Will IPTV Ride the Peer-to-Peer Stream?

Alexandro Sentinelli, Gustavo Marfia, Mario Gerla, and Leonard Kleinrock,
University of California Los Angeles
Saurabh Tewari, Yahoo!

ABSTRACT

The great success of P2P systems for the purpose of file-sharing set the path to the next killer application on the Internet, P2P video streaming. Although it solves scalability issues, P2P technology experiences problems of a long start time and churn-induced instability that can greatly affect the user experience. Moreover, technical and business solutions for digital rights management are still under investigation. Great efforts are underway in both academia and industry to solve these problems, whose solution will offer a scalable, affordable, and legal TV-quality-like broadcast of content.

In this article, we analyze what is available to the end user in terms of P2P video-streaming products and determine which of these are the most promising for IPTV and content distribution companies.

In the following, we offer:

- A survey of the available architectures.
- A set of experiments on a popular peer-to-peer system, SopCast.
- Guidelines for large scale deployment.

INTRODUCTION

Overlays and P2P (peer-to-peer) systems, initially developed to support IP multicast and file-sharing, have moved beyond that functionality. These technologies have greatly enhanced the distribution of information on the Internet by enabling efficient cooperation among end users. With the increasing bandwidth capacity provided by the Internet, they also are proving to be key technologies for the delivery of real-time video and audio streams and of video-on-demand files.

Skype is one of the best examples of how P2P technology can be exploited in the consumer market. Beyond the Skype case, which relies on P2P technologies but seldom involves more than two users in a communication environment, new applications are emerging that are more sophisticated in several aspects. Focusing on video streaming, we can distinguish two different scenarios.

One scenario corresponds to TV-viewer migration from regular TV to IP-based TV for watching sports events, news, and popular broadcast channels. We can regard this process as a natural technological migration, where IP technologies add flexibility (e.g., choice between video-on-demand and video streaming) and interactivity to traditional TV service.

The other scenario corresponds to a growing community of end users, who wish to share content that often is produced by themselves. An important contributor to this process is digital camera technology. A typical user uploads content related to personal life (e.g., a birthday video) from a PC and distributes it within a community (e.g., friends). Such a customer is a *pro-sumer*, that is, someone who produces and consumes content. Web sites such as youtube.com, Soapbox, and myspace.com are the main enablers of this phenomenon; they store and distribute content from and to millions of users.

Industry and academia are expecting great success from the introduction and development of IPTV. Consumers will have the opportunity to use an interactive TV that either streams real-time content or retrieves on-demand content. While an IPTV market barely exists today, industry analysts estimate there will be almost 80 million subscribers worldwide by 2011 [1]. Of the aforementioned systems, only Skype relies on P2P technology. The remaining systems unicast from server to receiver, an approach that does not scale well unless backed up by sufficient server replication.

In this article we explain how P2P solves scalability issues by exploiting its ability to distribute information. In the first part of this article, we survey available technologies for the support of video streaming. In the second part, we describe measurements and analysis based on an operational streaming system. Leveraging these results, we comment on the capability of real-time streaming systems to make efficient and fair use of the Internet. We also identify the missing components for a complete and satisfactory video-streaming experience with IPTV.

P2P SYSTEMS

There has been considerable work in the area of peer-to-peer live-video streaming. We refer to only a small subset of the application layer work here, omitting interesting approaches involving media encoding (e.g., [2]) and various network layer techniques (e.g., [3]).

P2P streaming systems strive to optimize

This work was partially supported by the Italian Ministry for Research via the ICTP/E-Grid Initiative and the Interlink Initiative, the National Science Foundation through grants Swarms and Whynet, and the UC-Micro Grant MICRO 04-05 private sponsor STMicroelectronics. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

0163-6804/07/$20.00 © 2007 IEEE

three important metrics: SETUP delay (i.e., the time from when the user first tunes on the channel to when the video is visible), END TO END delay (i.e., the delay between the content originator and the receiver, also known as playback delay), and PLAYBACK continuity (i.e., the percentage of received data packets). Most of the systems can be classified based on the type of distribution graph they implement.

Tree-based overlays implement a tree distribution graph, rooted at the source of content. In principle, each node receives data from a parent node, which may be the source or a peer. If peers do not change too often, such systems require little overhead, as packets are forwarded from node to node without a requirement for extra messages. However, in high churn environments (i.e., fast turnover of peers in the tree), the tree must be continuously destroyed and rebuilt, a process that requires considerable control message overhead. As a side effect, nodes must buffer data for at least the time required to repair the tree to avoid packet loss.

Mesh-based overlays implement a mesh distribution graph, where each node contacts a subset of peers to obtain a number of chunks. Every node must know which chunks are owned by its peers and explicitly *pulls* the chunks it requires. This type of scheme involves overhead, due in part to the exchange of buffer maps between nodes (i.e., nodes advertise the set of chunks they own) and in part to the pull process (i.e., each node sends a request to receive the chunks). Due to the fact that each node relies on multiple peers to retrieve content, mesh-based systems offer good resilience to node failures. On the negative side, they require large buffers to support the chunk pull (clearly, large buffers are required to increase the chances of finding a chunk).

In the following section, we begin with a brief overview of popular tree-based systems and then focus on mesh-based ones.

Tree-based Systems

Narada, by Chu et al. [4], is one of the first examples of end-system multicast targeting video-stream applications. In brief, Narada builds a mesh topology that connects the participating nodes, selecting the links based on round-trip-time estimates between nodes. On top of the mesh, Narada builds a source-rooted minimum-delay tree used for media delivery. Narada was implemented and tested with conferencing applications and is the underlying technology used in ESM (End System Multicast, http://esm.cmu.edu).

NICE, introduced by Banerjee et al. in [5], initially was designed for low-bandwidth, data-streaming applications with a large number of receivers. Based on round-trip-time information between hosts, NICE builds a hierarchy of nodes. In this structure, nodes keep detailed knowledge of *close* peers (in terms of hierarchy) and coarse knowledge of nodes in other groups. No global topological information is required. The hierarchy implies the routes.

Splitstream, by Castro et al. [6], improves fair sharing of resources among nodes. Tree-based systems, designed to limit end-to-end delay, tend to have a large number of leaf nodes. Leaf nodes do not contribute to the overall performance of the system. Splitstream fixes the problem by building multiple trees, where a node can be a leaf in all trees but one. Data, divided into stripes, is propagated using a different multicast tree per each stripe. A receiver that wishes to attain a certain quality of service by receiving a certain number of stripes, joins the trees that correspond to those stripes.

Mesh-based Systems

After the success of BitTorrent as a file-sharing P2P system, the same technology was applied to streaming applications. At a high level the protocol works as follows. A new node registers to the system and receives the addresses of a set of trackers. A tracker is a super-node that tracks the nodes that are downloading or have downloaded a file. When the node contacts the peers advertised by the tracker, the node receives from each of them a buffer map, that is, a map of the chunks of data they own and are able to share. At this point, based on various heuristics (e.g., bandwidth, delay), the node selects a subset of peers and requests chunks from them. This type of chunk exchange scheme is used in the streaming applications described in the following section.

PPLive is by far the most popular video-streaming client. The protocol is proprietary, but due to recent work, [7] it was classified as a mesh-based scheme. The major difference between BitTorrent and PPLive is that in PPLive, packets must meet the playback deadline. To relax the time requirements to have enough time to react to node failures and to smooth out the jitter, packets flow through two buffers; one is managed by PPLive, the second by the media player. A downside of such an architecture is the long start-up delay. There are actually two types of delay:

- The interval between channel selection and media display (10 to 15 seconds)
- The playback time, required for fluent playback in spite of jitter (an additional 10 to 15 seconds)

The time lag between nodes may range up to about one minute. It could be distressing to hear nearby viewers scream "GOAL" while you are still watching the pre-goal action. Nevertheless, PPLive proved to perform remarkably well in several important applications. On January 28, 2006, PPLive delivered a very popular TV program in China, hosting over 200,000 users, at data rates to users between 400 and 800 kb/s, reaching an aggregate bit-rate of 100 Gb/s.

DONet (or Coolstreaming) is probably the next most successful P2P streaming system implementation to date. DONet works the same as PPLive for features such as registration, peer discovery, and chunk distribution. Unlike PPLive, however, the creators of DONet published information about the internals of their scheme [8]. In addition to what was already said for PPLive, we add that DONet implements an algorithm that chooses to download first the chunks with the least number of suppliers. In the case of a tie, DONet chooses the chunks owned by nodes with the largest bandwidth.

The major difference between BitTorrent and PPLive is that in PPLive, packets must meet the playback deadline. To relax the time requirements to have enough time to react to node failures and to smooth out the jitter, packets flow through two buffers; one is managed by PPLive, the second by the media player.

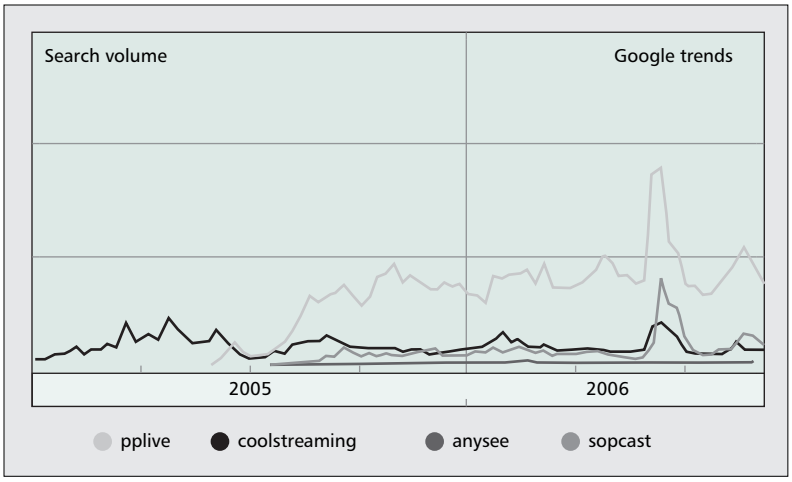

Figure 1. *Normalized search volume for different popular P2P schemes (www.google.com/trends).*

Unlike the previously mentioned schemes, an Anysee [9] node participates in mesh building but does not pull chunks from its peers. Every node in the mesh keeps an active path for data and a set of backup paths in case the active path fails to deliver within certain time constraints. Furthermore, this scheme first introduces the concept of inter-overlay optimization. Anysee involves all nodes (e.g., it uses the spare bandwidth capacity of the nodes that are receiving CNN to help those nodes that are receiving NBC), increasing network efficiency. Therefore, smaller buffers are required, compared to chunk-based schemes. In [9] the authors show experimental results based on the implemented system, where the average delay between source and destination is within 30 seconds.

In the following section, we describe a set of experiments we performed on SopCast [11] (www.sopcast.com). The popularity of SopCast can be seen, in relative terms, from Fig. 1. Even though it started after Coolstreaming, SopCast appears to have the same popularity. Figure 1 also shows that the number of search requests for such systems increased in coincidence with the 2006 World Cup in Germany.

SopCast

Born as a student project at Fudan University in China, SopCast was a success almost instantly. As reported by one of the developers in a Wall Street Journal interview [10], SopCast supported more than 100,000 simultaneous users just a few months after its deployment.

SopCast provides services to both consumers and producers of content. Consumers can log on and tune on the desired channel. Producers can register a channel and broadcast content. Likewise, a producer can stream a file by acquiring such a channel, similar to what can be performed with a system such as YouTube.

SopCast, as a protocol, is a closed system. To learn about its internals so that we could classify and compare it with other schemes, we had to perform measurements on it.

Experiments

We performed two sets of passive experiments. With the first set, we observed the system by tuning on a popular SopCast channel and collecting traffic measurements in different settings (i.e., campus network, 100 Mb/s upload rate versus residential network, 400–500 kb/s upload rate). The second set of experiments was a controlled set. The experiments were run on Planet Lab (www.planet-lab.org), where the source and all destinations were under our control.

Our first goal was to classify SopCast relative to the other P2P streaming systems. We determined that SopCast belongs to the mesh-based family. Inspecting the incoming and outgoing packets of each peer, we observed the exchange of a heavy amount of traffic with several nodes. At first, the SopCast client contacts a tracker (broker.sopcast.com) from which it presumably receives a list of contacts. After contacting the tracker, the client connects to and receives data from a number of other hosts. We infer that SopCast behaves as a chunk-driven protocol, akin to BitTorrent. In the sequel, we describe the details of these experiments.

To perform any experiment, we must differentiate data traffic from overhead. We sniffed a steady stream for an hour. SopCast relies on UDP (user datagram protocol) traffic for most of its functions. Plotting the packet size distribution, we noticed two peaks, one at 1362 bytes and the other at 70 bytes. This fact, together with the source/destination information, suggests that the 1362-byte packets are video traffic, and the 70-byte packets are the application-layer acknowledgments.

In the first set of experiments, we measured:

- The number of peers a node typically connects to
- The amount of data buffered at the client
- The start-up delay
- The playback continuity index

In both campus and residential settings, we observed that the client typically downloads from two to five peers. In our experiments, the lion's share of the stream is usually provided by two peers.

A media frame, before being displayed, must traverse two buffers. Received chunks are stored in the SopCast buffer. When this buffer is full, SopCast launches a media player (e.g., Windows Media Player) that accesses the client's buffer through a local Web server listening on port 8902. Then, chunks are placed in the player's buffer; the size of this buffer depends on the player's settings. For Windows Media Player, we chose the fixed buffer option that by default, is set to five seconds. We then measured the SopCast buffer. We waited for the video to stabilize first. We then simultaneously disconnected the host from the network and downloaded, with a Web browser, the data from the local server. This enabled us to take a snapshot of the client's buffer. Averaging over several trials, we found slightly different results for the two settings. In the residential setting, the buffer was approximately 2.3 MB. In the campus setting, it was 1.8 MB. With a stream rate of 360 kb/s, these results are consistent with the one-minute buffered data values observed in other systems.

Start-up delay in the two different settings was measured for 50 trials for a popular SopCast channel. In the campus setting test, 27 peers were connected to this channel while performing the test. We observed a minimum value of 37 seconds, an average of 56 seconds, and a maximum value of 176 seconds. In the residential setting, peers slightly increased to 35. Here we observed a minimum value of 100 seconds, an average of 111 seconds, and a maximum value of 118 seconds.

We tested the variation of the continuity index of the received stream as the available bandwidth capacity narrowed. The application-layer bit rate in this experiment was 271 kb/s. As we can see in Fig. 2, we observed a continuity index of 99.67 percent at an input rate of 380 kb/s. With such value, the video was clean and fluid. Below this value, the delivery rate began a steep decrease. The quality would remain acceptable up to 330 kb/s input capacity, for a corresponding continuity index of 88.27 percent. This is the smallest continuity index value that permitted one to appreciate the video in this experiment. In a unicast experiment, for the same video data rate, we obtained a similar continuity index value, around 87 percent, but for a 260 kb/s input rate. For the measured setting, we infer that the peer-to-peer system adds about 27 percent more traffic.

We used Planet Lab nodes to perform the second set of experiments. Planet Lab allowed us to place receivers across the United States on machines with known capabilities. SopCast allowed us to broadcast our own channel. In this controlled environment, we wanted to understand how peers cooperate, and how much bandwidth they reserve for uploading. As before, we performed the test using two different settings. The first setting had the source connected to the campus network, the second to the residential network. In these experiments we measured:

- The upload and download traffic pattern in a SopCast network
- The playback delay

To observe the upload and download traffic, we broadcasted a video with a bit-rate of 291 kb/s. The results of this experiment, for the two settings and per each node can be seen in Fig. 3 and Fig. 4. In the campus setting, the download and the upload average bit rates over all nodes including the source are 367 kb/s and 129 kb/s. In the residential setting these values become 339 kb/s and 300 kb/s. Note that the source upload rate decreased from 2.6 Mb/s (on campus) to 415 kb/s (at home). With an 84 percent decrease in source supply, the rate at the receivers decreased by only 8 percent. If the source is placed on a high speed network, clients exploit its capacity by downloading directly from it and supposedly reducing playback delay. We also observed that no nodes other than those connected to our channel were contacted. We then can deduce that no inter-overlay optimization is performed.

In the second set of experiments, the source bit-rate was 95 kb/s. To estimate the playback delay, we connected to the video Web server of each node. In both the campus and the residential case, we experienced an average playback delay of 63 seconds. This value is very close to the buffer we estimated previously. The new estimated buffer value of 0.67 MB is consistent with the conjecture that SopCast buffers one minute of video.

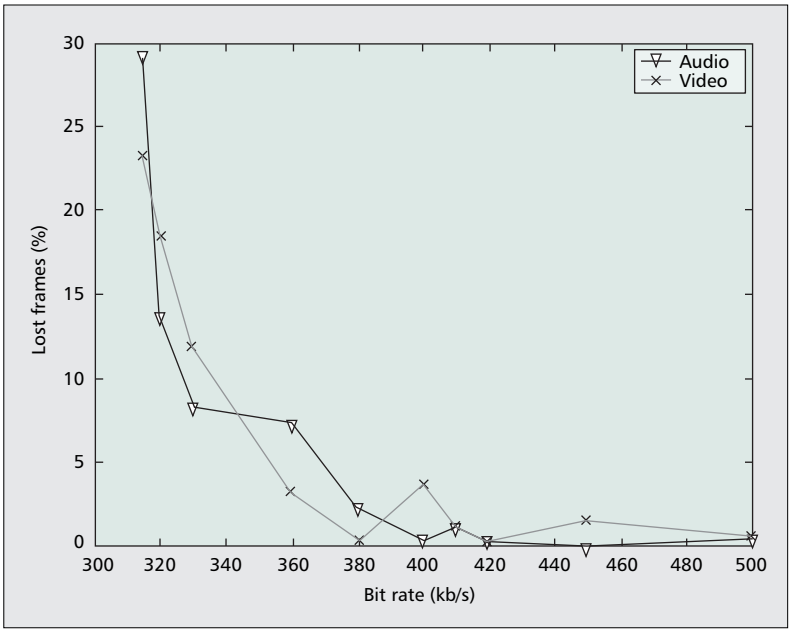

Figure 2. *Continuity index, expressed in percentage of lost frames.*

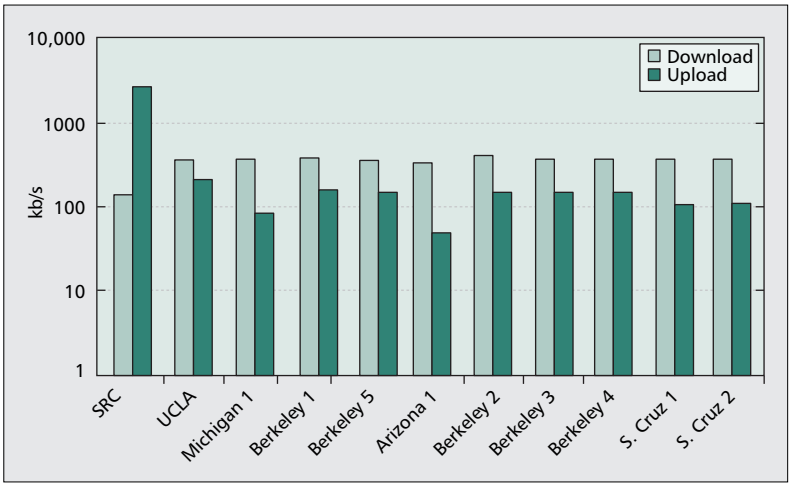

Figure 3. *Planet Lab: upload and download of nodes when the source is connected to the campus network.*

Discussion

Table 1 reports the typical performance figures for the applications at hand. For all of these systems, it is possible to experience tens of seconds in delay both at start up and at playback. These values suggest that, while peer-to-peer streaming systems such as SopCast enable availability of content that otherwise would not be available at a reasonable cost, these systems are still inferior to the conventional TV (on-demand) or client/server IPTV system in terms of performance.

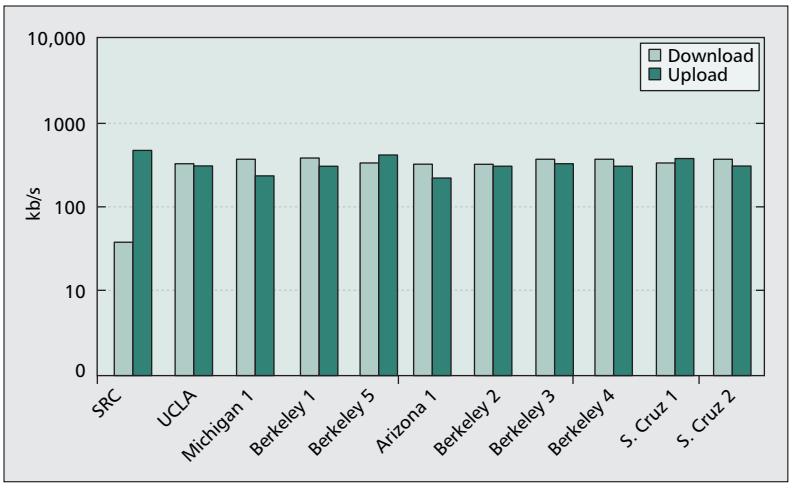

Figure 4. *Planet Lab: upload and download of nodes when the source is connected to the residential network.*

The Next Steps: From Near Real-Time to Real-Time IPTV

Our experiments with SopCast illustrate how the new generation of peer-to-peer live streaming solutions enable anyone with an ordinary broadband connection to broadcast their own TV station to a worldwide audience. However, given the asymmetry of typical residential broadband connections, it is not possible to sustain high-quality streaming using only the peer upload capacity. Unlike *delay tolerant* applications such as file downloads or on-demand streaming, where a peer storing the content can provide the capacity, in *live* streaming, the delay is very critical, and the content quickly loses relevance after a few seconds of delay. Hence, for every viewer watching a 1 Mb/s stream, some peer (or server) must supply 1 Mb/s upstream bandwidth in near real-time. Given the asymmetry in deployed broadband connections (at least in the North American market in 2007), an average residential user cannot supply 1 Mb/s upstream bandwidth. Thus IPTV-like (i.e., 1+ Mb/s) live streaming over the Internet would in near term require supplementing the peer-to-peer capacity with some server-side capacity. We offer guidelines for devising high bit-rate live-streaming solutions utilizing peer-to-peer capacity. Two interesting examples that follow illustrate the practical implications of the capacity-constrained nature of peer-to-peer live streaming.

Suppose we wish to offer a streaming rate of 700 kb/s and expect to have 100,000 users with residential broadband connections with 400 kb/s average upstream bandwidth. On average, if the peer-to-peer solution utilizes 90 percent of the 400 kb/s upstream bandwidth, each user provides 360 kb/s of upstream capacity. At 700 kb/s per user aggregated over 100,000 users, the total delivered streaming rate is 70 Gb/s. The total streaming rate provided by peers at 360 kb/s per user aggregated over 100,000 users is 36 Gb/s. Thus, servers would still be required to supply the remaining 34 Gb/s. This means that short of a revolutionary video-compression technique that can fit the high quality video stream to the limited upstream peer capacity, and until the fiber-to-the-home solutions are deployed, IPTV-quality peer-to-peer live streaming will not be possible. However, we do note that an existing streaming provider that installed capacity to serve, say, 340 kb/s live streaming to 100,000 users, can now provide 700 kb/s, that is, more than double the streaming rate without adding more infrastructure.

As a second example, assume that 5 percent of the 100,000 users are from university networks and can provide 7.5 Mb/s (75 percent of 10 Mb/s) upstream capacity. With this scenario, per 20 users (i.e., 1 corporate and 19 residential), peer upload capacity is 14.34 Mb/s ((19×0.9×400 kb/s) + 7.5 Mb/s), which can fully supply the 14 Mb/s (20×700 kb/s) streaming capacity required for the 20 users. Thus, if the user population has this type of user upstream capacity distribution, the peer-to-peer solution eliminates the requirement for any server-side capacity. Since fiber-to-the-home already is starting to be deployed, we expect IPTV-quality broadcast to become possible using peer-to-peer live streaming in the near future.

We now discuss the factors that affect the ability of the peer-to-peer solution to utilize the peer upstream capacities for chunk-based algorithms. At any time, the upstream capacity of a given peer, say peer A, can be utilized if peer A is connected to a peer that requires one of the chunks peer A has. This is directly determined by the number of peers a peer is connected to at any given time and the number of chunks relevant at any given time. A peer that is connected to a larger number of peers is more likely to find a peer that requires a chunk than a peer that is connected to a fewer number of peers. For example, if the content is assumed to be of little value 20 seconds after the actual event, that allows at most 20 seconds worth of streaming data to be

Scheme	Buffer	Playout delay	Startup delay	Data rate	Definition	Push/pull	Tree/mesh	Reference
PPLive	2 min	1 min	20 s–2 min	300–350 kb/s	320 × 240	Pull	Mesh	[7]
Coolstreaming	2 min	1 min	1 min	300–350 kb/s	320 × 240	Pull	Mesh	[8]
Anysee	40 s	20–30 s	20 s	300–350 kb/s	320 × 240	Push	Hybrid	[9]
SopCast	1 min	1 min	1–5 min	300–350 kb/s	320 × 240	Pull	Mesh	SopCast section

Table 1. *Performance of P2P schemes.*

shared among the peers. Clearly, the larger this delay tolerance, the better a streaming solution will perform — not only does a larger tolerance provide more opportunity for a peer to seek more sources for a particular chunk, but it also provides more chunks that a peer can share. This delay tolerance directly translates into the size of the buffer mentioned in the discussion of our SopCast experiments (e.g., if the delay tolerance is 20 seconds and the streaming rate is 700 kb/s, the buffer size is 14 Mbits or 1.75 MB).

Reference [12] offers an analytical model for chunk-based algorithms. We reproduce Fig. 5 and Fig. 6 from [12] that show the effect of the peer group size (i.e., the number of peers each peer connects to) and the buffer size. We note from Fig. 5 that the incremental benefit of connecting to more than eight peers is minimal. We also note that the authors in [8] (Coolstreaming) also reached a similar conclusion (empirical observations led them to conclude that being actively connected to four peers is adequate). Figure 6 shows the effect of buffer size on the efficiency (the fraction of total end-user upload capacity that can be utilized) of a chunk-based algorithm where the buffer size is described in terms of the number of relevant chunks in the buffer.

It is important to note that, while the number of relevant chunks in the buffer is an indicator of the size of the buffer, there is an important distinction between the two, which has enormous practical significance. As we saw earlier, a 20-second delay tolerance implies a buffer size of 1.75 MB at a 700 kb/s streaming rate. For a chunk size of 256KB, the default value in BitTorrent, this provides only seven chunks, which based on the analytical model of Fig. 6 implies around 87 percent efficiency. However, for example, if the chunk size is reduced to 16 KB, the same 1.75 MB buffer now is comprised of 112 chunks, which based on Fig. 6, would allow a chunk-based solution to utilize 99 percent of the peer upload capacity. (From the model used in [12], we note that the efficiency of a chunk-based solution is equal to [1 – (1/number of chunks)].) We note that decreasing the chunk size does increase overhead costs — smaller chunk size implies larger buffer maps and hence, larger bandwidth consumed in buffer maps exchanged between the peers; transport protocol overhead associated with the transfer of each chunk also increases (as a fraction of total useful bytes exchanged) when chunk size becomes smaller. An appropriate chunk size would balance the increased efficiency with smaller chunk size and this increase in overhead cost due to smaller chunk size.

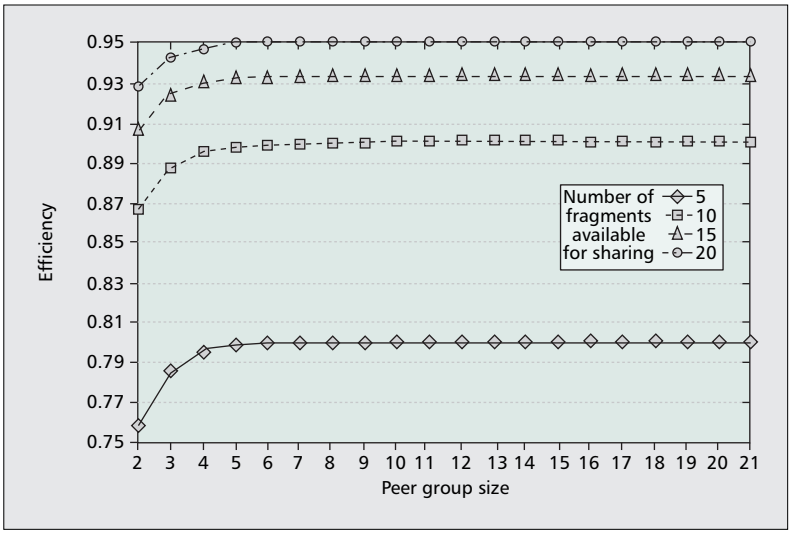

Figure 5. *Efficiency as a function of the number of peers to which a node connects.*

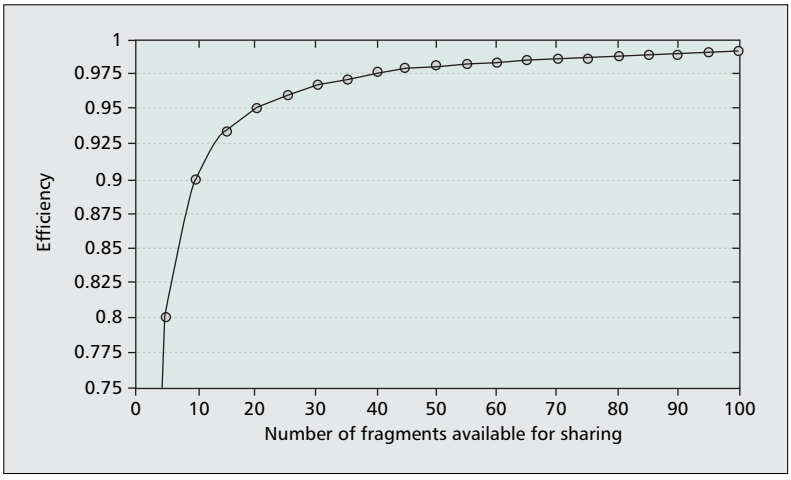

Figure 6. *Efficiency as a function of the buffer size.*

Conclusion

The aim of this article is to give an overview of popular P2P streaming systems, provide the results of some experiments on a real working system, and identify key trade offs between bandwidth, buffers, and latency in the design and deployment of large-scale P2P streaming. The results of these measurements, together with the data gathered in other papers, showed that such systems are adequate for on-demand viewing of delay-tolerant content. They are, however, far from adequate for delivering time-critical video streams; the major obstacle is asymmetric residential user connectivity.

We plan to take further advantage of SopCast and Planet Lab in our future work. Our plans include measurements of a controlled large-scale network. In this way, we aim to validate the model and the trade off herein described. Those results will be used to explore new peer selection strategies in P2P streaming.

References

[1] http://www.cedmagazine.com/article/CA6412165.html

[2] X. Jin, K.-L. Cheng, and S.-H. Chan, "Sim: Scalable Island Multicast for Peer-to-Peer Media Streaming," *IEEE Int'l. Conf. Multimedia Expo (ICME)*, July 2006.

[3] J. Li, "Peerstreaming: A Practical Receiver-Driven Peer-to-Peer Media Streaming System," Microsoft Research TR-2004-101, Sept. 2004.

[4] Y. hua Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," *Proc. 2000 ACM SIGMETRICS Int'l. Conf. Measurement and Modeling of Comp. Sys.*, 2000.

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. 2002 Conf. Apps., Technologies, Architectures, and Protocols for Comp. Commun.*, 2002.

Our plans include measurements of a controlled large-scale network. In this way, we aim to validate the model and the trade off herein described. Those results will be used to explore new peer selection strategies in P2P streaming.

[6] M. Castro *et al.*, "Splitstream: High-Bandwidth Multicast in Cooperative Environments," *Proc. 19th ACM Symp. Op. Sys. Principles*,2003, pp. 298–313.
[7] X. Hei *et al.*, "A Measurement Study of a Large-Scale P2P IPTV System," Tech. rep., Dept. of Comp. and Info. Sci., Polytechnic University, 2006.
[8] X. Zhang *et al.*, "Coolstreaming/DONet: A Data-driven Overlay Network for Efficient Live Media Streaming," *IEEE INFOCOM*, 2005.
[9] X. Liao *et al.*, "Anysee: Peer-to-Peer Live Streaming," *IEEE INFOCOM*, 2006.
[10] G. A. Fowler and S. McBride, "Newest Export from China: Pirated Pay TV", *Wall Street J.*, Sept. 2, 2005.
[11] S. Ali, A. Mathur, and H. Zhang, "Measurement of Commercial Peer-to-Peer Live Video Streaming," *Wksp. Recent Advances in Peer-to-Peer Streaming*, Aug. 2006.
[12] S. Tewari and L. Kleinrock, "Analytical Model for BitTorrent-Based Live Video Streaming," *IEEE Wksp. Networking Issues in Multimedia Entertainment*, to appear, 2007.

BIOGRAPHIES

ALEXANDRO SENTINELLI (alexsent@cs.ucla.edu) received a Laurea degree in electrical and telecommunications engineering from the Università di Roma 3 in 2003. Since 2005 he has been a researcher for STMicroelectronics at the University of California, Los Angeles (UCLA). In 2003 he joined the Research and Development Laboratories of Motorola Italia, Turin, Italy. His research interests are physical layer, information theory, and overlay networks in P2P environments for video streaming applications.

GUSTAVO MARFIA (gmarfia@cs.ucla.edu) graduated in 2003 in telecommunications engineering from the Università di Pisa. He joined the Network Research Lab at UCLA, pursuing a Ph.D. in computer science under the guidance of Prof. Mario Gerla. He worked for two years at Siemens Mobile Research Laboratories in Milan, where his work focused on new application development for 3G networks. His research interests include ad hoc networks, P2P networks, and congestion control.

SAURABH TEWARI (stewari@cs.ucla.edu) received his Ph.D. in computer science from UCLA in January 2007. He received his M.B.A. from the University of Texas at Arlington, his M.S. in electrical and computer engineering from Carnegie Mellon University, and his B.Tech. in electrical engineering from the Indian Institute of Technology, Kanpur. He works on streaming media distribution technologies at Yahoo. He worked for over eight years on core routing and optical transport at Alcatel, and optical sensors systems at Optomation.

MARIO GERLA [F '02] (gerla@cs.ucla.edu) received a graduate degree in engineering from the Politecnico di Milano in 1966, and his M.S. and Ph.D. degrees in engineering from UCLA in 1970 and 1973. He joined the Faculty of the Computer Science Department at UCLA, where he is now a professor. He worked for Network Analysis Corporation, New York, from 1973 to 1976. His research interests include distributed computer communication systems and wireless networks. He has designed and implemented various network protocols (channel access, clustering, routing, and transport) under DARPA and NSF grants. Currently, he is leading the ONR MINUTEMAN project at UCLA, with a focus on robust scalable network architectures for unmanned intelligent agents in defense and homeland security scenarios. He also conducts research on scalable TCP transport for the next-generation Internet (for recent publications, see http://www.cs.ucla.edu/NRL).

LEONARD KLEINROCK [F] (lk@cs.ucla.edu) received his B.E.E. degree from City College of New York (CCNY) in 1957 and received his Ph.D. from Massachusetts Institute of Technology in 1963. He serves as a professor of computer science at UCLA and served as chairman of the department from 1991 to 1995. He has also received honorary degrees from CCNY (1997), the University of Massachusetts, Amherst (2000), the University of Bologna (2005), and Politecnico di Torino (2005). He has published more than 240 papers and authored six books on a wide array of subjects including queuing theory, packet switching networks, packet radio networks, local area networks, broadband networks, gigabit networks, and nomadic computing. He is a member of the American Academy of Arts and Sciences, the National Academy of Engineering, an ACM Fellow, and a founding member of the Computer Science and Telecommunications Board of the National Research Council. Among his many honors, he is the recipient of the CCNY Townsend Harris Medal, the CCNY Electrical Engineering Award, the Marconi Award, the L.M. Ericsson Prize, the NAE Charles Stark Draper Prize, the Okawa Prize, the IEEE Internet Millennium Award, the UCLA Outstanding Teacher Award, the Lanchester Prize, the ACM SIGCOMM Award, the Sigma Xi Monie Ferst Award, the INFORMS President's Award, and the IEEE Harry Goode Award. He was listed by the *Los Angeles Times* in 1999 as among the "50 People Who Most Influenced Business This Century."