Reducing Data Request Contentions for Improved Streaming Quality

Yao Liu[1] Fei Li[1] Lei Guo[2] Songqing Chen[1]
[1]Department. of Computer Science George Mason University
[2]Yahoo! Inc. Sunnyvale, California, USA
{yliud, lifei, sqchen}@cs.gmu.edu lguo@yahoo-inc.com

ABSTRACT

In P2P assisted multi-channel live streaming systems, it is commonly believed that in unpopular channels, quality degradation is due to the small number of participating peers with almost-the-same set of available data; this phenomena prevents effective data exchanges among peers themselves and automatically leads to data request contentions once a new data chunk becomes available. In popular programs, our measurement on PPLive for a continuous three-month period at various locations also shows numerous occurrences of quality degradation because of the even higher ratio (up to 190%) of repetitive data requests for the same data chunks.

These results motivate us to investigate effective data exchange strategies to reduce data request contentions for improved streaming quality. IDEA, an Improved peer Data Exchange Algorithm, is proposed to carefully select chunks to request from different peers. We conduct extensive simulations and the results show that IDEA significantly outperforms the widely used algorithms in deployed systems.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Algorithms, Experimentation

Keywords

P2P Streaming, QoS, RE-request

1. INTRODUCTION

With the demonstrated scalability of many practical P2P applications, Internet streaming systems have widely adopted P2P techniques. For example, PPLive [1], UUSee [2] are both P2P assisted multi-channel Internet streaming systems, serving hundreds of millions of Internet users with hundreds of channels daily.

NOSSDAV'10, June 2–4, 2010, Amsterdam, The Netherlands.

The increasing Internet streaming services have attracted significant attention from both research and industrial communities to understand and to improve the performance of existing systems. For example, CoolStreaming [3] implements one of the earliest data-driven IPTV systems. Because the single most concern for clients is the streaming quality perceived, lots of research [4, 5, 6] has sought to improve the streaming quality from various aspects, such as topologies, incentive mechanisms and peer churns.

However, in practically deployed P2P assisted multi-channel live streaming systems, users in unpopular channels still often suffer un-satisfying playback quality due to poor buffered content diversity among the participating peers [7]. This has constrained the peer data exchange among themselves while the server often has limited amount of bandwidth allocated for an unpopular channel [8]. Studies have been conducted to alleviate such quality degradation in unpopular channels by re-provisioning the server resources [8], using peers in popular channels to help unpopular ones [9], etc. However, given limited resource available on the server and the majority ($> 70\%$ on average) of unpopular channels in a P2P assisted streaming multi-channel system, re-provisioning could only have limited effect with the increase of participating clients. On the other side, given that there is always resource shortage in a P2P system due to free-riders or physical constraints on the network uploading bandwidth, leveraging peers in popular channels to help unpopular ones is counter-effective, since the resource shortage in a popular channel is often much larger than that in an unpopular one.

In addition, through the Internet based continuous measurement in practically deployed P2P assisted multi-channel streaming system, PPLive, at various locations (both in USA and China), we have found that quality degradation not only commonly exists in unpopular channels, but also frequently occurs in popular ones because of the very high (up to 190%) repetitive request ratio (for the same data chunks). Such a high repetitive request ratio, caused by severe data request contentions, results in ever-increasing lag of data buffering and the drainage of the playback buffer.

The measurement results motivate us to examine better peer data buffering and exchange strategies. That is, for a peer at any time, in what kind of order it should connect with which available neighbors to request which data chunks. This problem, however, has not been thoroughly investigated before. Previous studies [3, 4, 5, 10, 11] on data delivery approaches mainly considered push (often associated with tree), pull (often associated with mesh), or the hybrid of pull and push (such as new CoolStreaming [10]).

In these studies, simple or heuristic data exchange strategies are used, mainly concerning the data chunk selection policies only, such as local rarest first [3], in which the rarest chunk among all peers has the higher priority of being distributed. However, at the stage of fewer copies of chunks, multiple peers may request the same chunk from the same parent, which results in higher contentions and eventually reduces the speed of proliferating copies.

In this work, noticing that peer data buffering and exchange are constrained by both the available neighbors and chunk selection strategies, we propose an Improved peer Data Exchange Algorithm, called IDEA. IDEA aims to reduce the data request contentions so that peers can minimize their data acquisition time. We conduct extensive simulations in order to compare our scheme with others from different perspectives. The results demonstrate that our scheme consistently outperforms the practically used algorithms in the deployed systems, such as the sequential strategy used in the current PPLive (based on our reverse-engineering), and the local rarest-first used in CoolStreaming for both popular and unpopular channels.

2. PPLIVE MEASUREMENT

In order to examine the streaming quality in current P2P streaming systems, we deployed probing hosts in both China (TeleCom and NetCom) and USA (on university campus) to view different channels of PPLive. While the playback continues, we capture all incoming and outgoing packets. Currently PPLive ranks the channel popularity based on participating peers, ranging from *star-0* to *star-5*. Figure 1 shows the distribution of the channels with different popularity in two weeks. Apparently, the majority ($> 70\%$ on average) are unpopular channels with *star-0* and *star-1*. In our measurements, the probing hosts were instructed to watch both unpopular (*star-0*) and popular channels (*star-5*). Due to page limit, we will only present results observed from China where most PPLive users reside. Since we captured the NBA playoff games, a highly popular event, for 4 days, we also present 4 day results of other unpopular and popular channels as comparisons. Note that while the unpopular and popular channels' streaming rate is 381kbps, the streaming rate of NBA playoff games is 700kbps.

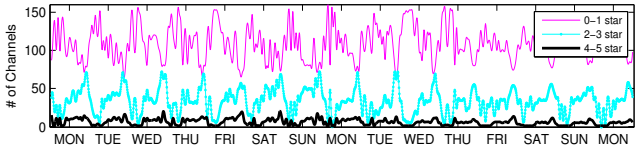


Figure 1: # of Channels with 0-5 stars (2009/08/24-2009/09/07)

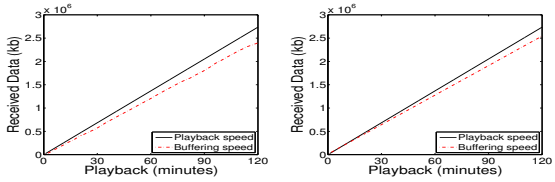
Table 1 shows the number of unique IP addresses that were returned in the peer-list reply messages. Although the result only shows a local view of the total online peers, it indicates the population difference in different channels.

2.1 Quality Degradation in both Unpopular and Popular Channels

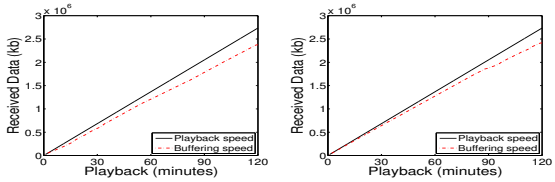
To analyze the streaming quality received at our probing hosts, we first examine if the data receiving is fast enough to catch up the playback. Figures 2, 3, and 4 show the accumulated data received along the playback for unpopular, popular, and highly popular channels, respectively.

Table 1: Observed neighbors from probing hosts

	Unpopular	Popular	NBA Playoff (extremely popular)
Day 1	75	1960	7398
Day 2	79	1773	6319
Day 3	63	1755	6268
Day 4	57	1701	11005

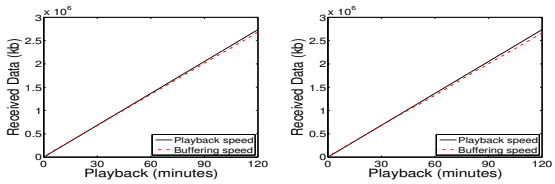


(a) 04/06/2009 (b) 04/07/2009

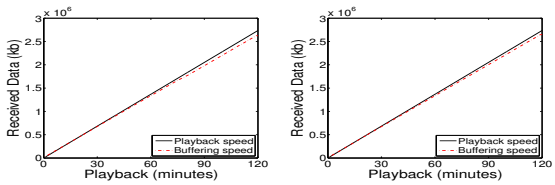


(c) 04/08/2009 (d) 04/09/2009

Figure 2: Buffering vs. Playback (Unpopular channel)



(a) 04/06/2009 (b) 04/07/2009

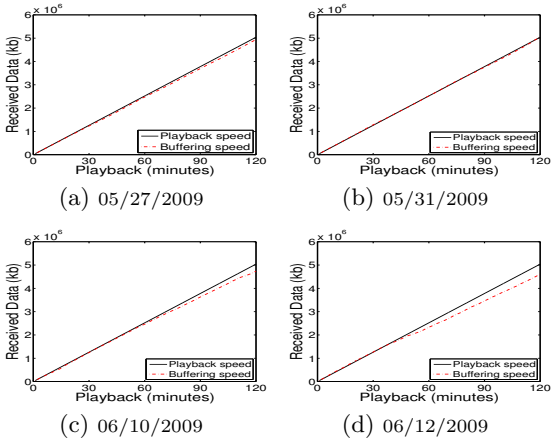


(c) 04/08/2009 (d) 04/09/2009

Figure 3: Buffering vs. Playback (Popular channel)

Figure 2 shows the data buffering versus playback in the unpopular channels from 04/06 to 04/09. Apparently, the initial playback buffer, commonly around 1 minute in practice, cannot smooth out the significant buffering lag, indicating the consistent inferior quality in unpopular channels.

In popular channels, Figures 3 and 4 show that compared to unpopular channels, the buffering lag is significantly reduced. One reason might be that in popular channels, there are more peers available for a peer to contact for data. However, along the playback, the buffering lag still increases in practice. While sometimes the buffering lag could be smoothed out by the initial playout buffer, such as 05/31 for NBA playoff, most of the time, the buffering lag goes beyond the initial playback buffer. Also, on our probing hosts,



(a) 05/27/2009 (b) 05/31/2009

(c) 06/10/2009 (d) 06/12/2009

Figure 4: Buffering vs. Playback (NBA Playoff)

we have observed frequent short-term fluctuations, making users frequently experience replaying the buffered content and ever-increasing playback delay.

2.2 High Data RE-request Rate for the Same Data Chunks

Buffering lags as shown in the last subsection for both popular and unpopular channels indicate our probing hosts could not receive data in time. We thus set to examine the data requests and find a significant amount of data RE-requests (for the same piece of data chunks) during our measurement, especially in popular channels. A data *RE-request* means a repetitive request for the same playback data chunk because a previous request was not replied or was not replied in time.

Table 2: Data RE-request ratio (Unpopular channels)

	# of RE-request	# of Received Unique Pieces	RE-request Ratio (%)	Same Neighbor Ratio (%)
04/06	202096	252343	79.7	80.5
04/07	190203	267074	71.1	78.6
04/08	261275	249830	103.5	85.1
04/09	294652	254171	115.5	86.6

Table 3: Data RE-request ratio (Popular channels)

	# of RE-request	# of Received Unique Pieces	RE-request Ratio (%)	Same Neighbor Ratio (%)
04/06	506815	286975	176.1	86.9
04/07	425014	280546	150.7	83.8
04/08	480251	283848	168.7	85.0
04/09	453577	286690	157.8	84.3

Table 4: Data RE-request ratio (NBA Playoff)

	# of RE-request	# Received Unique Pieces	RE-request Ratio (%)	Same Neighbor Ratio (%)
05/27	689924	467014	147.5	86.0
05/31	865435	449537	190.2	84.8
06/10	740971	452433	162.5	86.7
06/12	728959	450424	160.4	85.0

Tables 2, 3 and 4 show the number of data RE-request against the unique received chunks for playback and the RE-request ratio, respectively, in the second, third, and fourth columns. In unpopular channels, the *RE-request ratio*, defined as the number of data RE-requests divided by unique requested data chunks, is around 100%, indicating that almost all the data chunks were requested twice. For popular and highly popular channels shown in Tables 3 and 4, the RE-request ratio is even higher, reaching 190%. This means some data chunks were even requested three times. Consider the significant control overhead of data RE-requests and the time spent on waiting for data replies, it is not surprising that there is quality degradation in popular channels. Since a data RE-request indicates one unsuccessful data pull attempt, the larger the RE-request ratio, the less efficient the data exchange strategy is. On the other hand, simply increasing the number of reachable neighbors for a peer would not solve this problem as our probing hosts in popular channels have many more neighbors than in the unpopular channels, but have even more request failures. Further analysis shows that these RE-requests were still mainly sent to the same neighbor as before, as shown in the last column in the tables. Thus, the request failure is likely due to the request contention instead of the sudden neighbor departures.

3. DESIGN OF IDEA

We have shown in Section 2 that the quality degradation, in both unpopular and popular channels, comes from the following two underlying reasons: (1) when the number of peers is small, the requested data chunks are not available among the neighbors of the requesting peer at that time; (2) when the number of peers is large, multiple peers requesting the same data chunk overload the neighbors and result in high contentions and poor data exchange efficiency. In this section, we design an Improved Data Exchange Algorithm (IDEA) for peers requesting data chunks from their neighbors in distributed systems by tackling with both issues. On one hand, we would like to propagate quickly the data from the few peers holding them to increase the availability of each data chunk in the whole system; on the other hand, we let the peers request copies of chunks in a uniformly randomized way such that contention can be reduced significantly.

At time t, we consider a P2P streaming system consisting of n peers $P_1,\ P_2,\ \ldots,\ P_n$ that are requesting the same set of data chunks, where n may be changing over time. Here in our system, a peer employs a sliding window scheme to buffer a set of chunks of data and after a chunk is played back by the media player, it is not sharable any more (assuming the worst case). For simplicity of presentation, we regard one peer requesting different sets of chunks at different time as multiple different peers.

At any time, each peer P_j has a vector (which is termed as a *buffermap* in P2P streaming systems) $\mathbf{V}_j = < a_1^j,\ a_2^j,\ \ldots,\ a_m^j >$ to denote the data that it already has, where $a_k^j \in \{0,\ 1\}$ is the indicator of the availability of the chunk c_k for the peer P_j. For all $k = 1,\ 2,\ \ldots,\ m$, we define

$$a_k^j = \begin{cases} 1, & \text{if } P_j \text{ has the chunk } c_k, \\ 0, & \text{otherwise.} \end{cases}$$

We define a matrix $\mathbf{A}$ containing the information about the availability of all chunks of data. Under an ideal scenario, $\mathbf{A}$ is known to all the peers.

$$\mathbf{A}_{n\times m} = \begin{pmatrix} \mathbf{V}_1 \\ \ldots \\ \mathbf{V}_n \end{pmatrix} = \begin{pmatrix} a_1^1 & a_2^1 & \ldots & a_m^1 \\ \ldots & \ldots & \ldots & \ldots \\ a_1^n & a_2^n & \ldots & a_m^n \end{pmatrix}. \quad (1)$$

Given time t, for each peer, we need to specify the order of its connecting with other peers as well as the sequences of exchanging data chunks. Each chunk's data exchange cannot be preempted. Our target is to balance the distribution of the respective data chunks over all the system without jeopardizing individual peers' *performance*, defined as the earliest time of getting the whole set of chunks. Heuristically, an (almost)-evenly distributed placements of distinct data chunks and an optimal uniformly permutation over all copies of the same chunk for peers result in much less contention among peers with limited resources, and potentially improves the performance of P2P streaming systems.

Following the above heuristics, we sketch the idea of our algorithm. For each data exchange session, there is one *sender* and one *receiver*. The sender simply follows the First-Come-First-Serve (FCFS) policy, passing the chunks of data according to the order specified by the receiver. If there are multiple receivers' requests at the same time, ties are broken randomly. The main technique of our algorithm lies at the receiver side. (Roughly speaking, our algorithm is a pull-based one.) Given time t and a peer P_i, our goal is to design an algorithm specifying a sequence for P_i to connect with which peer to exchange which data chunk at which time such that P_i gets the whole set of data S at its earliest time. Among all such feasible sequences, we prefer P_i to choose the one which proliferates the rarer chunks faster. In order to reduce the contention among peers connecting with the same peer for the same chunk, a (uniformly) random permutation policy is employed.

Based on our discussion above, we design the algorithm. Given a time t and a peer P_j, let r_j denote the number of chunks that P_j already has at the current time. $r_j = \mathbf{I} \cdot \mathbf{V}_j^T$, where $\mathbf{I} = < 1,\ 1,\ \ldots,\ 1 >$. Then, we define d_j^k as the least remaining time for P_j to share its available data c_j^k in the system. Since peers' playback points may not be synchronized, d_j^k is defined as the difference between k and the estimated playback point of P_j divided by the playback rate of the peer P_j. Let $x_k = \sum_{j=1}^{n} a_k^j,\ k = 1,\ 2,\ \ldots,\ m$ denote the number of available chunks c_k. Let c_k^j denote the chunk k that a peer P_j has. The weight function for chunks c_k is defined as the inverse of the number of available copies. Thus, the rarest copy has the highest weight.

$$w_k = \frac{1}{x_k} = \frac{1}{\sum_{j=1}^{n} a_k^j}. \tag{2}$$

For each peer P_j, we initialize a queue with $m - r_j$ array spaces indicating the sequence of exchanging the chunks that P_j does not have for now. Then, for these chunks , we sort all copies of them in non-increasing weight order (according to the weight function defined above), with ties broken in favor of the larger deadline ones. At first, we maximize the number of chunks that the peer P_j can get before these chunks of data become unavailable. We insert a copy of each chunk c_k that the peer P_j does not have currently in a non-increasing weight order in the queue. Note that a chunk c_k may have multiple copies belonging to multiple different peers with different deadlines. In the iteration of examining each chunk c_k for P_j, we insert the copy c_k^l from a peer P_l in an empty position no later than d_k^l. If c_k^l cannot be inserted without violating any deadline of an existing copy of chunk in the queue after possible rearrangement over all chunks in the queue, we discard c_k^l; this indicates that we cannot get c_k^l or another chunk before they become unavailable. The feasibility test of c_k^l can be done using the Earliest-Deadline First (EDF) algorithm on all tentatively selected chunks in the queue (see details in Algorithm 1). After considering the copy c_k^l with the largest-deadline for the chunk c_k, we let all other copies c_k^s (if any) have weight reassigned 0; that is, P_j does not consider any other copy of the chunk c_k). We repeat doing above operations until all chunks that P_j does not have at time t are considered. Finally, after P_j maximizes the number of chunks that it can exchange from other peers, P_j uniformly randomizes the order of selecting copies of these chunks in order to reduce the contention among peers requesting the same chunk of data from the same parent. The pseudo-code for a peer P_j works as in Algorithm 1. For clarity, we do not address the role of bandwidth in the algorithm. A factor can be multiplied to the size of the queue storing the chunks unavailable for the peer to reflect the bandwidth limitation and the speed of exchanging data.

Procedure 1 `IDEA-Receiver`($\mathbf{A}_{n\times m},\ P_j$)

1: Get the buffermap information of all peers (that is, initialize a_k^i).
2: Initialize a queue Q to contain the copies of chunks that P_j currently does not have.
3: **for** $i = 1,\ \ldots,\ n$ **do**
4: Set d_i^k according to estimated playback point and playback rate of peer P_i.
5: **end for**
6: **for** $k = 1,\ \ldots,\ m$ **do**
7: $x_k = \sum_{i=1}^{n} a_k^i,\ w_k = 1/x_k$.
8: **end for**
9: Sort all a_k^i in non-increasing weight order w_k with ties broken in favor of larger deadline (d_k^i) ones.
10: **for** each chunk c_k that is in the queue Q **do**
11: Run EDF algorithm over all existing chunks with deadlines d_k^i in Q.
12: **if** some copy of chunk existing in Q cannot be aligned in a position no later than its deadline **then**
13: Reject c_k^l.
14: **else**
15: Insert a copy c_k^l of the chunk c_k belonging to a peer P_l into Q.
16: **end if**
17: Set all copies of c_k weight 0.
18: **end for**
19: Uniformly random permute the copies (from different peers) of each chunk in Q with possible replacement under the deadline constraints.

4. PERFORMANCE EVALUATION

To study the performance of our proposed IDEA, in this section, we evaluate the performance of IDEA against local-rarest first and sequential algorithms, represented as `RAREST` and `SEQUENTIAL`. The former, used in CoolStreaming, gives high priority to data chunks that are least available from neighbors, while the latter, used in PPLive, requests data chunks according to the sequence number of data chunks.

4.1 Experiment Setup

To thoroughly evaluate the various aspects of the algorithms, we use an open source P2P streaming simulator [12].

The following settings are used in our simulations. The streaming rate is 300kbps. Each participating peer has up to 25 neighbors, which is based on our findings from the measurement of PPLive. We set the peers' upload and download bandwidth based on the measurement results from [13]. We consider one streaming server and its outgoing bandwidth is set to 5 Mbps for one channel and the peers' end-to-end latency is randomly mapped to a node pair of a 2500×2500 matrix from the Meridian data set [14].

In the experiments, peers join the streaming channel with the rate of 500 peers per second, and the inter-arrival time follows the exponential distribution. The peers' online time distribution follows the measurement results of Gridmedia [12]. Peers exchange buffermaps with their neighbors every 0.5 seconds, and peers request streaming data from neighbors every 1 second. In each scenario, the peer's buffer size is set to 10 seconds, 15 seconds, and 20 seconds of the playback, respectively. The streaming lasts for 600 seconds. Each test is repeated 10 times and we calculate the average and the 95% confidence interval of our results across these runs for each setup with different random seeds.

We evaluated three different popularity scenarios: (i) unpopular channels in which a total of 50 peers participate in the streaming; (ii) medium popular channels with a total of 500 peers; and (iii) highly popular channels with 5000 peers. While various situations have been evaluated, we can only present the typical results for unpopular, medium popular, and highly popular channels due to page limit.

4.2 RE-request Ratio

As our measurement results shows, the RE-request ratio caused by request contentions is the major source of performance degradation. We thus first investigate the effectiveness of our IDEA algorithm in reducing data request contention.

Figure 5 depicts the RE-request ratio of channels with different popularity. BUF indicates the initial buffer size in these figures. Figure 5 (a) shows that the RE-request ratios of both local-rarest first and sequential are prohibitively high for unpopular channels, reaching about 15. In addition, Figure 5 also shows that in unpopular channels simply increasing the buffer size cannot address the performance degradation. On the contrary, it may actually aggravate the request contention.

As the channel becomes more popular , Figure 5 (b) shows that the performance of local-rarest first significantly improves, but is still worse than that of our IDEA algorithm, while the sequential algorithm performs the worst. The improvement of local-rarest first is due to the increase of the number of available peers that a peer can connect with and request data chunks from. However, when a channel becomes highly popular, Figure 5 (c) shows that the RE-request ratio of local-rarest first increases again. This is because of the faster increase of request contention among peers than the increase of the available peers to connect with since each peer can have a limited number of neighbors, up to 25 in these experiments. Among all the algorithms, IDEA performs the best in all scenarios.

4.3 Received Quality

While request contention is the cause of playback degradation, the ultimate concern from a user's perspective is the playback quality. Figure 6 shows the corresponding overall quality in different scenarios. The quality here is defined as the ratio between the amount of playback data received intime and the amount of data requested for smooth playback.

In consistency with the lower RE-request ratio, in unpopular channels, Figure 6 (a) shows that the received quality of our IDEA algorithm is close to 100%, while both local-rarest first and sequential can only achieve around 80%.

In medium popular channels, Figure 6 (b) shows that the streaming quality of local-rarest first becomes better ($> 95\%$), but it is still worse than that of our IDEA algorithm. On the other hand, the streaming quality with the sequential scheduling algorithm still suffers. In highly popular channels, as we can see from Figure 6 (c), our IDEA algorithm constantly achieves nearly 100% streaming quality. For local-rarest, the quality in the highly popular channel is worse than that in the medium popular channel due to the severely increased request contention as indicated by Figures 5 (b) and (c).

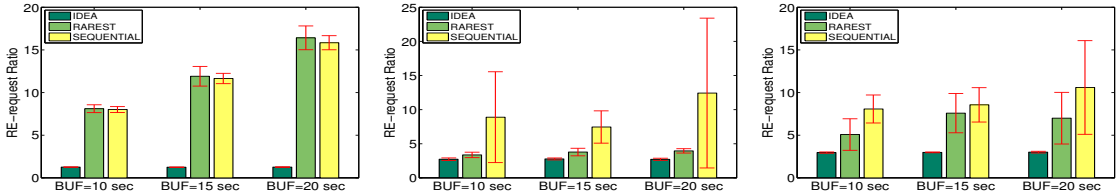
4.4 Packet Propagation Delay

Overall quality presented in the last subsection shows the playback quality once the playback starts. A playback can be started later in order to improve the overall quality by buffering more data. In Section 2, we have observed the increasing playback lag with time. To analyze this lag, we investigate the packet propagation delay at the network level in this subsection. Figure 7 shows the average packet propagation delay calculated over the entire streaming session. The packet propagation delay is defined as the difference between the time when a packet is delivered from the streaming server to the time when the packet actually arrives at a peer. Figure 7 (a) shows the average packet propagation delay in unpopular channels. Compared with Figures 7 (b) and (c) where more peers are participating in the streaming, the propagation delay in unpopular channels is relatively short. This is likely due to two factors. On one side, more peers in the medium and highly popular channels extend the number of hops of packet traveling. On the other hand, request contention also contributes to the prolonged delay we observe on these figures.

Figure 7 also shows that the packet propagation delay actually increases with the increase of the buffer size, although the effect is less pronounced on the medium and highly popular channels than the unpopular channels. Nevertheless, Figure 7 shows that our IDEA algorithm constantly outperforms the other two in all three scenarios.

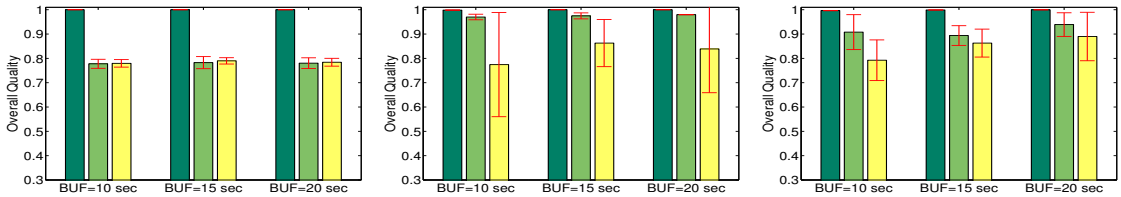
5. OTHER RELATED WORK

Internet streaming systems, particular the P2P assisted multi-channel systems like PPLive, PPStream, UUSee, are becoming more and more popular. Along with the increasing Internet streaming traffic, lots of studies [11, 10] have proposed new schemes to improve the streaming quality. Most recently, Huang et al. have conducted a large scale measurement to study the PPLive based on-demand streaming [13]. On the other hand, the increasing Internet streaming traffic has also raised concerns about the Internet resource utilization. For example, recently, Wu et al. investigated P2P streaming topologies in UUSee [15] and found only about 40% of total connected peers are from the same ISP as the observing peer. Work [8] also aimed to have full ISP awareness to constrain P2P traffic within ISP boundaries.



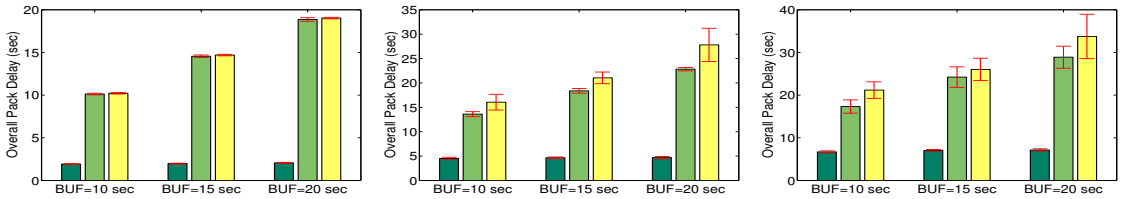
(a) Unpopular (50 Peers) (b) Medium popular (500 Peers) (c) Highly popular (5000 Peers)

Figure 5: Overall RE-request Ratio



(a) Unpopular (50 Peers) (b) Medium popular (500 Peers) (c) Highly popular (5000 Peers)

Figure 6: Overall Received Quality



(a) Unpopular (50 Peers) (b) Medium popular (500 Peers) (c) Highly popular (5000 Peers)

Figure 7: Packet Propagation Delay

6. CONCLUSION

Data buffering and exchange strategies play a key role in the peer data acquisition, and thus directly determine the peer inter-connections and impact peers' perceived streaming quality in the P2P assisted multi-channel live streaming systems. Existing systems use simple or heuristic schemes or adopt the effective schemes in P2P file sharing such as rarest first. The effectiveness of these schemes has not been carefully researched before. In this study, motivated by the measurement results from a large scale deployed system, we investigate the peer data buffering and exchange strategies. In order to improve the peer exchange opportunities, peers are equipped with a new peer selection and data chunk selection scheme. Our evaluation results show that our proposed algorithm can significantly improve the streaming quality for clients in both unpopular and popular channels.

7. ACKNOWLEDGEMENT

We appreciate constructive comments from anonymous referees and help from our shepherd Zhenyu Yang. The work is partially supported by NSF under grants CNS-0746649, CCF-0915681, and AFOSR under grant FA9550-09-1-0071.

8. REFERENCES

[1] "PPLive," http://www.pplive.com.
[2] "UUSee," http://www.uusee.com.
[3] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "Coolstreaming/donet: A data-driven overlay network for efficient live media streaming," in *Proc. of IEEE INFOCOM*, 2005.
[4] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," in *Proc. of IEEE INFOCOM*, 2007.
[5] F. Wang, Y. Xiong, and J. Liu, "mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast," in *Proc. of IEEE ICDCS*, 2007.
[6] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proc. of ACM IMC*, 2006.
[7] C. Wu, B. Li, and S. Zhao, "Diagnosing network-wide p2p live streaming inefficiencies," in *Proc. of IEEE INFOCOM Mini-Conference*, 2009.
[8] C. Wu, B. Li, and S. Zhao, "Multi-channel live p2p streaming: Refocusing on servers," in *Proc. of IEEE INFOCOM*, 2008.
[9] D. Wu, C. Liang, Y. Liu, and K.W. Ross, "View-upload decoupling: A redesign of multi-channel p2p video systems," in *Proc. of IEEE INFOCOM*, 2009.
[10] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements, and performance implications," in *Proc. of IEEE INFOCOM*, 2008.
[11] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming.," in *Proc. of IEEE INFOCOM*, 2007.
[12] "P2P Streaming Simulator," http://media.cs.tsinghua.edu.cn/ zhangm/.
[13] Y. Huang, T. Fu, D. Chiu, J. Lui, and C.Huang, "Challenges, design and analysis of a large-scale p2p-vod system," in *Proc. of ACM SIGCOMM*, 2008.
[14] "Meridian node to node latency matrix (2500 × 2500)," http://www.cs.cornell.edu/People/egs/meridian/data.php.
[15] C. Wu, B. Li, and S. Zhao, "Exploring large-scale peer-to-peer live streaming," in *IEEE Transactions on Parallel and Distributed Systems*, June 2008.