Evaluation of Swarm Video Streaming

David Erman, Yong Yao, Karel De Vogeleer, Adrian Popescu

International Congress On Ultra Modern Teland Control Systemsecommunications

2010 Moscow

Evaluation of Swarm Video Streaming

David Erman, Yong Yao, Karel De Vogeleer and Adrian Popescu
Dept. of Communications and Computer Systems
School of Computing
Blekinge Institute of Technology
371 79 Karlskrona, Sweden
Email: {david.erman, yong.yao, karel.de.vogeleer, adrian.popescu}@bth.se

***Abstract*—IPTV has been hailed as a "killer app" of the Internet for a long time, but it has yet to really make a major breakthrough, due to various technical and political reasons. The recent influx of community-driven video distribution has re-actualized the research into efficient distribution methods for streaming video. In this paper, we suggest a number of modifications and extensions to the BitTorrent distribution and replication system to make it suitable for use in providing a streaming video delivery service, and implement parts of these in a simulator. Also, we report on a simulation study as well as a large-scale real-world study of the extensions on PlanetLab. Results show that BitTorrent can be used as a bandwidth-efficient alternative to traditional IPTV solutions.**
Keywords: IPTV, media distribution, swarm

I. Introduction

Large-scale, real-time multimedia distribution over the Internet has been the subject of research for a long time. A large number of mechanisms, policies, methods, and schemes have been proposed for media coding, scheduling, and distribution. Internet Protocol (IP) multicast was expected to be the primary transport mechanism for this, but it was never deployed to the expected extent. Recent developments in overlay networks have reactualized the research on multicast, with the consequence that many of the previous mechanisms and schemes are being re-evaluated.

Peer-to-Peer (P2P) systems such as those provided by Conviva [1] and PeerCast [2] are being used to stream video and audio to large subscriber groups. Furthermore, approaches such as Distributed Prefetching Protocol for Asynchronous Multicast (dPAM) [3] and oStream [4] provide intelligent application layer multicast routing and caching services. One of the most popular distribution and replication systems is BitTorrent (BT) [5], developed to distribute large files and decrease the load on the initial file server. Our paper is a contribution towards why BT for video streaming.

The rest of this article is organised as follows: in Section II, we briefly describe the BT system. This is followed by a discussion of current solutions for BT streaming in Section III. In Section IV, we describe the piece selection algorithm of BT as well as present a set of modifications to them for enabling BT as a streaming solution. Furthermore, we describe a set of changes to the BT metadata, which together with the algorithm modifications are collectively denoted as BitTorrent Extensions for Streaming (BiTES). Section V describes the metrics we use for evaluating these modifications. In Section VI, we report on a simulation study of the modifications, and in Section VII we report in a real-world study of them. Finally, Section VIII concludes the paper.

978-1-4244-7286-4/10/$26.00 ©2010 IEEE

II. BitTorrent

A BT system is comprised of three distinct types of network entities: a *tracker* and peers, divided into one or more *seeds* and *leechers*. The tracker is a coordinating entity responsible for providing peers with the addresses of other peers as well as keeping up- and download statistics of peers. A seed is a peer that has all of the content, and a leecher is a peer that does not. The set of peers and the tracker are collectively known as a BT *swarm*. A tracker may manage several swarms, and the specific swarm is identified by the Secure Hash Algorithm One (SHA1) hash of the meta-data describing a specific set of content. Meta-data is encoded using a BT-specific encoding scheme called *bencoding*.

Data transfer in a BT swarm is performed by leechers downloading fixed-size parts, called *pieces*, typically of size 256 kB, 512 kB, or 1 MB [11]. The pieces are requested and downloaded in smaller parts called *blocks*, typically of size 16 or 32 kB. The blocks are downloaded from either other leechers or from seeds in the swarm. Without an initial seed, or at least leechers which collectively have all pieces, a swarm cannot be successful in that no peer will ever have the full content. In order to get fair reciprocation between participating peers a peer selection algorithm, known as the *choking* algorithm, is used. The algorithm prioritises peers that have a higher download rate for upload. Peers are only allowed to download from another peer, or be unchoked by that peer, if they show *interest* in that peer. Interest occurs if the other peer has pieces that the potential interested peer does not have. To join a specific BT swarm, a potential peer must first acquire a set of metadata, known as a *torrent file*. The torrent file contains, among other information, the address to the swarm tracker and a set of SHA1 hash values for the content pieces. The SHA1 hash for the torrent file itself acts as an identifier of the swarm and is used in both peer handshakes and tracker queries.

III. State of the Art

There are several proposals and systems, both academic and industrial, that use BT as a transport mechanism for

streaming delivery. For instance, BitTorrent Inc. have released a proprietary media streaming solution called BitTorrent DNA [6], which is based on a modified BT infrastructure with added management and Quality of Service (QoS) capabilities.

BitTorrent-Assisted Streaming System (BASS) [7] is a hybrid approach in which a traditional client-server Video-on-Demand (VoD) streaming solution is augmented with BT downloading of non-time-sensitive data. The rarest-first algorithm is modified to only consider for download pieces that are *after* the current playout point, storing them for later playback. Clients download pieces in a linear fashion from the media server, except for pieces that have already been downloaded by BT and pieces that are being downloaded by BT and are expected to finish before the playout deadline.

The authors show that BASS improves server bandwidth utilization by 34 % compared to the pure client-server solution. They also show that the initial delay before playback is significantly shorter than in both the pure BT and client-server cases.

One drawback with the analysis in [7] is that the authors assume that there are significantly more non-BASS clients than clients using BASS in the swarm. This assumption also indicates that there are several seeds in the swarm, so that the BASS clients are never starved of pieces. Another potential problem with the work is that the authors use simulation inputs that may not be representative for a streaming scenario. The inputs used are hyper-exponential fits to the download and arrival rates for peers participating in the distribution of a Linux release. Furthermore, the traffic is analysed from a single tracker log for one specific content.

BitTorrent Streaming (BiToS) [8] is a modification of the BT piece selection algorithm. In BiToS, the piece set is divided into three disjoint subsets: the *received pieces*, the *high priority* (HP) and *remaining pieces* (RP) sets. Pieces that are close to be consumed, *i. e.*, close to the current playout point, are placed in the HP set, while the remaining non-downloaded pieces are placed in the RP set. For instance, if the playout point is at piece 5, the HP set might be $\{6, 7, 8, 9\}$. The number of pieces in the HP set is fixed. The received pieces set contains those pieces that the client has downloaded and are available for sharing with the client's active peer set.

The Vuze [9] client is one of the most popular BT clients and integrates a video distribution service. The client is implemented in Java and incorporates advanced configuration options to control client behaviour. It also features advanced sharing and tracker functionality.

Tribler [10] originated as a research project at Delft University of Technology and De Vrije Universiteit Amsterdam. The Tribler client provides users with access to streaming videos from a variety of sources, including BT and YouTube. The client also includes a social networking component which discovers peers that consume similar types of content as each other.

The Tribler project is also associated with P2P-Next [13], a large European research consortium under the 7^{th} framework programme.

IV. Piece Selection and Metadata

The typical use of a BT client is to download static, non-streaming content. In this context, piece ordering is not relevant, and pieces are downloaded according to a *rarest-first* algorithm. The exact implementation details are left to individual developers, subject to the following constraints:

- The algorithm should quickly provide a client with full pieces, so that it can quickly engage in reciprocal up- and download with other peers in the swarm.
- The algorithm should increase the distribution of *rare* pieces, *i. e.*, the number of replicas of pieces that are replicated the least in the swarm should be increased. This is done by downloading the rarest pieces first and also gives name to the algorithm.

The reference implementation of this algorithm is described and implemented in the reference client, *a. k. a.*, mainline BT client. The algorithm is governed by four distinct policies:

1) *Random first:* A peer selects pieces to download at random until it has downloaded four full pieces.
2) *Strict priority:* When a block is downloaded, the remaining blocks of the corresponding piece are prioritized.
3) *End game mode:* When a peer has received or has outstanding requests for all pieces during the final phase of a download, requests for the not yet received blocks are sent to all connected peers that have the pieces. For every received block, the peer sends *cancel* messages to the peers that were sent requests previously.
4) *Rarest first:* A peer maintains a list of the rarest pieces, *i. e.*, the pieces that are available from the fewest number of peers. The peer then requests these pieces. This increases the popularity of the peer itself, since it now has a rare piece to share with the rest of the swarm.

A. Streaming Content

BT is very effective in distributing content without requirements on data ordering and timeliness. However, in the case of streaming media these are non-optional requirements and must be addressed if BT is to be used as a transport mechanism for such media. This means that in addition to the previously mentioned requirements for a piece selection algorithm, a streaming version of such an algorithm must also take into account another requirement: *keep the playback deadline for each piece*. To facilitate this, we propose a number of changes to the BT core algorithms and mechanisms, collectively called BiTES [18]. One of the major changes is the addition of per-piece deadlines to the swarm meta-data, to facilitate the algorithms below. The piece deadlines represent the maximum time for downloading the piece if it should be received in time for playout.

Further, we divide the entire piece set into three disjoint ranges: a *seeding range*, a *playback buffer* and a *picking range*, as shown in Figure 1.

Pieces in the seeding range are considered as being downloaded, regardless of the pieces actually having been downloaded or not. Once the playout point has passed a specific

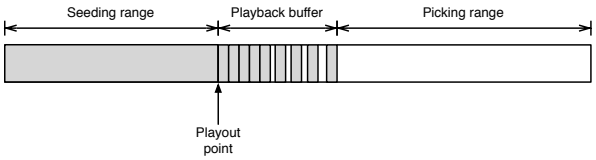

Fig. 1. Piece ranges.

piece, *i. e.*, the deadline for a piece has passed (regardless of whether the piece has been consumed or not), the piece is no longer considered selectable. Pieces in the playback buffer range are picked linearly, in the sense that the lowest available non-downloaded piece that is within the buffer range is selected for download. For instance, if the buffer size is 5 pieces, the playout point is piece 3 and the buffer contains pieces $\{4, 5, 6, 8\}$, piece 7 is picked. The remainder of the piece set is denoted by the picking range.

Once the pieces in the playback buffer are either downloaded or downloading, the pieces in the picking range are available for selection. We test several piece selection algorithms:

1) *Linear:* In linear selection, a peer picks pieces in a linear sequence, $k, k+1, k+2, \ldots$ from the picking range. In a sense, this extends the playback buffer size to the entirety of the piece set.
2) *Smoothing:* Select pieces that contribute most to network traffic load, *i. e.*, the pieces that have the shortest associated deadlines. Shorter deadlines means that a piece needs to be downloaded faster for it to meet its deadline.
3) *Rarest-first:* The default BT selection algorithm. Pick the pieces that are least replicated in the swarm.
4) *Hybrid probabilistic:* A probabilistic hybrid algorithm in which the smoothing algorithm is used with probability p and the rarest-first algorithm with probability $1 - p$.
5) *Hybrid deadline-based:* The deadline-based algorithm uses the smoothing algorithm to implement something more akin to conventional smoothing, *i. e.*, only perform smoothing when the link is underutilized. The following is a description of the algorithm:
   a) Pick a piece using rarest-first, p_r, with inter deadline time $t(p_r)$, and associated required download bandwidth $b(p_r) = 1/t(p_r)$
   b) If $b(p_r) < \bar{b}$, pick a piece p_s for smoothing that minimizes $t(p_r) + t(p_s) = t(p_r + 1)$. This means: find the piece with the deadline closest to the time left until the next deadline, given the current download bandwidth. If no such piece can be found, then pick a new piece using rarest-first.

B. Metadata

A peer interested in downloading some content by using BitTorrent must first obtain a set of meta-data, the so-called *torrent file*, to be able to join a set of peers engaging in the distribution of the specific content. The meta-data needed to join a BitTorrent swarm consists of the network address information (in BitTorrent terminology called the *announce URL*) of the tracker and resource information such as file and piece size. The torrent file itself is a bencoded version of the associated meta information.

An important part of the resource information is a set of SHA1 hash values, each value corresponding to a specific piece of the resource. The hash values are used to verify the correct reception of a piece. When rejoining a swarm, the client must recalculate the hash for each downloaded piece. This is a very intensive operation with regards to both CPU usage and disk I/O, which has resulted in certain alternative BitTorrent clients storing information regarding which pieces have been successfully downloaded within a specific field in the torrent file.

A separate SHA1 hash value, the `info` field, is also included in the meta-data. This value is used as an identification of the current BT swarm, and the hash value appears in both the tracker and peer protocols. The value is obtained by hashing the entire meta-data (except the `info`-field itself). If a third-party client has added extra fields to the torrent file that may change intermittently (such as the resume data or cached peer addresses), these should not be taken into account when calculating the `info`-field hash value.

For our purposes, the most important part of the torrent file is the value of the `pieces`-key in the `info`-dictionary. This value contains the concatenation of the SHA1 hashes for every piece. The number of pieces,N_p , is thus given by the size of this string, divided by 20, which is the size of the SHA1 hash.

The BiTES software adds an additional dictionary to the BT metadata, named `bites`, containing the following keys:

- *duration* – a string denoting the duration of the video object.
- *vcodec* – a string denoting the codec that the video stream in the video object is compressed with.
- *acodec* – a string denoting the codec that the audio stream in the video object is compressed with.
- *achannels* – a string denoting the number of channels in the audio stream. Permissible values are either integers or "x.1"-format indicating x-channel surround with subwoofer.
- *vstream* – an integer indicating which video stream in the video object is used for following keys.
- *average_bitrate* – a string denoting the average bitrate (in kbps) of the decompressed video stream.
- *deadlines* – a list containing $N_p - 1$ strings where each string i denotes the additive deadline, in seconds, for piece $i - 1$. The first piece in the content has the implicit deadline of 0. That is, the first string in the `deadlines` list denotes the duration of piece 0, or, the time since playback started by which piece 1 must have been downloaded. The absolute deadline for piece i is given by $\sum_{j=0}^{i} d_j$.

V. Evaluation Metrics

Our experiments are evaluated using a number of quantitative metrics and qualitative measures. It is important that these

metrics and measures capture the most important characteristics of the system under study. Consequently, the metrics have to address the following characteristics: how well the system behaves in BT terms, and how well it performs as a streaming solution. To assess these characteristics, we use three metrics: *piece distribution*, *continuity index* and *server load*.

A. Piece distribution

Piece distribution is a measure of how well a BT swarm is behaving. It is an important measure to ascertain that a piece selection algorithm does not behave too selfishly, so that pieces in the swarm are not under- or overrepresented. Piece distribution can be measured in several ways. The most common way in BT clients is by using the number of copies of the most rare piece in the swarm, denoted by p_r.

Ideally, the distribution of pieces in a BT swarm is uniform, i.e., every piece is replicated equally, if the swarm is observed over a long period of time. Another option assessing the swarm distribution is therefore to observe the piece distribution over time, for instance, the discrepancy between the distribution of pieces to the uniform distribution using standard statistical metrics such as the Anderson-Darling statistic. For our work, we opt for a visual measure and use distribution graphs to assess the quality of the piece distribution.

B. Server load

The server load, b, is a classic measure of the quality of a streaming mechanism. In the BT case, we define the server load as the number of pieces that are downloaded from the initial seeds, divided by the number of pieces in the piece set.

It is typically of interest to minimise the server load, which in the streaming BT case means that the system should rapidly distribute pieces to peers, so that there are no rare pieces (pieces only available from seeds), and only use seeds for downloading when pieces are rare. However, this may cause clients to be overloaded with requests, and in effect move the server load to more popular peers.

We measure the server load as the ratio of the number of sent pieces from the initial seeds to the number of total pieces in the piece set. For a completely new swarm in which peers do not have any of the pieces, the minimum server load is 1.

C. Continuity index

The Continuity Index (CI) is another measure of the quality of a streaming service. It is defined as the number of pieces that meet their deadlines, divided by the total number of pieces. While the server load and piece distribution are system-centric metrics, the continuity index is client-centric. A missed deadline results in distorted playback and an impaired experience for the user of the service, which is why it is also important to consider this metric.

VI. Simulations

To test the BiTES modifications, we have run a number of simulations. The simulator was implemented using the OMNeT++ [14] discrete event simulation system and was validated against real BT traffic characteristics reported in [17].

In total, we ran four sets of simulations, where each set consists of six simulation scenarios [18]. However, due to space limitations, we only report on the results for the first set of simulations in this paper. Each scenario represents a specific piece selection algorithm, and each set represents a slight change in the input parameters of the simulation. The simulation scenarios are denoted *linear*, *rarest-first*, *prob-0.1*, *prob-0.5*, *prob-1.0* and *deadline*, each representing the corresponding algorithm.

In Table I, we report the simulation parameter settings for the first set of simulations.

TABLE I
Simulation parameters for experiment scenarios, set 1.

Parameter	Value
Seeds	1
Peers	1000, including initial seed
Swarm inter-arrival time	exponential, mean 12.395 s
Maximum connections per peer	10
Peers requested from tracker	10
Asymmetric links	no
Initial piece distribution	0 %
Link bandwidth	Uniform, $[2, 10]$ Mbps
Link delay	Uniform, $[5, 400]$ ms
Piece inter-deadline time	Truncated Gaussian, mean 0.3 s, stddev 0.5
Playback buffer size	10 pieces
Request waiting time	exponential, mean 100 ms
Block size	$2^{15}(32768)$ bytes
Piece size	62144 bytes
Simulation runs	40

The main reason for selecting a single seed is to create a scenario similar to that of a conventional file transfer, such as using HyperText Transport Protocol (HTTP) or File Transfer Protocol (FTP), which is the typical scenario for a BitTorrent swarm. Choosing 1000 peers for the simulation represents an acceptable trade-off between simulation time and enough data to be able to perform adequate statistical analysis. For the same reason, we ran 40 simulations per scenario. After download, peers remain in the swarm for an exponentially distributed time, with mean 180 s. The simulation ends when all 999 non-initial seed peers have become seeds. The parameter value for the mean swarm inter-arrival time was selected at random as one of the component parameters for one of the measurements reported in [18]. The number of connections and maximum number of peers requested from the tracker are selected to have a maximum total up- and downstream bandwidth of 100 Mbps for each peer. Again, one connection per peer is reserved for a connection to the tracker. The link delay, bandwidth and asymmetry were selected to represent non-ADSL broadband users, at both regional and global areas. Further, the initial piece distribution of joining peers is set to 0 % to reflect typical video broadcasting behaviour. Typically, a video is not likely to be partially pre-fetched in advance. The request waiting time was added to account for various factors, such as scheduling of requests of a peer, snubbing, bandwidth limiting, and other delay-generating processes. It is the amount of time

that elapses before a piece request is responded. The block size of 32768 bytes was selected as it was the the default size recommended by the BT specification [11] at the time of the experiments.

A. Simulation results

In this section, we present the results from the first set of simulations. The remaining simulations are omitted due to space considerations, and are reported in [18].

1) Piece Distribution: we observe in Figure 2 that the linear scenario is significantly different from all the other scenarios. It is also one of the two scenarios, together with the pure smoothing, that does not use the rarest-first algorithm at all. The remainder of the algorithms employed all use this algorithm in part: in the *rarest*, *prob-0.1* and *prob-0.5* scenarios pieces are selected using rarest-first with probabilities 1, 0.9 and 0.5 respectively. In the *deadline* scenario, at least every other piece is selected according to the rarest-first policy. The

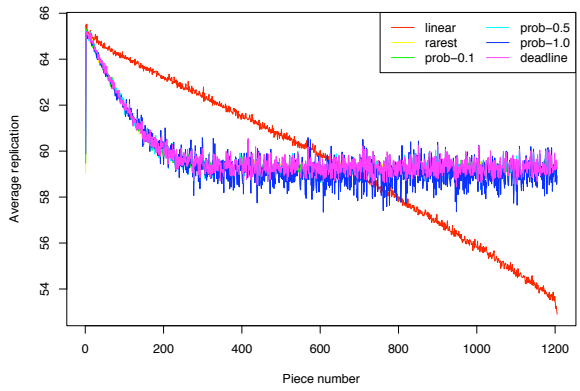

Fig. 2. Piece distribution results, set 1.

reason for the clear negative linear slope of the *linear* scenario and the higher replication of the initial pieces of the other scenarios is that all scenarios employ a playback-buffer. In the *linear* case, the playback buffer size is in effect the entire piece set. This means that the number of the piece next selected is directly related to the download rate and time in the swarm of the corresponding peer. Since lower-numbered pieces are downloaded earlier, they stay longer in the swarm, and are thus replicated to a higher degree.

For each scenario, we then average the continuity index and server load as well as the calculating of 95% confidence intervals.

2) Continuity Index: in Figure 3, we show a graph of the resulting per-run averages and confidence intervals. The solid lines in Figure 3 represent the average CI for all runs for the corresponding scenario. The most prominent result is the good performance of the *linear* algorithm. This is an expected result, as the playback buffer is more likely to never empty using this algorithm.

Furthermore, the algorithms that employ the *rarest-first* algorithm to some extent, *i. e.*, all but the *linear* and *prob-1.0* algorithms perform similarly with respect to the CI.

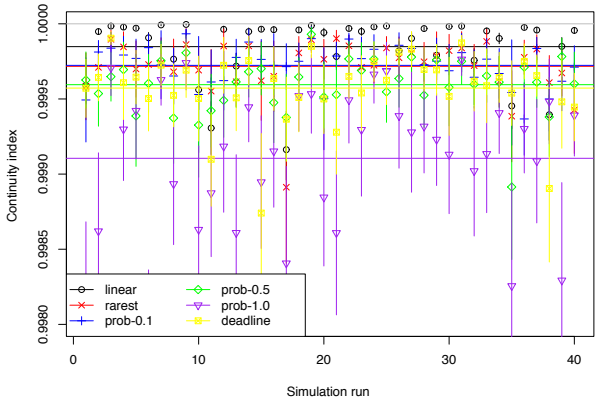

Fig. 3. Continuity index results, set 1.

Another feature observed in Figure 3 is the results for the *prob-1.0* scenario. First, that scenario is the only scenario with an average CI that indicates missing more than one piece deadline. Second, the large confidence intervals are evident on a per-run basis for the *prob-1.0* scenario.

3) Server Load: we present the results for the server load metric in Figure 4. As it was the case for the piece distribution, the linear selection algorithm stands out as being the worst performing of the evaluated algorithms. However, it still outperforms a conventional non-cooperative streaming solution by using only 2.08 % of the bandwidth for simulation set 1 (the corresponding server load for conventional web streaming is 999). Similar to the results for the piece distribution, we

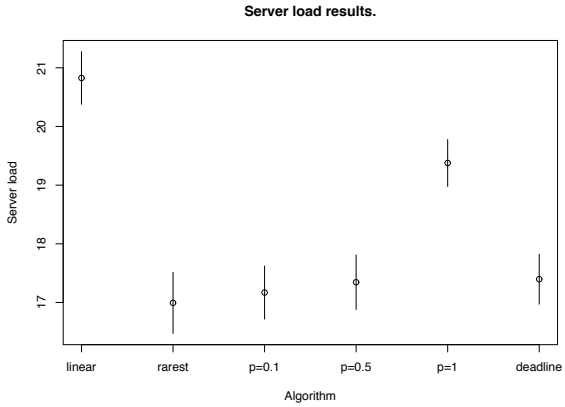

Fig. 4. Server load results, set 1.

observe a clear tendency of increasing server load as the chosen algorithm becomes more a pure smoothing algorithm. The reason for this is that when using the rarest-first algorithm, pieces are selected in different order by each peer, while in the pure smoothing case, peers select pieces in the same order, according to their deadlines. In this sense, the *prob-1.0* and *linear* algorithms are similar since the pieces are actually

picked deterministically. This also gives an explanation as to why the *linear* algorithm performs so poorly, as pieces numbered higher than half of the total number of pieces are more likely to be requested from the initial seed.

Another interesting observation is the similarity of the results for the *prob-0.5* and *deadline* scenarios. In both algorithms, pieces are on the average selected the same number of times from the initial seed using the *rarest-first* algorithm, and a smoothing algorithm. However, the *prob-0.5* algorithm requires less calculations, while providing equal or better results.

VII. Real-world Experiments

In addition to the simulation study reported in the previous section, we have performed two sets of real-world experiments: one in a Local Area Network (LAN) environment, consisting of 13 computers, and one in a PlanetLab [16] environment, to perform experiments as close to real-world environment as possible.

We have implemented a BT client library from scratch, incorporating the modified piece selection algorithms described in section IV. The client is written in C++ and runs on most modern operating systems. We have also written software to parse video files and to generate the metadata described in Section IV-B.

The video file used for the experiments is Elephants Dream [15], an open source short movie.

To facilitate comparison of the measurement study and the simulation study, we used the same algorithm configurations for each experiment, as defined in Section VI, except for changing the *prob-0.1* scenario to *prob-0.2*. This was due to the *prob-0.1* performing so close to the *rarest* algorithm that they were practically identical. Furthermore, since the *deadline* algorithm performed so similar to the *prob-0.5* algorithm, we have opted not to study this algorithm further in the real-world experiments.

A. Control System

In both LAN and Planet-Lab environments, multiple computers are involved as BT peers. Due to the high number of participating peers, manually managing these computers to participate in downloading swarm is cumbersome and error-prone. To automate the experiments, we have developed an automatic control system, called the BiTES Manager. Its main tasks are:

- Distribute the torrent file and BiTES executable file to all peers.
- Start all BiTES clients.
- Collect experimental results from all peers.
- Control incoming communications from leechers and provide them with *inter-arrival* and *inter-departure* times. The swarm *inter-arrival* time for every leecher is calculated according to two parameters: *global starting* time and *exponential random* time. The *global starting* time is used as a reference time for each experimental run, making all leecher peers start downloading relative to this time. Its value will be reset by the *central controller* after every downloading run has been finished. The exponential random time is a delay for each peer, relative to the starting time of the previous peer. In effect, the global starting time can be viewed as the experiment starting time, while the exponential random time is the swarm inter-arrival time for the specific peer and experiment.
- Control leechers to carry out the different scenarios.
- Control leechers to carry out multiple runs under different scenarios.

B. Results for LAN experiment

The parameters for the LAN experiment were set as shown in Table II. There are some differences compared to Table I, that are due to the fact that in the experimental study, certain parameters, such as inter-piece times, are not relevant for the running of the experiment, as they are inherently results of the peer-to-peer interactions.

TABLE II
Parameter values for LAN experiment.

Parameter	Value
Seeds	1
Leechers	12
Maximum connections per peer	20
Maximum uploads per peer	4
Peers requested from tracker	50
Swarm inter-arrival time	exponential, mean 12.395 s
Swarm inter-departure time	exponential, mean 12.395 s
Link bandwidth	100 Mpbs
Initial piece distribution	0%
Playback buffer size	10 pieces
Piece size	256 kB
Total number of pieces	1071

The experiments started on July 25, 2009 and lasted 15 days. They were carried out in the computer laboratory located at BTH. All computers adopted for the experiments have the same configurations of both hardware and software, *i. e.*, 2.39 GHz Intel Core Duo, 2.00 Gigabytes (GB) memory of RAM, 10.7 GB Disk drive and running the Ubuntu Linux distribution. The following scenarios were run: rarest-first, linear, hybrid of rarest and smoothing with probability 0.2 and 0.5, and pure smoothing. Each scenario was run 30 times. In other words, 150 runs on all 12 computers have been carried out. The results for the CI and server load, with the corresponding 95 % confidence intervals are reported in Table III.

1) Piece distribution: Figure 5 shows the average number of replications per piece for all 30 experimental runs. We observe that both the *rarest* and *smoothing* algorithms have significantly affected the piece distribution in the swarm. By using the smoothing probability for piece selection, these two algorithms become direct competitors.

Because of the *rarest* algorithm, replication of pieces in the picking range is more uniformly distributed in the swarm. On the other hand, the *smoothing* algorithm assigns higher priorities to pieces of lower frame duration. Consequently, these pieces are more replicated among the peers in the

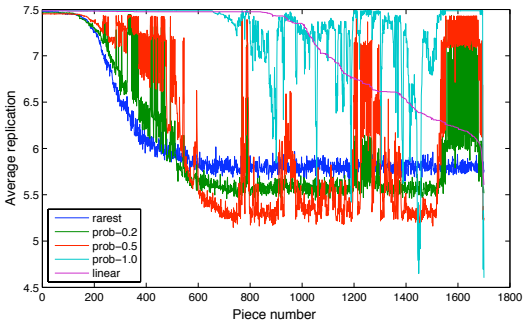

Fig. 5. Piece distribution in the LAN experiment.

swarm. We observe in Figure 5 that higher the smoothing probability is, higher the replication of pieces with shorter frame duration becomes. When the smoothing probability is 1.0 (pure smoothing algorithm), the piece with the shortest associated frame duration (deadline) is selected. This implies that the distribution of pieces in the picking range will be directly related to the frame duration of the corresponding pieces.

2) Server load: All leechers respectively update the number of pieces sent from seeds in their local log files during each run. The BiTES Manager collects these logs and calculates the server load.

Similar to the piece distribution, it is clear that the server load tends to increase as the smoothing probability increases. Another prominent feature observed in Table III is that the *rarest* scenario has the lowest server load and the *linear* scenario has the highest one. This is in agreement with the results of our simulation study.

3) Continuity index: Data regarding the continuity index also is recorded by every leecher for every run. According to Table III, we observe that the *linear* scenario performs well with respect to the CI. This is an expected result. The competition of the *rarest* algorithm and *smoothing* algorithms can also be observed in Table III. It is clear that the average CI of both the *prob-0.2* and *prob-0.5* scenarios are bigger than the corresponding values of both the *rarest* and *prob-1.0* scenarios (*pure smoothing*). This means there must exist a suitable smoothing probability value to maximize the average CI, under the cooperation effect of the *rarest* and *smoothing* algorithms.

TABLE III
LAN RESULTS.

Scenario	CI	Server load
linear	0.998171 ± 0.000708	9.88 ± 0.38
rarest	0.997918 ± 0.000158	4.04 ± 0.16
prob-0.2	0.998034 ± 0.000219	4.34 ± 0.23
prob-0.5	0.998035 ± 0.000500	4.46 ± 0.27
prob-1.0	0.995163 ± 0.001665	8.64 ± 0.30

C. Results for PlanetLab experiments

Performing experiments based on PlanetLab is as close to a real world environment as possible since PlanetLab uses the common Internet infrastructure. Since PlanetLab runs over the public Internet, several factors induced by the real network of the PlanetLab environment must be taken into account, *i. e.*, Transport Control Protocol (TCP)-traffic, routing, utilization of hardware resources, and so on. Moreover, the functionality of the BiTES software need to be tested and adjusted to make itself more suitable to the real network environments. The corresponding results also need to be compared with ones obtained in the LAN experiment.

1) PlanetLab issues: We experienced several issues during our PlanetLab experiment. First and foremost, there appears to exist an automatic upload limitation for all nodes in PlanetLab, *e. g.*, 10 GB per node per day. Furthermore, some nodes are not usable, due to either being offline, overloaded, having prohibitively high latencies, or being more severely bandwidth-limited than others. To make things worse, these issues often occur only intermittently. Moreover, some nodes are part the same research group, and as such share the available upload allotment, forcing us to only use a single of these nodes.

We have performed extensive testing of more than 900 nodes in PlanetLab for their TCP connectivity characteristics. 427 PlanetLab nodes were suitable with respect to delay and bandwidth. In order to make BT swarm more close to the real-world one, we chose nodes from different continents, *i. e.*, European countries, America, and Canada. However, some nodes, located at areas with very high TCP connection delay, have not been chosen, *i. e.*, Japan, HongKong, and China. Finally, only 100 peers have been selected to participate in the measurement BT swarm. Moreover, 10 of these nodes are assigned as seeds to provide higher speed, and circumvent the bandwidth limitation of PlanetLab. This is also the reason for the slight parameter change in Table IV compared to Table II

We report the corresponding values in Table IV.

TABLE IV
PARAMETER VALUES FOR PLANETLAB EXPERIMENT.

Parameter	Value
Seeds	10
Peers	90
Maximum connections per peer	100
Maximum uploads per peer	8
Peers requested from tracker	50
Swarm inter-arrival time	exponential, mean 12.395s
Swarm inter-departure time	exponential, mean 12.395s
Initial piece distribution	0%
Playback buffer size	10 pieces
Piece size	256 kB
Total number of pieces	1071

The experiments started from August 22, 2009. The following scenarios have been tested in turn, until September 22, 2009: *rarest*, *prob-0.2*, *prob-0.5*, *prob-1.0* and *linear*. Each of these scenarios is run 15 times, due to the longer running time, compared to the two previous experiments.

2) Piece distribution: Figure 6 shows the average number of replicas per piece for all of the 15 experimental runs for the *rarest*, *prob-0.2*, *prob-0.5* scenarios, and 2 experimental runs for the *prob-1.0* scenario. We observed that the graphs for the different scenarios in Figure 6 correspond rather closely with the ones in Figure 5 of the LAN environment. This illustrates that both the *rarest* and *smoothing* algorithms also compete in affecting the piece distribution in the entire BT swarm, when run in the PlanetLab environment. However, we found that the graph for the *rarest* and *linear* scenarios in Figure 6 does not correspond to the one in Figure 5. We attribute this discrepancy to the changing of the nodes, as described in Section VII-C1.

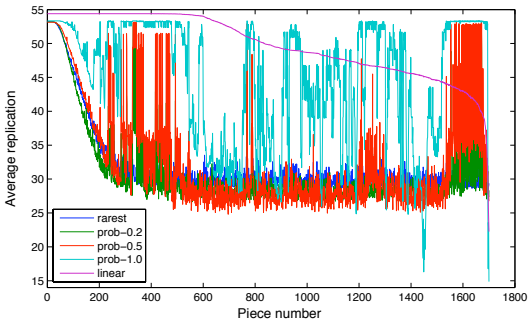

Fig. 6. Piece distribution in the PlanetLab experiment.

3) Confidence intervals: We calculate the 95 % confidence intervals of *server load* and *continuity index*, and report these in Table V. Here, the server load values are larger because 10 nodes were assigned as seeds during experiments, which means that more pieces are downloaded from seeds, compared to the simulations and the LAN experiment.

TABLE V
PLANETLAB RESULTS.

Scenario	CI	Server load
rarest	0.943560 ± 0.005332	338.11 ± 3.64
prob-0.2	0.947546 ± 0.004715	330.52 ± 4.27
prob-0.5	0.942406 ± 0.005448	335.63 ± 3.71
prob-1.0	0.953693 ± 0.004782	367.43 ± 3.11
linear	0.909031 ± 0.014132	368.72 ± 3.53

VIII. CONCLUSIONS

In this paper, we suggest a number of modifications to the BitTorrent (BT) distribution system in order to make it a suitable solution for distributing streaming media. Also, we present a simulation study of these modifications and report on selected results from this study. Furthermore, we report a comprehensive real-world measurement study of these modifications on both a local as well as a global scale.

The most salient result is that by downloading the pieces closest to consumption in a sequential fashion, BT becomes usable as a streaming media distribution transport mechanism. However, the entire media stream should not be downloaded sequentially, as this yields poor performance with respect to both piece distribution and server load. While the performance of the linear algorithm is poor compared to the other algorithms, it still outperforms conventional, non-collaborative, streaming.

Further, the rarest-first algorithm has been shown to perform well with respect to all the evaluated metrics, as long as pieces close to consumption are downloaded sequentially. However, the smoothing algorithm performed significantly worse than the simpler probabilistic counterpart.

The real-world results show that the LAN experiment corresponds well to the simulation study, with respect to all three metrics: server load, Continuity Index, and piece distribution. The PlanetLab experiments show more disparate results, which we attribute to several factors:

- The environment is highly dynamic compared to the previous scenarios and measurements.
- The number of seeds were higher, which also affects the results.
- The PlanetLab environment performs bandwidth throttling after a certain amount of data has been transmitted.

Future work is to carry out the *deadline* BT downloading scenario on both LAN and PlanetLab environments. The corresponding piece distribution, server load and continuity index will be evaluated. The evaluation results need to be compared with simulation and experimental study exhibited in this paper.

REFERENCES

[1] Conviva. http://www.conviva.com/.
[2] PeerCast. http://www.peercast.org/.
[3] A. Sharma, A. Bestavros, and I. Matta. dPAM: A Distributed Prefetching Protocol for Scalable Asynchronous Multicast in P2P Systems. In *Proceedings of IEEE International Conference on Network Protocols(ICNP)*, 2005.
[4] Y. Cui, B. Li, and K. Nahrstedt. oStream: Asynchronous streaming multicast in application-layer overlay networks. In*IEEE Journal on Selected Areas in Communications*, 2004
[5] Cohen B. Incentives build robustness in bittorrent. In *1st Workshop on the Economics of Peer-2-Peer Systems*, Berkley, CA, June 5-6 2003.
[6] BitTorrent, Inc. Bittorrent DNA. http://www.bittorrent.com/dna/.
[7] C. Dana, D. Li, D. Harrison, and C.N. Chuah. BASS: Bittorrent assisted streaming system for video-on-demand. In *International Workshop on Multimedia Signal Processing (MMsP)*, 2005.
[8] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for supporting streaming applications. In *9th IEEE Global Internet Symposium (GI2006)*, Barcelona, Spain, April 2006.
[9] Vuze, Inc. Vuze. http://www.vuze.com/.
[10] The Tribler Team. Tribler. http://www.tribler.org/.
[11] BitTorrent protocol specification. http://wiki.theory.org/BitTorrentSpecification/.
[12] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 203–216, 2006.
[13] The P2P-next project. http://www.p2p-next.org.
[14] A. Varga, R. Hornig. An overview of the OMNeT++ simulation environment. In*Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems*, 2008.
[15] Orange Open Movie Project. Elephants dream. http://www.elephantsdream.org/.
[16] The PlanetLab project. http://www.planet-lab.org/.
[17] David Erman. Bittorrent traffic measurements and models, October 2005. Licentiate thesis, Blekinge Institute of Technology.
[18] David Erman. *On BitTorrent Media Distribution*. PhD thesis, Blekinge Institute of Technology, March 2008.