Improving VoD Server Efficiency with BitTorrent *

Yung Ryn Choe, Derek L. Schuff, Jagadeesh M. Dyaberi, and Vijay S. Pai
Purdue University
West Lafayette, IN 47907
yung@purdue.edu, dschuff@purdue.edu, jdyaberi@purdue.edu, vpai@purdue.edu

ABSTRACT

This paper presents and evaluates Toast, a scalable Video-on-Demand (VoD) streaming system that combines the popular BitTorrent peer-to-peer (P2P) file-transfer technology with a simple dedicated streaming server to decrease server load and increase client transfer speed. Toast includes a modified version of BitTorrent that supports streaming data delivery and that communicates with a VoD server when the desired data cannot be delivered in real-time by other peers.

The results show that the default BitTorrent download strategy is not well-suited to the VoD environment because it fetches pieces of the desired video from other peers without regard to when those pieces will actually be needed by the media viewer. Instead, strategies should favor downloading pieces of content that will be needed earlier, decreasing the chances that the clients will be forced to get the data directly from the VoD server. Such strategies allow Toast to operate much more efficiently than simple unicast distribution, reducing data transfer demands by up to 70–90% if clients remain in the system as seeds after viewing their content. Toast thus extends the aggregate throughput capability of a VoD service, offloading work from the server onto the P2P network in a scalable and demand-driven fashion.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems–Distributed Applications; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks–Internet; H.5.1 [**Multimedia Information Systems**]: Video

General Terms

Performance, Experimentation, Design, Measurements

*This work is supported in part by the National Science Foundation under Grant Nos. CCF-0532448, CNS-0532452, and CCF-0621457.

MM'07, September 23–28, 2007, Ausburg, Bavaria, Germany.

Keywords

Video-on-Demand, Multimedia Streaming, BitTorrent, Peer-to-Peer, Experimental Systems

1. INTRODUCTION

With the proliferation of inexpensive broadband connections, many applications have arisen to take advantage of this widespread bandwidth. One of the most commercially important and technically challenging applications is Video-on-Demand (VoD) service for high-quality, full-length movies. For example, the most recent annual report of the largest cable company (Comcast) starts with the sentence "On Demand is in" and states "Our growing ON DEMAND library attracted more than 1.4 billion views in 2005, nearly a 150 percent jump over the previous year" [9]. This is an average of 62 views per year per customer household. These numbers can only be expected to increase as higher bandwidth links proliferate and library capacity increases, allowing a greater number of high-quality video selections for consumers. Forms of VoD range from news and entertainment clips on the Internet to high-quality full-length movie services offered by ISPs or cable companies.

Numerous systems have targeted VoD through powerful storage servers that use disk striping and scheduling to multiplex a large number of distinct request streams across a storage array while meeting real-time network data delivery targets [3, 6, 15, 21, 27]. However, VoD systems may also experience difficulties in achieving high throughput as a result of their network data delivery method. Whether they use UDP, TCP, or another transport protocol, VoD servers communicate with each client separately since each client may request a different piece of content, and since even multiple clients requesting the same piece of content may be at different points in the stream. This scenario potentially requires the server to deliver the same content repetitively as it transmits data separately to each client.

An alternative model for data delivery on high-bandwidth communication channels is peer-to-peer (P2P) communication, which makes all participants perform both client and server functions. Using P2P technology can greatly reduce the cost of distributing content, because the peers contribute their resources as well. Among various P2P systems, the most popular is BitTorrent. BitTorrent breaks each file into pieces to improve the efficiency and speed of file transfers. Participants seeking a particular file form a *swarm*. Entries in the swarm that have the complete file are called *seeds*. Members that join the swarm obtain data pieces from seeds and other new members depending on data availability, while also providing data to other new members as well. A "tit-for-tat" policy aims to insure that every participant must also provide service to others. Once a member has the full file, it also becomes a first-class seed and remains such until its user closes the BitTorrent session.

By allowing an individual client to download pieces of a file simultaneously from multiple sources, BitTorrent improves user latency while also avoiding bottlenecks at a centralized server.

BitTorrent has many uses, such as distribution of Linux ISOs and other software updates. It is also extensively used for video data transfer (including piracy, though this use is officially discouraged) [30]. Despite its use in video environments, however, BitTorrent is fundamentally based on a download model rather than a streaming model: actual end-user access to the media content is not possible until the entire video has arrived. The default BitTorrent policy is for a given swarm member to prefer to get the rarest piece of data available from another swarm member. This is beneficial for keeping more copies of this rare piece in the swarm and allowing other members to pull this data from multiple sources, but is not necessarily well-suited to the goals of the end-user who actually wants to view the data in real-time.

This paper presents a new system called Toast (Torrent Assisted Streaming) which aims to improve the efficiency of VoD servers by using P2P delivery through a video-targeted version of BitTorrent. The BitTorrent version in Toast has been adapted to support streaming real-time data delivery and to communicate with a traditional back-end VoD server when none of the swarm participants is able to provide the desired data in time for the user to view it. In essence, a P2P network acts as a distributed cache for a VoD server, offloading work from the server in a scalable and demand-driven fashion. The VoD server remains in the system, however, to allow clients to receive data pieces in real-time even when BitTorrent swarms cannot provide those pieces efficiently. Though Toast could be used on the Internet for the type of content seen on popular video sites such as YouTube, the focus of this paper is higher-quality streams and a faster network; for example, a cable company could deploy a system like Toast on its set-top boxes.

The contributions of this paper are as follows. First, this paper describes the design and implementation of Toast, providing experimental verification of its basic principles. Second, this paper explores the strategies by which BitTorrent picks the next piece of content to download. The results show that the default BitTorrent policy is not well-suited to the VoD environment. Instead, policies should favor downloading earlier pieces of content from other swarm participants to make it more likely that the end clients will not have to get the data from the VoD server instead. Third, this paper explores the impact of seeding on the effectiveness of Toast. The experimental results show that intelligent piece picking and persistent client seeding allow Toast to be quite effective, offloading 70–90% of the network traffic from the VoD server onto the P2P network (depending on the upload bandwidth available at the clients).

Section 2 discusses background material on the BitTorrent system and on VoD. Section 3 presents the overall Toast system and discusses the client and server implementations, along with the various client options that are tested. Section 4 discusses overall evaluation goals, the testbed, and the methodology used to investigate Toast. Section 5 presents and discusses our findings, and Section 6 concludes the paper.

2. BACKGROUND

2.1 BitTorrent

BitTorrent has become the most popular file distribution protocol on the Internet. This is primarily due to its efficiency and speed in transferring files. Previous P2P systems usually consisted primarily of a method to search for and locate files shared on the network. Once found, a peer simply requested the file from another peer, which transferred it using HTTP or a similar protocol. These systems were primarily differentiated by their methods of locating content, but were all similar with respect to their transfer methods. BitTorrent on the other hand ignores the search problem. Instead, it relies on web sites or other common distribution methods to distribute small files called *torrent* files (sometimes called "dot torrent" files, due to their filename extension), each of which is essentially a descriptor of a file or group of files to be downloaded. Each file to be distributed has its own torrent file, and the group of clients downloading a particular torrent is called a *swarm*. Each swarm is independent and self-contained, but individual clients may participate in more than one swarm at a time. The swarm is managed by a simple network server called a *tracker*, which is responsible for keeping track of all clients in the swarm, and informing clients about each other. The tracker does not upload or download any file data; to begin file distribution requires at least one client which has the entire file, and which will upload to other clients in the swarm. Such a client is called a *seed*, and downloading clients which do not yet have the whole file are called *peers* or *leechers* (we will use the general term *client* to refer to either a peer or a seed). Peers become seeds once they have the whole file, and there are no distinctions between the seeds. (In particular, there are no differences between the original seed run by the original distributor of the file and other clients that have become seeds and are still participating in the swarm.) In most cases, once they have the entire file, clients will continue to participate as seeds until the user closes them.

There are two major innovations in the BitTorrent approach. The first is that each file is split into a number of small pieces (often 256 kB), and these pieces are transferred out of order. This means that peers that have different pieces of the file can exchange them, and that a peer can download different pieces from several other peers at once. In fact, since transfers are made at a granularity even smaller than the piece size (usually 16 kB), even a single piece can be downloaded from several peers at once. This can greatly increase the speed at which a file is transferred compared to simply downloading all of it from a single peer.

The second important feature of BitTorrent is an incentive strategy designed to reduce the impact of "freeloaders" (clients who download a lot of data but rarely or never upload anything) [8]. Such freeloaders are common in many P2P networks and do not contribute positively to overall system performance. In the BitTorrent system, each client maintains connections with many others, but is not necessarily willing to upload to all of them. Remote clients which have active connections with the local client, but to which the local client is not willing to upload, are said to be *choked*; all remote clients start out choked. A remote client may be randomly selected to be unchoked (called "optimistic unchoking"), or may be unchoked when the local client recieves file data from it. Thus, sending pieces of the file to remote clients increases the performance seen by the local client. This strategy is called *tit-for-tat* because of its reactive nature; similar strategies have been shown effective in solving a variety of optimization problems in game theory [2].

Using the above mechanisms, BitTorrent deals with heterogeneity in peer upload rates in two different ways. First, the tit-for-tat scheme rewards peers that have higher upload rates and are thus contributing more to the swarm. Second, the ability to download parts of the same piece from multiple peers at the same time not only accelerates the download process, but also shields the downloader from the possibility of waiting too long on a slow uploader.

For BitTorrent to operate effectively, peers communicating in a swarm need to have different sets of pieces so that they can exchange them. The choice of which piece to request from another

peer can thus be critically important. Selecting pieces from the other peer's set uniformly at random generally does a good job of maintaining this "piece diversity" throughout the system. However, the standard BitTorrent client uses a "rarest-first" policy in which the client keeps track of how many copies of each piece exist among its peers, and selects the pieces with the fewest copies. Preferentially choosing the rarest piece has three benefits. First, it helps to ensure that all the pieces will still be available if all the seeds leave the network. Second, this scheme also improves the aggregate upload bandwidth available for the chosen piece since this peer will now be able to provide the piece to other swarm members. Third, it can maximize the possibility that a peer has something to exchange with other peers since it is unlikely that other peers have this content as well. Otherwise, other peers do not have the incentive to serve this peer due to the tit-for-tat nature of the algorithm.

To maintain piece diversity, the standard client uses random selection until a specified number of pieces have been downloaded (4 by default), and then switches to rarest first, randomly selecting among pieces with the same rarity.

2.2 Video-on-Demand

Video-on-Demand (VoD) has long been a research goal for system architecture, networking, and audio/video coding researchers, and hundreds of systems and solutions have been developed in these areas. A common way of implementing a VoD server is to use unicast and send each client a copy of the media, using one of several protocols designed for this purpose (e.g., RTP and RTSP [25, 26]). However, this unicast approach is inefficient with hundreds or thousands of clients. By taking advantage of the fact that the same files are requested by many of the clients, many techniques have been developed using multicast for nearly on-demand viewing, or using multiple unicast or multicast streams to reduce server load while still providing true on-demand service. Such schemes include patching, staggered broadcasting, hierarchical multicast stream merging, adaptive piggybacking, and periodic broadcast protocols [4, 11, 14, 17, 31]. However, IP multicast is rarely seen on the Internet or even intra-ISP networks, so these solutions have not had much impact.

Many newer techniques for video data delivery are based on some form of peer-to-peer or overlay multicast technology, all with their own protocols to manage peer communication and organization [10]. The first such system uses linear chains of clients to achieve functionality similar to IP multicast and uses these chains to implement a generalized batching technique for on-demand video [28]. GloVE combines these chaining and batching techniques, allowing multiple streams of data between different clients, but relies on IP multicast to make these streams efficient [13].

On-demand streaming systems such as CoopNet, PALS, PROP, and BiToS are closely related to this work. Each of these systems seeks to support an infrastucture-based system with P2P networks and thus achieve scalability and reliability. CoopNet provides both live and on-demand streaming using a multicast tree rooted at the server and divides the streaming media content into multiple substreams using *multiple description coding (MDC)* to provide robustness [22]. When CoopNet is used for on-demand streaming, the P2P network is used only when the server is overloaded. The server is required to keep track of the peers and the content held by them, and redirect requests to peers when it is overloaded. (BitTorrent and Toast clients do this by themselves, preferring to get content from peers rather than the server). PALS uses layered-encoding to allow receivers to fetch data from multiple sources, including either other peers or servers [24]. The receiver chooses sender peers in such a way as to optimize throughput and quality of service. Unlike BitTorrent and Toast, PALS distinguishes between senders and receivers and thus only uses peers as senders when they have the complete data of the stream. PROP is designed for intranets which deploy a proxy server [18]. At any time, the requesting client receives data from either a peer or the proxy server. If data is not available in the peers or the proxy, the proxy server requests the missing data from the media server. The BitTorrent approach allows each peer to retrieve pieces of the video stream from multiple other clients simultaneously. This provides robustness by allowing multiple sources for each piece in case of node failure, and provides better performance by allowing downloaders to get pieces well in advance, reducing the chance that they will have to get them from the server. BiToS actually uses BitTorrent for content distribution and balances rarest-first piece selection with a need for real-time delivery [32]. However, BiToS does not include any backing servers to guarantee real-time delivery. As a result, the measured results for all variants indicate that at least 5% of pieces are not received in time, degrading quality of service.

Other works have provided analysis and simulations of proposed peer-to-peer VoD systems backed by servers, in which the peer-to-peer network is used to reduce load from the servers. Cui et al. propose oStream, a system using overlay multicast trees to stream most of the video from peers instead of the server [10]. They offer extensive analysis and some simulation results. Huang et al. describe a peer-assisted VoD service with different fetching policies based on mathematical models of client needs and the capabilities of the server and P2P network [20]. They provide the results of a discrete-event simulation model based on real-world traces of accesses to a video to show the potential of such an approach. Chen et al. describe and simulate a system that employs topology-aware algorithms and provides economic incentives to peers that provide data, thus providing additional encouragement to contribute resources and coordinating delivery to reduce overall network traffic [5]. Dana et al. propose a system similar to Toast called BitTorrent-Assisted Streaming System (BASS) [12]. Their work uses Torrent trace data from the distribution of Fedora Core 3 and develops a simplified model of BitTorrent client performance, which they simulate for several metrics. Although each of the above simulation-based and analytical works provides valuable conceptual insights, Toast is a real implementation and thus allows for more detailed insight and analysis on actual performance issues. For example, using a real implementation allows this paper to explore various modifications to BitTorrent that better target this system for VoD.

2.3 Other Related Work

Numerous web caching systems and content distribution networks (CDNs) (e.g., Akamai) have sought to offload the delivery of streaming video content onto geographically-distributed servers. Peer-to-peer systems represent an alternative approach to this problem. P2P has advantages in dynamically provisioning resources as demand rises since each requesting node must also be a provider. If the peer nodes are actually controlled by a system administrator (e.g., cable company or PCs on a LAN), they can be set to remain as background seeds long after their viewing is complete, thus offloading work from remote network servers without requiring additional infrastructure support through a CDN.

A related problem to on-demand video streaming is that of live streaming. Live streaming accomplishes a similar goal (distributing video or other content to large number of people) but does not have the requirement that the user be able to skip to arbitrary points in the stream (for example, in the future). However, this is the primary advantage that BitTorrent exploits: the ability of clients

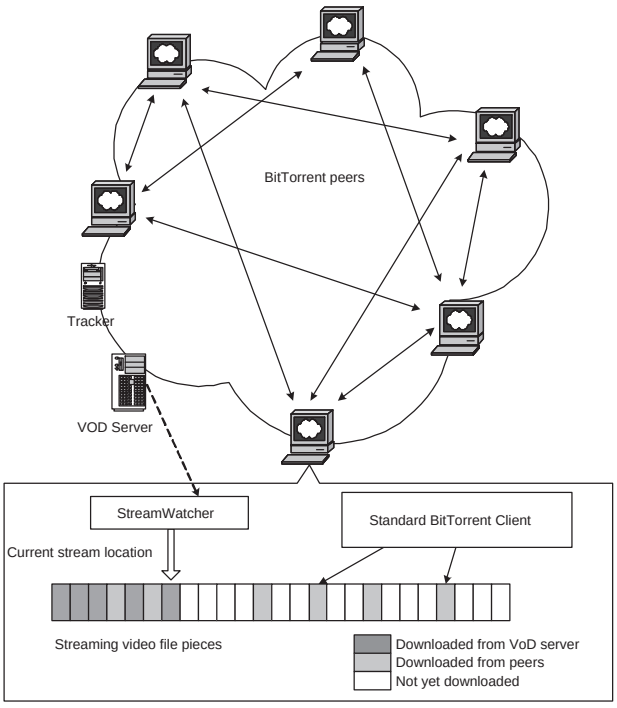

Figure 1: System and client overview

to exchange pieces from anywhere in the file. BitTorrent-based systems for live streaming have been proposed, implemented, and analyzed. Such systems typically introduce a slight delay in playback to allow for limited exchange of a small window of content pieces that have already been published but not yet distributed to all peers [23, 29]. Alternative peer-to-peer approaches include overlay-based multicast networks (e.g., [7]). These systems also provide the benefits of peer-to-peer in terms of demand-driven scalability, but BitTorrent-based systems benefit from BitTorrent's more richly-connected peering, use of multiple simultaneous peers at each peer, and tit-for-tat incentive system.

3. IMPLEMENTATION

3.1 Overview

Figure 1 shows an overview of the Toast system, along with some details of the client. The Toast system is essentially a hybrid of a modified BitTorrent and a simple unicast VoD system. The tracker and the seed client (which is optional in Toast) are unmodified. The Toast clients implement standard BitTorrent functionality, but also have an additional component. As in BitTorrent, each file has its own instance of the overall system, although some components may be shared. However, Toast also includes a VoD media server that can satisfy any client's request for a piece of the file.

The fundamental problem with BitTorrent for VoD applications is that the file pieces are downloaded out of order. This means that if a user is watching a video while it is being downloaded, it may come to a piece of the file that has not been downloaded yet (even though still later pieces of the file may have already been downloaded). If this situation arises or will arise soon, a Toast client simply makes an extra request to the dedicated VoD server, outside of the BitTorrent system, to prevent interruption of the video stream. In this way, the VoD server serves as a kind of backup for the BitTorrent system. The "on-demand" nature of VoD is still satisfied, but with much lower overhead on the server, compared to systems where the server sends the whole file itself. The guiding principle of Toast is thus to acquire data ahead of time from the peers whenever possible while using the VoD server as a source of data only if the peers do not have the data or cannot supply it in real-time. This goal is different from that of standard BitTorrent, so the software must also be modified accordingly.

3.2 Modified Client

The Toast client is based on version 4.4 of the official BitTorrent client from `http://www.bittorrent.com`, also known as the "mainline" client. This implementation was chosen from the many that exist because it was the original implementation by the author of the protocol (making it a popular de-facto standard), and it is written in Python, which facilitates relatively easy reading and modification of its code.

StreamWatcher. The primary addition is a new module called StreamWatcher, which tracks the progress of a conceptual video stream that the user is watching. (No actual video functionality is implemented; the StreamWatcher merely tracks a file position.) This is done by simply keeping track of time and multiplying the elapsed time by the bitrate of the video stream (assuming no use of VCR operations such as pause, fast-forward, and rewind). Alternatively, if an interface to a movie viewer were implemented, the viewer could inform the StreamWatcher of the current position in the file, updating this position as time elapses or as VCR operations are invoked. By knowing the file position, the StreamWatcher knows which block will be next needed by the viewer, and it uses the interfaces to BitTorrent's internal components to keep track of the download in progress.

Whenever the stream reaches a point of the file (whether through simple play or through VCR-style operations), the StreamWatcher checks the file store to see if the piece corresponding to that point has been downloaded and is available. If so, then nothing must be done until the next piece is needed. If the piece is not present, then it must be downloaded immediately or the video stream will suffer delays or breaks. So, the StreamWatcher sends a request to the VoD server (which is carried out in a separate thread) and downloads the piece. Although this download takes place out-of-band with respect to the BitTorrent swarm, a complete piece received in this way is still written into the file using the standard internal BitTorrent component which manages the downloaded file. In addition, the internal PiecePicker (which keeps track of which pieces have arrived and is responsible for selecting which piece to request next) is also updated. These measures ensure that the piece will not be selected and downloaded again from another peer. If there are any outstanding requests to other peers for parts of this piece, they are cancelled. Finally, this new piece is advertised to adjacent peers (just as if it had been downloaded from a peer) so that they can request it from the local client. Taken together, these procedures effectively add the new piece to the existing swarm.

The BitTorrent portion of the client operates as usual, downloading pieces of the file from peers and writing them into the BitTorrent file store. The StreamWatcher simply keeps track of the current stream position and downloads missing pieces from the VoD server when necessary, writing them into the same file store.

PiecePicker. The other major modification to the client is to the policy that determines which piece will be selected when the local client is ready to request a new piece from a peer. Unlike the standard BitTorrent, Toast has less reason to favor the rarest piece first since there is no danger that the file will become unavailable. Instead, the primary goal should be to reduce load on the VoD server as much as possible. This means that there is a potential trade-off in

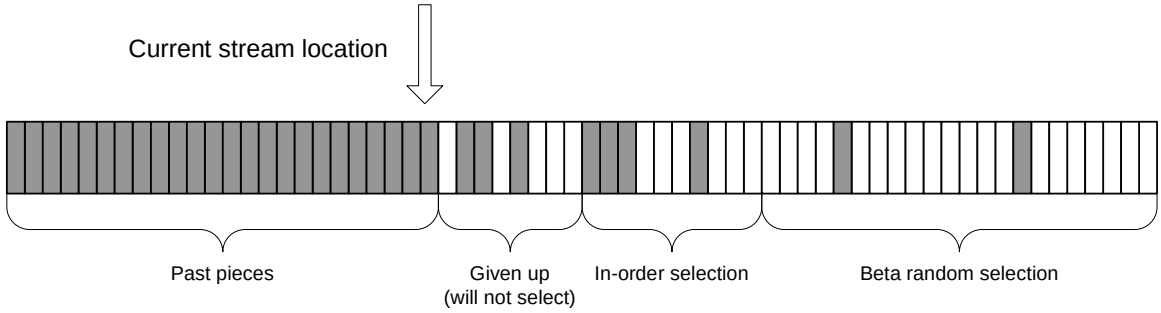

Figure 2: Hybrid piece picking policy

piece selection: biasing selection toward pieces that will be needed sooner would make it more likely that pieces are present when they are needed by the StreamWatcher. However, good piece diversity is still required for efficient BitTorrent operation, and more even distribution would facilitate this. It should be noted that the piece selection policy is only a preference. The PiecePicker is called when a remote client has indicated its willingness to upload (that is, it has unchoked the local client), but the remote client may not have the picker's preferred piece. In this case, the picker chooses the next most desirable piece, according to the policy. It is also possible that there are no pieces available that the client needs; in this case it must simply wait and try again later.

The Toast client implements three piece selection schemes in addition to the default. All policies share the common feature of "giving up" and not selecting pieces that are too close to the current stream position to download on time. For example, if a missing piece that represents 4 seconds of video will be needed in less than 4 seconds, and the remote client's upload rate is less than the video bitrate, then the download cannot finish before the piece is needed, so the picker will skip it and choose a piece further ahead. The amount of this lookahead is estimated based on the length of the piece, the client's upload rate, and the video bitrate. The first selection policy, called "in-order," simply selects the piece that will be needed soonest by the StreamWatcher; that is, the lowest-numbered available piece. This shows a sort of greedy approach where piece diversity is ignored in favor of pieces which will be needed soon. The second policy, called the "beta" policy, selects randomly among all needed pieces using a distribution that favors earlier pieces over later pieces. This is implemented using Python's generator for a beta distribution with an α parameter of 1.0 and a β parameter of 2.0. This gives a probability density function that decreases linearly as piece number increases. This is an attempt to strike a balance between getting pieces that will be needed soon, and maintaining piece diversity.

Figure 2 shows the third policy, a hybrid of the previous two. Aside from the pieces in the past and those given up, it maintains two ranges of pieces: those that will be needed "soon" (e.g., in the next few minutes), and those that will not. It first attempts to ensure that it has all the pieces that will be needed soon by choosing in-order in that range. Then, if it has all the available pieces in this range, it chooses with the beta distribution among the remaining pieces. The size of the in-order range can be specified in seconds because the video has a known bitrate.

Local buffer size limit. In order to model situations in which the local client has only a limited amount of hard disk space to devote to the file, Toast has an option to simulate a limited local buffer. The size of the downloaded data is tracked, and when the size limit is reached, one of two actions is taken. First the client attempts to reduce the local data size by ejecting pieces which are no longer needed (i.e., pieces earlier than the current file position, which have presumably already been watched). To simulate ejection of pieces, the client simply sends a new "bitfield" message to all of its peers, informing them that it no longer has the ejected pieces. If there are no pieces available to eject, the client then suspends all download activity by informing all peers that it is not interested in any of their pieces and by not requesting any new pieces until it can free space by ejecting pieces (new pieces will become eligible for ejection as the current file position passes them).

3.3 VoD Server

The server is a modified HTTP server. Instead of streaming the movie from start to finish, the server sends the pieces that are requested by the clients. The clients send an HTTP request with the name of the file, and the starting and ending byte (using the HTTP/1.1 Content-Range entity-header [16]) that correspond to the piece they need. The piece is sent using as much bandwidth as the server has available and the client is able to receive (rather than just at the video bitrate) to ensure that the client has the piece as soon as possible for its own playback and to share the piece with other clients.

4. EXPERIMENTAL METHODOLOGY

Toast is implemented in version 4.4 of the official (mainline) BitTorrent client and is tested using an actual peer-to-peer network and VoD server. To model the impact of a large number of peers, many instances of the client can be run on each of a small number of machines. The clients are managed by a script that ensures that each client runs in its own directory. Each client also uses a different TCP port for incoming connections. To keep connection speeds realistic, a client's upload bandwidth is limited by the client itself (in fact, this is an existing feature of the mainline client). The client uses only a small amount of CPU time, and provided that bandwidth limits can keep the network interface from being saturated, many clients can be run on a single machine. Our testbed consists of 6 dual-processor 2.8 GHz Intel Pentium 4 systems and 4 4-processor 2.2 GHz AMD Opteron systems, all with 4 GB of RAM and connected by a Gigabit Ethernet switch. Because clients running on cable set-top boxes would share a low-latency network, using Gigabit Ethernet with appropriate delays added to the network could be a realistic setup. For all tests shown, 300 clients are distributed across 9 systems, and the server has its own Intel-based system. A seed client also runs on the server system during these tests, though Toast does not require this.

To emulate a cable-LAN, a Linux tool called NetEm was used to add a fixed amount of delay to all packets as they are sent, including those destined for other ports on the same machine [19]. NetEm

is used to control traffic characteristics like delay and packet loss, allowing a lab environment to simulate a wide-area network. It is part of the Linux kernel version 2.6 and is controlled by "tc" (traffic control), a tool which is part of the iproute2 package. All machines used for testing (including the server machine) were assigned a one-way send-side delay of 4 ms using NetEm. The combination of this delay with processing time and propagation delay gives an average round-trip time of about 10 ms, corresponding to the latency characteristics of a metro-area LAN [1]. Our tests do not model a specific network topology, but rather only the impact of topology on packet latency.

While there are many types of VoD services on many types of networks, one of the main challenging goals remains to serve high quality full-length movies, for example as an ISP service for its customers. In this type of situation, clients often have low latency and high download bandwidth to the ISP's network, though upload rates are more limited. A typical DVD quality movie in MPEG format would be about 4 GB streamed at a rate of 6 Mbps. Due to disk space and bandwidth capacity, the number of clients that can be run on a single machine is limited when using such large files. Instead, the experiment here scales down the system to use 2 Mbps streams, each of which has a capacity of 858 MB. Note that even this lower bitrate exceeds the quality of one of the most popular legal video distribution schemes (iTunes store) and is thus still a realistic operating range. Toast is evaluated using maximum client upload rates of 1 Mbps, 1.5 Mbps, and 2 Mbps.

The testing scenario models a period in which clients come and go over time. For example, a set of 300 clients might watch the same 2 hour movie over a period of 8 hours, arriving at different times during the first 6 hours. To keep the test runs shorter but still model this scenario, the actual tests were run over a period of 4 hours, with clients starting in the first 3 hours, and all being finished by the end of the 4th. The clients arrived every 36 seconds at a constant rate. Three different client behaviors are tested: in the "download-only" behavior the client quits and leaves the swarm as soon as all the pieces are downloaded (which is before the stream reaches the end of the file). This is the behavior of many Internet users of BitTorrent who do not wish to share their upload bandwidth more than necessary. The streaming behavior downloads the file and continues to seed it on the network until the stream reaches the end of the file (1 hour from when the client started), which might be the behavior of a user who kept the BitTorrent client open until the end of the movie. The seeding behavior does not exit the client but seeds the file continually until the end of the test, resulting in all 300 clients being active at the end. Because the default behavior of most BitTorrent clients is to seed the file until the client is closed, this behavior is also commonly seen on Internet swarms, and might also be used if the VoD distributor controls all of the clients (for example, in cable set-top boxes).

The key evaluation metric is reduction of load on the VoD server. A simple unicast server would have to send the entire file to every client separately, with a total data transfer equal to the file size times the number of clients. Allowing the clients to send data to each other reduces this load on the server, ultimately allowing it to serve more clients. All traffic sent by the server machine (including the VoD server, BitTorrent tracker, and BitTorrent seed) was tracked at the network interface and compared to the amount of file data that would have been sent by a unicast server, equal to the size of the file multiplied by the number of clients. All measurements include startup costs where no content has been distributed to any of the clients yet (thus making them slightly conservative with regards to the benefits of Toast).

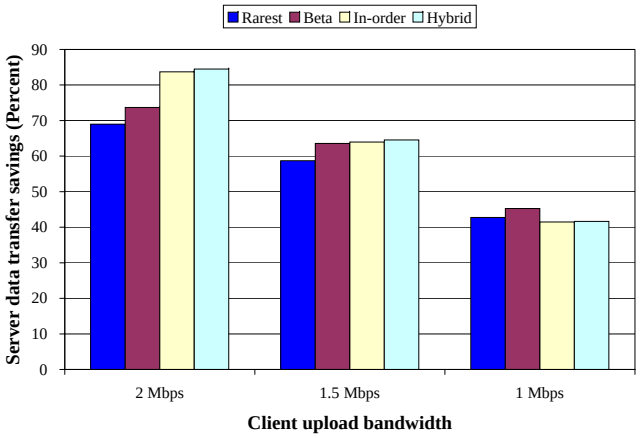

Figure 3: Comparison of picker policies at different maximum client upload rates using streaming behavior

5. RESULTS AND DISCUSSION

Impact of piece selection policies and upload rates. Figure 3 shows the test results as a percent reduction in total data uploaded by the server machine (including the VoD server, BitTorrent tracker, and the seed client) compared to a simple unicast server that sends the entire file to each client. In other words, it shows the percentage of total data transfer that was transferred between peers instead of from the server. Each client upload bandwidth rate is grouped together, comparing each of the four piece selection policies. The 2 Mbps upload bitrate matches the client upload bitrate with the video stream rate, while the 1.5 Mbps and 1 Mbps upload rates represent more constrained environments ($\frac{3}{4}$ and $\frac{1}{2}$ of the video bitrate, respectively). In these tests, all clients use the streaming behavior as described in Section 4: they leave the swarm once the StreamWatcher reaches the end of the video content. These tests also do not add artificial delays using NetEm.

For upload rates lower than the video bitrate, it is impossible for the percent reduction to exceed the ratio of the upload rate to the video bitrate with the streaming client behavior. For example, if the upload rate is half the video stream rate, and each client uploads data for exactly the length of time of the video, then it can only contribute half of the total size of the video into the swarm. The same is true of each client, and the remaining data must be contributed by the VoD server. There was also a greater difference among piece picking policies for greater upload bandwidth, with the better overall performance giving them more ability to differentiate. As expected, the piece selection policies which are designed with streaming in mind almost always perform better than the default rarest-first policy. In addition, the in-order and hybrid policies perform much better with large upload bandwidth, whereas the beta policy performs better with low upload bandwidth. When most of the downloaded pieces can be successfully delivered by peers, it is most profitable to use the available bandwidth to deliver those pieces that will be needed soon rather than attempt to maintain diversity or fetch far into the future.

The 1 Mbps upload case is the only case where rarest-first performs slightly better than in-order and hybrid policies. Since rarest-first has no bias for earlier pieces, it is more likely than the others to select further in the future. This divergence arises as follows. If a piece in the near future is selected and the system has a low upload rate, there is a higher probability that the data will not come fast enough and will need to be downloaded from the VoD server any-

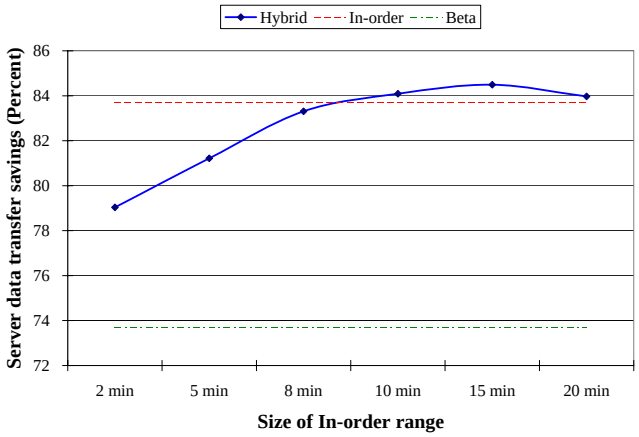

Figure 4: Effect of in-order range size on hybrid policy using streaming behavior and 2 Mbps maximum client upload rate

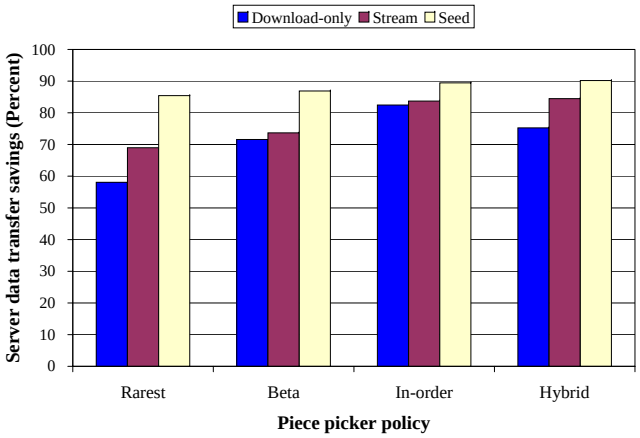

Figure 5: Effect of client sharing behavior in 2 Mbps maximum client upload rate

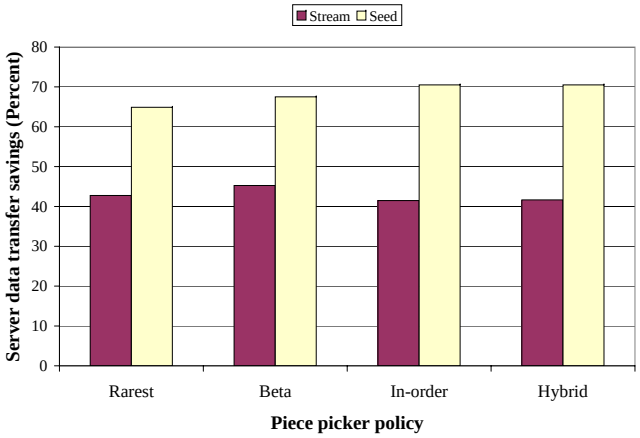

Figure 6: Effect of client sharing behavior in 1 Mbps maximum client upload rate

way, despite the attempt to compensate for this effect by not selecting on nearby pieces as described in Section 3. Predicting the right number of nearby pieces to avoid is difficult because a client does not know what percent of the remote peer's upload bandwidth it is getting, or the number of remote peers from which it can download a single piece at any given time. Consequently, the actual number of pieces that cannot be obtained from peers successfully will vary dynamically. Because the pieces in the expected in-order range are also less likely to be available, the hybrid policy never gets a chance to escape to the randomized region. Consequently, it behaves just like the in-order policy.

Figure 4 shows the effect of varying the in-order range of the hybrid piece picking policy. This range is tested with values from 2 to 20 minutes. In these tests, the client upload rate was equal to the video bitrate and again the streaming client behavior is used. In-order and beta policy results are included for reference. Hybrid outperforms beta throughout the range considered; it also outperforms in-order from approximately 9 minute mark on, peaking at around 15 minutes. (The 15 minute in-order range was used in Figure 3.) This result, as well as the other results in this section show that the hybrid policy achieves its goal of capturing the benefits of early piece selection and piece diversity over a wide range of conditions and represents a good compromise between the more randomized rarest-first and beta policies and the stricter in-order policy.

Client sharing/termination behavior. Figure 5 shows the effect of three testing scenarios when the client upload bandwidth is 2 Mbps. The clients are tested using the download-only, stream, and seed termination conditions described in Section 4 for each of the piece picker policies. As expected, the seeding behavior performs the best with server bandwidth reduced by up to 90% in the hybrid policy. However, even the download and stream scenarios see at least 70% savings with the three new piece picker policies. The download-only scenario performs the worst since clients stop sharing the pieces when they are done getting the movie. The actual sharing behavior is likely to depend heavily on the client population. The stream behavior models a likely PC-based VoD scenario where the viewers will close the program upon completing the movie. The seed behavior aims to model a situation where the clients are always connected and responsive, such as one where the client machines are controlled by an external system administrator. One possible deployment that could exhibit such behavior would be set-top boxes in a cable system; thus, the addition of a BitTorrent peer-to-peer network among nodes in a given area has the potential to greatly reduce the server infrastructure requirements of cable-based VoD.

Figure 6 shows the impact of the seed scenario when the maximum client upload rate is just 1 Mbps (half of the video streaming rate). As described above, the stream scenario would be theoretically limited to only reducing half the load; in practice, the server savings only reaches 45% at most. However, with the seed scenario, the savings reach 71%. This is actually superior to the savings that the stream scenario achieves with 1.5 Mbps upload bandwidth. This result indicates that even with the upload rate only half of the video bitrate, Toast can be effective in reducing the server load if there are enough peers contributing.

Impact of choking policy. As discussed in Section 2, BitTorrent employs a tit-for-tat choking policy, in which peers prefer to upload to other peers from which they have already received data. This strategy is designed to provide incentives for the peers to serve data to others rather than merely acting as freeloaders. In a more controlled environment such as an ISP or cable set-top box, however, this may not be necessary, and a more equal scheme might be more efficient. The tit-for-tat choking policy is compared against a

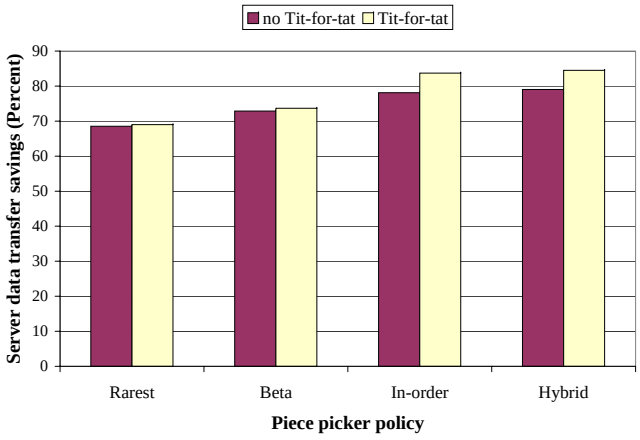

Figure 7: Effect of peer choking policy using streaming behavior and 2 Mbps maximum client upload rate

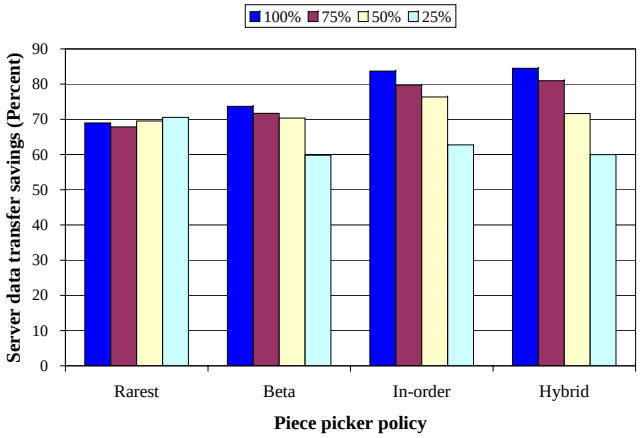

Figure 8: Effect of limited local storage on clients using streaming behavior and 2 Mbps maximum client upload rate

policy that does not consider peer download rates. This is the same policy that is used by a peer when it finishes downloading and becomes a seed. Instead of using download rates (which are no longer relevant for a seed), it uses upload rates and prefers peers which are not downloading from other peers [8]. Figure 7 shows the results achieved by both this seed choking strategy and the standard tit-for-tat for various piece-picker policies with the stream sharing behavior. The experimental setup for these tests is the same as for the previous tests. The results indicate that the standard tit-for-tat policy performs better for all piece picker policies studied, with substantial differences in the in-order and hybrid cases. Thus, even in a controlled environment, cooperation between clients plays an important role in the performance of the system.

Impact of limited local storage. Figure 8 shows the results when each of the new piece picking policies is subjected to a limited storage space. Such a situation may arise if the storage space at the client machine is shared among multiple swarms or is used for other purposes (such as digital video recording). The buffer space in each test is limited to the percentage of the video file size indicated in the legend. This limitation is enforced by the clients themselves; a client downloads normally until the limit is reached, and after that it limits its stored size using two mechanisms. If the

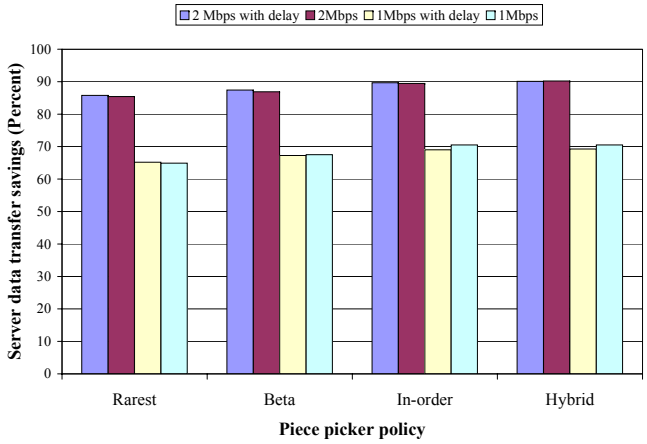

Figure 9: Effect of latency on server bandwidth reduction using seeding behavior

client has any pieces that have already been viewed (that is, pieces before the current stream location) one such piece can be ejected from the local store for each new piece downloaded. If no piece can be ejected, the client must stop the download process and not accept any new piece from peers, a process which carries significant overhead because of the nature of the BitTorrent protocol.

Unsurprisingly, the overall performance is reduced under this restriction, since the ejected pieces are then unavailable to send to other peers. This effect is particularly marked when the size is reduced from 50% to 25% because the clients are forced to stop and start the download process much more often. The behavior of the hybrid piece picking policy is also notable in that it performs closely to the in-order policy for large buffer limits but performs much more like the beta policy for small storage capacities. Clients that are forced to eject pieces will eject the earliest pieces first, thus making it more likely that the pieces in the in-order range are unavailable for delivery to other peers. As a result, those other peers must instead choose pieces further in the future more often. In the rarest-first policy (the default BitTorrent policy), there are only minor differences in performance with respect to the storage space. Interestingly, the performance is slightly better with only 25% of the storage than with more available. Rarest-first is more effective than the other policies with very limited local storage, but cannot take advantage of additional storage to reduce server bandwidth further. For the other policies to be effective, a storage space equivalent to 50% or more of the total stream length should be available.

Impact of latency. Figure 9 shows the effect of network latency as described in Section 4 on the Toast system, comparing the 1 Mbps and 2 Mbps results using additional network delay (10 ms round-trip time) to the base results with no added delay. These tests use the seed termination policy. The server transfer reduction is almost identical, differing by less than 2.1% in all cases, indicating that the additional delay has almost no effect on performance. Two additional delay scenarios were tested, in which not all clients had the same delay. In the first additional scenario, 4 machines were assigned no delay, 4 were assigned 1 ms, one was assigned 2 ms, and one was assigned 5 ms of send-side delay, resulting in overall round-trip times of up to 7 ms. A second additional scenario used 3 machines with no delay, 2 machines with 4 ms, and the rest with 5 ms, resulting in round-trip times of up to 10 ms. These tests differed by less than 2.8% in all cases from the base results.

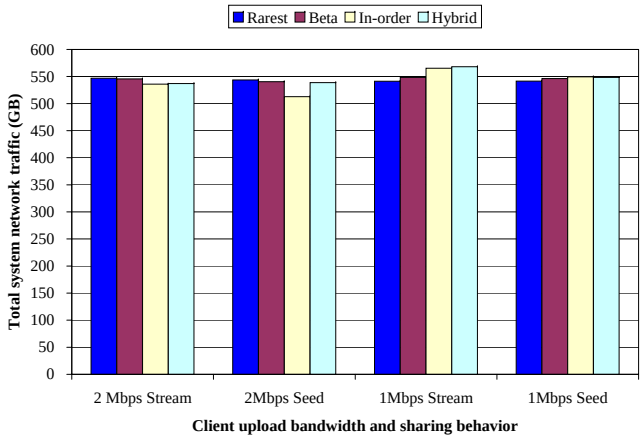

Figure 10: Total network traffic generated

Total system network traffic generated. Besides server data transfer savings, another important measure of the efficiencies of various Toast policies is the overall network traffic generated by the system. Differences in total network traffic may arise because of duplicate network traffic generated by the peers due to inefficiencies in the policy or because of clients fetching data from the server after already requesting the same data from a peer. Figure 10 shows the total network traffic in Gigabytes generated by the system for various piece-picker policies, at both 2 Mbps and 1 Mbps client upload rates, and for both stream and seed sharing patterns. This traffic metric includes both incoming and outgoing network traffic generated by the server, the seed client, and all the BitTorrent clients (including intra-machine transfers in our emulation testbed). These tests were done with network delay added as described in Section 4. This figure shows only small differences in total network traffic despite using different piece-picker policies, client upload bandwidths, and sharing behaviors. This result indicates that Toast has few inefficiencies in total network traffic; the idea of "giving up" on the data pieces closest to the current stream position makes it unlikely that the client will have to fetch data from the server after first requesting it from a peer. The only exceptions are slight degradations in the case of in-order or hybrid piece-picking policies with 1 Mbps upload bandwidth and stream sharing behavior; this result follows the earlier discussion about the slightly degraded performance of these policies in this client configuration.

Impact of client arrival behavior. For the primary tests, clients used a uniform arrival rate, arriving every 36 seconds. In a scaled down version of the tests using 90 clients and a smaller video file, two additional arrival processes were considered. In the "all-at-once" arrival behavior, all clients entered the swarm at approximately the same time, simulating the release time of a popular movie or a scheduled showing. In the Poisson arrival behavior, clients arrived according to a Poisson process with an average interarrival time of 2 minutes. These behaviors were compared to uniform arrivals with 2 minute intervals. The all-at-once behavior performed 21% worse on average than uniform. Because all clients were at approximately the same point in the stream, fewer future pieces were available in the swarm. Poisson arrival behaviors performed very similarly to uniform, within 2% on average.

Discussion. The results presented here show that Toast can be an effective solution for improving the efficiency and aggregate throughput of a Video-on-Demand service. Toast achieves these results using a video-targeted adaptation of BitTorrent backed by a full-fledged VoD server. The results show that the default BitTorrent policy is not well-suited to the VoD environment since it makes no attempt to use available bandwidth or storage to achieve real-time delivery constraints. Instead, policies that favor earlier content pieces make it more likely that clients can avoid using the backing VoD server. Further, allowing clients to remain in the swarm as seeds can help improve VoD server efficiency even for limited client upload bandwidth. Combining BitTorrent, a backing VoD server, a hybrid piece-picking policy, and client seeding allows Toast to offload as much as 90% of the network traffic from the VoD server onto the P2P network.

This paper uses BitTorrent to optimize the distribution of a single file. However, the scheme described here could be deployed using separate swarms for distributing different popular files. Alternatively, each peer could run a single "swarm manager" allowing it to better partition its storage and network resources among multiple active streams. Such capacity management should lead to further interesting avenues for research.

This paper also does not evaluate the effect of the number of participating peers or the scalability of this system. As with any P2P system, Toast should provide throughput scalable with the number of the peers so long as peers are not sharing common bottlenecks. Although a greater number of peers would also yield some benefit in terms of the likelihood of receiving a piece from a peer instead of the server, Tewari and Kleinrock showed in the context of live streaming that BitTorrent-based peer groups reach a critical mass with as few as 10–15 peers [29]. The critical group size should be even smaller with on-demand streaming since data can be served from further ahead in the stream.

6. CONCLUSIONS

This paper presents and evaluates Toast, a system to improve the efficiency of a Video-on-Demand service using the popular and proven BitTorrent peer-to-peer technology. Toast includes a modified version of BitTorrent that supports streaming data delivery rather than just downloads. The BitTorrent client is adapted to communicate with a traditional VoD server when the desired piece of data cannot be delivered by the swarm participants in a timely fashion. Toast thus extends the aggregate throughput capability of a VoD service, offloading work from the server onto the P2P network in a scalable and demand-driven fashion.

This paper evaluates Toast under several usage configurations. The results show that the default BitTorrent piece selection policy is not well-suited to the VoD environment. Instead, policies should favor downloading earlier pieces of content from other swarm participants to make it less likely that the end clients will have to get the data directly from the VoD server. Further benefits are possible if clients remain in the system as seeds after they are done viewing their content. Such configurations allow Toast to operate very efficiently, reducing data transfer demands by up to 70–90% compared to simple unicast distribution. Toast is thus an attractive choice to serve as the substrate of a VoD data delivery system, effectively utilizing client resources and network bandwidth while also reducing server infrastructure requirements.

Acknowledgments

The authors thank Sanjay Rao (Purdue) for valuable feedback in the early stages of this work. The authors also thank the anonymous reviewers for their input and particularly acknowledge the efforts of Carsten Griwodz (Oslo), the PC shepherd for this paper.

7. REFERENCES

[1] Aspera Inc. Overcoming the challenges of network data delivery. White paper, 2006.

[2] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[3] Bitband Technologies Ltd. Vision 680. Data Sheet.

[4] S. Carter and D. Long. Improving video-on-demand server efficiency through stream tapping. In *Computer Communications and Networks, 1997. Proceedings., Sixth International Conference on*, pages 200–207, 22-25 Sept. 1997.

[5] Y.-F. Chen et al. When is P2P Technology Beneficial for IPTV Services. In *Proceedings of the 17th International Workshop on Network and Operating System Support for Digital Audio and Video*, May 2007.

[6] Y. R. Choe and V. S. Pai. Achieving Reliable Parallel Performance in a VoD Storage Server Using Randomization and Replication. In *Proceedings of the 21st International Parallel and Distributed Processing Symposium*, March 2007.

[7] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast (keynote address). In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 1–12, New York, NY, USA, 2000. ACM Press.

[8] B. Cohen. Incentives build robustness in bittorrent. Technical report, May 2003.

[9] Comcast Corporation. Annual Report to Shareholders, 2005.

[10] Y. Cui, B. Li, and K. Nahrstedt. oStream: asynchronous streaming multicast in application-layer overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):91 – 106, 2004.

[11] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*, pages 15–23, New York, NY, USA, 1994. ACM Press.

[12] C. Dana, D. Li, D. Harrison, and C.-N. Chuah. BASS: BitTorrent assisted streaming system for video-on-demand. *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, October 2005.

[13] L. de Pinho, E. Ishikawa, and C. de Amorim. GloVE: A distributed environment for scalable video-on-demand systems. *Int. J. High Perform. Comput. Appl. (USA)*, 17(2):147 – 61, Summer 2003.

[14] D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *Knowledge and Data Engineering, IEEE Transactions on*, 13(5):742–757, Sept.-Oct. 2001.

[15] Entone Technologies, Inc. Entone Video Server Architecture. White paper, 2005.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP 1.1. IETF RFC 2616, June 1999.

[17] L. Golubchik, J. C. S. Lui, and R. Muntz. Reducing i/o demand in video-on-demand storage servers. In *SIGMETRICS '95/PERFORMANCE '95: Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 25–36, New York, NY, USA, 1995. ACM Press.

[18] L. Guo, S. Chen, S. Ren, X. Chen, and S. Jiang. PROP: a Scalable and Reliable P2P Assisted Proxy Streaming System. *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 778–786, 2004.

[19] S. Hemminger. Network Emulation with NetEm. In *Proceedings of the 2005 Linux Conference Australia (LCA-2005)*, April 2005.

[20] C. Huang, J. Li, and K. Ross. Peer-Assisted VoD: Making Internet Video Distribution Cheap. *Proceedings of Sixth Internaltional Workshop on Peer-to-Peer Systems*, 2007.

[21] Kasenna, Inc. Kasenna Media Servers. Data Sheet, August 2003.

[22] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 177–186, 2002.

[23] http://www.ppstream.com/. Lsat checked June 21, 2007.

[24] R. Rejaie and A. Ortega. PALS: Peer-to-Peer Adapative Layered Streaming. In *Proceedings of the 13th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 153–161, June 2003.

[25] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. IETF RFC 1889, January 1996.

[26] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). IETF RFC 2326, April 1998.

[27] P. Shenoy and H. Vin. Multimedia storage servers. In K. Jeffay and H. Zhang, editors, *In Readings in Multimedia Computing and Networking*. Morgan Kaufmann Publishers, 2002.

[28] S. Sheu, K. Hua, and W. Tavanapong. Chaining: a generalized batching technique for video-on-demand systems. *Proceedings IEEE International Conference on Multimedia Computing and Systems*, pages 110 – 17, 1997.

[29] S. Tewari and L. Kleinrock. Analytical Model for BitTorrent-based Live Video Streaming. In *Proceedings of the IEEE NIME 2007 Workshop*, January 2007.

[30] C. Thompson. The BitTorrent Effect. *Wired Magazine*, January 2005.

[31] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(4):197–208, August 1996.

[32] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–6, April 2006.