Proxy-P2P Streaming Under the Microscope: Fine-Grain Measurement of a Configurable Platform

(Invited Paper)

Joe Wenjie Jiang[†], Mung Chiang[*], Jennifer Rexford[†]
[†]Department of Computer Science
[*]Department of Electrical Engineering
Princeton University, Princeton, New Jersey, USA 08540
Email: {wenjiej, jrex, chiangm}@princeton.edu

S.-H. Gary Chan, K.-F. Simon Wong, C.-H. Philip Yuen
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
Email: {gchan, cssmw, chyuen}@cse.ust.hk

***Abstract*—Although peer-to-peer (P2P) streaming can efficiently deliver live video content to large user populations, existing applications often suffer from limited video quality, periodic hiccups, and high delays. To overcome some of the limitations of today's unstructured (mesh-based) designs, we have developed and deployed FastMesh-SIM, a novel P2P streaming system that leverages proxies, push-mechanism and IP multicast to achieve lower playback delay and better stream continuity. Having control over a real P2P streaming system also gives us a rare opportunity to conduct controlled experiments where we vary major design parameters (*e.g.*, push vs. pull delivery, IP multicast support, streaming rate, and video segment size) under a range of operating conditions (*e.g.*, dynamics of peer churn, and different network configurations), while collecting detailed, fine-granular measurements (*e.g.*, the various components of end-to-end delay). Analysis of the measurement data, consisting of seven trials of streaming several live TV channels for more than 100 hours to 140 peers, sheds light on how design decisions and the operating environment affect important performance metrics. Our experiments show that a push-based, proxy-P2P system can achieve low delay and good video quality, though network bottlenecks on long-haul connections can sometimes cause disruptions in a global deployment. Theory-practice gaps observed from the data are also discussed. Large-scale, global experiments are now being carried out.**

I. Introduction

Unstructured (or mesh-based) peer-to-peer (P2P) systems, where peers asynchronously swap segments of the video, can substantially reduce server bandwidth and system cost [1], though often at the cost of high latency and periodic hiccups [2]–[4]. An attractive alternative for efficient, low-latency video streaming is to explicitly organize the peers into an application-level multicast tree, and use proxies and recovery protocols to enhance streaming quality. Still, many important questions remain about how to most effectively design and operate a tree-based P2P streaming system. In this paper, we perform an in-depth measurement study of a proxy-P2P network, FastMesh-SIM [5], [6], which broadcasts live TV channels and is deployed in Hong Kong and Princeton. Through controlled experiments on our heavily instrumented system, we provide a unique viewpoint into issues such as the origins of delay and jitter, and performance disruptions in a realistic wide-area deployment.

This work was supported, in part, by the General Research Fund from the Research Grant Council of the Hong Kong Special Administrative Region, China (611209), and the Hong Kong Innovation Technology Fund (ITS/013/08). We would also like to thank the team of engineers and developers, in particular Mr. Joe Tsoi and Mr. Tony Ren at the HKUST for their contributions to the work.

Acquiring a deeper understanding of live P2P streaming systems is challenging, for several reasons. First, there are many performance metrics of interest, including delay, jitter, throughput, and loss. Some of these, such as delay and jitter, are less understood than throughput. We need to understand both the individual components of these metrics (*e.g.*, the breakdown of end-to-end delay into transmission delay, propagation delay, queueing delay, and protocol incurred delay) and the underlying system bottlenecks that limit the achievable performance. Second, many architectural choices have a significant impact on system performance and scalability, including push vs. pull delivery, the size of video segments, the use of proxies to assist in streaming, and the use of IP multicast. Third, performance also depends on the conditions the system experiences, such as the dynamics of peer churn and the behavior of the underlying communication network. Our goal, then, is to gain a quantified understanding of how major design decisions and environmental conditions affect the important performance metrics in the system.

Previous measurement studies have consisted mainly of "black box" characterizations of commercial P2P streaming services [3], [4], [7], [8]. These kinds of studies are invaluable for providing a better understanding of how popular applications, and real users, behave in large commercial deployments. However, black-box monitoring of a proprietary system understandably does not enable experiments that dissect the performance metrics and quantify how they depend on specific design decisions or environmental conditions. Similarly, [9]–[12] inspect data contributed from service providers, yet, these system design and implementation may not even be allowed to be publicly revealed, let alone be freely changed by researchers while serving a customer population.

Other work have relied on high-level models that can be evaluated via analysis and simulation [13]–[19]. While clearly important for advancing our understanding, these kinds of models or tools typically do not capture realistic network cross traffic or lower-level system issues in each peer (*e.g.*,

978-1-4244-7116-4/10/$26.00 ©2010 IEEE

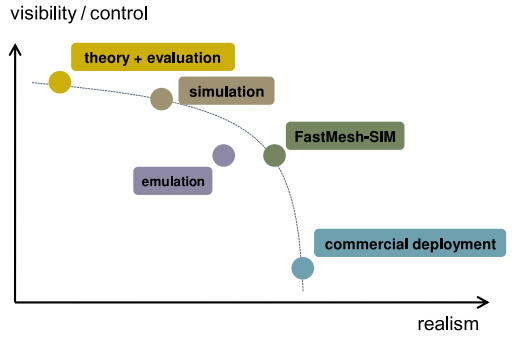

Fig. 1. Illustration of the tradeoffs between programmability and visibility on the one hand and realism on the other hand attained by different approaches.

I/O overheads, interactions with TCP, and ISP rate limit).

Complementary to the above approaches, this paper presents a new point on the tradeoff-curve between visibility/control and realism (as illustrated in Figure 1): controlled experiments on a configurable P2P streaming system that has been designed, implemented, and deployed by the same team that carried out the measurement study. This enables us to collect fine-grained measurements under a range of design settings, while running controlled experiments under realistic wide-area conditions over the public Internet . We collected and analyzed more than 100 hours of streaming logs from seven experimental trials, involving 140 peers at Hong Kong University of Science and Technology (HKUST), China, and Princeton University in NJ, USA. Analysis of the data allows us to identify performance bottlenecks, pros and cons of architectural choices, good settings for tunable parameters, and suitable models and assumptions for future analytical work on P2P streaming. In particular, our evaluation explores the following questions:

- What are the dominant components of delay and what aspects of the system design are primarily responsible?
- What causes "hiccups" in the video playback and how can we prevent these disruptions due to delay jitter?
- What are the tradeoffs in push vs. pull based delivery, and can we benefit from the best aspects of each approach?
- How can network-layer support for IP multicast improve system efficiency and video quality?
- How do network conditions and peer churn affect the scalability and performance of the system?
- How does a local error, such as a lost packet, propagate to affect the rest of the P2P system?

The rest of the paper is organized as follows. Section II gives an overview of the FashMesh-SIM system. Section III presents the measurement methodology and the design factors we vary in the experiments. Section IV presents the data analysis and the derived answers to the questions listed above. Section V discusses the theory-practice gaps illuminated by this study, and Section VI concludes.

II. Design and Deployment of FastMesh-SIM

In this section, we briefly review our design choices in FastMesh-SIM that improve the end-to-end delay, stream quality, and system scalability. FastMesh-SIM constructs application-level multicast trees. The system consists of two main protocols: (i) FastMesh that constructs a streaming backbone among an upper tier of highly-stable proxies, and (ii) SIM (Scalable Island Multicast) by which a lower tier of end-user peers self-organize into one or multiple push-based trees, leveraging local IP multicast support wherever possible. While FastMesh and SIM have been separately reported in prior publications [5], [6], this paper reports the first deployment of an integrated service and an in-depth measurement study. Our system consists of the following building blocks:

Push-based stream delivery. FastMesh-SIM is inherently a push-based system that constructs one or multiple application-level multicast trees. The video is divided into multiple sub-streams, each of which is delivered by one tree. The tree structure is explicitly maintained at every peer. As segments arrive, parent nodes immediately start sending the data to their children in push-based "hot potato" fashion.

Low-delay backbone mesh. The video content is divided into multiple substreams (or bundles) that are each sent through the proxies before reaching the lower tier of peers. The proxies are stable and bandwidth-rich machines, such as dedicated infrastructure nodes contributed by the content provider. The proxies run the fully distributed protocol FastMesh, which builds multiple low-delay spanning trees. The protocol is adaptive such that the delay is constantly optimized given any change of server availability.

Scalable local streaming tree. Local peers run the lightweight SIM protocol to construct low-delay trees with minimal overhead. As a new peer joins the system, it contacts a Rendezvous Point (RP) that returns its local proxy, which in turn gives a random list of existing peers to bootstrap the process. Through successive local probing of these nodes and their neighbors, the peer selects an optimal node as its parent, taking both RTT and bandwidth into consideration. The tree formation is distributed and adapt, as the underlying network conditions and node availability change over time.

IP-multicast integration. A unique feature of SIM is its integration of IP-multicast with application-layer multicast to improve bandwidth efficiency. Although IP multicast is not globally available on today's Internet, many local area networks are multicast-capable. SIM capitalizes on local support for IP multicast by embedding these "multicast islands" into the overlay streaming tree. Peers within a multicast island receive data using IP multicast, and communicate with the outsiders through border nodes via the overlay connections. SIM allows an arbitrary fraction of IP-multicast capable nodes to form one or multiple islands, thus improving the efficiency of bandwidth usage in a local network.

Reactive error recovery. FastMesh is able to adapt to mild server churns by re-optimizing the mesh construction. SIM

implements a more sophisticated error-recovery mechanism to respond to temporary or unexpected packet or stream losses. Through backup parents and distributed tree re-formation, SIM minimizes the latency incurred by the error recovery, even under frequent peer churns.

III. Measurement Metrics and Methodology

Conducting experiments with FastMesh-SIM system enables an unusual degree of both visibility (through fine-grained instrumentation of the software) and control (by programming many tunable design parameters).

A. Visibility: Fine-Grained Instrumentation

Traditional measurement studies in P2P live streaming often resort to a large-scale user pool that generates real traffic in Internet-like environment. However, due to a large user population and peer churns, they often rely on random or statistical sampling, which is partial and coarse granular.

As an alternative approach, we carefully choose a relatively smaller set of peers but distributed diversely over the public Internet. We perform an in-depth segment examination of the system and analyze the performance metrics for the entire user population. In our measurement, each peer generates a record for every segment it receives, which includes a list of metrics that reflect a peer's current status. All records are sent to a log server periodically. With these information, we can reconstruct the lifespan of every segment, allowing us to efficiently characterize and diagnose the system.

Topology: a peer's parent/children list, which helps us trace the path of data delivery and error propagation.

Bandwidth: a peer's upload/download data bytes via different sources, *e.g.*, overlay tree, IP-multicast or recovery server.

Delay: every segment is time-stamped as it is transmitted over the P2P network. We measure various delay information, *e.g.*, the source-to-peer latency, delay incurred on each hop and in each component.

Buffer: a peers' playback buffer state is recorded, including the latest, oldest and currently played segment, and instantaneous segment loss rate.

B. Control: Configurable Streaming Platform

Conventional measurement studies usually leverage data sets collected from or contributed by commercial streaming applications. Visibility is limited since most softwares are proprietary and many specific implementation choices and protocol details are not exposed. They also have to maintain commercial operability and cannot experiment over the entire design space with their customers.

In contrast, in this measurement study, we intentionally turn on/off some features and tune system parameters to perform a set of controlled experiments. The following control knobs allow us to easily configure the system to operate under different settings, which reflects different design tradeoffs:

Architecture: Our system can be configured to operate in push or pull mode. In pull mode, a peer sends requests to its parent for the desired segments. In push mode, a parent immediately forwards the data as soon as it is received.

IP-multicast: The IP-multicast knob can be configured to be on or off. It helps analyze the delay efficiency and bandwidth efficiency achieved by network layer support.

Streaming rate: We tune the video rate such that the system operates under different workloads. This allows us to explore the fundamental limit of streaming capacity over the public Internet and the system overhead under higher workload.

Peer churns: Peer churns in our experiment are synthesized to follow a Poisson arrival and exponential sojourn time.

Recovery: Proxy servers also serve as data backup plane to feed peers during transient data loss. The bandwidth reservation for error recovery is a configurable parameter.

Segment size: Segment size is the smallest replication unit used in the system. It determines how quickly a peer can disseminate the data it holds. Smaller segment size implies lower transmission delay, but greater protocol overhead.

Geographical diversity: We tune locality of participating peers, from same NATted peers to different firewalls, and from the same LAN to Internet-wide locations.

TABLE I
Multi-factored Experiment Trials

Trial	Parameter Configuration
1	Baseline: pushed stream delivery, IP-multicast disabled, no peer churns, 150Kbps stream, segment size of 25 packets (1.4KB)
2	Pulled stream delivery
3	Synthesized peer churns
4	IP-multicast capable peers
5	Varying segment size 1- 700 IP MTU
6	IP-multicast capable and churning peers
7	Varying stream rate 50 - 400Kbps

C. Experiment Setup

In preparation for our experimental trials, we deployed 140 peers (desktops and laptops) in Hong Kong and Princeton. Peers from each network are connected to the Internet via campus LAN access, which are IP-multicast capable. Each network deploys 5 proxy servers that form a FashMesh. Local peers that run SIM protocol join the streaming session as our shell script instructs.

With multiple tunable control knobs, we carefully plan a set of trials that quantify the impact of these factors on various aspects of system performance. We selectively tune on and off a subset of them and design 7 trials, as shown in Table I. Trial 1 is designed as the baseline approach. Trial 2-7 follow the same configuration as Trial 1, but varying one or two factors respectively. These trials are collected during July 10-14 2009, and each lasts around 10 hours.

IV. Data Analysis

In this section, we analyze the traces we collected from the 7 experiment trials.

TABLE II
DELAY COMPONENT ANALYSIS OF A TWO-HOP PEER

Components of Delay (sec)	Push (mean)		Pull (mean)	
local transmission	0.1942	11.2%	0.0454	0.90%
proxy transmission	0.9567	55.2%	0.9282	18.6%
propagation	0.1491	8.60%	0.1738	3.50%
protocol	0.4332	25.0%	3.8268	77.0%
total	1.7332	100%	4.9742	100%

A. *Optimizing Delay Performance*

We first show the latencies in both push and pull based data delivery, and then investigate delay components and the source of delay variation.

End-to-end latency: push vs pull. Figure 2 shows the trend of increasing average end-to-end latency as the tree depth increases. The "push" curve is generated from Trial 1, and "pull" curve from Trial 2, both of which disable IP-multicast. Conventional wisdom is that more hops results in higher delay, which is clearly shown in the "pull" curve as the delay is proportional to the tree depth. On the other hand, the "push" curve is flatter, which is due to the fact that, when data is pushed from the upstream, the one hop transmission delay from Hong Kong to Princeton is the main component.

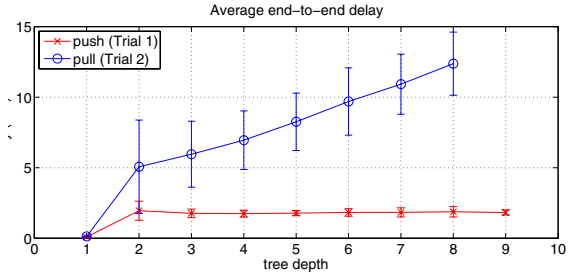

Fig. 2. Average end-to-end delay: push vs pull

Data also quantify the intuition that push can significantly reduce delay. In pull mode, a child requests the segment bitmap information from its parent before transmission begins. The parent-child handshake delay in pull mode can quickly accumulate as the number of hops increases. In particular, we set the buffer map update interval to be 1 second, which is considered low in practice. As the peer population grows, the protocol overhead incurred by pull may become the bottleneck.

Differentiating delay components. After showing the delay performance, a question that naturally arises is what components constitute the end-to-end delay. The source-to-peer delay is accumulated hop by hop, where each hop includes the segment transmission delay, propagation delay, protocol delay (*e.g.*, handshaking and peer-request in pull), and delay due to other system overhead. To differentiate the total end-to-end delay into these components, we choose a peer of depth 2, which eases our analysis since the delay is decoupled into two hops where the first one traverses the Internet from Hong Kong to Princeton, and the second one is on the campus.

Table II summarizes the percentage of each component of delay. When using push, the main delay component is the proxy transmission delay since the throughput over the public Internet is regulated by TCP. Long RTT results in low throughput, leading to a bottleneck. This also explains why the push curve in Figure 2 is flat. When using pull, the proxy transmission delay component becomes less significant, and the protocol delay turns out to be the bottleneck. The buffer map probing period was set 1 second in our implementation, plus the two-way handshake delay, resulting in a total protocol delay of 3.8 seconds for two hops. As the number of hops increases, the factor of protocol delay will further dominate the total end-to-end delay.

Diagnosing delay jitter. Delay jitter is an important metric that affects the continuity of the streaming video. It originates from the delay fluctuations of different segments. Conventional approaches remove delay jitter by creating large playback buffer, which leads to higher latency. We next look into the delay hiccups in our system, and analyzes its causes.

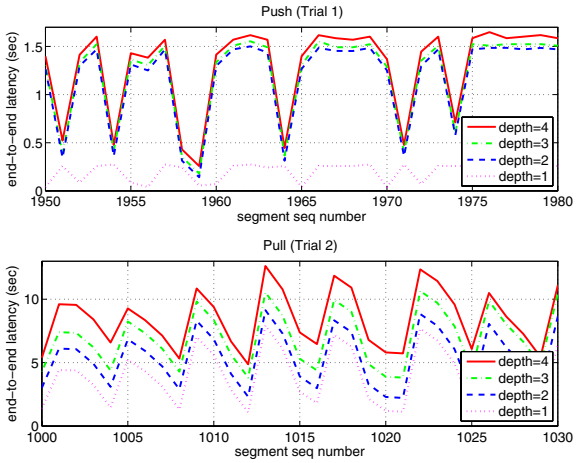

Fig. 3. Segment-wise end-to-end path delay.

Figure 3 shows the evolution of segment-wise delays for peers on the same end-to-end path, for both push and pull without using IP-multicast. In both cases, a delay hiccup occurs roughly once every 5 seconds. It is clear to see that some hops do not contribute to delay variance, *e.g.*, local peer connections. Most delay hiccups originate from the first hop, *i.e.*, the source-proxy connections. Though FastMesh is a push-based protocol, proxies still pull data from the source, which results in such jitters. The pull-request interval is set to be 1 second, and each segment contains around 1.8 seconds of video. Protocol delay overhead is minimized when the request time coincides the generation of every segments, which is every 9 seconds, which is 5 segments. This explains why the delay hiccup happens roughly every 5 segments. It is commonly believed that delay jitter originates from fluctuating bandwidth, but the protocol overhead is another important source of delay jitter.

B. Improving Bandwidth Efficiency by IP-Multicast

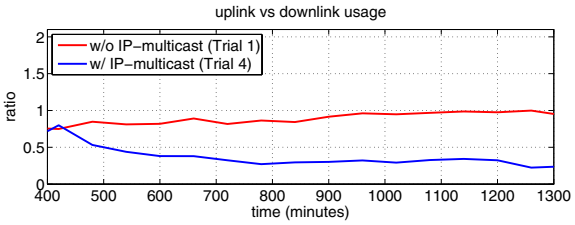

Fig. 4. Streaming with IP-multicast reduces uplink cost.

One of the major benefits of IP-multicast is to improve the bandwidth efficiency in a P2P streaming system, which can be quantified by the ratio between the total uplink usage and total downlink usage. For pure P2P streaming, this metric should be close to one, since every downloaded byte is uploaded from another peer. When the metric is above one, it indicates some data is retransmitted due to packet loss.

IP-multicast improves the bandwidth efficiency by saving the uplink resource, *i.e.*, the multicast source only needs to send one copy of the data to the multicast tree, rather than multiple unicasts between a parent and its children. However, the benefit is limited by two factors. First, some networks have not enabled IP multicast. In our experiments, most peers in Princeton campus network do not have IP-multicast support. These peers receive data from the overlay tree instead. Second, IP multicast relies on UDP which cannot recover packet losses automatically. Hence, the error recovery mechanism is triggered to re-fetch the lost data via the overlay tree approach.

Figure 4 plots the uplink vs downlink ratio of normal peers, for disabled (Trial 1) and enabled (Trial 4) IP-multicast. This metric varies over time due to new peer arrivals and packet losses. In Trial 1, the metric is slightly less than 1 due to the fact that the information of newly arriving peers is not up to date. It converges to one, as predicted by theory, when the system population stabilizes. In Trial 4, this ratio decreases, as more and more IP-multicast capable peers join the system. It eventually achieves a ratio of 30%, *i.e.*, saving 70% of the uplink capacities. This demonstrates that integrating IP-multicast into P2P streaming, though not globally available, has potential to improve the bandwidth utilization.

C. Tradeoff of Segment Sizes

Smaller size reduces per segment transmission time at the expense of system scheduling overhead. To quantify the best tradeoff, we run experiments that vary the segment size, while keeping other parameter settings fixed. Figure 5 shows the delays for the same peer under different segment sizes. A general observation is that delay grows quasi-linearly with the segment size. This is due to the fact that the proxy transmission delay is the bottleneck, which is proportional to segment size. So one possible strategy is use as small a segment size as possible. However, as the segment size decreases, other delay components, such as protocol-induced delay and processing delay, become more significant. This explains why the curve

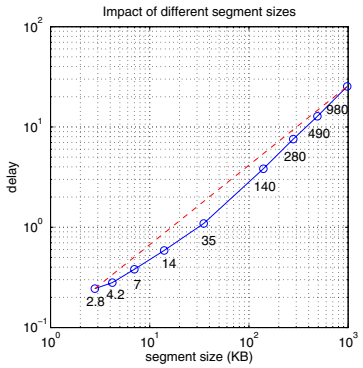

Fig. 5. Trial 5: impact of segment size on end-to-end delay.

is not strictly linear, and the slope is less steep for smaller segment sizes. We did not explore the lower-bound of the segment size, though one that is smaller than the MTU, *i.e.*, 1.5KB, is obviously inefficient due to packet header overhead.

D. Mitigating Peer Churns by Error Recovery

To understand how peer churns affect the system stability and the streaming performance, we introduce a high rate of peer churns in Trial 3. We focus on the dynamics of one selected peer, which is representative of the whole user population. Figure 6(a) shows the topological changes reflected on this peer's tree depth. The user initially joins the system with a depth of 5, and later decreases to 4 and 2, respectively, due to departures of its parents. Such dynamics have severe adverse impact on the peer's delay, as illustrated in Figure 6(b). When the peer loses its parent, it suffers from temporary stream loss and needs to find a new parent, which might incur a very high delay. Eventually the peer's depth becomes 2, meaning that it cannot find a stable parent and connects to a proxy server. Observe that the delay continues to grow even if its topology stabilizes, which looks counter-intuitive but later becomes clear as we zoom into the proxy server.

Figure 6(c) shows the fanout degrees of two proxy servers deployed in this trial, which fluctuates quite heavily and can reach as high as 9. This shows that peer churns are so severe that quite a number of peers cannot find stable parents and eventually have to resort to the proxy server for data recovery. Increasing proxy workload results in an increase of of the peer delay. The peer shown in Figure 6(a)(b) is the child of proxy 1, and its delay shows high correlation with the proxy's fanout. This experiment demonstrates that high peer churns can be quite detrimental to stable viewing performance, causing consistent stream losses and large delays. Deploying more proxy servers, or building a more stable tree based on the history can potentially mitigate this problem.

V. Uncovering Gaps Between Theory and Practice

Most theoretical studies of P2P streaming make simplifying assumptions about the underlying system. The results from our measurement study allow us to revisit these assumptions, confirming some while placing qualifiers on others:

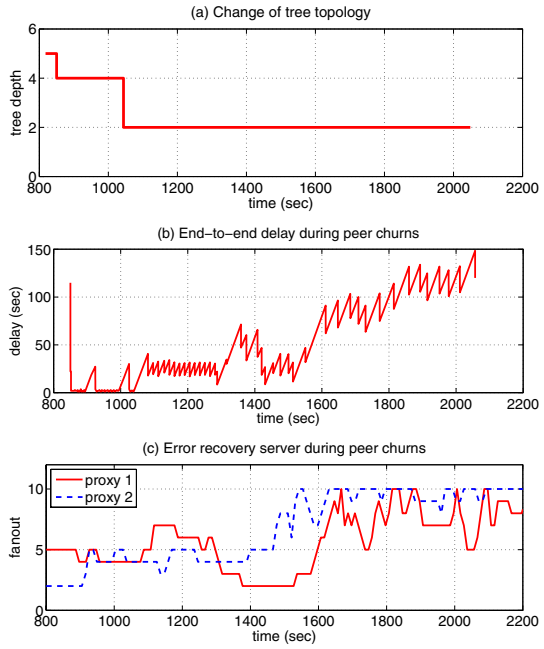

Fig. 6. System robustness and performance during peer churns.

Where is the bandwidth bottleneck? It is commonly believed that a peer's upload capacity is the bottleneck since most access technologies have very asymmetric uplink and downlink bandwidth. The caveat is that long Internet connections may suffer from low throughput, due to high RTT, existence of firewalls, and ISP policing. These factors may start to dominate the upload capacity bottleneck, especially for geographically dispersed or relatively sparse network.

Which is the main delay component? Much previous work assumes that transmission and propagation delay are the dominant components of end-to-end latency. In an operational P2P streaming application, delays induced by peer churns or protocol overhead may instead become more important factors.

Why is there delay jitter? Delay jitter is commonly believed to originate from bandwidth fluctuations. Yet, protocol overhead, especially control handshakes and requesting buffer maps, can become an non-negligible source of hiccups.

How large should node fanout in multicast trees be? In building a tree-based streaming system, many theoretical works optimize peer delay by optimizing the tree fanout. Our data confirm that a large fanout can significantly increase the delay, due to large buffering delays at the high-degree parents.

What kind of trees is more robust? Conventional wisdom is that large tree fanout (shallow tree) may be, on average, less robust than small tree fanout (deep tree). Large fanout implies higher error correlation among children. In practice, a deeper tree is indeed more robust. With proxies and proper error control, a peer's failure does not necessarily trigger the error recovery of all its descendants. The peer's child detects error more quickly than its grandchildren, since the loss of the TCP connection is detected faster than the usual application-layer timeouts.

VI. Conclusion

Compared to several other more mature areas of study in networking, proxy-P2P streaming is still in need of more empirical data collected from deployed systems. On the trade-off curve between high visibility and control on the one hand, and a high degree of realism on the other, this paper presents a valuable complement to existing approaches: fine-grain measurement data collected from the highly configurable FastMesh-SIM platform. Controlled experiments were designed with target research questions in mind, and trials were carried out over Hong Kong and Princeton. Analysis of the data shed light on architectural decisions (such as push vs. pull and use of IP multicast), parameter selection (such as segment size and tree depth), and the impact of system dynamics (such as peer churn and error recovery).

References

[1] "PPLive." http://www.pplive.com/.
[2] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," in *Proc. ACM SIGCOMM*, 2004.
[3] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, 2007.
[4] S. Ali, A. Mathur, and H. Zhang, "Measurement of commercial peer-to-peer live video streaming," in *Proceedings of the Workshop in Recent Advances in Peer-to-Peer Streaming (WRAIPS)*, 2006.
[5] R.-N. Ren, Y.-T. H. Li, and S.-H. Chan, "FastMesh: A low-delay high-bandwidth mesh for peer-to-peer streaming," *IEEE Transactions on Multimedia*. An earlier version appeared in INFOCOM 2008.
[6] X. Jin, H.-S. Tang, S.-H. Chan, and K.-L. Cheng, "Deployment issues in scalable island multicast for peer-to-peer streaming," *IEEE Multimedia Magazine*, vol. 16, pp. 72–80, January-March 2009.
[7] X. Hei, Y. Liu, and K. W. Ross, "Inferring network-wide quality in P2P live streaming systems," *IEEE J. on Selected Areas in Communications*, vol. 25, no. 9, pp. 1640–1654, 2007.
[8] T. Silverston and O. Fourmaux, "P2P IPTV measurement: A comparison study," *CoRR*, 2006.
[9] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain, "Watching television over an IP network," in *Proc. Internet Measurement Conference*, pp. 71–84, 2008.
[10] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, design and analysis of a large-scale P2P-VOD system," *Proc. ACM SIGCOMM*, vol. 38, no. 4, pp. 375–388, 2008.
[11] C. Wu, B. Li, and S. Zhao, "Exploring large-scale peer-to-peer live streaming topologies," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 3, pp. 1–23, 2008.
[12] S. Xie, B. Li, G. Keung, and X. Zhang:, "Coolstreaming: Design, theory, and practice," in *IEEE Transactions on Multimedia*, 2007.
[13] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for P2P streaming systems," in *Proc. IEEE INFOCOM*, 2007.
[14] D. Wu, Y. Liu, and K. W. Ross, "Queuing network models for multi-channel live streaming systems," in *Proc. IEEE INFOCOM*, 2009.
[15] Y. Liu, "On the minimum delay peer-to-peer video streaming: How realtime can it be?," in *ACM Multimedia*, pp. 127–136, 2007.
[16] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance bounds for peer-assisted live streaming," in *Proc. ACM SIGMETRICS*, 2008.
[17] S. Liu, S. Sengupta, M. Chiang, J. Li, and P. A. Chou, "Achieving streaming capacity in P2P." Microsoft Research Tech. Report, April 2008.
[18] C. Feng, B. Li, and B. Li, "Understanding the performance gap between pull-based mesh streaming protocols and fundamental limits," in *Proc. IEEE INFOCOM*, 2009.
[19] Y. Zhou, D. Chiu, and J. C. Lui, "A Simple Model for Analysis and Design of P2P Streaming Algorithms," in *Proc. International Conference on Network Protocols*, 2007.