Is Random Scheduling Sufficient in P2P Video Streaming?

Chao Liang
ECE Dept.
Polytechnic University
Brooklyn, NY, 11201
Email: cliang@photon.poly.edu

Yang Guo
Corporate Research
Thomson Inc.
Princeton, NJ 08540
Email: Yang.Guo@thomson.net

Yong Liu
ECE Dept.
Polytechnic University
Brooklyn, NY, 11201
Email: yongliu@poly.edu

***Abstract*—Peer-to-Peer (P2P) technology has recently been employed to deliver large scale video multicast services on the Internet. Considerable efforts have been made by both academia and industry on P2P streaming design. While academia mostly focus on exploring design space to approach the theoretical performance bounds, our recent measurement study on several commercial P2P streaming systems indicates that they are able to deliver good user Quality of Experience with seemingly simple designs. One intriguing question remains:** *how elaborate should a good P2P video streaming design be?* **Towards answering this question, we developed and implemented several representative P2P streaming designs, ranging from theoretically proved optimal designs to straightforward "naive" designs. Through an extensive comparison study on PlanetLab, we unveil several key factors contributing to the successes of simple P2P streaming designs, including system resource index, sever capacity and chunk scheduling rule, peer download buffering and peering degree. We also identify regions where naive designs are inadequate and more elaborate designs can improve things considerably. Our study not only brings us better understandings and more insights into the operation of existing systems, it also sheds lights on the design of future systems that can achieve a good balance between the performance and the complexity.**

I. Introduction

With the fast penetration of broadband residential accesses, Video-over-IP applications are quickly becoming the new generation "killer" applications on the Internet. Traditionally, videos are streamed from a server to clients through unicast connections. Due to the unpredictable packet losses, delays and delay jitters inside the "best-effort" Internet, it is challenging to maintain a continuous video playback on a client. Recently, several ISPs provide IPTV services through the combination of multicast and unicast inside their own private IP networks, such as FiOS of Verizon and U-Verse of AT&T. Content Delivery Networks (CDNs) have also been employed to push video content to the edges of networks, e.g. Youtube [1]. More recently, Peer-to-Peer (P2P) technology has enabled large scale video multicast on the Internet with minimum server and network infrastructure support [2], [3]. Compared with traditional client-server based solutions, P2P streaming faces additional challenges, such as random peer arrivals and departures, peer uploading bandwidth bottlenecks [4] and content bottlenecks [5]. Considerable efforts have been made by academia and industry on P2P streaming design. While academia mostly focus on exploring design space to approach theoretical performance bounds, e.g. [6], [7], commercial P2P streaming systems, such as PPLive [8], are able to deliver good user Quality of Experience with seemingly simple designs [9], [10].

One intriguing question remains: *how elaborate should a good P2P video streaming design be?* There is a whole spectrum of answers representing different tradeoffs between the performance and the complexity. Optimal streaming designs from academia reside at one end of the spectrum where the optimal performance is obtained through elaborate designs. Commercial system designs reside at the other end of the spectrum where the protocol simplicity is a high-order design rule, which is often followed at the price of a certain degree of performance loss. In this paper, we address this question through an extensive comparison study of several proposed P2P video streaming designs, ranging from theoretically proved optimal designs to straightforward "naive" designs. Among all the proposed P2P streaming mechanisms, our study unveils several key mechanisms contributing most to the successes of P2P streaming on the Internet, including system resource index, sever capacity and chunk scheduling rule, peer download buffering and peering degree. We also identify regions where naive designs are inadequate and more elaborate designs can improve things considerably. Our study not only brings us better understandings and more insights into the operation of existing systems, but also sheds lights on the design of future systems that can achieve a good balance between performance and complexity.

A comparison study like this can be done through different approaches. The first approach is modeling and analysis. However, tractable models are often based on unverified assumptions and many system details have to be abstracted out. The second approach is simulation. Due to the difficulty of simulating packet level events in large networks, P2P systems are mostly simulated at application layer only. Consequently, important details of the underneath native networks, such as topology, congestion, and packet delays, cannot be faithfully represented in the evaluation. The third approach is experiments on the Internet. The ideal places to conduct such experiments are deployed commercial P2P streaming systems. However, many commercial systems are proprietary, and they cannot afford user performance degradations introduced by large scale experiments on their running systems. In this study, we resort to controlled experiments on PlanetLab [11], a testbed overlaid on the Internet. We implemented five different P2P streaming designs and conducted extensive compara-

tive experiments using machines available on PlanetLab. Our experimental results provide us with valuable quantitative answers regarding the sensitivity and the insensitivity of P2P streaming performance to different design mechanisms. The contribution of this paper is three-fold:

1) We developed and implemented several representative P2P streaming designs, and conducted a thorough comparison study. Those designs were systematically evaluated through extensive experiments on the PlanetLab.

2) Our experiments demonstrated that the performance of P2P streaming is *insensitive* to scheduling when the streaming rate is low and long playback delays are tolerable. Even simple random scheduling can achieve a close-to-optimal performance. We further quantitatively showed that P2P streaming performance becomes *highly sensitive* to scheduling when the streaming rate approaches the maximum supportable rate of the system and the tolerable delays become small.

3) Our rich experimental results along different dimensions deepened our understandings on several fundamental tradeoffs in P2P streaming designs. Firstly, we showed that high streaming rates and short playback delays are achievable through elaborate designs with fine tuned peer management and data scheduling. Secondly, our study quantitatively demonstrated the importance of server infrastructure and super peers. We showed that increasing server capacity and exploring peer heterogeneity can significantly "bootstrap" the streaming performance on all peers. Thirdly, we demonstrated that elaborate peer buffer management allows one to simultaneously achieve good peer delay performance and playback continuity.

The remaining of this paper is organized as follows. We briefly describe the related work in Section II. Different designs and implementations are introduced in Section III. We then describe our evaluation methodology in Section IV. Major experimental results and findings are presented in Section V. The paper is concluded with discussion in Section VI.

II. Related Work

P2P streaming systems can be broadly classified into two categories, namely tree-based and mesh-based. The tree-based systems, such as ESM [2], have well-organized overlay structures and typically distribute video by actively pushing data from a peer to its children peers. One major drawback of tree-based streaming systems is their vulnerability to peer churn. A peer departure will interrupt video delivery to all peers in the subtree rooted at the departed peer. In a mesh-based P2P streaming system, peers are not confined to a static topology. Instead, the peering relationships are established/terminated based on the content availability and bandwidth availability on peers. A peer dynamically connects to a subset of random peers in the system. Peers periodically exchange information about their data availability. Video chunk is pulled/pushed between a peer and its neighbor who has already obtained that chunk. Many mesh-based P2P streaming systems have been proposed, such as Chainsaw [12], DONet/CoolStreaming [3], PRIME [13], etc. Most of the real P2P streaming systems deployed over Internet are based on this type of design. More recently, there have been ongoing efforts to improve resource utilization and achieve maximum streaming rate allowed by the system. Random useful packet forwarding [6] is a randomized distributed algorithm that can converge to the maximum streaming rate. Adaptive queue based chunk scheduling [14] is another distributed optimal P2P streaming algorithm.

The authors in [15] studied simple pull-based peer-to-peer streaming system. With proper configuration, simple pull-based scheduling comes close to optimum in terms of bandwidth utilization and throughput. Our study confirms some of their results. In addition, we examine the system performance from multiple angles, e.g., system resource index, server scheduling rule, buffering/playback delay, peering degree, etc. Hence our work offers a more comprehensive treatment of the performance comparison study.

III. P2P Streaming: Design and Implementation

In this section we describe several P2P streaming algorithms that will be used later on in performance comparison. We select these algorithms either due to their provable optimality, or due to their design simplicity and their adoptions in existing P2P streaming solutions. For each scheduling algorithm, we fine tune its design and carefully select the configuration parameters. This ensures individual scheduling algorithm to operate at its peak performance. Our goal is to compare the performance of these representative scheduling algorithms and identify factors that contribute most to the performance of a P2P streaming system. The performance comparison also allows us to answer fundamental questions such as if the scheduling is indeed important in P2P streaming.

There are three basic scheduling algorithms: Random Pull (RP), Adaptive Queue Based Chunk scheduling (AQCS), and Random Useful Packet Forwarding (RUPF). AQCS and RUPF have been theoretically proved to be optimum [14], [6]. RP scheduling is a simple scheduling and bears similarity to the scheduling used in real P2P streaming solution [9]. Two variations of RP, Random Pull with Fresh chunk first (RPF) and Random Pull with Most-Deprived peer selection (RPMD), are also introduced for comparison.

A. Baseline Scheduling : Random Pull Scheduling

As a baseline scheduling, Random-Pull (*RP*) employs pull mode at the data exchange layer. Pull mode is widely employed in P2P streaming since it avoids duplicate data chunks. A peer in RP scheduling randomly select M neighbors to serve. The value of M is chosen to be proportional to the peer's upload capacity. A peer can request up to K data chunks from one serving peer. The value of K is set so that K chunks can be downloaded within *one window sliding period*, the epoch at which download window moves forward. At the data exchange layer, RP randomly selects missing chunks to pull. Figure 1 illustrates a simple example of the above process. Peer P randomly selects peer P_a and P_b to serve by issuing pull tokens to them. Peer P_a then sends a pull request back to P for some missing chunks which P owns. P_b sends a pull waiver message instead since all of its missing chunks have been scheduled to be pulled from other peers. After receiving

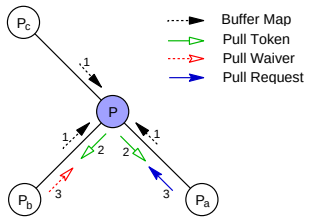


Fig. 1. Random pull scheduling

the waiver message, P can send the token to another unselected peer. The value K is selected by peer P according to its uplink capacity, and is piggybacked on the pull tokens.

B. Optimal Scheduling I: Adaptive Queue-based Chunk Scheduling

Adaptive queue-based chunk scheduling (*AQCS*) [14] is a deterministic distributed scheduling algorithm. AQCS can support the optimal streaming rate in practical networking environment. The mesh construction layer in AQCS maintains a fully connected mesh among participating peers. Data chunks are pulled/pushed from the server to all peers, cached at peers' forwarding queues, and relayed from peers to their neighbors. The availability of spare upload capacity on a peer is inferred from the occupancy of its forwarding queue. Signals are passed between the server and peers to adaptively allocate chunk forwarding workloads to peers based on their current bandwidth availabilities. Figure 2 illustrates a simple example with one source server and three peers. Each peer maintains

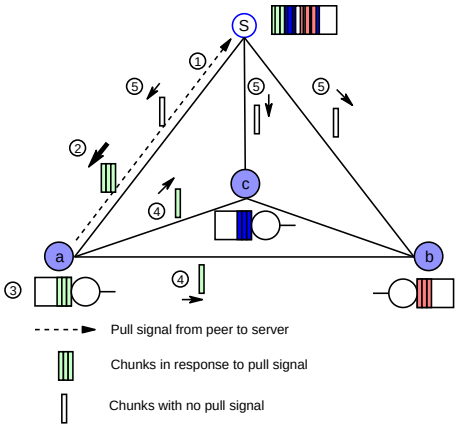


Fig. 2. Queue-based P2P system with four nodes. 1. peer a sends pull signal to the server; 2. the server replies with three chunks in response to the pull signal; 3. three chunks are cached in the peer a forward queue; 4. cached chunks are forwarded to neighbor peers sequentially; 5. duplicate chunks are sent to all peers when the server has finished all 'pull' signals processing.

several queues including a forwarding queue. Using peer a as an example, the signal and data flow is described next. A pull signal are sent from peers a to the server whenever its forwarding queue becomes empty (or falls below a threshold) (step 1 in Figure 1). The server responds to a pull signal by sending three data chunks back to peer a (step 2). These chunks will be stored in the forwarding queue (step 3) and be relayed to peer b and peer c (step 4). When the server has responded to all 'pull' signals in its 'pull' signal queue, it uploads one data chunk to all peers (step 5). This data chunk will not be stored in peer's forwarding queue and will not be relayed further.

The server's peer scheduling layer selects the peers that issue the pull signals. The received chunks at peers are relayed to other peers hence peers' peer scheduling layer in AQCS is not required. Server's data exchange layer always pushes out the fresh chunks that have not been served before, while peers' data exchange layer conducts chunk relaying.

C. Optimal Scheduling II: Random Useful Packet Forwarding

Random Useful Packet Forwarding (*RUPF*) [6] is a randomized distributed algorithm that can converge to the optimal streaming rate. The optimality of the method is proved under the assumption that the mesh is fully connected. However the scheduling can be extended to arbitrary mesh networks. The key in RUPF is for peers to identify the neighboring peer that needs data the most, namely the *most-deprived peer*. Peers serve data chunks to their respective most-deprived peers. The served data chunks are randomly selected from the missing chunk set of the receiving peer. The source server in RUPF always gives highest priority to the fresh data chunks that have not been sent out before. It turns out that the fresh-data-first policy plays an important role in determining P2P system's performance as shown in Section V.

Although RUPF is proved to be optimal, its performance is not satisfactory in practice. Our experiment conducted over the PlanetLab shows that a large number of duplicate chunks are observed at receiving peers and a high percentage of chunks miss their playback deadlines. In the original RUPF design, once the most-deprived peer is chosen, a peer pushes randomly selected chunks based on its local view of the available chunks at the most-deprived peer (derived from the buffer maps exchanged between peers). Since the information of available chunks at the most-deprived peer may be outdated, and more importantly, a peer may be chosen as the most-deprived peer by multiple peers, collisions becomes inevitable. In addition, since a peer only chooses a single most deprived peer, its uplink capacity sometimes cannot be fully utilized. To address the above issues, we replace RUPF's data chunk push method by a data chunk pull method. Each peer is allowed to serve the top M in the most-deprived peer list. A peer sends tokens to its most-deprived peers. The token indicates that this peer is willing to serve the data. The most-deprived peers then issue the data chunk request with bit-map defining which chunks it wants to download. In a nutshell, the improved RUPF is similar to random pull (RP) scheduling except that: (1) RUPF selects M most deprived peers while RP select M random peers; and (2) the source server in RUPF pushes out fresh data chunks first while the server in RP does not give preferential treatment to fresh chunks. To examine the impact of most-deprived-peer-selection policy and fresh-chunk-first policy, we also implement two variations of random pull by augmenting random pull with fresh chunk first (RPF) and the most-deprived peer selection (RPMD) respectively.

Finally we briefly describe how to find the most deprived peers in RUPF. Let $P(u)$ and $P(v)$ denote the set of chunks

that peer u and peer v have received. The expected chunk list of peer v from peer u is $E(v,u) = P(u) \setminus P(v)$, i.e., the chunks peer u has while peer v does not. The most deprived peer is the peer whose $|E(v,u)|$ is the maximum. In addition, buffer maps of peers need to be synchronized in order to determine the most deprived peers. In practice, peers join the system at different time instants. The network and computation delay can cause asynchronous download windows. A peer is only interested in missing chunks in its local download window. The buffer maps received from other peers should be adapted based on the position of the local download window. Figure 3 shows how to transform peers' buffer map to choose the most deprived peer. A local peer received buffer maps from

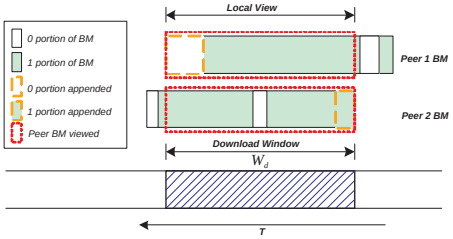


Fig. 3. Peer buffer map alignment for choosing most-deprived peer

peer 1 and peer 2. The local peer's window lies in between the windows of peer 1 and 2. To align peer 1's buffer map with the local download window, we truncate peer 1's buffer map outside the local download window, and append 0s to the beginning of peer 1's buffer map, assuming that peer 1 has not received these chunks. Similarly, peer 2's buffer map is truncated according to the local view and 1s are appended to the end of buffer map, assuming that peer 2 still caches these chunks in its buffer.

IV. Evaluation Methodology

A. Experiment Setting

The prototypes of RP, AQCS, RUPF, RPF, and RPMD are implemented and the experiments are conducted over PlanetLab [11]. The prototypes have more than $10,000$ lines of C++ code. More than 100 PlanetLab nodes are used with most of them located in North America. In our experiments, all connections between nodes are TCP connections. This avoids network layer data losses, and allows us to use software package Trickle [16] to set nodes' uplink bandwidth.

We follow the common assumption that the downlink bandwidth is not the bottleneck and thus do not set limit on them. To make the experiments more realistic, the uplink bandwidth of peers are assigned according to the distribution as in [14]. Specifically, 20% peers have upload capacity of 128kbps, 40% have 384kbps, 25% have 1Mbps, and 15% have 4Mbps. The resource index, defined as the total upload bandwidth over the number of peers times the streaming rate, reflects the resource availability in the system. It is necessary for the resource index to be greater than one so that the peer-to-peer streaming system can support the corresponding streaming rate. In our experiments, the average upload bandwidth is around 1Mbps.

In order to facilitate the system performance analysis, we also implement an auxiliary *log* system. During the experiment, each node collects related statistics every second and estimates the average upload and download rate per ten seconds. These information is recorded at local log files and is transmitted to one public sink after the experiment is over. The experiment results are analyzed off-line.

B. Performance Metrics

Following performance metrics are investigated to evaluate the performance of peer-to-peer streaming systems.

1) *chunk miss ratio*: During the video playback, chunks have to be received before their playback deadline. The chunks missing the deadlines are skipped or the video playback freezes. Both downgrade users' viewing quality. *Chunk miss ratio* is defined as the number of data chunks which miss the playback deadline over the total number of data chunks that peer should receive. Since chunks are of the same size, this term directly reflects the playback quality at user end. *Average chunk miss ratio* is the average of chunk miss ratios across all peers. It takes time for system to reach steady state. In our off-line analysis, the first 60 seconds data is skipped to minimize the impact of any factor due to unsteady state.
2) *peer bandwidth utilization*: The bandwidth utilization reflects how effective scheduling algorithms utilize the bandwidth resources. A poor scheduling scheme may underutilize a peer's upload capacity while other peers are "starved" of content, so-called *content bottleneck* phenomenon. The bandwidth utilization is computed by estimating the average upload rate and dividing it by the peers' pre-set upload capacity.
3) *playback delay*: Playback delay is the elapsed time from the moment when a chunk is generated at the source until it is played at the peer side.

In the following experiments, we study the impact of different design parameters by examining these performance metrics.

V. Experimental Results

In this section, we present experiment results obtained in our comparison study of five different P2P streaming designs, Random Pull (RP), Random Pull with Fresh first policy (RPF), Random Pull with Most-Deprived peer selection (RPMD), Random Useful Packet Forwarding (RUPF) and Adaptive Queue-based Chunk Scheduling (AQCS).

A. Impact of System Resource Index

As shown in [4], the maximum supportable streaming rate in a P2P streaming system with n peers is

$$r_{max} = \min\left\{u_s, \frac{u_s + \sum_{i=1}^{n} u_i}{n}\right\} \quad (1)$$

where u_s is the uploading capacity of the server and u_i is the uploading capacity of peer i. Denote by $\bar{u}$ the average upload capacity per peer, i.e., $\bar{u} = \frac{u_s + \sum_{i=1}^{n} u_i}{n}$. If $u_s \leq \bar{u}$,

$r_{max} = u_s$. The video rate is bounded by the server uploading rate u_s. We call such scenario *server resource poor scenario*. In contrast, if $u_s > \bar{u}$, $r_{max} = \bar{u}$. The server is no longer the performance bottleneck. We call it *server resource rich scenario*. The bandwidth resource index ρ is defined as

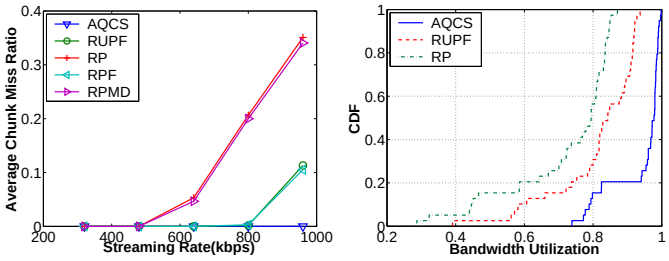
$$\rho \triangleq \frac{\bar{u}}{r} = \frac{u_s + \sum_{i=1}^{n} u_i}{nr}, \quad (2)$$

which is the ratio of the total uploading resources in the system to the total resource demand at streaming rate of r.

Below we compare the performance of scheduling algorithms at different resource indexes by varying the streaming rate. In AQCS, the chunks size is set to be 1KByte, and the server replies each pull signal with only one chunk. We experiment with other parameters and the current setting gives us the best performance. The server increases the streaming rate by increasing the number of chunks generated per second. In RUPF and RP, we fix the buffer map size at 120bits (15 bytes). The download window is set to be 30 seconds and moves forward every 10 seconds. The server produces four chunks per second and increases the streaming rate by increasing the chunk size (this way the buffer map size remains the same). We select 40 stable PlanetLab nodes for this experiment. Using Trickle, we set the uplink bandwidth according to the distribution. A full mesh is formed among nodes. The content source server's uploading capacity is set at 1 Mbps. We gradually increase the streaming rate from 480 kbps to 960 kbps. Correspondently, the resource index of the system decreases from 2.2 to 1.1. For each streaming rate, we conduct one set of experiments for each scheduling algorithm. Each experiment lasts for 300 seconds. Based on our off-line data analysis, the experiment duration of 300 seconds is appropriate since the systems goes into the steady state within tens of seconds. Figure 4(a) shows the average chunk miss ratios of the various scheduling algorithms. When the streaming rate is 480kbps ($\rho = 2.2$) or smaller, all scheduling methods have zero chunk miss ratio. This indicates that when the system has high resource index, the chunk miss ratio is insensitive to scheduling algorithms. Even random scheduling can have very good performance. Indeed, most current commercial P2P streaming systems on the Internet operate at streaming rates of 400kbps or lower. Our experimental results explain the successes of simple P2P streaming designs.

As we increase the streaming rate above 480kbps, the performance of RP and RPMD drops sharply. Meanwhile, with more elaborate designs, RPF and RUPF still perform satisfactorily until the streaming rate approaches 960kbps. At this point, AQCS outperforms all others and sustains zero chunk miss ratio. This suggests that intelligent scheduling does make difference when the resource index is low.

To understand the root cause of performance difference among different scheduling algorithms, we look into peers' uploading bandwidth utilization when the streaming rate is 960kbps ($\rho = 1.1$). At this streaming rate, RP and RUPF has around 35% and 11% chunk miss ratio, respectively, while AQCS has zero loss. Figure 4(b) shows the distribution of peer average bandwidth utilization during the experiment. Peers in



(a) Chunk Miss Ratio (b) Bandwidth Utilization CDF

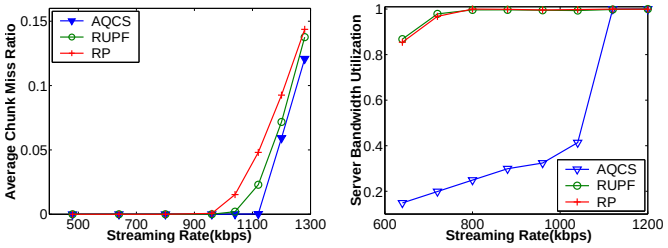
Fig. 4. Streaming performance of different scheduling algorithms at different resource indexes when server capacity is 1Mbps. (a) Average chunk miss ratio at different streaming rates. The performance is insensitive to scheduling at high resource indexes, and becomes sensitive to scheduling at low resource indexes; (b)At low resource indexes, scheduling with higher peer bandwidth utilization achieves better performance.

AQCS have average bandwidth utilization around 94%, while that in RUPF and RP is around 80% and 72%, respectively. When the system resource index is close to one, it becomes critical to utilize all peers' uploading bandwidth. In this experiment, RUPF and RP encounter content bottleneck [5] with small resource index. Content bottleneck wastes bandwidth since the peers cannot obtain new chunks in time and have nothing to upload to other peers.

B. Impact of Server Scheduling Rule and Capacity

It is interesting to notice the performance difference between random pull (RP) and random pull with fresh chunk first (RPF) as shown in Figure 4(a). RPF scheduling is a variation of RP and implements fresh-chunk-first scheduling at the server. The server gives strict priority to push out fresh chunks so that new content can be quickly distributed among peers. RPF out-performs RP significantly and its performance is very close to the performance of RUPF. In contrast, random pull with most deprived peer scheduling (RPMD) implements most deprived peer scheduling as in random useful packet forwarding (RUPF). The chunk miss ratio in RPMD is almost as bad as in RP. The experiment results suggest that (i) the fresh-chunk-first scheduling at source server plays an important role in improving the system performance; and (ii) most deprived scheduling, although has been theoretically shown to be optimal [6], does not seem to bring performance improvement in our experiments.

Next we investigate the impact of server bandwidth. In the previous experiment, the server uploading bandwidth is 1Mbps. Since the average peer uploading capacity in the system is slightly above 1Mbps, it corresponds to the server resource poor scenario. In the following experiments, we increase the server bandwidth to 3.2Mbps, leading to the server resource rich scenario. We repeat the previous experiment with a wider range of streaming rates and plot the results for AQCS, RUPF and RP in Figure 5(a). (We skip RPMD and RPF because their performance lies in between that of RP and RUPF). The curves show that all miss ratios are nearly zero, even for RP, when the streaming rate is less than 960kbps. The chunk miss ratio goes up rapidly for RUPF and RP when streaming rate goes beyond 960kbps while AQCS still maintains zero miss ratio up to 1,100kbps.



(a) Chunk Miss Ratio (b) Server Bandwidth Utilization

Fig. 5. Streaming performance of different scheduling algorithms at different resource indexes when server capacity is 3.2Mbps. (a) Increasing server bandwidth dramatically extends the scheduling insensitive region; (b) More efficient P2P scheduling impose lighter load on server.

Since the system resource index drops below one when the streaming rate is larger than $1,100$kbps, the miss ratio for AQCS increases linearly at the end. This again verifies that *the streaming performance becomes insensitive to scheduling when the system resource index is high, and sensitive to scheduling when resource index is low.*

More interestingly, increasing server capacity from 1Mbps to 3.2Mbps extends the scheduling insensitive region dramatically from 480kbps ($\rho = 2.2$) in Figure 4(a) to 960kbps ($\rho = 1.155$) in Figure 5(a). Obviously, increasing server capacity can increase the resource index of the system. And system performance will improve as resource index increases. However, according to (2), the increase in server bandwidth will be factored down by the number of peers in the system. In fact, in this experiment, the resource index at 960kbps is 1.155 which is only slightly higher than the resource index of 1.1 at 960kbps in the previous experiment. The average chunk miss ratios for RP and RUPF improve from 35% and 11% to zero respectively. This suggests that increasing server uploading capacity brings in more dramatic performance improvement than barely increasing system resource index. Figure 5(b) illustrates the average server bandwidth utilization under different streaming rates. The server utilizations in RPUF and RP are always high. This suggests that the server bandwidth plays an important role in reducing the chunk miss ratio for these two scheduling algorithms. On the contrary, the server utilization in AQCS is low until the streaming rate exceeds 1Mbps and the resource index falls below 1. This suggests that an efficient P2P streaming design can effectively bring down the server load and improve the system scalability.

To further examine the impact of server bandwidth, we conducted additional experiments by continuously varying server upload capacity. We fix the streaming rate at 640kbps and increase the server bandwidth from 600kbps to 1.2Mbps. All other experiment setting is the same as the previous experiments. Figure 6 shows the average chunk miss ratio for five different scheduling algorithms at different server bandwidth. When the server bandwidth is low, there are performance gaps among them. Again, algorithms with fresh-chunk-first rule have better performance. As the server bandwidth approaches 1.2Mbps (twice the streaming rate), the performance becomes insensitive to scheduling and all of them have nearly zero chunk miss ratios. This again shows the unique impact of the

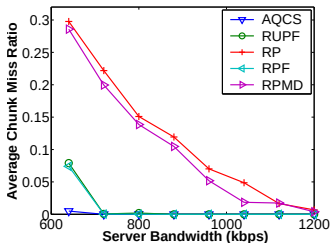


Fig. 6. Average chunk miss ratio under different server bandwidth. Increasing the server bandwidth can quickly improve the system performance.

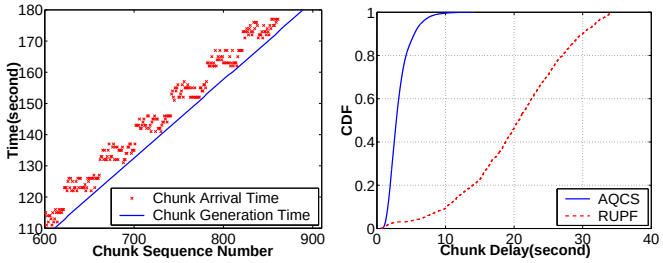
server bandwidth on the whole system. One explanation is that a server with high bandwidth can simultaneously upload a chunk to many peers. This effectively increases the fan-out degrees of chunk delivery trees at their roots thus chunks can be delivered to all peers quicker. The same amount of bandwidth increase on a regular peer does not have such significant impact. Our results here suggest that *investing on server bandwidth can dramatically bootstrap the performance of P2P streaming systems*, which is consistent with findings from other analysis [7] and experimental studies [15].

C. Impact of Buffering Delay

In client-server based video streaming, client side video buffering is necessary for continuous playback in face of network bandwidth variations. In P2P streaming, each peer maintains a moving window that specifies the range of video chunks to be downloaded. The window normally advances at the video playback rate. The window size determines the length of playback delay. The larger window size gives peers more time to download chunks. However, the larger window size imposes longer playback delay. The dimensioning of buffering delay is indeed a trade-off between the streaming delay performance and playback continuity. In this section, we investigate the impact of buffering delay on the performance of different P2P streaming designs.

We first study the video chunk delays under different streaming rates. We measure the chunk delays in the server resource rich experiments presented in Section V-B. For each chunk, the server records its generation time, and peers record the chunk's arrival time. In RUPF, the download window is 30 seconds and moves forward every 10 seconds. In AQCS, the download window is 15 seconds and moves forward every 1 seconds. Figure 7(a) shows the arrival times of chunks on one peer and their generation times at server side when the streaming rate is 480kbps in RUPF. With this low streaming rate, the system has high resource index. We can observe from Figure 7(a) that chunks arrive at the monitored peer in batches. The moving window advances once every 10 seconds. After each window advance, the peer can quickly finish the downloading of all missing chunks entering into the moving window. The peer stays idle to wait for the next window advance. The fast downloading at low streaming rate enables us to reduce the download window size to achieve shorter playback delays on all peers.

We then compare the delay performance of RUPF and AQCS at the streaming rate of 1120kbps. From Figure 5(a),

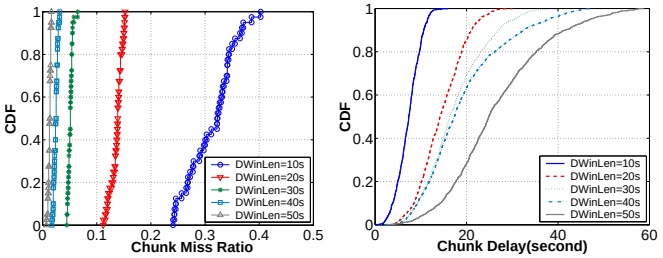


(a) Chunk Arrivals at Low Rate (b) Chunk Delays at High Rate

Fig. 7. Delay performance under different streaming rates. (a) Chunk arrival patterns at rate 480kbps, high resource index leads to fast chunk downloading. (b) Chunk delays at rate 1120kbps, low resource index leads to high delay variability in RUPF.

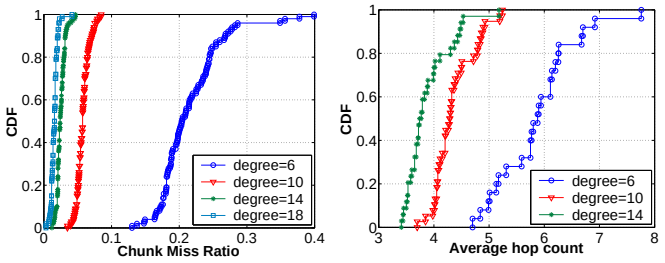
the chunk miss ratio in RUPF is around 3%. For chunks downloaded before their playback deadlines, we plot their delay distribution in Figure 7(b). The chunk delays in RUPF have a wide variability with the average delay of 20.2 seconds. In comparison, AQCS not only achieves zero chunk miss ratio, but also has superb delay performance. All chunks can be received within 10 seconds after they are generated by the server. The average chunk delay is 3.2 seconds. This shows us that *efficient scheduling can help reduce the video delays and require less buffering at peers.*

To reduce chunk miss ratio under uploading and downloading rate oscillations, one can introduce a large buffering delay on peers. We examine the performance of RP with various buffer lengths. The streaming rate and the server bandwidth are fixed to be 640kbps and 1Mbps. We then vary the download window size of all peers from 10 seconds to 50 seconds. Figure 8(a) shows the chunk miss ratio distribution of all nodes with different buffer lengths. As the buffer length increases, the chunk miss ratio decreases. With 10 seconds buffer length, all peers encounter larger than 23% chunk loss. The average chunk miss loss drops to 5% when the buffer length increases to 30 seconds. As the buffer length continues to increase, the improvement becomes less because fewer and fewer chunks cannot meet the playback deadline. To examine the impact of buffer length on the delay performance, we plot the delay distribution of chunks received by one specific peer with different buffer lengths in Figure 8(b). As the buffer length increases, the average and worst chunk delay also become larger. The results confirm that the chunk miss ratio can be reduced by increasing peer buffering, although that may



(a) Chunk Miss Ratio (b) Chunk Delay

Fig. 8. Chunk miss ratio and delay trade-offs under different buffer lengths. (a) chunk miss ratio decreases as buffer length increases. (b) chunk delay increases as buffer length increases.



(a) RUPF Homogenous Degree (b) RUPF Average Hop Number

Fig. 9. System performance with different peering topologies and degrees. Larger peering degrees lead to lower chunk miss ratios and shorter delays. Heterogeneity peering topology has better performance.

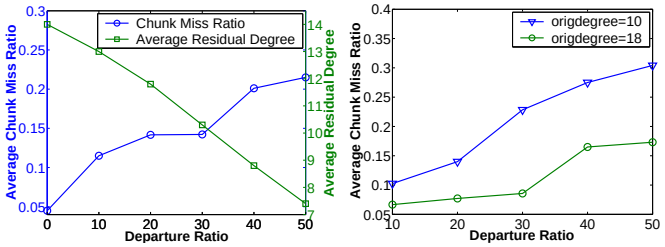
introduce longer delays. From this experiment of RP, we see that *while random scheduling can achieve zero chunk miss ratio with long buffering delays, a careful scheduling design is necessary to simultaneously achieve low chunk miss ratio and short chunk delay.*

D. Impact of Peering Topology and Degree

The peering degree is another important parameter in P2P streaming designs. In a random mesh based streaming system, a peer only connects to a subset of randomly selected neighbors and exchanges data with them. Due to connection and signaling overhead, the number of neighbors of each peer cannot be too large. In this section, we compare the performance of two random mesh based designs, RUPF and RP, with different peering degrees. We use 100 Planetlab nodes for the evaluation. The streaming rate is 640 kbps and the server bandwidth is 1 Mbps.

We start with homogeneous random peering topology where each peer has the same number of randomly selected neighbors. First we compare the streaming performance of RUPF scheduling algorithm at different peering degrees. Figure 9(a) shows the cumulative distribution of all peer chunk miss ratio under different peering degrees. As the peering degree increases, the chunk miss ratio drops. When each peer only connects to 6 neighbors, the average chunk miss ratio is around 20%. Chunk miss ratio drops to 5% when peering degree increases to 10. At peering degree 14, the chunk miss ratio drops to 3%. This is because that with more neighbors a peer has a better chance to locate and download missing chunks in its download window. We can also observe that as the peer degree increases, the improvement slows down. The miss ratio improvement from peering degree 14 to 18 is less than the improvement from 6 to 10. Similar trends can be observed as that in RP. We can conclude that *increasing peering degree can reduce chunk miss ratio on peers.*

Even in mesh-based P2P streaming systems, the delivery paths of a single chunk still form a tree. As peer degree increases, a peer can upload one chunk to more neighboring peers, which in turn increases the fan-out of the chunk delivery tree. Therefore, the average number of hops that a chunk traverses to reach all peers decreases. Consequently, peers can download chunks faster and the chunk miss ratio becomes smaller. To verify the path length of each chunk, we append a hop counter to each chunk, which records how many hops



(a) Miss ratio & Residual degree (b) Different initial degree

Fig. 10. Chunk miss ratio of distorted network caused by peer churn. large peer degree can increase the resilience to peer churn.

a chunk has traversed. Upon receiving a chunk, each peer increments the chunk's hop counter and forward it to its descendants in the chunk delivery tree. Figure 9(b) depicts the distribution of the average hop count of randomly sampled chunks received by all peers. When all peers connect to 6 neighbors, each chunk needs to traverse in average 6 hops from server. As peer degree increases to 14, the average path length drops down to below 4.

Other than achieving good streaming performance, such as small chunk missing ratio and short chunk delay, how to make P2P streaming systems robust against peer churn is another major design consideration. In the rest of this section, we investigate the impact of peering degree on the resilience of P2P streaming systems. In this experiment, all 100 nodes join the system at the beginning and each peer has homogenous number of neighbors. After the system enters its steady state, we then create peer churn events by removing a certain percentage of peers from the system in a batch. To evaluate the resilience of random mesh topologies at different degrees, we sample the performance of the system after each batch peer removal without implementing any churn recovery mechanisms. In practice, a peer losing neighbors can obtain more neighbors through directory service such as tracker. The performance shown here is hence the worse case scenario and represents the performance peers may experience during the transitional period. After each peer removal, random peers in the system are sampled to evaluate their playback performance. Figure 10(a) shows the average chunk miss ratio and residual peer degree of the selected peers when different percentage of peers leaves the system. At the beginning of the experiment, every peer connects to 14 neighbors to construct the mesh network. When 10% peers leave, the average chunk miss ratio on the sampled peers is around 12% and the average number of connections of each peer drops to around 13. While 40% peers leave, the chunk miss ratio increases to 20% and the average residual peer number drops below 9. As larger percentage of peers leave, the remaining peers have less neighbors and the performance becomes worse.

Large peering degree helps in improving system robustness in the face of peer churn. We compare the resilience of the system when the initial peering degree is set to 10 and 18, respectively. Figure 10(b) illustrates the chunk miss ratios. The system with initial peer degree 18 has better resilience than others. Even when 50% peers leave, it can still has around 15% chunk loss, while the ratio of that with initial peer degree 10 is nearly twice.

VI. Conclusions and Future Work

P2P streaming has recently attracted lots of attentions from academia and industry. Numerous conceptual and practical P2P streaming designs have been proposed and deployed. In this paper, we conducted an extensive comparison study for five representative designs. Rich numerical results obtained from experiments on the PlanetLab allow us to quantitatively investigate the sensitivity and insensitivity of P2P streaming performance to different design mechanisms. Our study unveils several key factors contributing most to the successes of simple P2P streaming designs on the current Internet. More importantly, our study sheds lights on the design of the future generation of P2P streaming systems that achieve a good balance between performance and complexity.

Due to the size of the current PlanetLab and the heavy usage from the research community, the scale of our experiments is confined. Nevertheless, many findings from our current experiments are consistent with results obtained from measurement studies of various commercial systems with thousands of real users. In the future work, we are interested in exploring different ways to extrapolate our results to draw more general conclusions on P2P systems with different sizes.

References

[1] Youtube, "Youtube Homepage," http://www.youtube.com.
[2] Y.-H. Chu, S. G.Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM SIGMETRICS*, 2000.
[3] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "DONet/CoolStreaming: A data-driven overlay network for live media streaming," in *Proceedings of IEEE INFOCOM*, 2005.
[4] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *Proceedings of IEEE INFOCOM*, 2007.
[5] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," in *Proceedings of IEEE INFOCOM*, 2007.
[6] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proceedings of IEEE INFOCOM*, 2007.
[7] Y. Liu, "On the Minimum Delay Peer-to-Peer Video Streaming: how realtime can it be?" in *Proceedings of ACM Multimedia*, 2007.
[8] PPLive, "PPLive Homepage," http://www.pplive.com.
[9] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, November 2007.
[10] X. Hei, Y. Liu, and K. Ross, "Inferring Network-Wide Quality in P2P Live Streaming Systems," *IEEE Journal on Selected Areas in Communications, the special issue on advances in P2P streaming*, 2008.
[11] PlanetLab, "PlanetLab Homepage," http://www.planet-lab.org.
[12] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *IPTPS*, 2005.
[13] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming," in *Proceedings of IEEE INFOCOM*, 2007.
[14] Y. Guo, C. Liang, and Y. Liu, "AQCS: Adaptive Queue-based Chunk Scheduling for P2P Live Streaming," in *Proceedings of IFIP Networking*, 2008.
[15] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the Power of Pull-based Streaming Protocol: Can We Do Better?" *IEEE Journal on Selected Areas in Communications*, 2007.
[16] Trickle, "Trickle Homepage," http://monkey.org/~marius/pages/?page=trickle.