Simulating Large-Scale P2P Assisted Video Streaming

Ryan LaFortune and Christopher D. Carothers
Rensselaer Polytechnic Institute
{laforr, chrisc}@cs.rpi.edu

William D. Smith, Joe Czechowski, and Xi Wang
General Electric Global Research Center
{smithwd, czechowski, wangx}@research.ge.com

Abstract

In this paper, we show through detailed simulations that BitTorrent can be used to assist a server or content distribution network (CDN) for large-scale streaming delivery. Specifically, a content owner can distribute a file to over 131,072 users/peers with a peak swarm size of 16,384 concurrent users and a CDN bandwidth savings of 73%, while achieving an average user buffer time of under 2 seconds, which is an A+ on the StreamQ user performance rating system. Current research and even deployed systems rarely have swarms in excess of 1,000 concurrent peers for a single piece of content. Consequently, this is the first study that reports on peer-to-peer streaming for swarms of this scale. Further, the simulation model demonstrates that a high quality of service can be provided, while significantly reducing the distributor's transit costs for these large swarms.

1. Introduction

In 2005, peer-to-peer (P2P) accounted for roughly 60% of all Internet traffic [36] (a European study from 2007 claimed this number was close to 95% during off-peak hours [1]). Most of this traffic was due to file-sharing protocols such as BitTorrent. At any time, there are approximately 8 million users sharing 10 petabytes of data (mostly media files). This accounts for nearly 10% of the worldwide broadband connections [35]. While many files transferred using P2P are done so illegally, P2P has become an attractive alternative for content owners to legally distribute files because of reduced transit costs, and also the fact that centralized content server performance deteriorates rapidly as the number of clients increases.

BitTorrent is a robust P2P protocol that takes advantage of peers' bandwidth to efficiently replicate content. It is a scalable mechanism that has been successful at distributing large files quickly without overwhelming the capacity of the origin servers [3]. BitTorrent also features a game theoretical incentive mechanism used to ensure the fair distribution of content and prevent selfish peer behavior.

With so much digital media content being transferred, it is natural to examine P2P's potential for streaming delivery. If content is streamed, the user can begin to enjoy it sooner, and can evaluate its quality early on in order to preserve valuable resources [46]. While many P2P protocols exist for streaming, none have achieved the degree of performance, scalability, user-fairness, and popularity as BitTorrent has for accomplishing time-insensitive mass-downloads. For this reason, we explore using BitTorrent to assist in streaming downloads.

Studied is a P2P streaming model based on BitTorrent Assisted Streaming System (BASS) [15] over a wide range of scenarios. Our work has two main differences from [15]. First, our simulation scenarios are based on the large-scale (100's of thousands of users compared to 350 users) distribution of television content using a combination of subscription and on-demand usage patterns. Furthermore, we employ a variation on the buffering scheme used in [15]. Specifically, we add an additional buffer designed to improve streaming quality by reducing the number of rebuffers a user experiences.

Having streaming swarms of this size is by no means far-fetched. CBS experienced 4.3 million (unique) downloads of its media player just before the 2008 NCAA March Madness basketball tournament (an increase of 147% from the prior year) [42]. Even more recently, 1.1 million viewers signed on in Great Britain to watch the 2008 Summer Olympics in Beijing [23]. While the systems in these examples were server-only systems, they demonstrate significant growth in the online streaming community, and also the need to incorporate P2P technologies in order to drive down distribution costs in the near future.

In our model, a portion of the *swarm* (peering network) downloads the content immediately upon its release (the *flash crowd*). Soon after the release, on-demand viewers join the swarm with the intention of streaming the content. Simulations are performed using our full-detail "mainline" BitTorrent simulator [8] and Internet topology model [28]. The two coupled together allow us the means to simulate BitTorrent more accurately and on a scale orders of magnitude larger than previous efforts (as is required to simulate television-size audiences). Further, our model preserves the designated market areas as defined by Nielsen Media Research [33], allowing us to analyze trends by geographic region in the United States.

Discussed next is related work and our contributions. In Section 2, an overview of the BitTorrent protocol is provided. Our implementation is discussed in Section 3. Simulation results are described in Section 4. Finally, our conclusions are presented in Section 5.

1.1. Related work

P2P streaming is often accomplished using *application layer multicast*, where an overlay network is constructed containing the participating nodes. The content owner injects the stream into the overlay, where the nodes may consume it and forward it to their children. The structure of the overlay is typically a tree, forest, or mesh.

A multicast tree is the simplest structure. Each node receives content from a single parent, and forwards it to its children. The height of this tree translates into its latency, and the width translates into the number of bandwidth bottlenecks. ZIGZAG [44] is an architecture composed of a clustering hierarchy and a multicast tree of logarithmic height and constant node degree. Overcast [21] also builds a tree, but attempts to maximize a metric such as bandwidth or latency from all nodes to the root. A common problem characteristic of multicast trees is their lack of fault-tolerance. If a single node fails, it may disconnect portions of the tree, rendering them useless until the failing node's children can recognize the failure and reconnect. Bayeux [50] (an architecture that leverages Tapestry [48]) attempts to solve this problem using secondary pointers.

Traditional tree-based multicast is not well suited for P2P, as the burden of duplicating and forwarding traffic is carried by a small subset of peers that are interior nodes of the tree. This conflicts with the expectation that all nodes will share this burden equally. SplitStream [11] splits the stream into stripes, each delivered with a separate multicast tree. It attempts to create a structure where interior nodes of one tree are leaf nodes in all the remaining trees, in order to fairly distribute the forwarding burden. Other systems that use forests include Narada [13] and PALS [40].

Fundamentally, forest overlays suffer from the same problems as tree overlays [47], since a node in any stripe may fail. Like a forest, a mesh overlay allows for simultaneous downloads, but also allows parts of the file to come from perpendicular nodes. If a node fails, other nodes can continue to receive content while reconnecting to the overlay. However, a protocol is needed to locate missing content in the network. Bullet [27], CollectCast [17], and DONet/CoolStreaming [49] are examples of systems that use mesh overlays. [2] uses network coding, segment scheduling, and peer-matching algorithms to show that high quality of service (QoS) video-on-demand (VoD) is feasible in mesh-based overlays. Further, they show that poor performance could be a result of under-utilizing network resources. The swarms studied in this paper were small (typically 500 nodes), and it is unclear how these techniques scale for larger swarms.

Under ideal conditions, application layer multicast works well for streaming media. But, even with clever techniques to ensure performance, scalability, and fault-tolerance, all these schemes lack user incentives. Users upload in good-faith [43], and are not penalized if they choose not to contribute their resources to the P2P network. This has sparked some interest in using BitTorrent for streaming, since BitTorrent employs an incentive mechanism.

A peer-assisted VoD architecture that uses a BitTorrent-like network for topology management and a distributed hash table (DHT) for data management is proposed in [20]. They show it is possible to achieve a good QoS, a low startup time, and cost savings with the assistance of a P2P network. BiToS [46] is a BitTorrent derivative that imposes minimal changes to the protocol's piece-picker to allow for streaming. It organizes needed pieces into two queues, the *high-priority pieces* and the *remaining pieces*. Any piece that misses its playback deadline will be removed from both queues and will no longer be considered for download, thus degrading the video quality. With a probability of p, the earliest deadline piece of the high-priority piece set is requested, and with a probability of $1 - p$, the rarest remaining piece is requested (p can be fixed or dynamically assigned). The goal is to download pieces in order as they are needed for playback, and occasionally download rare pieces to make the peer an attractive trading partner as per BitTorrent's incentive mechanism. Thus, the different values of p affect the video quality, and the download time remains roughly the same. [7] describes another piece-selection policy that attempts to achieve a good compromise between high piece diversity and the in-order retrieval of pieces. BASS [15] assists a server or content distribution network (CDN) infrastructure with a P2P network (BitTorrent) to reduce transit costs. [12] presents another BitTorrent-assisted streaming technology called Torrent Assisted Streaming (Toast), and provides results from small-scale sessions (300 peers). In this paper, we simulate BitTorrent-assisted video streaming for very large swarms (up to 131,072 peers).

There is an emerging market for streaming media content using P2P. This trend is due to the fault tolerance, scalability, and potential for transit cost savings characteristic of P2P. Some companies currently offering streaming video via P2P include: PPLive [37], PPStream [38], MySee [31], Roxbeam [41], UUSEE [45], BitTorrent.com [5], Verisign Kontiki [26], ITIVA [19], Joost [24], Pando [34], and Red Swoosh [39]. In China alone, over 12,000,000 users have accessed a service or downloaded software within the last two years [47].

1.2. Contributions

Through the use of detailed simulations over a wide range of large-scale scenarios, we show that using BitTorrent to assist a server or CDN infrastructure can result in a high QoS for users, while reducing distributor transit costs significantly. Specifically, a content owner can distribute a file to up to 131,072 users/peers with a peak swarm size of 16,384 users and a bandwidth savings of 73%, while achieving an average user buffer time of under 2 seconds (see Table 1). Current research and even deployed systems such as CoolStreaming rarely have swarms in excess of 1,000 concurrent peers for a single piece of content. Consequently, this is the first study that reports on P2P streaming for swarms of this scale. Further, we show that CDN utilization scales roughly linearly with swarm size, demonstrating that swarm size estimates can be used to estimate CDN utilization, allowing content owners to optimize their CDN bandwidth agreements.

2. The BitTorrent protocol

The BitTorrent protocol creates a virtual P2P overlay network using five major components: (i) *a torrent file*, (ii) *a web site*, (iii) *a tracker server*, (iv) *seeders peers*, and (v) *non-seeder peers*.

A *torrent* file is composed of a header plus a number of SHA-1 block hashes of the original file, where each block or *piece* of the file is a 256 KB chunk of the whole file. These chunks are further broken down into 16 KB subchunks called *slices*. The header information denotes the IP address or URL of the *tracker* for this torrent file. Once created, the torrent file is then placed on a publicly accessible web site, from which anyone can download. In addition, the original content publisher will start a *seeder* peer with a complete copy of the content file as well as the torrent file. The seeder uses the torrent file to register with the tracker. The tracker server matches peers interested in the same content with each other. New peers interested in downloading the content pull the torrent file and then register with the tracker. When a *non-seeder* completes downloading the content, it becomes a seeder. Furthermore, both seeder and non-seeder peers share content that they have downloaded with other peers. Because of the size of the *peer-set* (maximum of 80) and the random peer selection, the probability of creating an isolated clique in the overlay network graph is extremely low, which ensures robust network routes for piece distribution. The downside to this approach is that topological locality is completely ignored, resulting in much higher network utilization (i.e. more network hops and consumption of more link bandwidth). Thus, the protocol makes a robustness for locality tradeoff.

Peers in the swarm will begin to transfer slices of the file amongst themselves in a cooperative *tit-for-tat* manner, which is governed by BitTorrent's complex *choker* algorithm. The choker is designed to refuse cooperation with selfish peers that do not contribute to the swarm [14].

When a piece is downloaded, the BitTorrent software will validate that piece against the SHA-1 hash value for that piece. When a piece is validated, that peer is able to share slices of it with other peers who request it. Pieces within a peer-set are exchanged using a *rarest piece first* policy, which is used exclusively after the first few randomly selected pieces have been obtained (typically four pieces). Because each peer announces to all peers in its peer-set every piece it obtains (via a HAVE message), all peers are able to keep copy counts on each piece and determine within their peer-set which piece or pieces are rarest (i.e. lowest copy count). When a peer has obtained all pieces of the file, it then switches to being a pure seeder of the content. At any point during the exchange process, peers may join or leave the swarm. Because of the highly volatile nature of these swarms, a peer will re-request an updated list of peers from the tracker periodically (typically every 300 seconds). This ensures the survival of the swarm, assuming the tracker remains operational.

More recently, BitTorrent has added a DHT based tracker mechanism. This approach increases swarm robustness even in the face of tracker failures. However, DHTs are beyond the scope of the current investigation.

3. Implementation

The BitTorrent model is written on top of *ROSS* [9, 10], which is an optimistically synchronized parallel simulator based on the Time Warp protocol [22]. In this modeling framework, simulation objects, such as peers, are realized as logical processes (LPs) that exchange time-stamped event messages in order to communicate. Each message within the BitTorrent protocol is realized as a time-stamped event message, where the time stamps are generated by delays from the network topology model [28], which realistically approximates today's home broadband Internet service.

The topology model is designed to abstract away details that are non-pertinent to Internet simulations, where delays experienced in the core are negligible compared to those in the *last mile* [16]. It is comprised of several components such as the network connectivity model, which defines the nodes and links present in the simulated network. The backbone portion of the connectivity model comes from actual traces, and therefore represents a subset of the actual Internet backbone. Lower-tiered ISPs and their home users are synthesized based on typical structures of cable and DSL networks, and actual population and device usage distributions. Because of the hierarchical nature of these types of networks, no home user adjacencies need to be stored (saving a significant amount of memory), since relationships between users, such as hop counts, can be calculated. From these hop counts, we can generate realistic delays based on distributions around delays we observed in actual traces, and then estimate available end-to-end bandwidth.

In order to simulate at such a large scale, the simulator needs to be further abstracted, and must use an efficient and compact underlying data structure. This is accomplished in part by simulating flows (at the slice-level) rather than simulating at the packet-level. Further, the BitTorrent data structure saves memory and processing time by eliminating redundant data throughout the system, and also by not storing or transferring any data payloads for the file transfer. Instead, we make adjustments based on the size of the data that would have been transferred.

BASS [15] is a hybrid server/P2P streaming system for large-scale VoD. In BASS, clients can stream via BitTorrent connections and media servers simultaneously. File pieces are downloaded from a server sequentially, with the exception of pieces already obtained using BitTorrent. Similarly, the BitTorrent piece-picker will not choose to download pieces scheduled prior to the current playback point, as they have already been obtained. In [15], a P2P contribution rate of 34% has been reported for a scenario of 350 peers distributing a 692 piece file (at 1 Mbps).

For our model to simulate BASS, two new entities needed to be added. The first is a *streaming server*. A streaming server is an LP that represents a highly-capable peer, which answers all requests in FIFO order. With the exception of using the same piece/slice scheme, a server does not run the BitTorrent protocol (it does not choke peers). The implementation allows for any number of streaming servers in the system. It should be noted that having these servers in an environment with malicious or selfish peers would require new security considerations. The second entity added to the model is a *streaming peer*. This peer is a BitTorrent peer, but is also responsible for keeping track of buffer state and playback deadlines. Streaming peers may co-exist and cooperate with non-streaming peers in a simulation run. Algorithm 1 demonstrates the minor modifications required at birth to accommodate streaming peers.

Algorithm 1: BIRTH Event
```
1  if $streaming\_peer$ then
2      Initialize buffers;
3      Schedule PLAYBACK now;
4  end
   // All peers
5  Initiate BitTorrent protocol;
```

A streaming peer employs a double-buffering scheme consisting of a *playback buffer* and a *look-ahead buffer* (see Figure 1). The playback buffer contains all video data that will be played next. If this buffer is not full when required, a re-buffer is triggered, causing the remaining buffer slices to be requested from a server. In addition, all remaining slices in the look-ahead buffer are also requested (see Algorithm 2). The purpose of the look-ahead buffer is to download pieces that have not missed their deadlines yet, but are needed soon. The chance that these pieces will be downloaded via BitTorrent is small, and the goal is to reduce the amount of re-buffers necessary, and the total buffering time. Data coming from a server is treated in the same way as data coming from a BitTorrent peer. This forces a peer to send out HAVE messages for pieces downloaded from a server. Further, the BitTorrent piece-picker does not need to be modified, since a peer will not request data that it already has, regardless of where it came from.

In the event of a re-buffer, the client will play the content as soon as it becomes available. Whenever content is successfully played, the next playback is scheduled for the point in the future when the current playback finishes. To initiate the first buffering and playback, a streaming peer has a playback event scheduled at its birth time. Clearly, the size of the buffers impact both the QoS and the distribution costs (larger buffers result in less P2P contribution but better QoS).

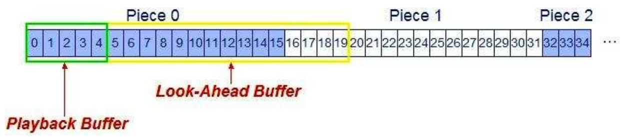

Figure 1. Double-buffering scheme. The playback buffer is 5 slices and the look-ahead buffer is 15 slices in this example.

Algorithm 2: PLAYBACK Event
```
   // Check playback buffer
1  for $i = 1$ to $playback\_buffer.size$ do
2      if $playback\_buffer[i].missing$ then
3          Request $playback\_buffer[i].piece$;
4          Set $missed\_playback$;
5      end
6  end
   // Check look-ahead buffer
7  for $j = 1$ to $lookahead\_buffer.size$ do
8      if $lookahead\_buffer[j].missing$ then
9          Request $lookahead\_buffer[j].piece$;
10     end
11 end
   // Handle the successful or
      unsuccessful playback
12 if $missed\_playback$ then
13     if Last playback was successful then
14         Store $first\_unsuccessful\_time$;
15     end
16 else
17     if Last playback was unsuccessful then
18         Increment $rebuffers$;
19         Update $buffer\_time$;
20         Clear $first\_unsuccessful\_time$;
21     end
       // Playback was successful
22     Update both buffers;
23     Schedule next PLAYBACK event;
24 end
```

4. Simulation results

In this section, we demonstrate system performance, cost savings, and server utilization (average and peak) for several streaming scenarios using our model of BASS.

In the following experiments, the distributed video consists of 512 pieces at 700 Kbps (approximately 24 minutes, which is the average length of a television show without commercials). The simulated swarms have 16,384 (flash crowds of 1,024 and 2,048) and 32,768 (flash crowds of

2,048 and 4,096) peers, where the flash crowds consist of peers that are not streaming, consistent with the subscription model for new content. Through prior experimentation, it was determined that the flash crowds should be at least 1,024 peers. If the flash crowd is too small, there is not enough content in the network to satisfy the streamers, and buffering times are much higher. Further, our results suggest that the flash crowd does not need to be very large, since larger flash crowds do not impact the buffering times, the number of re-buffering events, or the P2P contribution. The peers in each swarm arrive for approximately two hours.

Different IP Television (IPTV) services, whether they are server-only or P2P, offer a wide range of bit rates. For example, PPLive [37] typically offers bit rates between 250 Kbps and 800 Kbps [18], and the server-oriented NeuLion [32] offers playback at 700 Kbps. For this reason, no single bit rate is more suitable than any other in our study, and we chose 700 Kbps, which is roughly average amongst those we have seen (later on we explore a higher bit rate of 1.5 Mbps). VCD quality video is typically 1.25 Mbps, DVD quality is 5 Mbps, and HDTV is 15 Mbps [4].

In terms of our network topology and BitTorrent model configuration, there are over 100 different parameters. Here, we provide a brief overview of the most important. Each host computer has upstream and downstream bandwidths of 64 KBps and 256 KBps respectively for DSL subscribers, and 128 KBps and 512 KBps respectively for cable subscribers. Each peer can support up to 80 connections (default BitTorrent setting), which are established using BitTorrent's random peer selection algorithm from a single tracker. Each connection's data rate is based on the TCP network bandwidth equation [30], which is a function of the maximum segment size (1380 bytes in BitTorrent), round trip time as determined by the path delay between peers, and a probability of packet loss, which is set to 5%. The path delay between peers is determined by the hop count between them (assuming a nominal 6 ms delay per hop). The topology, delay per hop, and loss probability were determined by real measurement data taken from home broadband networks and various looking glass sites. This data enabled us to efficiently construct a reasonable Internet topology model for the top 31 Nielsen markets, which includes over 100 million TV viewers [33]. As evidence of this efficiency, we find that a 131,072 peer scenario requires only 6.5 GB of memory to represent the network topology. All simulations were executed on a 16-way AMD Opteron 2.8 GHz system with 64 GB of RAM, using only 1 of the 16 processors. The full details of the BitTorrent model and Internet topology model can be found in [8] and [28] respectively.

As mentioned above, the main discriminating variable is the size of the look-ahead buffer. In our experiments, the size of this buffer ranges from 500 to 3,000 slices by increments of 500. If the buffer is too small, the P2P contribution is very high, however the buffering times are very poor. Further, if the buffer is very large, there is a drastic decrease in P2P contribution, with only a marginal increase in quality.

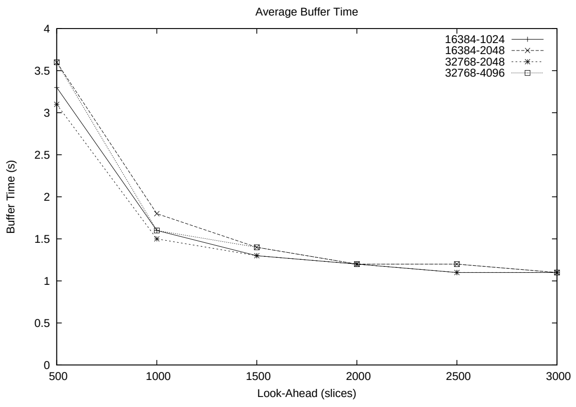

Figure 2. Average buffer times experienced by streaming peers as a function of look-ahead buffer size (in slices). Swarm scenarios range from 16,384 total peers with 1,024 peers in the initial flash crowd (legend label: $16384 - 1024$**) to 32,768 total peers with 4,096 peers in the initial flash crowd.**

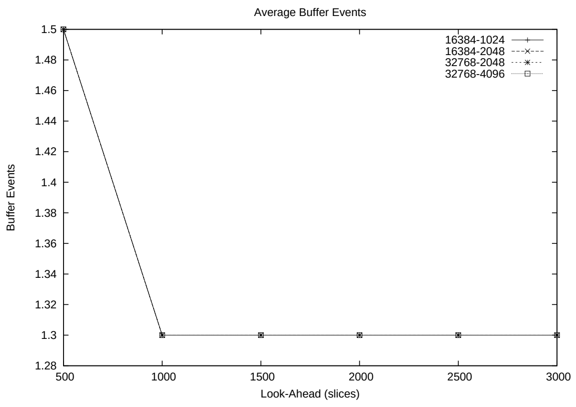

Figure 3. Average number of buffer events experienced by streaming peers as a function of look-ahead buffer size (in slices). Note that the first buffer event is mandatory for all streaming peers. Swarm scenarios range from 16,384 total peers with 1,024 peers in the initial flash crowd (legend label: $16384 - 1024$**) to 32,768 total peers with 4,096 peers in the initial flash crowd.**

In each run, a look-ahead buffer of size 500 results in an average buffering time of 3.3 seconds (1.5 average buffer events per user) with a P2P contribution of 78% (see Figures 2, 3, and 4 respectively). While this sounds good, 3.3

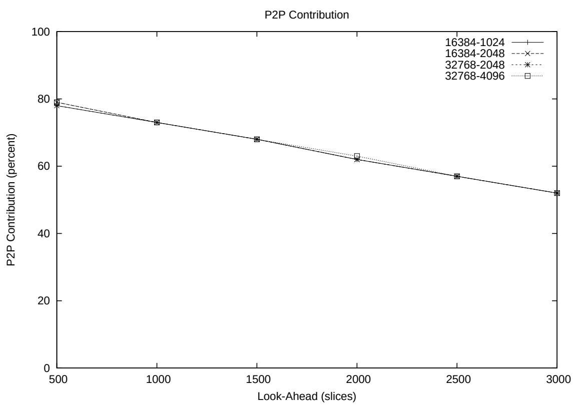

Figure 4. Percentage of bandwidth contributed by the P2P network as a function of look-ahead buffer size (in slices). Swarm scenarios range from 16,384 total peers with 1,024 peers in the initial flash crowd (legend label: $16384 - 1024$) to 32,768 total peers with 4,096 peers in the initial flash crowd.

Simulated Peers	Avg. Buffer Time (s)	Avg. Buffers	P2P Contribution	Avg. CDN Util. (MBps)	Peak CDN Util. (MBps)
16,384	1.8	1.3	73%	104	145
32,768	1.5	1.3	73%	158	312
65,536	1.4	1.3	73%	314	617
131,072	1.4	1.3	73%	633	1,228

Table 1. Streaming performance across several metrics as a function of swarm size ranging from 16,384 peers to 131,072 peers. The flash crowd is 2,048 peers and the look-ahead buffer is 1,000 slices.

seconds may be too long for some applications. A buffering time of under 2 seconds (1.3 average buffer events per user) with up to a 73% contribution from the P2P network can be achieved. This time can be further reduced by 0.4 seconds, but doing so reduces the P2P contribution by 16%. Depending on the distributor's budget and needs, the size of the look-ahead buffer can be established. For an average buffering time of 1.1 seconds, it is still possible to achieve a P2P contribution of 53%. Thus, using BitTorrent to assist a streaming server or CDN infrastructure can significantly lower transit costs, while achieving an excellent QoS for users.

4.1. QoS

Although the average user buffer times and number of buffer events are very good, we would like to know how all peers fare. Consider the case of 16,384 peers, a flash crowd of 1,024, and a look-ahead buffer of 1,500 slices. The average buffering time is 1.3 seconds (with a standard deviation of 2.6 seconds). The histogram in Figure 5 shows that most peers do indeed experience good performance, with 99.76% of peers experiencing a buffer time of under 3 seconds. Similarly, on average, each peer requires only 1.3 buffering events (with a standard deviation of 0.6 buffer events), with 99.15% of peers requiring at most 2 buffer events (including the initial mandatory buffering, see Figure 6).

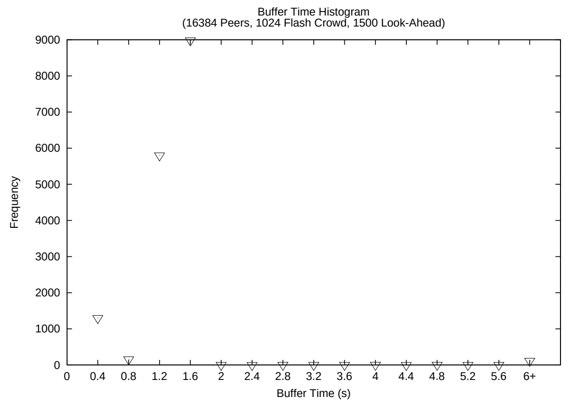

Figure 5. Histogram of the buffering times. Observe that most streaming peers experience a buffering time of under 2 seconds.

Overall, streaming quality can be measured by a QoS metric called *adjusted frustration time* [25]. Adjusted frustration time is defined as the total sum of buffering times, plus a 2 second penalty for every re-buffering event. This metric is used as part of the StreamQ user performance rating system. In this system, an adjusted frustration time of under 6 seconds is given the grade of A+, 6 to 9 seconds an A, 9 to 12 seconds a B+, and so on. Figure 7 is a histogram of the swarm's adjusted frustration times (the average is 1.8 seconds, with a standard deviation of 3.1 seconds, and 99.12% of users experience an adjusted frustration time of under 3.6 seconds), indicating that the overall QoS is very good. The StreamQ user performance ratings for this run can be found in Table 2.

4.2. CDN utilization

Figures 8 and 9 show the average and peak CDN server utilizations for each scenario. The peak never exceeds 390 MBps, and the averages are usually around half the peaks. Further, for the simulated scenarios, the server utilization

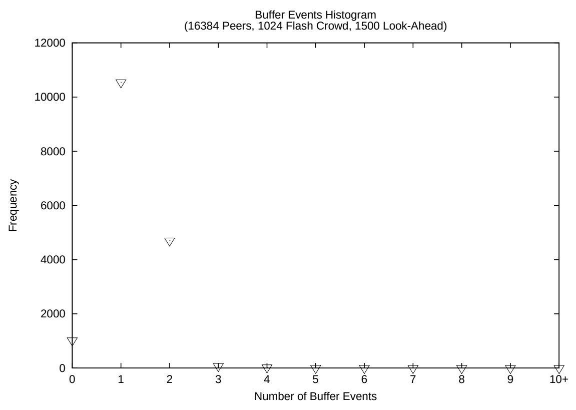

Figure 6. Histogram of the number of buffering events. Observe that most streaming peers experience less than 3 buffer events.

Grade	Frequency
A+	16,245
A	21
B+	15
B	13
C+	18
C	15
D+	5
D	10
F	42

Table 2. Distribution streaming peers grades using the StreamQ performance rating system. Note, that a grade of F is given when a peer's adjusted frustration time is 27 seconds or more.

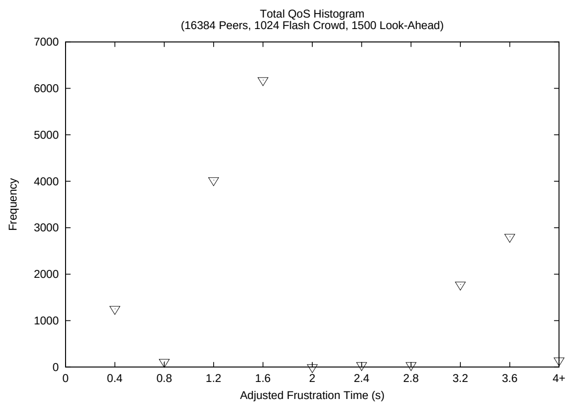

Figure 7. Histogram of the adjusted frustration times. Observe that most streaming peers experience an $A+$ StreamQ performance.

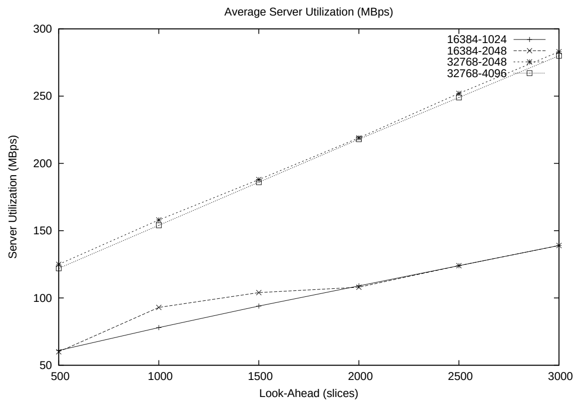

Figure 8. Average server utilization as a function of look-ahead buffer size (in slices). Swarm scenarios range from 16,384 total peers with 1,024 peers in the initial flash crowd (legend label: $16384 - 1024$) to 32,768 total peers with 4,096 peers in the initial flash crowd.

scales roughly linearly with the size of the swarm. Thus, if a content provider can estimate swarm sizes, an excellent estimate of server requirements can be made.

Most CDNs and ISPs use a method called *burstable billing* [6] to charge their customers. This method charges based on a regular sustained utilization, allowing brief usage peaks to occasionally exceed the threshold without penalty. Typically, customers are billed at the 95^{th} percentile of their usage. This method is beneficial for customers whose usage is fairly steady. If usage is bursty or unpredictable, a flat-rate system that charges per byte (or GB) delivered may be the best option. Table 3 shows the 95^{th} percentiles and the total volume of data delivered with and without the assistance of the P2P network.

We confirm the claims published in [15] that most CDN contribution occurs in the first pieces of the file, with very little towards the end. This is due to the fact that BitTorrent (which usually employs a rarest piece first algorithm) does not have a chance to obtain early pieces because they are needed too soon, while the later pieces have more time before playback, and thus more opportunities to be downloaded via BitTorrent. Figure 10 shows how many times a slice (16 KB) is downloaded from the server, for each piece (256 KB) of the file.

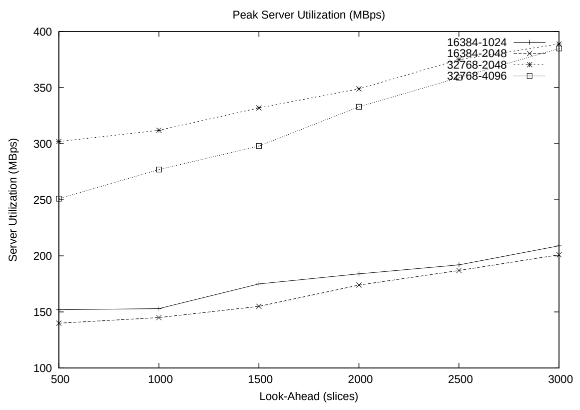

Figure 9. Peak server utilizations as a function of look-ahead buffer size (in slices). Swarm scenarios range from 16,384 total peers with 1,024 peers in the initial flash crowd (legend label: $16384 - 1024$) to 32,768 total peers with 4,096 peers in the initial flash crowd.

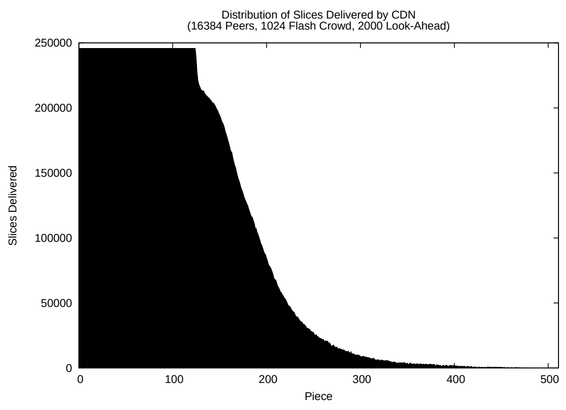

Figure 10. Distribution of slices delivered by the CDN throughout the file, for the 16,384 peer, 1,024 flash crowd, and 2,000 slice look-ahead scenario.

Simulated Peers	95th Percentile CDN Util. (MBps)	CDN Delivered (GB) With P2P Network	CDN Delivered (GB) Without P2P Network
16,384	112	567	2,098
32,768	224	1,133	4,195
65,536	442	2,265	8,389
131,072	922	4,530	16,778

Table 3. 95^{th} percentiles and the total volume of data delivered (GB) by the CDN with and without the assistance of the P2P network for several large swarms (16,384 peers to 131,072 peers). The flash crowd size is 2,048 peers and a look-ahead buffer is 1,000 slices.

Simulated Peers	Bit Rate (Kbps)	Look-Ahead (slices)	Avg. Buffer Time (s)	Avg. Buffers
16,384	700	1,500	1.4	1.3
16,384	1,500	2,000	1.4	1.3
32,768	700	1,500	1.3	1.3
32,768	1,500	2,000	1.4	1.3

Table 4. Impact of bit-rate and look-ahead buffer size on QoS (buffer times and re-buffers) across different swarm sizes.

4.3. Video bit rate

To this point, all results in this paper have been for a bit rate of 700 Kbps. In this section, we present results from simulations at 1.5 Mbps (a 12 minute video), and show that with a nominal increase to the look-ahead parameter, it is possible to achieve a similar QoS and CDN requirements. Table 4 shows that an increase of 500 slices to the look-ahead buffer will allow us to achieve the same QoS as the 700 Kbps scenario (this is the point where both curves begin to converge to 1.1 seconds and 1.3 buffer events). Table 5 shows the P2P contribution and the server utilizations for these same scenarios. To achieve the same QoS, more CDN involvement is required for the higher bit rate, which is expected since the only difference is that now all playback deadlines occur sooner.

When the look-ahead buffer is 500 slices for all scenarios (the worst case for both bit rates), the average buffering time for the 1.5 Mbps video is approximately twice that of the 700 Kbps video (a maximum of 7.2 seconds compared to 3.6 seconds), and there are on average 0.6 more buffer events per user. For any look-ahead buffer size, the P2P contribution is consistently 10 to 13% less for the higher bit rate. While the average and peak server utilizations appear to go down occasionally, they typically increase for the higher bit rate by 30 Mbps and 60 Mbps respectively for the 16,384 peer swarms, and by 60 Mbps and 90 Mbps respectively for the 32,768 peer swarms.

These results show that while P2P contribution decreases and server utilization increases (an increase in overall CDN involvement), the same QoS can be achieved at higher bit rates as can with lower ones.

Simulated Peers	Bit Rate (Kbps)	Look-Ahead (slices)	P2P Contribution	Avg. CDN Util. (MBps)	Peak CDN Util. (MBps)
16,384	700	1,500	68%	104	155
16,384	1,500	2,000	50%	145	239
32,768	700	1,500	68%	188	332
32,768	1,500	2,000	51%	143	225

Table 5. P2P contribution and CDN utilization as a function of swarm size, bit rate, and buffer size.

5. Conclusions

In this paper, we have demonstrated the feasibility of large-scale P2P assisted video streaming, and discussed its inherent tradeoffs. Through a comprehensive campaign of simulation scenarios, we have shown that a content distributor can fine-tune parameters in order to reduce CDN utilization, making it possible to distribute files to large audiences on a finite budget. One simulation run shows that a file can be distributed to 131,072 users with a CDN savings of 73%, while providing users with an average buffering time of fewer than 2 seconds (an A+ on the StreamQ performance rating system).

In the future, we will modify the simulated clients to have dynamic behavior, in order to study network impacts and security considerations. Another direction would look at local caching strategies, so that a distributor can also offer long-tail (less popular) content while continuing to save in transit costs.

Since our study began, we have seen an increase in the adoption of online video streaming (for example, the 2008 NCAA March Madness tournament [42] and the 2008 Summer Olympics in Beijing [23]). As online audiences continue to grow, more content owners will use a P2P network to assist in distribution in order to reduce their data transit costs. We have demonstrated how this can be accomplished while keeping viewers satisfied, and have paved the way for further research of this technology.

References

[1] N. Anderson. Nocturnal P2P Transmissions Account for 95 Percent of Internet Traffic. `http://arstechnica.com/news.ars/post/20071128-nocturnal-p2p-transmissions-account-for-95-percent-of-internet-bandwidth.html`.

[2] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez. Is High-Quality VoD Feasible using P2P Swarming? In *Proceedings of the 16th International conference on World Wide Web*, pages 903–912, Banff, Alberta, Canada, 2007.

[3] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving BitTorrent Performance. Technical Report, MSR-TR-2005-03, Microsoft Research, February 2005.

[4] Bit Rate. `http://en.wikipedia.org/wiki/Bitrate`, accessed August 22, 2008.

[5] BitTorrent. `http://www.bittorrent.com`.

[6] Burstable Billing - Wikipedia, November 2007. `http://en.wikipedia.org/wiki/Burstable_billing`.

[7] N. Carlsson and D. L. Eager. Peer-Assisted On-Demand Streaming of Stored Media using BitTorrent-Like Protocols. In *Proceedings of IFIP/TC6 Networking 2007*, pages 570–581, Atlanta, GA, May 2007.

[8] C. D. Carothers, R. LaFortune, W. D. Smith, and M. R. Gilder. A Case Study in Modeling Large-Scale Peer-to-Peer File-Sharing Networks using Discrete-Event Simulation. In *Proceedings of the International Mediterranean Modeling Multiconference*, pages 617–624, Barcelona, Spain, October 2006.

[9] C. D. Carothers, D. Bauer, and S. Pearce. ROSS: Rensselaer's Optimistic Simulation System User's Guide. Technical Report, #02-12, Department of Computer Science, Rensselaer Polytechnic Institute, 2002.

[10] C. Carothers, D. Bauer, and S. Pearce. ROSS: A High-Performance, Low Memory, Modular Time Warp System. In *Proceeding of the 14th Workshop on Parallel and Distributed Simulation*, pages 53–60, May 2000.

[11] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Multicast in a Cooperative Environment. In *SOSP'03*, Lake Bolton, New York, October 2003.

[12] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai. Improving VoD Server Efficiency with BitTorrent. In *Proceedings of the 15th international conference on Multimedia*, pages 117–126, Augsburg, Germany, 2007.

[13] Y. H. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *Measurement and Modeling of Computer System*, pages 1–12, 2000.

[14] Bram Cohen. Incentives Build Robustness in BitTorrent. In *The Workshop on Economics of Peer-to-Peer Systems* , Berkeley, CA, June 2003.

[15] C. Dana, D. Li, D. Harrison, and C. Chuah. BASS: Bittorrent Assisted Streaming System for Video-on-Demand. In *International Workshop on Multimedia Signal Processing IEEE Press*, 2005.

[16] TJ Giuli and Mary Baker. Narses: A Scalable Flow-Based Network Simulator. *ArXiv Computer Science e-prints*, CS0211024, November 2002.

[17] M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev. CollectCast: A Peer-to-Peer Service for Media Streaming. *ACM/Springer Multimedia Systems Journal*, October 2003.

[18] X. Hei, C. Liang, J. Liang, Y. Liu, and K.W. Ross. Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System. In *Proceedings of IPTV Workshop, International World Wide Web Conference*, 2006.

[19] ITIVA NETWORKS. `http://www.itiva.com`.

[20] V. Janardhan and H. Schulzrinne. Peer Assisted VoD for Set-Top Box Based IP Network. In *Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*, pages 335–339, Kyoto, Japan, 2007.

[21] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole Jr. Overcast: Reliable Multicasting with an Overlay Network.

[22] D.R. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[23] Tim Johnson. As TV Ratings Soar, Olympics Organizers Flex Muscles, Look Beyond Beijing. `http://www.miamiherald.com/sports/sports-wire/story/649490.html`, accessed August 22, 2008.

[24] Joost. `http://www.joost.com`.

[25] Keynote Systems - Hosted Streaming Quality Measurement, 2006. `http://www.keynote.com/products/voip_and_streaming/streaming_performance/streaming_perspective_stremq.html`.

[26] Kontiki Delivery Management System. `http://www.verisign.com/products-services/content-messaging/broadband- delivery/kontiki-delivery-management`.

[27] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of ACM SOSP*, 2003.

[28] R. LaFortune, C. D. Carothers, W. Smith, and M. Hartman. An Abstract Internet Topology Model for Simulating Peer-to-Peer Content Distribution. In *Principles of Advanced and Distributed Simulation*, San Diego, CA, June 2007.

[29] A. Legout, G. Urvoy-Keller, and P. Michiardi. "Understanding BitTorrent: An Experimental Perspective". Technical Report, INRIA, Eurecom, France, November 2005.

[30] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), 1997.

[31] MySee. `http://www.mysee.com`.

[32] NeuLion Inc. `http://www.neulion.com/technology/technology.php`.

[33] Nielsen Media – Home Page, 2006. `http://www.nielsenmedia.com/dmas.html`.

[34] Pando. `http://www.pando.com`.

[35] A. Parker, "The True Picture of Peer-to-Peer File-Sharing", `http://www.cachelogic.com/research/slide1.php`.

[36] A. Parker, "P2P in 2005", `http://www.cachelogic.com/research/2005_slide01.php`.

[37] PPLive. `http://www.pplive.com/en/index.html`.

[38] PPStream. `http://www.ppstream.com`.

[39] Red Swoosh, an Akamai Company. `http://www.akamai.com/redswoosh`.

[40] R. Rejaie and A. Ortega. PALS: Peer-to-Peer Adaptive Layered Streaming. In *Proceedings of ACM NOSSDAV*, pages 153–161, June 2003.

[41] Roxbeam. `http://www.roxbeam.com`.

[42] Stream Line: CBS To Provide Live Online Broadcast Of Final Four. `http://www.sportsbusinessdaily.com/article/119776`, accessed August 22, 2008.

[43] S. Tewari and L. Kleinrock. Analytical Model for BitTorrent-Based Live Video Streaming. In *Proceedings of IEEE NIME 2007 Workshop*, Las Vegas, NV, Jan 2007.

[44] Duc A. Tran, Kien A. Hua, and Tai T. Do. A Peer-to-Peer Architecture for Media Streaming. *Journal on Selected Areas in Communications, Special Issue on Advances in Service Overlay Networks*.

[45] UUSEE. `http://www.uusee.com`.

[46] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. *IEEE INFOCOM 2006 Global Internet Workshop*, April 2006.

[47] G. Wen, H. Longshe, and F. Qiang. Recent Advances in Peer-to-Peer Media Streaming Systems. In *China Communications*, October 2006.

[48] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. *Technical Report, UCB/CSD-01-1141, UC Berkeley*, April 2001.

[49] X. Zhang, J. Liu, B. Li, and T-S. Peter Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In *Proceeding of IEEE/INFOCOM*, Miami, FL, March 2005.

[50] S. Zhuang, B. Zhao, A. Joseph, R. Katz, J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video*, June 2001.