)(c)(d).

With nodes arriving dynamically at a constant rate (Figure 10c), we expect the playback rates to be low, as the nodes which join later do not have anything to offer to the early nodes. We can see, however, that all policies perform remarkably well under such a scenario. The reason is that the early nodes act as seeds for the content they have already downloaded for other nodes, thus minimizing the scheduling problem. In essence, early nodes use their spare capacity to serve later nodes. From the figure, we can also see that pre-fetching does not provide large benefits since future blocks are always within reach of a given node.

In Figure 10(d), we study the case where the node arrivals are batched and more bursty. By comparing Figures 10(c) and (d), we observe that the performance of the system is higher as nodes arrive more spaced apart since the chances for earlier nodes to have content that they can offer to later nodes is higher.

## VIII. DEPARTURES

In this section, we evaluate the robustness of our scheduling schemes, including coding, under various departure scenarios. While the robustness of coding techniques has been studied before, P2P VoD systems are more fragile as we need to ensure a steady delivery of blocks in order to the nodes. Despite these constraints, we find that network coding is very effective against sudden and massive failure of nodes in the system and provides significant benefits in comparison with other heuristics.

We try to capture the system behaviour when short-lived nodes get a block from the server and depart from the system without passing on the block to other nodes. We have considered block departures where 1/3 of the blocks simply disappear from the system as soon as they are injected (the nodes which receive those blocks stay though). We are interested in studying how fast the missing blocks can be re-populated. If blocks take long to re-appear in the system, the playback rate decreases since blocks can only be played in order.

To this end, we have investigated the following policies: a) Departure-Counter, b) Benefit-Neighbourhood and c) Netcoding. In the *departure-counter* policy, the server tries to estimate the number of copies of each block in the system. The server keeps track of the nodes to

which it has given a particular block. If the server finds that all the nodes to which it has a given a particular block have left the system, it reinjects that block into the system. In the *benefit-neighbourhood* policy, the server serves with 50% probability whatever content is local rarest in the client neighbourhood. The *Netcoding* policy uses the network coding techniques described earlier. All the experiments were done with 500 nodes and a server link capacity of 2 unless otherwise mentioned. We present our results in Figure 11.

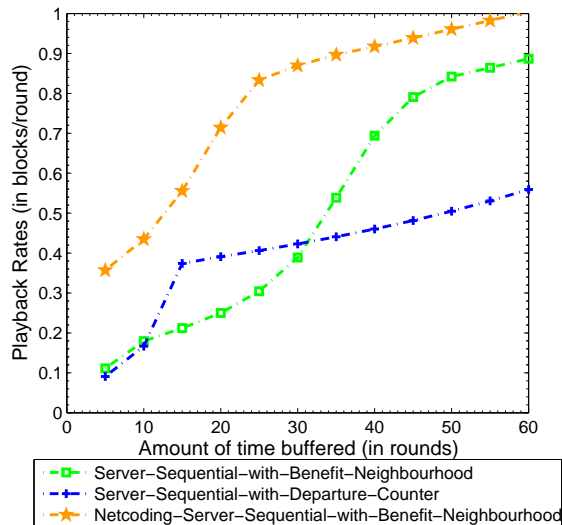
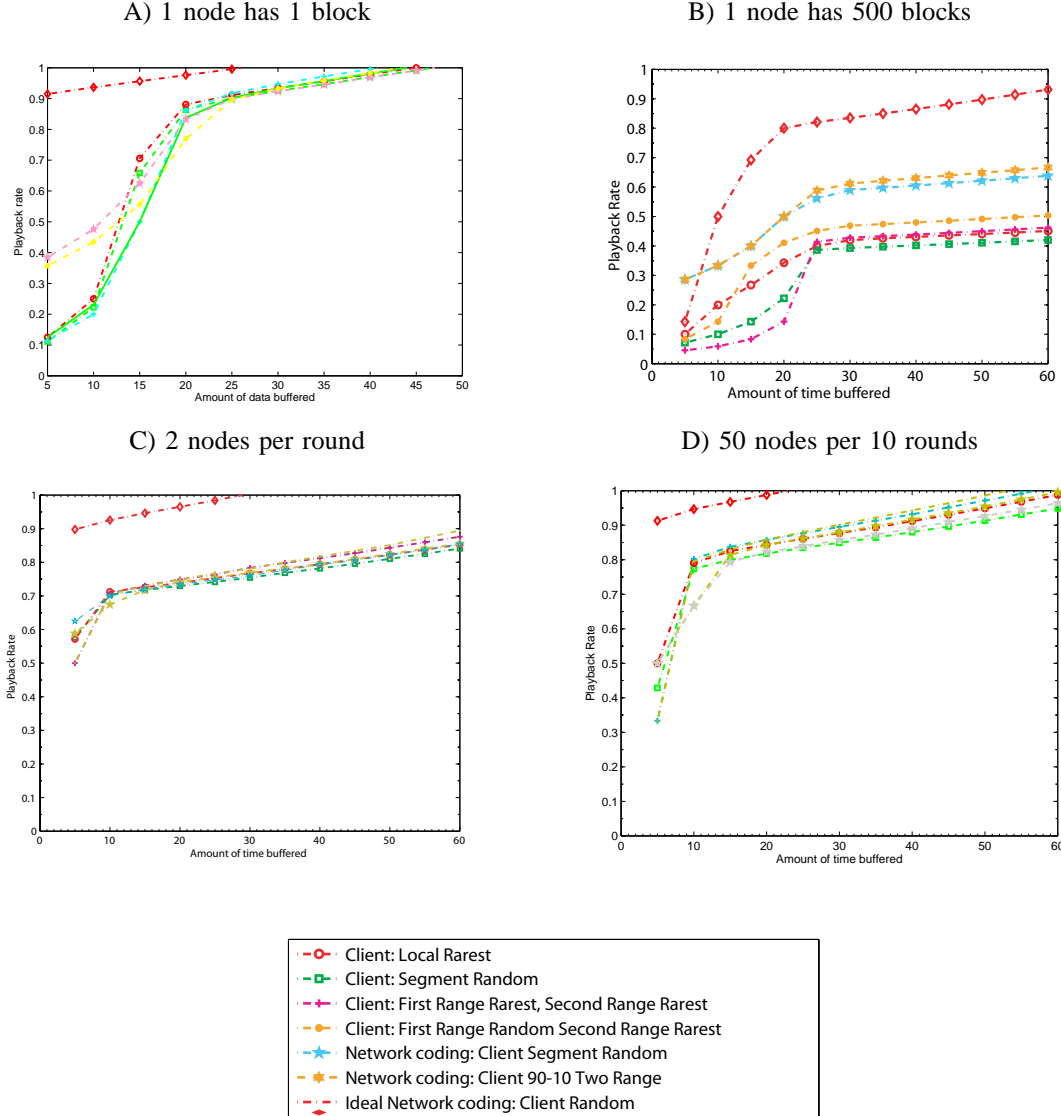


Fig. 11. This graph shows the impact of block departures and server link capacity on different server policies. Network coding proves to be very robust against mass block departures.

The departure-counter policy performs the best amongst all the non-netcoding schemes. The reason is that this policy detects in only a few rounds that a particular block is missing from the system and reinjects it. Still, network coding is far more effective and makes the system robust against such mass departures since an encoded block is able to repair multiple missing blocks. The Netcoding policy delivers playback rates close to 80% of the access link rate even under such massive departures. Network coding introduces a lot of randomization into the system and nodes do not ever wait on a particular block – they wait only on a particular segment. Hence, nodes quickly fetch an encoded block from the respective segment, which is used to repair the missing blocks. We have also tried increasing the capacity of the server (by 2x, 5x times) but have not observed any significant improvements. Thus, network coding is a key



Notes: 1) Playback rate is in blocks per round, 2) The amount of time buffered is in rounds.

Fig. 10. Playout rates vs initial buffering for various arrival patterns (a)-(d) and for various block exchange policies.

ingredient in ensuring the robustness of the system.

## IX. HETEROGENEOUS CLIENTS

In this section, we consider the efficacy of our scheduling policies when the system has heterogeneous clients, i.e., nodes that have differing link capacities. We again consider 500 initial nodes with one node having all the content. This node has a link capacity of 2. For the other nodes, we assign their link capacities according to the following distribution – 250 of the nodes (50%) are free-riders, i.e., they have an upload capacity of 0 though their download capacity is 1, 100 nodes (20%) have a link

capacity of 2, 50 nodes (10%) a link capacity of 6 and the remaining 50 nodes (10%) a link capacity of 10. This distribution of the link capacities for the clients is based on the data collected by [25]. Except for the free-riders, all the other nodes have symmetric upload and download capacities. We present our results in Figure 12.

We assume that the video file is pre-encoded at a given streaming rate. Hence, we expect that fast nodes would buffer only a small period of time and watch the video without interruptions, while slow nodes would need to buffer much longer. From Figure 12, we see that non-encoded strategies have a very low average playback rate

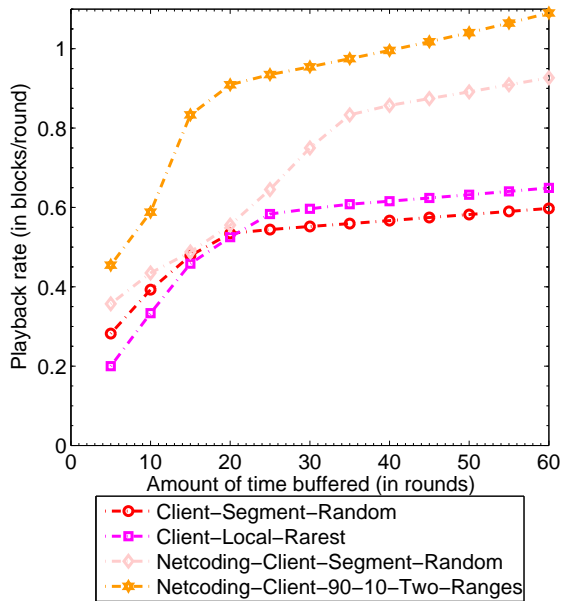


Fig. 12. This graph shows the performance of our scheduling policies in the presence of heterogeneous clients, i.e., nodes with varying access link capacities.

compared to coding strategies. In fact, coding strategies provide about 40% better playback rates. Given that the aggregate capacity of the system is  $250 * 0 + 100 * 1 + 50 * 2 + 50 * 6 + 50 * 10 = 1000$  units, we find that network coding achieves a much higher utilization while the non-coding strategies fail to leverage this extra capacity. This underlines the importance of network coding in achieving optimal scheduling of the dissemination of blocks. Also, the slow nodes tend to fall back in the download and when they contact the “server” node (which has all the content), they are likely to download blocks that already exist in other faster nodes that are leading the download. Thus, a lot of duplicate content is injected in the system by the “server” node. With network coding, on the other hand, the probability of injecting duplicate content is reduced, and the overall efficiency improves considerably.

## X. RELATED WORK

Video streaming over the Internet has been on of the most prolific research areas for over a decade, see [11], [13], [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 [24], [27], [32]. 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 [9]. More elaborate schemes propose to divide the video into segments and distribute each segment in different multicast channels [21], [22], [35]. 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 [7], [12], [19], [23], [34], [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], [35]. 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 is the BASS system [16]. BASS extends the current BitTorrent system [15] to provide a near-Video-on-Demand service. 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 (in the order of minutes as opposed to seconds).

## XI. CONCLUSIONS

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 the algorithms for scheduling the propagation of blocks are very important for determining both the utilization of the system and the ability of the nodes to play the content as they download it. We show that simple strategies, such as choosing the next block to transmit at random (in order to improve the network efficiency and is similar to the algorithms used by current P2P systems), or choosing the earliest block

that is not locally available (in order to support a play-as-you-download service) have very bad performance. We propose heuristics to overcome these problems and show that efficient VoD is possible with little buffering.

In summary, our findings are as follows:

- a) Policies that rely on clients satisfying their needs, perform poorly. We further observe that if the server transmits the globally rarest block, irrespectively of the needs of the receiving client, the performance of the system increases. However, estimating the global rarest block is difficult in a decentralized way, and, requires efficient methods for approximating such information.
- b) Local rarest policies improve the performance of the system. However, they do not eliminate the problem of slow progress during the ramp-up phases (i.e. when a new set of blocks enters the system). Pre-fetching, i.e. fetching blocks from later segments, can be used to reduce the effects of such ramp-up phases.
- c) The system performs well under dynamic arrivals, even without more advanced techniques like coding or pre-fetching. The reason is that older nodes in the system can help nodes that have arrived recently, and, hence, improve the play-out rates of the new nodes.
- d) Network coding performance is better, or equivalent to the optimal non-coding technique in all scenarios. Users can often target streaming rates that are close to 80% of their download rate with small buffers (e.g. 10% of the movie file).
- e) Amongst the strategies which do not use network coding, the *probabilistic-first-range-random-second-range-rarest* policy performs best and with network coding, *Netcoding-Client-90-10-Two-Ranges* works best across all scenarios.
- f) The benefits of pre-fetching and network coding are most visible when there are flash crowds. Thus, while pre-fetching (*probabilistic-first-range-random-second-range-rarest*) gives a 10% improvement, network coding (*Netcoding-Client-90-10-Two-Ranges*) provides a significant 40% increase in the playback rates.
- g) The system is highly robust even in the face of massive departures by using network coding techniques.

The focus of this paper has been in evaluating mesh-based systems for near Video-on-Demand applications, understanding the parameters that affect the performance of such systems, and giving guidelines and design principles for enabling low buffering delays and high play-out rates. Based on the results of this paper, we are building RedCarpet, a real NVoD system that includes most of the design principles discussed here. Preliminary evaluation confirms that a play-as-you-download experience can be efficiently supported using unstructured P2P networks.

## REFERENCES

- [1] eDonkey 2000 - overnet. <http://www.edonkey2000.com/>.
- [2] Feidian. <http://tv.net9.org/>.
- [3] Gnutella. <http://p2pjournal.com/main/gnutella.htm>.
- [4] KaZaA. <http://www.kazaa.com/>.
- [5] Pplive. <http://www.pplive.com/>.
- [6] Splitstream: High-bandwidth content distribution. <http://research.microsoft.com/~antr/SplitStream/default.htm>, Aug 2003.
- [7] End system multicast. <http://esm.cs.cmu.edu/>, 2005.
- [8] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. on Information Theory*, 46:1204–1216, 2000.
- [9] 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.
- [10] Mostafa Ammar. Why johnny can't multicast: Lessons about the evolution of the internet. NOSSDAV Keynote Speech, June 2003.
- [11] 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.
- [12] 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.
- [13] Shanwei Cen, Calton Pu, Richard Staehli, Crispin Cowan, and Jonathan Walpole. A distributed real time MPEG video audio player. In *NOSSDAV*, 1995.
- [14] P. A. Chou, Y. Wu, and K. Jain. Practical network coding. In *Allerton Conference on Communication, Control, and Computing*, Oct 2003.
- [15] Bram Cohen. BitTorrent. <http://www.bittorrent.com>.
- [16] 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.
- [17] Martin Reisslein Despina Saparilla, Keith Ross. Periodic broadcasting with vbr-encoded video. In *IEEE Infocom*. IEEE Press, 1999.
- [18] Christos Gkantsidis and Pablo Rodriguez. Network coding for large scale content distribution. In *IEEE Infocom*, 2005.
- [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. Mard, M. Effros, and D. Karger. On randomized network coding. In *41st Allerton Annual Conference on Communication, Control and Computing*, Oct 2003.
- [21] Ailan Hu. Video-on-demand broadcasting protocols: A comprehensive study. In *IEEE Infocom*, pages 508–571. IEEE Press, Apr 2001.
- [22] 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.
- [23] Yang hua Chu, Aditya Ganjam, T.S. Eugene Ng, Sanjay G. Rao, Kunwadee Sripanidkulchai, Jibin Zhan, and Hui Zhang. Early experience with an internet broadcast system based on overlay multicast. In *USENIX Annual Technical Conference*. USENIX, 2004.
- [24] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicasting for multimedia applications. In *IEEE Infocom'92*, volume 3, pages 2078–2085. IEEE Press, May 1992.
- [25] Bruce Maggs Kunwadee Sripanidkulchai, Aditya Ganjam and Hui Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. Aug 2004.
- [26] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Transactions on Information Theory*, 2003.

- [27] Xue Li, Sanjoy Pauly, and Mostafa 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. Mdard, R. Koetter, and P. A. Chou. Network coding: A new network design paradigm. In *IEEE International Symposium on Information Theory*, Adelaide, Sep 2005.
- [30] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proc. FOCS*, 2001.
- [31] Andrew Parker. P2P in 2005. [http://www.cachelogic.com/research/2005\\_slide01.php](http://www.cachelogic.com/research/2005_slide01.php), 2005.
- [32] J. C. Pasquale, G. C. Polyzos, and G. Xylomenos. The multimedia multicasting problem. *ACM Multimedia Systems*, 6:43–59, 1998.
- [33] 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.
- [34] Duc A. Tran, Kien A. Hua, and Tai Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE Infocom*. IEEE Press, 2003.
- [35] S. Viswanathan and T. Imiehnski. Pyramid broadcasting for video on demand service. In *IEEE Multimedia Computing and Networking Conference*. IEEE Press, 1995.
- [36] 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.
- [37] Dongyan Xu, Mohamed Hefeeda, Susanne Hambrusch, and Bharat Bhargava. On peer-to-peer media streaming. In *Proceedings of the 22nd 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.