AQCS: Adaptive Queue-based Chunk Scheduling for P2P Live Streaming

Yang Guo[1], Chao Liang[2], and Yong Liu[2]

[1] Corporate Research, Thomson, Princeton, NJ, USA 08540
Yang.Guo@thomson.net
[2] ECE Dept., Polytechnic University,Brooklyn, NY, USA 11201
cliang@photon.poly.edu,yongliu@poly.edu

Abstract. P2P streaming has been popular and is expected to attract even more users. One major challenge for P2P streaming is to offer users satisfactory Quality of Experience (QoE) in terms of video resolution, startup delay, and playback smoothness, all require efficient utilization of bandwidth resources in P2P networks. In this paper, we propose AQCS, adaptive queue-based chunk scheduling, that can support the maximum streaming rate allowed by a P2P streaming system with small signaling overhead and short startup delay. AQCS is a distributed algorithm with minimum requirement on peers. Queue-based design enables peers to be self-adaptive to the bandwidth variations and peer churn, and automatically converges to the optimal operating point. The prototype of AQCS is implemented and various implementation issues are examined. The experiments over the PlanetLab further demonstrate AQCS's optimality and its robustness against changing system/network environment.

1 Introduction

Video-over-IP applications have recently attracted a large number of users on the Internet. Youtube [1] 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 [2], CoolStreaming [3], and PPLive [4], have attracted millions of users to watch live or on-demand video programs on the Internet [5].

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 [6] 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 [7]. 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 support high resolution video streaming service. 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, AQCS, 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. The data chunks are relayed at most once by intermediate peers. Hence the video content can be disseminated to all peers quickly. The simplicity of the design also reduces the signaling overhead. Our contributions are three-fold:

- 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 support the optimal streaming rate in idealized network environments. A practical algorithm is designed to achieve the close-to-optimum performance in realistic network environment.
- A full-feature prototype is developed to test the feasibility and efficiency of the proposed scheduling algorithm. Various design considerations are explored to handle dynamics in realistic network environments, including peer churns, peer bandwidth variations, and inside network congestion.
- The performance of the prototype system is examined through experiments over PlanetLab [8]. Both the optimality and the adaptiveness of the proposed chunk scheduling method are demonstrated.

The remaining part of this paper is organized as follows. Background and related work is included in Section 2. The queuing model and scheduling algorithm of queue-based chunk scheduling are described in Section 3. Implementation considerations are explored in Section 4. The experiment results are reported in Section 5. Finally, Section 6 ends the paper with concluding remarks.

2 Background and Related Work

In [9], we propose HCPS - Hierarchically Clustered P2P Streaming system that can support the streaming rate approaching the optimum upper bound with short delay, yet

is simple enough to be implemented in practice. In HCPS, the peers are grouped into small size clusters and a hierarchy is formed among clusters to retrieve video data from the source server. By actively balancing the uploading capacities among clusters, and executing the perfect scheduling algorithm [10] within each cluster, the system resource can be efficiently utilized.

Fig. 1 depicts a two-level HCPS system. At the lower level, all peers are organized into bandwidth-balanced clusters. Each cluster consists of a cluster head and a small number, e.g. from 20 to 40 normal peers. Cluster heads of all clusters form a super cluster at upper level and retrieve video from the video source server in P2P fashion. After obtaining video at the upper level, a cluster head acts as a local video proxy server for normal peers in its cluster at lower level. Normal peers within the same cluster collaborate according to the perfect scheduling algorithm to retrieve video from their cluster head. In this new architecture, the server only distributes video to cluster heads; a normal peer only maintains connections to its neighbors in the same cluster; and a cluster head connects to both other cluster heads and normal peers in its own cluster. Two-level HCPS already has the ability to support a large number of peers with mild requirement on the number of connections on the server, cluster heads and normal peers.

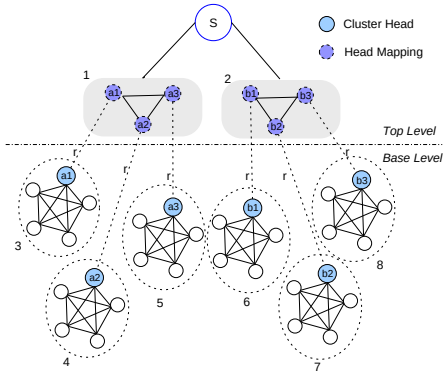


Fig. 1. Hierarchically Clustered P2P Streaming System

The perfect scheduling employed in HCPS does not work well in practical, though. It requires a central controller that collects all peers' upload capacity information, and computes the sub-stream rates sent from the server to individual peers. In practice, available upload capacity may not be known and can vary over time. The central co-ordinator needs to continuously monitor peers' upload capacity and re-compute the sub-stream rates. The adaptive queue-based chunk scheduling method is a distributed solution. Peers only exchange information with the server and make local decisions. Upload capacity information of peers/source is not required, and the scheme adapts to the changing peer membership and network environment automatically. Hence AQCS is a more suitable peer scheduling method for HCPS at cluster level.

There have been ongoing efforts intending to improve resource utilization in P2P live streaming. To improve the resource utilization in mesh-based P2P streaming, [11] 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. [12] performs a reality check by using network coding for P2P live streaming. Nevertheless neither approach can provably achieve the maximum streaming rate. The authors in [13] design a randomized distributed algorithm, Random Useful Packet Forwarding (RUPF), 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 with small control overhead. No data chunk need to be skipped to achieve short startup delay. AQCS also incurs less signaling overhead since no bit-maps carrying the data chunk availability information are exchanged between peers. Our initial experiments show that AQCS out-performs the randomized broadcasting scheme.

3 Adaptive queue-based chunk scheduling

It is desirable to support high streaming rate in P2P streaming. Higher streaming rate allows the system to broadcast video with better quality. It also provides more cushion to absorb the bandwidth variations (caused by peer churn, network congestion, etc.) 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 describe the queue-based chunk scheduling algorithm that can achieve close to optimal peer uploading bandwidth utilization in practical P2P networking environment.

In AQCS, 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. 2 depicts a P2P streaming system using queue-based chunk scheduling with one source server and three peers. Each peer maintains 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 peer a to the server whenever the queues become empty (or have fallen below a threshold) (step 1 in Fig. 2). 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.

3.1 Peer side scheduling and its queuing model

Fig. 3(a) 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

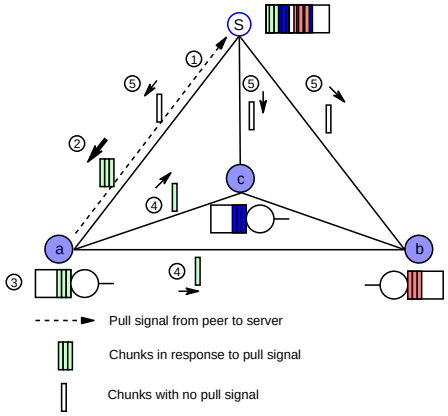
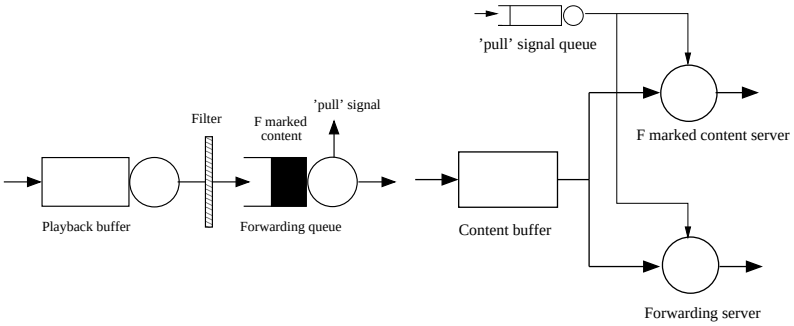


Fig. 2. Queue-based chunk scheduling example with four nodes

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 source server marks the data content either as F-marked content or NF-marked content before transmitting them. *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. *NF* content is filtered out at peers. *F* content is stored into the forward queue, marked as NF content, and forwarded to other peers. Because the relayed content is always marked as *NF* at the relaying peer, data content is relayed at most once in AQCS, which reduces the content distribution time and startup delay. 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 source server are described next.



(a) Queue Model of Peers (b) Queue Model of Source Server

Fig. 3. Queue Models of Peers and Source Server

3.2 Server side scheduling algorithm and its queuing model

Fig. 3(b) illustrates the server-side queuing model of AQCS. The source server has two queues: a content queue and a signal queue. The signal queue buffers the 'pull' signals issued by peers. The content queue is a multi-server queue with two dispatchers: an F-marked content dispatcher and a forward dispatcher. Which dispatcher is invoked depends on the status of the 'pull' signal queue. Specifically, if there are 'pull' signals in the signal queue, a small chunk of content is taken off from the content buffer, marked as *F* content, and dispatched by the F-marked content dispatcher to the peer that issued the 'pull' signal. The 'pull' signal is then removed from the signal queue. In contrast, 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. The forwarding dispatcher marks the chunk as *NF* and sends it to all peers in the system.

3.3 Optimality of queue-based chunk scheduling

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 [10]:

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

where u_s is content source server's upload capacity, u_i is peer i's upload capacity, and n is the number of 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. The maximum/optimal 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 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.*

Sketch of proof: In server resource poor scenario, the source server bandwidth is the bottleneck and cannot handle all 'pull' signals issued by peers. The signal queue at the server side is hence non-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 supportable streaming rate is equal to the server's upload capacity. Optimal rate is achieved.

In server resource rich scenario, the server has the bandwidth to service the 'pull' signals. During the time period when the 'pull' signal queue is empty, the server transmits duplicate NF-marked content to all peers. It can be shown that the streaming rate is $\frac{u_s+\sum_{i=1}^{n} u_i}{n}$. Optimal rate is again reached. The detailed proof can be found in [14].

4 Implementation considerations

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, propagation delay, network congestion, and peer churn.

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 kilo-bytes 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 (2). Denote by T_i the threshold for peer i to issue 'pull' signal. A 'pull' signal is sent to server whenever the number of chunks in the queue is less than or equal to T_i. 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:

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

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 set t_q to be zero. This leads to the following 'pull' signal threshold formula that can be used to guide the threshold selection:

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

The architectures of source server and peer are described next. Figure 4 illustrates the architecture of the source server. 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 these messages. If the output of handling these messages needs to be transmitted to remote peers, the output is put on the per-peer out-unit.

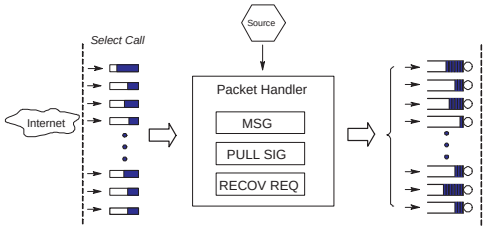


Fig. 4. Server Architecture

There is one out-unit for each destination peer to handle the data transmission process. Each out-unit 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 joined the P2P system and requests 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. The content source server replies to each 'pull' signal with 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 streaming rate.

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. This is similar to the head-of-line blocking problem in input-queued switch design. Separate queues avoid inefficient blocking caused by slow peers.

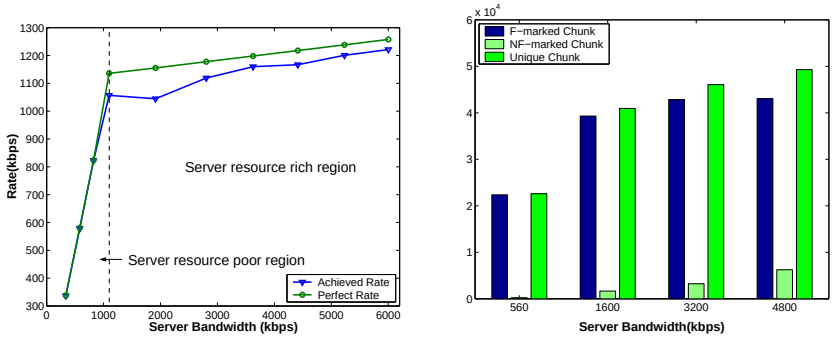
Peers' architecture is similar to the server's and is omitted here. In addition, we design the missing chunk recovery scheme that enables the peers to recover the missing chunks to avoid viewing quality degradation. Refer to [14] for more details.

5 Experiment results

In this section, we examine the performance of AQCS via experiments over PlanetLab [8]. $40+$ 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 [15] 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 graphs. The upload capacity of peers are assigned randomly according to the distribution obtained from the measurement study conducted in [16]. The largest uplink speed is reduced from 5000 kbps to 4000 kbps, which ensures that PlanetLab nodes have sufficient bandwidth to support the targeted rate. Specifically, 20% of peers have upload capacity of 128 kbps, 40% have 384 kbps, 25% have 1 Mbps, and 15% have 4 Mbps.

•**Optimality evaluation**. 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. 5(a)



(a) Achieved Rate vs. Optimal Rate (b) Distribution of Chunks from Server

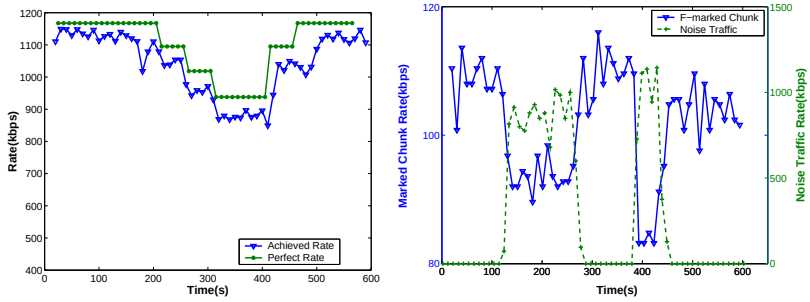
Fig. 5. Optimality Evaluation Results

shows the achieved streaming rate vs. the optimal 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. Notice that AQCS performs better in the server resource poor scenario than in the server resource rich scenario. We plot the numbers of F-marked and NF-marked chunks sent out by the source server to explain what causes the difference.

As shown in Fig. 5(b), when the server bandwidth is 560 kbps (source resource poor scenario), very few NF-marked chunks are transmitted. In theory, no NF-marked chunks should be sent in this scenario since signal queue is always non-empty. We do see several NF-marked chunks, which 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 (source resource rich scenarios). 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 due to the bandwidth variations or peer churn, peers can not receive F-marked chunks promptly. Peers' forward queues become idle and upload bandwidth is wasted. Nevertheless, AQCS always achieve the streaming rate within 10% of the theoretical optimal rate.

•**Adaptiveness to peer churn and bandwidth variations**. Peer churn has been identified as one of the major disruptions to the performance of p2p system. We study how AQCS performs in face of peer churns next. 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. 6(a) depicts the achieved rate vs. optimal



(a) Streaming Rate under Peer Churn (b) Marked Chunk Receiving Rate with Background Traffic

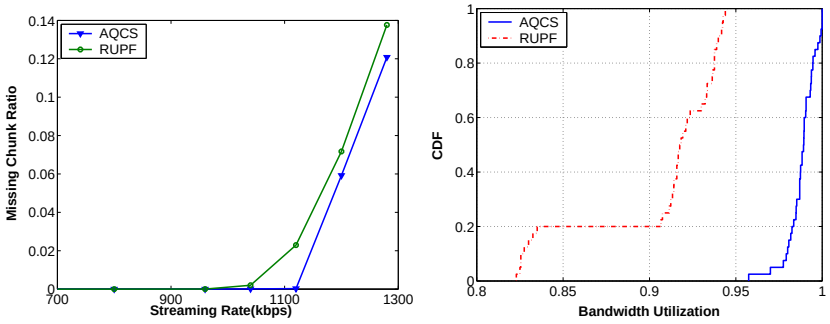
Fig. 6. Adaptiveness to Peer Churn and Bandwidth Variations

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.

In addition to peer churn, the network bandwidth varies over time due to cross traffic. To evaluate AQCS'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.

Fig. 6(b) 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. The queue-based chunk scheduling method adapts quickly to the decreasing 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.

• **Optimality comparison**. Finally the performance of AQCS is compared with that of the Random Useful Packet Forwarding (RUPF) scheme [13], also proved to be optimal theoretically. We implemented the prototype of randomized broadcasting scheme, and carefully tune the configuration parameters so that it performs at its peak performance. The server bandwidth is chosen to be 3.2Mbps, however, the conclusion is true for different scenarios we tested.



(a) Chunk Missing Ratio (b) Bandwidth Utilization

Fig. 7. Comparison with RUPF

Fig. 7(a) depicts the chunk missing ratio, the fraction of chunks that are lost or miss the playback deadline, with different streaming rates. Both adaptive queue-based chunk scheduling (AQCS) and the randomized scheduling (RUPF) achieve zero loss when the streaming rate is low. However, as the streaming rate increase beyond 1Mbps, AQCS remains no loss while RUPF starts to experiences missing chunks. In fact, AQCS maintains zero loss until the streaming rate reaches 1.1Mbps, the maximum streaming rate allowed by the system. The missing ratio of AQCS does increase linear after that due to un-sufficient bandwidth in P2P system. Fig. 7(b) explains why AQCS out-performs RUPF consistently. The CDFs of peer bandwidth utilization for both AQCS and RUPF are plotted at streaming rate 1.12Mbps. It is evident that peers in AQCS are able to utilize their upload bandwidth more efficiently than the peers in RUPF.

6 Conclusions

In this paper, we propose a simple queue-based chunk scheduling method that supports the streaming rate close to the maximum rate allowed by a P2P streaming system. A prototype is implemented and various design considerations are explored to ensure that the algorithm works in realistic network environment. The experiments over PlanetLab further 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. The other direction is to apply this approach to other P2P content distribution applications such as file sharing and video-on-demand. Our work demonstrated the effectiveness of application layer queue management in eliminating content bottlenecks in P2P live streaming. It is a natural extension to explore its applicability in other P2P applications.

References

1. Youtube: (Youtube Homepage) `http://www.youtube.com`.
2. Chu, Y.H., G.Rao, S., Zhang, H.: A case for end system multicast. In: Proceedings of ACM SIGMETRICS. (2000)
3. Zhang, X., Liu, J., Li, B., Yum, T.S.P.: DONet/CoolStreaming: A data-driven overlay network for live media streaming. In: Proceedings of IEEE INFOCOM. (2005)
4. PPLive: (PPLive Homepage) `http://www.pplive.com`.
5. Hei, X., Liang, C., Liang, J., Liu, Y., Ross, K.: A Measurement Study of a Large-Scale P2P IPTV System. IEEE Transactions on Multimedia (2007)
6. BT: (Bittorent Homepage) `http://www.bittorrent.com`.
7. Gkantsidis, C., Rodriguez, P.R.: Network Coding for Large Scale Content Distribution. In: Proceedings of IEEE INFOCOM. (2005)
8. PlanetLab: (PlanetLab Homepage) `http://www.planet-lab.org`.
9. Liang, C., Guo, Y., Liu, Y.: Hierarchically clustered p2p streaming system. In: Proceedings of GLOBECOM. (2007)
10. Kumar, R., Liu, Y., Ross, K.: Stochastic fluid theory for p2p streaming systems. In: Proceedings of IEEE INFOCOM. (2007)
11. Magharei, N., Rejaie, R.: PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming. In: Proceedings of IEEE INFOCOM. (2007)
12. Wang, M., Li, B.: Lava: A reality check of network coding in peer-to-peer live streaming. In: Proceedings of IEEE INFOCOM. (2007)
13. Massoulie, L., Twigg, A., Gkantsidis, C., Rodriguez, P.: Randomized decentralized broadcasting algorithms. In: Proceedings of IEEE INFOCOM. (2007)
14. Guo, Y., Liang, C., Liu, Y.: Adaptive Queue-based Chunk Scheduling for P2P Live Streaming. Polytechnic U., Tech. Rep., (2007)
15. Trickle: (Trickle Homepage) `http://monkey.org/~marius/pages/?page=trickle`.
16. Ashwin R. Bharambe, C.H., Padmanabhan, V.N.: Analyzing and Improving a BitTorrent Network Performance Mechanisms. In: Proceedings of IEEE INFOCOM. (2006)