On Piece Selection for Streaming BitTorrent

David Erman
Dept. of Telecommunication Systems
School of Engineering
Blekinge Institute of Technology
371 79 Karlskrona, Sweden
david.erman@bth.se

Karel De Vogeleer
Dept. of Telecommunication Systems
School of Engineering
Blekinge Institute of Technology
371 79 Karlskrona, Sweden
karel.de.vogeleer@bth.se

Adrian Popescu
Dept. of Telecommunication Systems
School of Engineering
Blekinge Institute of Technology
371 79 Karlskrona, Sweden
adrian.popescu@bth.se

***Abstract*—Video-on-Demand (VoD) 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 reactualized the research into efficient distribution methods for streaming video. In this paper, we propose a number of modifications and extensions to the BitTorrent (BT) 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 of the implemented extensions to the BT system.**

I. Introduction

Large-scale, real-time multimedia distribution over the Internet has been the subject of research for a substantial amount of 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, though 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.

Systems such as End-System Multicast (ESM) [ESM] and Peer-Cast [PCast] are being used to stream video and audio to large subscriber groups. Furthermore, approaches such as Distributed Prefetching Protocol for Asynchronous Multicast (dPAM) [SBM05] and oStream [CLN03] provide intelligent application layer multicast routing and caching services. One of the most popular distribution and replication systems is BitTorrent (BT) [Coh06], developed to distribute large files and decrease the load on the initial file server.

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 IV, we describe the piece selection algorithm BT as well as present a set of modifications to them for enabling BT as a streaming solution. In Section V, we present the experiment details on a simulation study we have performed to test our modifications to the BT algorithms. We present the results of this study in Section VI. Finally, Section VII conludes the paper.

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.

Data transfer in a BT swarm is performed by leechers downloading fixed-size parts, called *pieces*, in smaller parts, called *blocks*, of the content 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.

A more detailed description of the BT protocol and algorithms is found in [Coh05]. The choking algorithm is more precisely defined in [LUKM06].

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 [Bit], which is based on a modified BT infrastructure with added management and Quality of Service (QoS) capabilities.

Another solution offered by industrial players is the service offered by Vuze [Vuz], which is based on the open source Azureus BT client. Vuze offers streaming delivery of both free and pay-per-view Standard-definition (SD) and High-definition (HD) content.

BitTorrent-Assisted Streaming System (BASS) [DLHC05] is a hybrid approach in which a traditional client-server 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 utilisation 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 [DLHC05] 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 distribution. Furthermore, the traffic is analysed from a single tracker log for one specific content.

BitTorrent Streaming (BiToS) [VIF06] 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.

Pieces from the HP set are downloaded with priority p, while pieces from the RP set are downloaded with priority $1 - p$.

IV. Piece Selection

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 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, 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 BitTorrent Extensions for Streaming (BEST) [Erm08]. One of the major changes is the addition of per-piece deadlines to the swarm meta-data, to facilitate the algorithms below.

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.

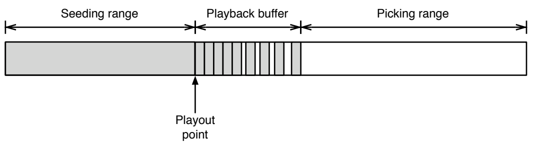

Fig. 1. Piece ranges.

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 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 underutilised. The following is a description of the algorithm:
   a) Pick a piece using rarest-first, p_r, with interdeadline 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.

V. Simulations

To test the BEST modifications, we have run a number of simulations. The simulator used was validated against real BT traffic characteristics.

In total, we have run four sets of simulations, where each set consists of six simulation scenarios [Erm08]. 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). 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 [Erm08]. 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 is the default size recommended by the BT specification [Coh05].

A. Evaluation Metrics

We evaluate the algorithm performance usign three distinct metrics and measures, namely *piece distribution*, *continuity index* and *server load*.

1) Piece distribution: Piece distribution is a measure of how well a BT swarm is behaving. 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. For our work, we opt for a visual measure, and use distribution graphs to assess the quality of the piece distribution.

2) 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.

3) 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.

VI. 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 [Erm08].

A. 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 reason for

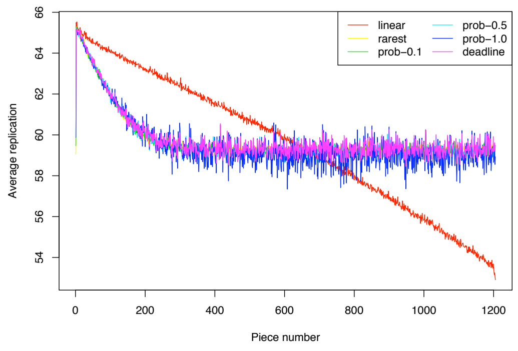

Fig. 2. Piece distribution results, set 1.

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.

B. 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

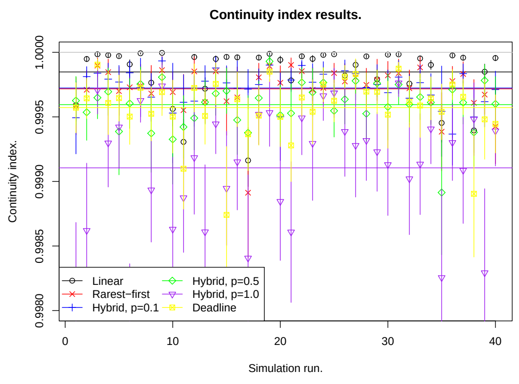

Fig. 3. Continuity index results, set 1.

perform similarly with respect to the CI. 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 indicate missing more than one piece deadline. Second are the large confidence intervals evident on a per-run basis for the *prob-1.0* scenario.

C. Server Load

We present the results for the server load metric in Figure 4. As 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 observe

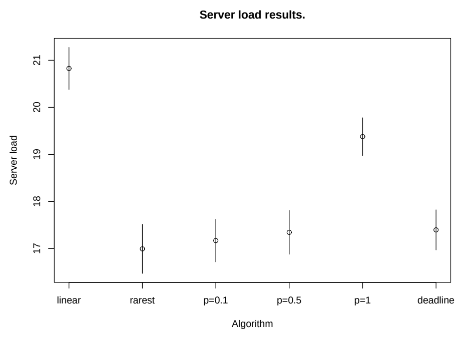

Fig. 4. Server load results, set 1.

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 amount of time 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. Conclusions

We have presented a number of modifications to the BitTorrent (BT) distribution system in order to make it a suitable solution for distributing streaming media. Also, we have presented a simulation study of these modifications and report on selected results from this study.

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 web 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 deadline-based smoothing algorithm performed significantly worse than the simpler probablistic counterpart.

VIII. Future Work

While the results of the simulations presented in this paper and related publication [Erm08] are encouraging for using BT as a transport for streaming media, there is additional work we wish to pursue. One set of improvements would be to make the inputs to the simulator more indicative of real-world conditions. Primarily, this would entail using a better topology generator, and extending the input torrent files with deadlines from real video streams. Furthermore, we intend to implement the extensions in a real client and to deploy it in lab environments as well as on a global scale.

References

[Bit] BitTorrent, Inc. Bittorrent DNA. http://www.bittorrent.com/dna/.

[CLN03] Y. Cui, B. Li, and K. Nahrstedt. oStream: Asynchronous streaming multicast in application-layer overlay networks, 2003.

[Coh05] Bram Cohen. BitTorrent protocol specification. http://www.bitconjurer.org/BitTorrent/protocol.html, February 2005.

[Coh06] Bram Cohen. BitTorrent. http://www.bittorrent.com/, March 2006.

[DLHC05] Chris Dana, Danjue Li, David Harrison, and Chen-Nee Chuah. BASS: Bittorrent assisted streaming system for video-on-demand. In *Proceedings of IEEE 7th Workshop on Multimedia Signal Processing*, pages 1–4, October 2005.

[Erm08] David Erman. *On BitTorrent Media Distribution*. PhD thesis, Blekinge Institute of Technology, March 2008.

[LUKM06] Arnaud Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 203–216, New York, NY, USA, 2006. ACM Press.

[PCast] PeerCast.org. Peercast – p2p casting for everyone. Online at http://peercast.org. URL verified on March 27, 2007.

[SBM05] Abhishek Sharma, Azer Bestavros, and Ibrahim Matta. dPAM: A Distributed Prefetching Protocol for Scalable Asynchronous Multicast in P2P Systems. In *Proceedings of Infocom'05: The IEEE International Conference on Computer Communication*, Miami, Florida, March 2005.

[ESM] The ESM Project. ESM – end system multicast. Online at http://esm.cs.cmu.edu/. URL verified on March 26, 2007.

[VIF06] Aggelos Vlavianos, Marios Iliofotou, and Michalis Faloutsos. BiToS: Enhancing BitTorrent for supporting streaming applications. In *9th IEEE Global Internet Symposium (GI2006)*, Barcelona, Spain, April 2006.

[Vuz] Vuze, Inc. Vuze. http://www.vuze.com.