Adaptive Queue-based Chunk Scheduling for P2P Live Streaming

Yang Guo
2 Independence Way
Thomson Lab
Princeton, NJ 08540
Email:Yang.Guo@thomson.net

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

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

***Abstract*— P2P streaming has been popular and is expected to attract even more users. The challenges for P2P streaming have been on its scalability and video viewing quality. Both require efficient utilization of resources in P2P networks. This paper proposes an adaptive queue-based chunk scheduling method. The proposed scheme can achieve high bandwidth utilization and optimal streaming rate possible in a P2P streaming system. The prototype implementing the queue-based scheduling is developed and used to evaluate the scheme in the real network. The experiment results show the queue-based chunk scheduling method is capable of achieving the streaming rate close to the optimum, and adapting to the peer churns and underlying network dynamics nicely.**

I. Introduction

Video-over-IP applications have recently attracted a large number of users on the Internet. In 2006, the number of video streams served increased 38.8% to 24.92 billion even without counting the user generated videos [1]. Youtube [2] alone hosted some 45 terabytes of videos and attracted 1.73 billion views by the end of August 2006. With the fast deployment of high-speed residential access, such as Fiber-To-The-Home, video traffic is expected to dominate the Internet in near future. Traditionally, video content can be streamed to end users either directly from video source servers or indirectly from servers in Content Delivery Networks (CDNs). Peer-to-Peer video streaming has emerged as an alternative with low server infrastructure cost. P2P video streaming systems, such as ESM [3], CoolStreaming [4], PPLive [5], and SopCast [6], have attracted millions of users to watch live or on-demand video programs on the Internet [7].

The P2P design philosophy seeks to utilize peers' upload bandwidth to reduce servers' workload. However, the upload bandwidth utilization might be throttled by the so called *content bottleneck* where a peer may not have any content that can be uploaded to its neighbors even if its link is idle. The mechanism adopted by P2P file sharing applications is to increase the content diversity among peers. For example, the *rarest-first policy* of BitTorrent [8] encourages peers to retrieve the chunks with the lowest availability among their neighbors. Network coding has also been explored to mitigate the content bottleneck problem [9]. The content bottleneck problem in live streaming is even more severe. Video content in live streaming has strict playback deadlines. Even temporary decrease in peer bandwidth utilization leads to peer playback quality degradation, such as video playback freezing or skipping. To make things worse, at any given moment, peers are only interested in downloading a small set of chunks falling into the current playback window. This greatly increases the possibility of content bottleneck. One way to address this problem is to compromise user viewing quality. For example, a lower video playback rate would impose lower peer bandwidth utilization requirement. Allowing a longer playback delay also allows a larger set of chunks to be exchanged among peers. The opposite solution lies in designing more efficient peering strategies and chunk scheduling methods.

This paper deals with the second solution. Our focus is on the design of a chunk scheduling method that can achieve high peer bandwidth utilization. We assume collaborative P2P systems. Peers help each other and forward received video chunks to other peers. Motivated by the effectiveness of buffer control on switches and Active Queue Management (AQM) on routers, we propose a novel queue-based chunk scheduling algorithm to adaptively eliminate content bottlenecks in P2P streaming. Using queue-based signaling between peers and the content source server, the amount of workload assigned to a peer is proportional to its available upload capacity, which leads to high bandwidth utilization. The queue-based signaling also enables the proposed scheme to adapt to the changing network environment. Our contributions are three-fold:

1) We propose a simple queue-based chunk scheduling method achieving high bandwidth utilization in P2P live streaming. We theoretically show that the proposed scheme can achieve full bandwidth utilization in idealized network environments. A practical algorithm is designed to achieve a close-to-optimum performance in realistic network environment.
2) For distributed implementation, various design considerations are explored to handle dynamics in realistic network environments, including peer churns, peer bandwidth variations, and inside network congestion. A full-feature prototype is developed to test the feasibility and efficiency of the proposed scheduling algorithm.
3) The performance of the prototype system is examined

through a series of carefully designed experiments over PlanetLab [10]. Both the optimality and the adaptiveness of the proposed chunk scheduling method are demonstrated.

A. Related Work

The authors in [11] derived the upper bound for the streaming rate of a P2P live streaming system. They also developed a centralized solution that can fully utilize peer uploading bandwidth and achieve the streaming rate upper bound. The centralized solution collects all peers' upload capacity information, and computes the sub-stream rates sent from the server to peers. In practice, available upload capacity varies over time and peers join and leave the system. The central coordinator needs to continuously monitor peers' upload capacity and re-compute the sub-stream rate to individuals. The proposed queue-based chunk scheduling method is a decentralized version of the above. Peers only exchange information with other peers/server and make local decision. Global peer upload capacity information does not need to be collected, and the scheme adapts to the changing peer membership and network environment nicely.

There have been ongoing efforts intending to improve resource utilization in P2P live streaming. The study in [12] shows the mesh-based scheme can better utilize peers' upload capacity than tree-based scheme, thanks to the dynamic mapping of content to the delivery paths. To improve the resource utilization in mesh-based P2P streaming, [13] proposes a two-phase swarming scheme where the fresh content is quickly diffused to the entire system in the first phase, and peers exchange available content in the second phase. Network coding is also applied to P2P live streaming. [14] performs a reality check by using network coding for P2P live streaming. However, neither approach can fully utilize the resources and achieve the maximum streaming rate. The authors in [15] give a randomized distributed algorithm that can converge to the maximum streaming rate. They also study the delay that users must endure in order to play the stream with a small amount of missing data. The queue-based chunk scheduling method is a deterministic distributed algorithm. No data chunk need to be skipped to achieve a small playback startup delay.

One shortcoming with the queue-based chunk scheduling method is it requires a fully connected topology among the server and all peers, and therefore is not practical. In [16], we proposed a clustered approach to achieve close to full bandwidth utilization. Peers are organized into clusters based on their capacities and locations. Within a cluster, the queue-based chunk scheduling method can be used. Thus the aforementioned problem is mitigated.

The remaining part of this paper is organized as follows. The queuing model and scheduling algorithm of queue-based chunk scheduling are described in Section II. Implementation considerations are explored in Section III. The experiment results are reported in Section IV. Finally, Section V ends the paper with concluding remarks.

II. Adaptive queue-based chunk scheduling

The capability to achieve high streaming rate is desirable for P2P streaming. Higher streaming rate allows the system to broadcast video with better quality. It also provides more cusion to absorb the bandwidth variations caused by peer churn and network congestions when constant-bit-rate (CBR) video is broadcasted. The key to achieve high streaming rate is to better utilize peers' uploading bandwidth.

In this section, we propose a queue-based chunk scheduling algorithm that can achieve close to 100% peers' uploading bandwidth utilization in practical P2P networking environment. In P2P system, the resource utilization is determined by the overlay topology and collective behavior of chunk scheduling at individual peers. At system level, queue-based adaptive chunk scheduling requires fully connected mesh among participating peers. At peer level, data chunks are pulled/pushed from server to peers, cached at peers' queue, and relayed from peers to its neighbors. The availability of upload capacity is inferred from the queue status such as the queue size or if the queue is empty. Signals are passed between peers and server to convey the information if a peer's upload capacity is available.

Fig. 1 depicts a P2P streaming system using queue-based chunk scheduling with one source server and three peers. Each peer mainstains several queues including a forward queue. Using peer a as an example, the signal and data flow is described next. Pull signals are sent from peers a to the server whenever the queues become empty (or have fallen below a threshold) (step 1 in Fig. 1). The server responds to the pull signal by sending three data chunks back to peer a (step 2). These chunks will be stored in the forward queue (step 3) and be relayed to peer b and peer c (step 4). When the server has responded to all 'pull' signals on its 'pull' signal queue, it serves one duplicated data chunks to all peers (step 5). These data chunks will not be stored in forward queue and will not be relayed further.

Next we first describe in detail the queue-based scheduling mechanism at the source server and peers. The optimality of the scheme is shown afterwards.

A. Peer side scheduling and its queuing model

Fig. 2 depicts the queuing model for peers in the queue-based scheduling method. A peer maintains a playback buffer that stores all received streaming content from the source server and other peers. The received content from different nodes is assembled in the playback buffer in playback order. The peer's media player renders/displays the content from this buffer. Meanwhile, the peer maintains a forwarding queue which is used to forward content to all other peers. The received content is partitioned into two classes: F-marked content and NF-marked content. *F (forwarding)* represents content that should be relayed/forwarded to other peers. *NF (non-forwarding)* indicates that content is intended for this peer only and no forwarding is required. The content forwarded by neighbor peers is always marked as *NF*. The content received

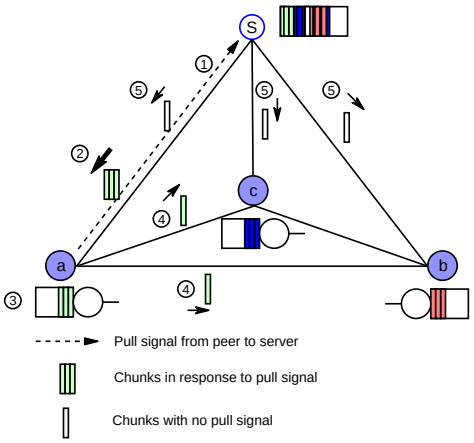


Fig. 1. Queue-based P2P system with four nodes. 1. peer a sends pull signal to the content source server; 2. content source server send three chunks in response to the pull signal; 3. three chunks are cached in the forward queue; 4. cached chunks are forwarded to neighbor peers; 5. duplicate chunk is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

from the source server can be marked either as *F* or as *NF*. *NF* content is filtered out. *F* content is stored into the forward queue, marked as NF content, and forwarded to other peers. In order to fully utilize a peer's upload capacity, the peer's forwarding queue should be kept busy. A signal is sent to the source server to request more content whenever the forwarding queue becomes empty. This is termed a 'pull' signal. The rules for marking the content at the content source server are described next.

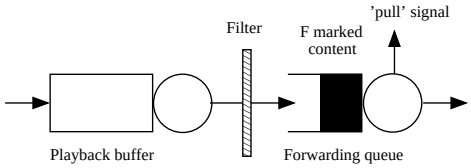


Fig. 2. Queue Model of Peers

B. Server side scheduling algorithm and its queuing model

Fig. 3 illustrates the server-side queuing model of the decentralized method. The source server has two queues: a content queue and a signal queue. The content queue is a multi-server queue with two dispatchers: an F-marked content dispatcher and a forward dispatcher. The dispatcher that is invoked depends on the control/status of the 'pull' signal queue. Specifically, if there is 'pull' signal in the signal queue, a small chunk of content is taken from the content buffer. This chunk of content is marked as *F* and dispatched by the F-marked content dispatcher to the peer that issued the 'pull' signal. The 'pull' signal is then removed from the 'pull' signal queue. If the signal queue is empty, the server takes a small chunk of content from the content buffer and puts that chunk of content into the forwarding queue to be dispatched. The forwarding dispatcher marks the chunk as *NF* and sends it to all peers in the system.

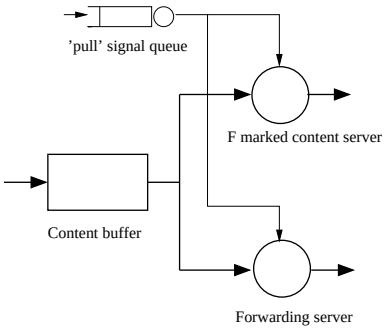


Fig. 3. Queue Model of Source Server

C. Proof of optimality for queue-based chunk scheduling

Next, we show that the queue-based scheduling method for both the peer-side and the server-side achieves the maximum P2P live streaming rate of the system. Given a content source server and a set of peers with known upload capacities, the maximum streaming rate, r^{max} , is governed by the following formula [11]:

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

where u_s is content source server's upload capacity, u_i is peer i's upload capacity, and there are n peers in the system. The second term on the right-hand side of equation, $\frac{u_s+\sum_{i=1}^{n} u_i}{n}$, is the average upload capacity per peer. More specifically, the maximum streaming rate is $r^{max} = u_s$ if $u_s \leq \frac{u_s+\sum_{i=1}^{n} u_i}{n}$, and $r^{max} = \frac{u_s+\sum_{i=1}^{n} u_i}{n}$ if $u_s > \frac{u_s+\sum_{i=1}^{n} u_i}{n}$. The first case is termed as *server resource poor scenario* where the server's upload capacity is the bottleneck. The second case is termed as *server resource rich scenario* where the peers' average upload capacity is the bottleneck.

Theorem 2.1: Assume that the signal propagation delay between a peer and the server is negligible and the data content can be transmitted at an arbitrary small amount, then the queue-based decentralized scheduling algorithm as described above achieves the maximum streaming rate possible in the system.

Proof: Suppose the video content is divided into small chunks. The server sends out *one* chunk each time it serves a 'pull' signal. A peer issues a pull signal to the server whenever the forwarding queue becomes empty. δ denotes the chunk size.

For peer i, $i = 1, 2, \ldots, n$, it takes time of $(n-1)\delta/u_i$ to forward one data chunk to all peers. Let r_i be the maximum rate at which the 'pull' signal is issued from peer i. Hence $r_i = u_i/(n-1)\delta$.

The maximum aggregated rate of 'pull' signal received at server, r, is $r = \sum_{i=1}^{n} r_i = \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$. It takes server δ/u_s to serve a pull signal. Hence the maximum 'pull' signal rate a

server can accommodate is u_s/δ. Now consider the following two scenarios/cases:

- Case 1: $u_s/\delta \leq \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$

In this scenario, the server cannot handle the 'pull' signal at maximum rate. The signal queue at the server side is hence never empty and the entire server bandwidth is used to transmit F-marked content to peers. In contrast, a peer's forward queue becomes idle while waiting for the new data content from the source server. Since each peer has sufficient upload bandwidth to relay the F-marked content (received from the server) to all other peers, the peers receive content sent out by the server at the maximum rate.

The supportable streaming rate is equal to the server's upload capacity. The condition $u_s/\delta \leq \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$ is equivalent to $u_s \leq \frac{u_s+\sum_{i=1}^{n} u_i}{n}$, i.e., the server resource poor scenario in Equation (1). Hence the streaming rate is consistent with the Equation (1) and the maximum streaming rate is reached.

- Case 2: $u_s/\delta > \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$

In this scenario, the server has the upload capacity to service the 'pull' signals at the maximum rate. During the time period when the 'pull' signal queue is empty, the server transmits duplicate NF-marked content to all peers. The amount of upload capacity used to serve F-marked content is $r\delta = \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}\delta = \frac{\sum_{i=1}^{n} u_i}{n-1}$.

The server's upload bandwidth used to serve NF-marked content is therefore $u_s - \frac{\sum_{i=1}^{n} u_i}{n-1}$. For each individual peers, the rate of receiving NF-marked content from server is $(u_s - \frac{\sum_{i=1}^{n} u_i}{n-1})/n$ since there are n peers in the system. The streaming rate at peers is:

$$\frac{\sum_{i=1}^{n} u_i}{n-1} + (u_s - \frac{\sum_{i=1}^{n} u_i}{n-1})/n = \frac{u_s + \sum_{i=1}^{n} u_i}{n}. \quad (2)$$

The condition $u_s/\delta > \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$ is equivalent to $u_s > \frac{u_s+\sum_{i=1}^{n} u_i}{n}$, i.e., the scenario in which the server is resource rich described above. Again, the streaming rate reaches the upper bound as indicated in Equation (1). This concludes the proof. ∎

Note that in case 2 where the aggregate 'pull' signal arrival rate is smaller than the server's service rate, it is assumed that the peers receive F-marked content immediately after issuing the 'pull' signal. The above assumption is true only if the 'pull' signal does not encounter any queuing delay and can be serviced immediately by the content source server. This means that (i) no two 'pull' signals arrive at the exact same time and (ii) a 'pull' signal can be serviced before the arrival of next incoming 'pull' signal. Assumption (i) is commonly used in queuing theory and is reasonable since a P2P system is a distributed system with respect to peers generating 'pull' signals. The probability that two 'pull' signals arrive at exactly the same time is low. Assumption (ii) means that the data can be transmitted in arbitrary small amounts, i.e., the size of data chunk, δ, can be arbitrarily small. In practice, the size of data chunks is limited in order to reduce the overhead associated with data transfers. Below we discuss the implementation considerations in realizing the above scheme in practice.

III. Implementation considerations

Implementation considerations in realizing the above scheme in practice are now discussed. The architecture of content source server and peers using the queue-based data chunk scheduling are now described with an eye toward practical implementation considerations including the impact of chunk size, network congestion, and peer churn.

A. Impact of chunk size and propagation delay

In the optimality proof, it was assumed that the chunk size could be arbitrarily small and the propagation delay was negligible. In practice, the chunk size is on the order of kilobytes to avoid excessive transmission overhead caused by protocol headers. The propagation delay is on the order of tens to hundreds of milliseconds. Hence, it is necessary to adjust the timing of issuing 'pull' signals by the peers and increase the number of F-marked chunks served at the content source server to allow the decentralized scheduling method to achieve close to the optimal live streaming rate.

At the server side, K F-marked chunks are transmitted as a batch in response to a 'pull' signal from a requesting peer (via the F-marked content queue). A larger value of K would reduce the 'pull' signal frequency and thus reduce the signaling overhead. This, however, increases peers' threshold to be shown in Equation (3). When the 'pull' signal queue is empty, the server's forwarding queue forwards one chunk at a time to all peers in the system. The arrival of a new 'pull' signal preempts the forwarding queue activity and the F-marked content queue services K chunks immediately.

The peer sets a threshold of T_i for the forwarding queue. The 'pull' signal is issued when the number of chunks of content in the queue is less than or equal to T_i. It takes at least twice the propagation delay to retrieve the F-marked content from the server. Issuing the 'pull' signals before forwarding queues become entirely empty avoids wasting the upload capacities.

How to set the value of T_i properly is considered next. The time to empty the forwarding queue with T_i chunks is $t_i^{empty} = (n-1)T_i\delta/u_i$. Meanwhile, it takes $t_i^{receive} = 2t_{si} + K\delta/u_s + t_q$ for peer i to receive K chunks after it issues a pull signal. Here t_{si} is the propagation delay between the source server and peer i, $K\delta/u_s$ is the time required for server to transmit K chunks, and t_q is queuing delay seen by the 'pull' signal at the server pull signal queue. In order to receive the chunks before the forwarding queue becomes fully drained, $t_i^{empty} \geq t_i^{receive}$. This leads to:

$$T_i \geq \frac{(2t_{si} + K\delta/u_s + t_q)u_i}{(n-1)\delta}. \quad (3)$$

All quantities are known except t_q, the queuing delay incurred at the server side signal queue. In server resource poor scenario where the source server is the bottleneck, the selection of T_i would not affect the streaming rate as long as the server is always busy. In server resource rich scenario, since the service rate of signal queue is faster than the pull signal rate, t_q is very small. So we let t_q be zero, i.e.,

$$T_i \geq \frac{(2t_{si} + K\delta/u_s)u_i}{(n-1)\delta}. \quad (4)$$

The peers' startup delay is computed next. τ denotes the startup delay. Given a peer has a full queue with T_i number of marked chunks, it takes $T_i\delta(n-1)/u_i$ to empty the queue. Hence the startup delay is:

$$\tau = \max_i\{T_i\delta(n-1)/u_i\}. \quad (5)$$

B. Source server/peer architecture

The content source server responds to the 'pull' signals from peers and pushes NF-marked content proactively to peers. The content source server is also the bootstrap node. As the bootstrap node, the content source server manages peer information (such as peer id, IP address, port number, etc.) and replies to the request for peer list from incoming new peers.

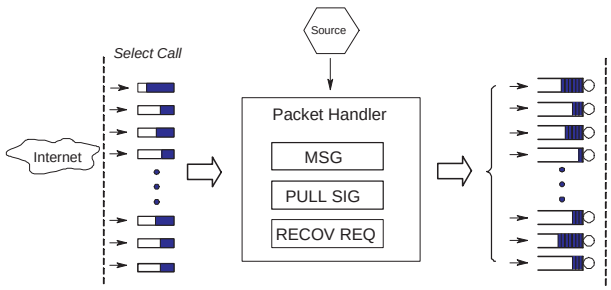


Fig. 4. Server Architecture

Figure 4 illustrates the architecture of the source server. In the queue-based adaptive P2P live streaming, the server and all peers are fully connected with full-duplex TCP connections. Using the 'select call' mechanism to monitor the connections with peers, the server maintains a set of input buffers to store received data. There are three types of incoming messages: management message, 'pull' signal, and missing chunk recovery request. Correspondingly three independent queues are formed for the messages respectively. If the output of handling these messages needs to be transmitted to remote peers, the output is put on the per-peer out-unit to be sent.

There is one out-unit for each destination peer to handle the data transmission process. Figure 5 depicts an exemplary out-unit that has four queues for a given peer: management message queue, F-marked content queue, NF-marked content queue, and missing chunk recovery queue. The management message queue stores responses to management requests. An example of a management request is when a new peer has just

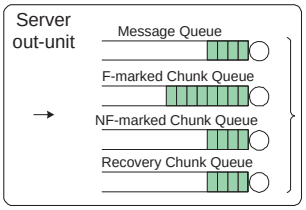


Fig. 5. Server Out-unit Queues

joined the P2P system and requests the peer list. The server would respond by returning the peer list. The F/NF marked content queue stores the F/NF marked content intended for this peer. Finally, chunk recovery queue stores the missing chunks requested by the peer.

Different queues are used for different types of traffic in order to prioritize the traffic types. Specifically, management messages have the highest priority, followed by F-marked content, and NF-marked content. The priority of recovery chunks can be adjusted based on the design requirement. Management messages have the highest priority because it is important for the system to run smoothly. For instance, by giving management messages the highest priority the delay for a new peer to join the system is shortened. When a new peer issues a request to the content source server to join the P2P system, the peer list can be sent to the new/joining peer quickly. Also, management messages are typically small in size compared to content messages. Giving higher priority to management message reduces overall average delay. The content source server replies to each 'pull' signal with K F-marked chunks. F-marked chunks are further relayed to other peers by the receiving peer. The content source server sends out a NF-marked chunk to all peers when the 'pull' signal queue is empty. NF-marked chunks are used by the destination peer only and will not be relayed further. Therefore, serving F-marked chunk promptly improves the utilization of peers' upload capacity and increases the overall P2P system live streaming rate. Locating and serving recovery chunks should be a higher priority than NF-marked chunk delivery since missing chunk affects the viewing quality significantly. If the priority of forwarding recovery chunks is set to be higher than that of F-marked chunks, viewing quality gets preferential treatment over system efficiency. In contrast, if F-marked chunks receive higher priority, the system efficiency is given higher priority. The priority scheme selected depends on the system design goal.

Another reason for using separate queues is to deal with bandwidth fluctuation and congestion inside the network. Many P2P researchers assume that server/peer's upload capacity is the bottleneck. In our experiments over PlanetLab, it has been observed that some peers may slow down significantly due to congestion. If all the peers share the same queue, the uploading to the slowest peer will block the uploading to remaining peers. The server's upload bandwidth will be wasted. This is similar to the head-of-line blocking problem

in input-queued switch design: an input queue will be blocked by a packet destined for a congested output port. The switching problem was solved by placing packets destined to different output ports in different virtual output queues. A similar solution is adopted by using separate queues for different peers. Separate queues avoid inefficient blocking caused by slow peers. Separate queues allow more accurate estimation of the amount of queued content, too. This is important for peers to determine when to issue 'pull' signals.

Fig. 6 depicts the architecture of a peer. The architecture of a peer in the P2P system described herein is similar to that of the content source server. The server and all peers are fully connected with full duplex TCP connections. A peer stores the received chunks into the playback buffer. The management messages from server (e.g., the peer list) or other peers (missing chunk recovery message) are stored in management message queue. The chunk process module filters out NF-marked chunks. F-marked chunks are duplicated into the out-units of all other peers.

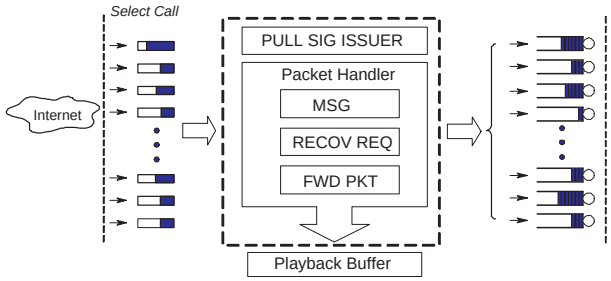


Fig. 6. Peer Architecture

Fig. 7 depicts the structure of peer side out-unit. It has three queues: management message queue, forward queue, and recovery chunk queue. Chunks in the forward queue will be marked as NF and will not be relayed further at receiving peers. 'Pull' signal issuer monitors the out-units. It employs the queue threshold as defined in Equation (3) to decide when to issue 'pull' signals to the content source server. When calculating the 'pull' signal threshold in Equation (3), the underlying assumption is that remote peers are served in a round-robin fashion using a single queue. Here a one-queue-per-peer design is used to avoid head-of-line blocking problem. The average of the forward queue size is used in Equation (3). If a peer always experiences a slow connection, some chunks may be forced to be dropped. Peers have to use missing chunk recovery mechanism to recover from the loss.

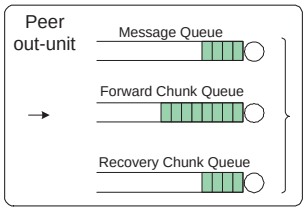


Fig. 7. Peer Out-unit Queues

C. Missing chunk recovery

Peer churn and network congestion may cause chunk losses. Sudden peer departure, such as node or connection failure, leaves the system no time to reschedule the chunks still buffered in the peer's out-unit. In case the network routes are congested to some destinations, the chunks waiting to be transmitted may overflow the queue in the out-unit, which leads to chunk losses at the receiving end. The missing chunk recovery scheme enables the peers to recover the missing chunks to avoid viewing quality degradation.

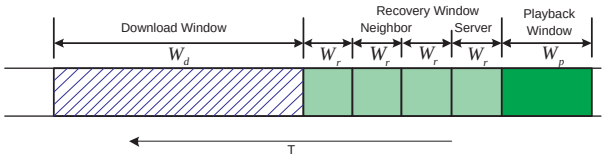


Fig. 8. Missing Chunk Recovery

Each peer maintains a playback buffer to store the video chunks received from the server and other peers. The playback buffer maintains three windows: playback window, recovery window, and download window. W_p, W_r and W_d denote the size (in terms of number of chunks) of playback window, recovery window, and download window, respectively. The media player renders/displays the content from the playback window. Missing chunks in the recovery window are recovered using the method described below. Finally, the chunks in the downloading window are pulled and pushed among the server and the other peers. The size of download window, W_d, can be estimated as follows:

$$W_d = \sum_{i=1}^{n} T_i + \left(u_s - \frac{\sum_{i=1}^{n}}{n-1}\right)^{+} \tau/\delta. \qquad (6)$$

where τ is the startup delay, and $(\cdot)^+$ is the non-negative function that takes value of zero if the input is negative. The first term in Equation (6) is the sum of all F-marked chunks cached at all peers. The second term is the number of NF-marked chunks sent out by the server.

Heuristics are employed to recover the missing chunks. If peers leave gracefully, the server is notified and the F-marked chunks waiting in the out-unit will be assigned to other peers. The missing chunks falling into the recovery window are recovered as follows. First, the recovery window is further divided into four sub-windows. Peers send the chunk recovery messages to the source server directly if the missing chunks are in the window closest in time to the playback window. These chunks are urgently needed otherwise the content quality will be impaired. An attempt is made to recover the missing chunks in the other three sub-windows from other peers. A peer randomly selects three recovery peers from the peer list, and associates one with each sub-window. The peer needs recovery chunks sends chunk recovery messages to the corresponding recovery peers. By randomly selecting a recovery peer, the recovery workload is evenly distributed among all peers.

IV. Experiment results

In this section, we examine the performance of queue-based chunk scheduling method via experiments. Experiments are conducted over PlanetLab [10]. $40+$ nodes (one content source server, one public sink and 40 users/peers) are used with most of them located in North America. All connections between nodes are TCP connections. TCP connections avoid network layer data losses, and allow us to use software package Trickle [17] to set a node's upload capacity. In our experiments, we observe the obtained upload bandwidth is slightly larger ($< 8\%$) than the value we set using Trickle. To account for this error, we measure the actual upload bandwidth, and use the measured rate for plotting the graphes. The upload capacity of peers are assigned randomly according to the distribution obtained from the measurement study conducted in [18], as listed in Table I. The largest uplink speed is reduced from

TABLE I
Bandwidth Distribution

Uplink(kbps)	Fraction of nodes
128	0.2
384	0.4
1000	0.25
4000	0.15

5000 kbps to 4000 kbps. This ensures that PlanetLab nodes have sufficient bandwidth to support the targeted rate.

A. Optimality Evaluation

The optimality of the queue-based scheduling method is evaluated first. All 40 peers join the system at the beginning of the experiment, and stay for the entire duration of the experiment. The content source server's upload capacity is varied from 320 kbps to 5.6 Mbps. For each server upload capacity setting, we run an experiment for 5 mins. The achieved streaming rate is collected every 10 seconds and the average value is reported at the end of each experiment. Fig. 9 shows the achieved streaming rate vs. the optimal

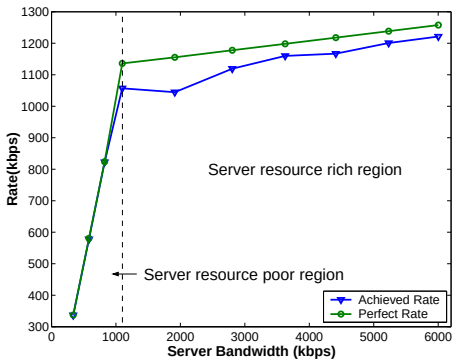


Fig. 9. Achieved Rate vs. Optimal Rate

rate with different server bandwidths. The difference never exceeds 10% of the optimal rate possible in the system. The curves exhibit two segments with turning point at around 1.1 Mbps. According to Equation (1), the server bandwidth is the bottleneck when it is smaller than 1.1 Mbps (server resource poor scenario). The streaming rate is equal to the source rate. As the server bandwidth becomes greater than 1.1 Mbps, the peers' average upload capacity becomes the bottleneck (server resource rich scenario). The streaming rate still increases linearly, however, with a smaller slope. This is evident in Fig. 10. We plot in Fig. 10 the numbers of F-marked and NF-marked chunks sent out by the source server. When the server bandwidth is 560 kbps, very few NF-marked chunks are transmitted (Note: in theory, no NF-marked chunks should be sent in this scenario. We do see several NF-marked chunks. We believe this is caused by the bandwidth variation in the network. The variation occasionally causes the server's pull signal queue becomes empty). In contrast, more and more NF-marked chunks are sent by the server as its uplink capacity increases beyond 1.1 Mbps. Another observation is that the queue based chunk scheduling performs better in the server resource poor scenario than in the server resource rich scenario. In the server resource poor scenario, the server sends out F-marked chunks exclusively. As long as the pull signal queue is not empty, the optimal streaming rate can be achieved. In the server resource rich scenario, the server sends out both F-marked and NF-marked chunks. If F-marked chunks are delayed at server or along the route from the server to peers, peers can not receive F-marked chunks promptly. Peers' forward queues become idle and upload bandwidth cannot be fully utilized. Furthermore, in the server resource rich scenario, F-marked chunk queues and NF-marked chunk queues compete for the server bandwidth. Although higher priority is given to F-marked chunks at the application level, NF-marked chunks sometimes have already been stored into the kernel level TCP buffer. This also slows down the F-marked chunk delivery.

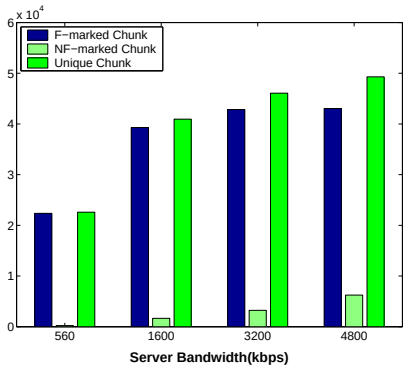


Fig. 10. Distribution of Chunks from Server

B. Adaptiveness to Peer Churn

Peer churn has been identified as one of the major disruptions to the performance of p2p system. Fig. 11 depicts how queue based chunk scheduling method performs in face of peer churns. In this 10 minutes experiment, the server bandwidth is set to be 2.4 Mbps. Three peers with the bandwidth of

4 Mbps are selected to leave the system at time of 200 seconds, 250 seconds, and 300 seconds, respectively. Two peers rejoin the system at time of 400 seconds, and the third one rejoins the system at time of 450 seconds. Fig. 11 depicts

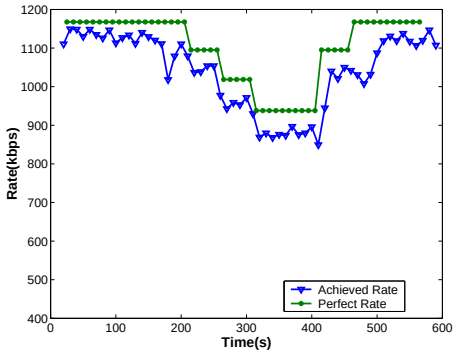


Fig. 11. Streaming Rate under Peer Churn

the achieved rate vs. optimal rate every 10 seconds. Although the departure and the join of a peer does introduce disruptions to the achieved streaming rate, overall the achieved streaming rate tracks the optimal rate closely. The difference between them never exceeds 12% of the optimal rate.

C. Adaptiveness to Network Bandwidth Variations

In addition to peer churn, the network bandwidth varies over time due to cross traffic. To test the system's adaptiveness to network bandwidth variations, the following experiment is conducted. We set up a sink on a separate PlanetLab node not participating in P2P streaming. One peer in the streaming system with upload capacity of 4 Mbps is selected to establish multiple parallel TCP connections to the sink. Each TCP connection sends out garbage data to the sink. The noise traffic generated by those TCP connections causes variations in the bandwidth available for the P2P video threads on the selected peer. We control the timing of the noise traffic generation.

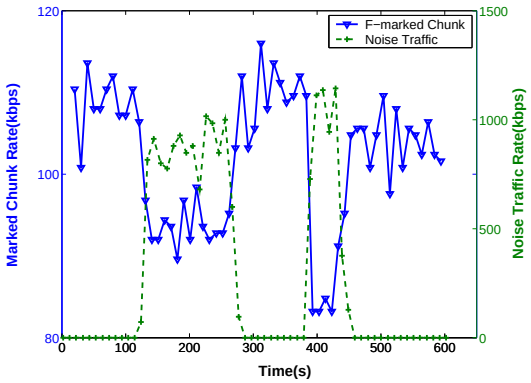


Fig. 12. Marked Chunk Receiving Rate with Background Traffic

Fig. 12 depicts the rate at which the F-marked chunks are received at the selected peer together with the sending rate of noise traffic. During time periods of (120 sec, 280 sec) and (380 sec, 450 sec), the noise traffic threads are on. Whenever the noise traffic is on, the queue-based chunk scheduling method adapts quickly to the changing available bandwidth by reducing its pull signal rate. Consequently, the server reduces the rate of F-marked chunks sent to the selected peer. When the noise traffic is turned off, the server sends more F-marked chunks to the selected peer to fully utilize its available uploading bandwidth. The self-adaptiveness of the queue-based chunk scheduling method makes the overall achieved streaming rate close to the optimal rate.

D. Supporting constant bit-rate (CBR) video

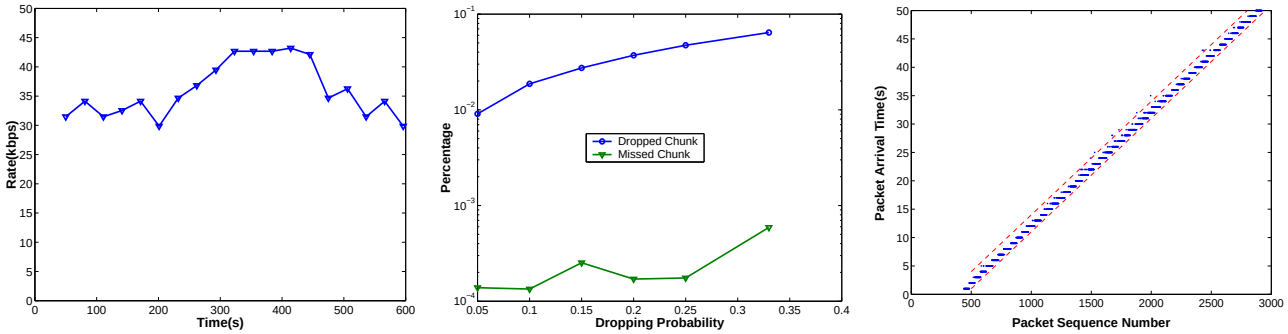
The performance of supporting CBR video is examined next. CBR video is widely used in online streaming. CBR video with the rate of 400 kbps offers reasonable video quality and is used by many p2p streaming systems. Hence we choose the video data rate of 400 kbps in the following experiments.

Although the queue-based chunk scheduling method aims to achieve high streaming rate, the system still benefits when supporting CBR video. The queue-based chunk scheduling method can tap into the redundant bandwidth resources whenever necessary, which enhances the system robustness against peer churns and network congestion. It also offers users small startup delays.

•**Performance with peer churns.** A peer's contribution is proportional to its upload bandwidth in queue-based chunk scheduling method. When peers join and depart the system, queue-based chunk scheduling is able to adapt to the change and adjust individual peer's contribution so that the constant streaming rate is maintained without disruptions. Fig.13(a) depicts the receiving F-marked chunk rate (from server) of a randomly selected peer. The churn pattern here is the same as that in Section IV-B: three peers leave the system at time of 200 seconds, 250 seconds, and 300 seconds, respectively. Two peers rejoin the system at time of 400 seconds, and the third one rejoins the system at time of 450 seconds. The selected peer receives around 32 kbps of F-marked chunk from server and relays them to all other peers. During the peer churn period, it boosts its receiving rate to 43 kbps, which compensates the resource loss due to the peer churn. When three peers rejoin the system, the receiving comes back the 32 kbps.

• **Missing chunk recovery.** The queue-based chunk scheduling is able to fully utilize all resources. For CBR videos, the system uses redundant resources to recover lost packets. To demonstrate the effectiveness of the missing chunk recovery mechanism, we intentionally inject packet losses into the system. Specifically, we select eight nodes according to the percentage in Table I. Whenever these nodes forward F-marked chunk to other peers, the chunk is dropped with a pre-defined dropping probability. Let missed chunk denote the chunk that is actually lost. Fig. 13(b) depicts the dropped chunk percentage and missed chunk percentage vs. dropping probability. Note that Y-axis is in log scale. Almost all chunks are recovered even as much as 7% of chunks are dropped.

• **Playback startup delay.** Finally we examine the startup delay, another important performance metric in p2p live



(a) Chunk Receiving Rate under Peer Churn (b) Ratio of Dropped and Missed Chunk (c) Chunk Sequence vs. Arrival Time

Fig. 13. Experiment Results with the Constant Bit-rate Video Source

streaming. Fig. 13(c) plots the chunk arrival time at a randomly selected node. Two linear lines sandwich chunk arrival times. The vertical time difference of these two lines are the startup delay. The startup delay for our experiment with 40 nodes and 50 kbps CBR video is 3 seconds. The slowest peer with upload bandwidth of 128 kbps is slowest in emptying the queue. With the server bandwidth of 2.4 Mbps, the forward queue threshold is less than one chunk according to Equation (3). Hence threshold is set to be one chunk. The chunk size is 1 KByte. Therefore the startup delay (according to Equation (5)) is $1 * 8 * 39/128 = 2.4seconds$ (128 kbps is the uplink bandwidth, 39 is the number of peers a peer needs to forward F-marked chunk to). The actually startup delay is little bit longer due to the network bandwidth variations and parallel queues in the out-units.

V. Conclusions

In this paper, we propose a simple queue-based chunk scheduling method that can achieve full bandwidth utilization in P2P live streaming. To study the effectiveness of the proposed scheme in practice, a prototype is developed and is used to conduct experiments over the real network. The results demonstrate the optimality and the adaptiveness of the proposed queue-based chunk scheduling method.

Future work can develop along several avenues. As the first attempt of applying queue management to P2P streaming, we used simple queue control schemes. We will explore queue control design space to further improve its performance. Secondly, we did not compare the performance of queue-based chunk scheduling with other existing methods. Although we feel confident that the queue-based chunk scheduling can outperform existing approaches due to its optimality, simplicity, and flexibility, it will be an interesting exercise to do the head-to-head comparisons. The third direction is to apply the queue-based chunk scheduling to hierarchically clustered P2P streaming framework [16]. The combination solves the scalability issue faced by the queue-based chunk scheduling method and will be a truly scalable and efficient P2P streaming solution. Our work demonstrated the effectiveness of application layer queue management in eliminating content bottlenecks in P2P live streaming. We are interested in applying this approach to other P2P content distribution applications such as file sharing and video-on-demand.

References

[1] "Accustream iMedia Research Homepage," 2007. `http://www.accustreamresearch.com`.

[2] Youtube, "Youtube Homepage." `http://www.youtube.com`.

[3] Y.-H. Chu, S. G.Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM SIGMETRICS*, 2000.

[4] 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.

[5] PPLive, "PPLive Homepage." `http://www.pplive.com`.

[6] SopCast, "SopCast Homepage." `http://www.sopcast.org`.

[7] 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.

[8] BT, "Bittorent Homepage." `http://www.bittorrent.com`.

[9] C. Gkantsidis and P. R. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proceedings of IEEE INFOCOM*, 2005.

[10] PlanetLab, "PlanetLab Homepage." `http://www.planet-lab.org`.

[11] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *Proceedings of IEEE INFOCOM*, 2007.

[12] 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.

[13] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming," in *Proceedings of IEEE INFOCOM*, 2007.

[14] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *Proceedings of IEEE INFOCOM*, 2007.

[15] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proceedings of IEEE INFOCOM*, 2007.

[16] C. Liang, Y. Guo, and Y. Liu, "Hierarchically clustered p2p streaming system," in *Proceedings of GLOBECOM*, 2007.

[17] Trickle, "Trickle Homepage." `http://monkey.org/~marius/pages/?page=trickle`.

[18] C. H. Ashwin R. Bharambe and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Network Performance Mechanisms," in *Proceedings of IEEE INFOCOM*, 2006.