ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία
Μεταπτυχιακού Διπλώματος Ειδίκευσης

«Implementation and Performance Evaluation of Video Streaming in the BitTorrent Protocol»

Άγγελος Αρχοντοβασίλης

Επιβλέπων: Γεώργιος Ξυλωμένος

ΑΘΗΝΑ, ΙΟΥΝΙΟΣ 2010

Abstract

BitTorrent has been one of the most effective mechanisms for P2P content distribution in recent years. Although BitTorrent was created for distribution of time insensitive content, in this work it is shown that with minimal changes BitTorrent can support streaming. The importance of this streaming capability is that the peer may have the ability of watching the video before the complete download of the file. This way, the peer can evaluate the quality of the video content early and decide if this particular video is worth spending time and resources.

The research was conducted using our full featured and extensible implementation of BitTorrent for the Omnet++ simulation environment developed in the Mobile Multimedia Laboratory of Athens University of Economics and Business. The implementation includes modifications in the BitTorrent piece selection strategy, giving higher download priority to pieces that are close to be reproduced by the player, in contrast to the original rarest-first BitTorrent policy. This was achieved by using a sliding window containing the pieces to be played next and by awarding them a higher download priority than the rest of the remaining pieces. Whenever a piece does not meet the player's deadline, it is considered as missed and it is not downloaded. In the simulation scenarios missed pieces for each peer are counted; this is the most critical parameter for proper video streaming, and with this metric we came up with our results.

Key words

BitTorrent, video streaming, peer-to-peer, module, Omnet++, simulation, peer, seeder, piece, block, swarm, sliding window, high priority set, buffering time, missed pieces, piece loss, bit rate.

Περίληψη

Τα τελευταία χρόνια το πρωτόκολλο BitTorrent έχει εξελιχθεί στον πιο αποτελεσματικό P2P μηχανισμό για διαμοιρασμό αρχείων. Παρόλο που ο σκοπός δημιουργίας του πρωτοκόλλου αφορούσε διανομή περιεχομένου χωρίς χρονικούς περιορισμούς, στην παρούσα εργασία γίνεται εμφανές ότι με ελάχιστες αλλαγές το BitTorrent μπορεί να υποστηρίξει video streaming. Η σημασία της ικανότητας υποστήριξης streaming είναι ότι ο χρήστης έχει τη δυνατότητα να παρακολουθήσει ένα video πριν από την ολοκλήρωση της λήψης του αρχείου. Επιπρόσθετα, ο χρήστης μπορεί να αξιολογήσει την ποιότητα του video πολύ νωρίς και να αποφασίσει εάν το συγκεκριμένο αρχείο αξίζει τον χρόνο και τους πόρους που ξοδεύονται για χάρη του.

Για την διεξαγωγή της έρευνας χρησιμοποιήθηκε ένα BitTorrent module για το περιβάλλον προσομοίωσης Omnet++, το οποίο αναπτύχθηκε στο Mobile Multimedia Laboratory του Οικονομικού Πανεπιστημίου Αθηνών. Η υλοποίηση περιλαμβάνει τροποποιήσεις στην στρατηγική επιλογής των κομματιών (pieces) του BitTorrent, απονέμοντας υψηλότερη προτεραιότητα λήψης στα κομμάτια εκείνα που είναι κοντά στην αναπαραγωγή τους από τον player. Η υλοποίηση όπως περιγράφηκε, έρχεται σε αντίθεση με την αρχική rarest-first πολιτική του BitTorrent. Η αλλαγή της κλασσικής αυτής πολιτικής, επιτεύχθηκε με την χρήση ενός «κυλιόμενου παραθύρου», το οποίο περιέχει τα κοντινότερα στην αναπαραγωγή κομμάτια. Στα κομμάτια αυτά δίνεται υψηλότερη προτεραιότητα έναντι των υπολοίπων. Τα κομμάτια τα οποία δεν έχουν ληφθεί κατά τη χρονική στιγμή που ο player τα χρειάζεται για να παίξουν, χαρακτηρίζονται ως «χαμένα» και σταματάει η διαδικασία λήψης τους. Στα σενάρια προσομοίωσης βασική μετρική είναι τα χαμένα κομμάτια για κάθε peer του συστήματος. Πρόκειται για την κρισιμότερη παράμετρο, όσον αφορά το video streaming και με αυτήν την μετρική κατέληξα στα αποτελέσματά της εργασίας.

Λέξεις κλειδιά

BitTorrent, video streaming, σύστημα ομότιμων, module, πρωτόκολλο, Omnet++, προσομοίωση, χρήστης, peer, seeder, piece, block, swarm, κυλιόμενο παράθυρο, high priority set, buffering time, missed pieces, piece loss, bit rate.

Acknowledgements

First of all, I would like to thank my supervisor, George Xylomenos for giving me the chance to work on this topic and for his continual support and advice throughout this work. I also would like to thank Professor George Polyzos for evaluating my work.

I owe my deepest gratitude to Charilaos Stais, PhD student of our department, for helping and encouraging me to keep going when things did not go as expected. Furthermore, I am extremely grateful to PhD student – and soon to be – Dr. Konstantinos Katsaros for his valuable advices whenever I needed help.

Last but not least, I want to thank my parents, Pantelis and Eleni, for supporting and encouraging me in all these years of my academic adventures.

Ευχαριστίες

Καταρχάς, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Γεώργιο Ξυλωμένο που μου έδωσε την ευκαιρία να ασχοληθώ με το συγκεκριμένο θέμα και για τη συνεχή υποστήριξη και καθοδήγηση του καθ' όλη την πορεία αυτής της διπλωματικής. Θα ήθελα επίσης να ευχαριστήσω τον καθηγητή κ. Γεώργιο Πολύζο, που αξιολόγησε την εργασία μου.

Οφείλω μεγάλη ευγνωμοσύνη στον υποψήφιο διδάκτορα του τμήματος μας, Χαρίλαο Στάη, για την βοήθεια και τη μεγάλη υποστήριξη που μου προσέφερε όποτε τα πράγματα δεν πήγαιναν όπως αναμενόταν. Επιπρόσθετα, είμαι ιδιαίτερα ευγνώμων στον υποψήφιο διδάκτορα – και πολύ σύντομα – Δρ. Κωνσταντίνο Κατσαρό για τις πολύτιμες συμβουλές του όποτε χρειαζόμουν βοήθεια.

Τέλος, θέλω να ευχαριστήσω τους γονείς μου, Παντελή και Ελένη, για την αμέριστη υλική και ηθική υποστήριξή τους και το μεγάλο κουράγιο που μου προσέφεραν όλα αυτά τα χρόνια της ακαδημαϊκής μου περιπέτειας.

Table of Contents

1. BitTorrent ... 8
1.1 Overview ... 8
1.2 Operation ... 9
2. BitTorrent Module for Omnet++ ... 10
2.1 The Tracker Protocol ... 10
2.2 The Peer-wire Protocol ... 10
2.2.1 Protocol overview ... 10
2.2.2 Connections ... 11
2.2.3 Piece downloading strategy ... 11
2.2.4 Queueing ... 12
2.2.5 Endgame mode ... 12
3. Video Streaming ... 13
3.1 The Uses and Benefits ... 13
3.2 Video Streaming using BitTorrent ... 13
3.3 Related work ... 13
4. BitTorrent Module modifications ... 17
4.1 Piece selection strategy for video streaming ... 17
4.2 Buffering time ... 17
4.3 Video Player ... 17
5. Simulation Parameters ... 18
5.1 Swarm size ... 18
5.2 Piece size ... 18
5.3 HPS size ... 18
5.4 HPS probability p ... 18
5.5 Video bit rate ... 19
6. Simulation Results ... 20
6.1 Simulation metrics ... 20
6.2 Scenarios – Graph description ... 20
6.3 Piece size ... 20
6.4 Block size ... 21
6.5 Swarm size ... 22
6.6 HPS size ... 22
6.7 Buffering delay ... 23
6.8 Video bit rate ... 24
6.9 Overview ... 26
7. Comparison with related work ... 27
8. Conclusions ... 28
9. Future work ... 28
9.1 Adaptation of HPS probability p ... 29
9.2 Missed pieces decision ... 29
10. References ... 30

List of Figures

Figure 1: Sliding window algorithm 14
Figure 2: BiToS approach for supporting streaming in BitTorrent . . . 15
Figure 3: Windowing Algorithms 16
Figure 4: Piece size scenarios 21
Figure 5: Block size scenarios 21
Figure 6: Swarm size scenarios 22
Figure 7: HPS Size scenarios 23
Figure 8: Additional buffering delay scenarios 23
Figure 9: Decreasing bit rate scenario 24
Figure 10: 300 Kbps bit rate snapshot 25
Figure 11: 150 Kbps bit rate snapshot 25
Figure 12: Simulation scenarios overview 26

List of Tables

Table 1: Peer-wire protocol parameters 12
Table 2: Simulation parameters 19
Table 3: P2P Multimedia streaming using BitTorrent results . . . 27
Table 4: Video streaming for BitTorrent module for Omnet++ results . . 27

1. BitTorrent

1.1 ***Overview*** [1]

BitTorrent is a peer-to-peer file sharing protocol used for distributing large amounts of data. BitTorrent is one of the most common protocols for transferring large files, and it has been estimated that it accounts for roughly 27-55% of all Internet traffic (depending on geographical location) as of February 2009. [2]

The BitTorrent protocol allows users to distribute large amounts of data without the heavy demands on their computers that would be needed for standard Internet hosting. A standard host's servers can easily be brought to a halt if high levels of simultaneous data flow are reached. The protocol works as an alternative data distribution method that makes even small computers (e.g. mobile phones) with low bandwidth capable of participating in large data transfers.

First, a user playing the role of file-provider makes a file available to the network. This first user is called a *seeder* and its availability on the network allows other users, called *peers*, to connect and begin to download from the seeder. As new peers connect to the network and request the same file, their computer receives a different piece of the data from the seeder. Once multiple peers have multiple pieces of the file, BitTorrent allows each to become a source for that portion of the file. The effect of this is to take on a small part of the task and relieve the initial user, distributing the file download task among the seeder and many peers. With BitTorrent, no one computer needs to supply data in quantities which could jeopardize the task by overwhelming all resources, yet the same final result—each peer eventually receiving the entire file—is still reached.

After the file is successfully and completely downloaded by a given peer, the peer is able to shift roles and become an additional seeder, helping the remaining peers to receive the entire file. This eventual shift from peers to seeders determines the overall 'health' of the file (as determined by the number of times a file is available in its complete form).

This distributed nature of BitTorrent leads to a flood like spreading of a file throughout peers. As more peers join the swarm, the likelihood of a successful download increases. Relative to standard Internet hosting, this provides a significant reduction in the original distributor's hardware and bandwidth resource costs. It also provides redundancy against system problems, reduces dependence on the original distributor [12] and provides a source for the file which is generally temporary and therefore harder to trace than when provided by the enduring availability of a host in standard file distribution techniques.

1.2 ***Operation*** [1]

A BitTorrent client is any program that implements the BitTorrent protocol. Each client is capable of preparing, requesting, and transmitting any type of computer file over a network, using the protocol. A peer is any computer running an instance of a client.

To share a file or group of files, a peer first creates a small file called a "torrent" (e.g. MyFile.torrent). This file contains metadata about the files to be shared and about the tracker, the computer that coordinates the file distribution. Peers that want to download the file must first obtain a torrent file for it and connect to the specified tracker, which tells them from which other peers to download the pieces of the file.

Though both ultimately transfer files over a network, a BitTorrent download differs from a classic download (as is typical with an HTTP or FTP request, for example) in several fundamental ways:

- BitTorrent makes many small data requests over different TCP connections to different machines, while classic downloading is typically made via a single TCP connection to a single machine.
- BitTorrent downloads in a random or in a "rarest-first" [13] approach that ensures high availability, while classic downloads are sequential.

Taken together, these differences allow BitTorrent to achieve much lower cost to the content provider, much higher redundancy, and much greater resistance to abuse or to "flash crowds" than regular server software. However, this protection, theoretically, comes at a cost: downloads can take time to achieve full speed because it may take time for enough peer connections to be established, and it may take time for a node to receive sufficient data to become an effective uploader. This contrasts with regular downloads (such as from an HTTP server, for example) that, while more vulnerable to overload and abuse, rise to full speed very quickly and maintain this speed throughout.

The strength of BT lies in its ability to resist to the Free-Riders phenomenon, in which selfish peers choose only to download the file without uploading. BT uses a Tit-for-Tat policy, where each peer chooses to upload to its peer as long as it takes something in return. If the neighbor peer behaves selfishly the Choking mechanism is invoked and the peer stops uploading to its neighboring peer. [4]

In general, BitTorrent's non-contiguous download methods have prevented it from supporting "progressive downloads" or "streaming playback". However, comments made by Bram Cohen in January 2007 suggest that streaming torrent downloads will soon be commonplace and ad supported streaming appears to be the result of those comments.

2. BitTorrent Module for Omnet++ [3]

Even though peer to peer content distribution remains one of the most active research areas, little progress has been made towards the study of the BitTorrent protocol, and its possible variations, in a fully controllable but realistic simulation environment. In the Mobile Multimedia Laboratory a full featured and extensible implementation of BitTorrent for the OMNeT++ simulation environment was developed. This implementation is briefly described in the following chapters.

2.1 ***The Tracker Protocol***

Typically, the distribution of a new file [14] with BitTorrent starts by publishing a .torrent metafile; this metafile is distributed to peers using an out-of-band channel, usually by posting the metafile on a web page. Among other information, a .torrent metafile contains the tracker address, file size, piece size, and the hashes for the file pieces. Trackers are responsible for helping peers discover each other so as to form a swarm. In most cases, each .torrent metafile is served by a single tracker; the *trackerless* approach employs *Distributed Hash Tables* (DHTs) for decentralized peer discovery. Clients communicate with the tracker via a simple text-based protocol, layered on top of HTTP/HTTPS, using the tracker's URL stored inside the metafile. During the download phase, each client communicates with the tracker and publishes its progress (in terms of total bytes downloaded/uploaded), as well as its *contact details* (e.g., IP address, TCP port, identification info). These parameters are passed from the client to the tracker using the standard HTTP GET method [14]. Note that most of the information *announced* by the client is for statistical purposes; only the IP address and TCP port of a client are crucial for the tracker. After such a message, called a *tracker request*, the tracker randomly selects a set of peers and returns their contact details in a *bencoded* dictionary [10]. Since each tracker request provides the contact details of a client to the tracker, the tracker can return such details back in its replies. In this manner, over time the peers discover increasing subsets of the swarm.

2.2 ***The Peer-wire Protocol***

The peer-wire protocol provides the core BitTorrent functionality: interaction with remote peers. In the following an overview of the implemented peer-wire protocol is presented and then some details of its operation, focusing on the most important features available in the implementation.

2.2.1 Protocol overview

After contacting the tracker, a client attempts to establish TCP connections with the peers listed in the tracker response. Upon connection establishment, the two peers exchange HANDSHAKE messages in order to verify their peer identities and ensure that they are interested in the same torrent metafile. This handshake is then followed by an exchange of BITFIELD messages that contain the bitfield of each client, which is a bitmap denoting the availability of each piece at the client. Based on that information a client can determine whether it is interested in one or more pieces offered by the remote peer. By following the above procedure over multiple [15] peer connections, a client collects information regarding the availability of the pieces that it is still missing in the subset of the swarm that has explored using that information. Based on this information, it then decides which pieces to

request from each peer. In general, if a peer does not hold any pieces that the client does not already hold, a NOT INTERESTED message is sent to that peer to indicate the lack of interest for its data. At the beginning of a connection, peers are assumed not to be interested in each other's pieces by default. Although at this stage a client knows the peers interested in, it cannot make any requests yet as data cannot be exchanged until the remote peer actively permits this by sending an UNCHOKE message. This means that each client is by default blocked, or, in Bit-Torrent parlance, *choked* by the corresponding remote peer. The decision to choke or unchoke a client is made based on several criteria embodied in the *choking algorithm* [10] of the protocol:

Reciprocation: Peers unchoke the clients providing the best upload rates.

TCP performance: TCP behaves better when the number of simultaneous uploads is capped.

Fibrillation avoidance: Frequent choking/unchoking causes data transfer interruptions that deteriorate protocol performance.

Optimistic unchoking: New peers are occasionally unchoked so as to discover potentially better connections. Moreover, peers are thus given the chance to acquire their first pieces. When a client is unchoked by a peer, it starts sending REQUEST messages, each asking for a specific block of the selected piece. The peer sends back the requested data using PIECE messages. Upon completing the downloading of a piece, a client informs via HAVE messages the peers that it has established connections to. These peers update the bitfield for that client and may then, potentially, express their interest for that piece with an INTERESTED message.

2.2.2 Connections

A client learns about other peers by employing the Tracker protocol and parsing the peer list returned by the tracker. The client then joins the swarm by establishing connections with some peers. However, as noted in [10], each connection incurs an increase in signaling traffic, especially for bitfield maintenance via the exchange of HAVE messages.

2.2.3 Piece downloading strategy

The piece downloading strategy refers to the policy followed in the selection of the pieces that will be requested from a peer. It is an important aspect of BitTorrent as it heavily affects the diversity of the pieces available at each peer. A low degree of diversity would result in low interest for a peer's pieces, thus causing degraded application performance. The two most prevalent piece downloading strategies, *Rarest First* and *Random First* are implemented. Based on the information gathered during the BITFIELD – HAVE message exchange, the *Rarest First* strategy selects those pieces that appear less frequently in a client's set of connected peers. This selection is randomized among several of the less common pieces, according to the RAREST LIST SIZE configuration parameter (see Table 1), in order to avoid multiple peers converging on the same piece. This way, peers download pieces that most other peers probably want, therefore facilitating data exchange. However, rare pieces are present only in a few peers, and it is possible that downloading them may be interrupted due to a choking decision. Clients with no pieces in their possession would therefore have to wait for an optimistic unchoking event (see Section 2.2.1) from a peer holding the same rare piece in order to continue downloading. The *Random First* strategy avoids this problem by selecting a random piece which is more likely to be available from multiple peers, so that a choking decision would not have such an adverse effect.

2.2.4 Queueing

As mentioned above, REQUEST messages refer to specific blocks of a piece. This facilitates fine-grained data exchange by enabling the queueing of data requests. As common piece sizes vary from 256 KB to 1 MB [10] or even larger, per piece requests would result in a multitude of redundant packet retransmissions in the event of a choking decision during piece transfer. A window-based queueing mechanism is employed for these requests; otherwise propagation delays would dominate the total download time. The implemented queueing policy includes a generic queueing mechanism in which the user can specify the exact size of the queue. In this mechanism a client may send to a peer up to REQUEST QUEUE LENGTH (see Table 1) REQUEST messages for blocks. Once a PIECE message has been received, the client may send the next REQUEST message. Once a piece has been requested in its entirety, if the request queue is not full, the client chooses another desired piece from that peer's bitfield according to the piece selection strategy (see Section 2.2.3) and starts sending REQUEST messages for its blocks.

2.2.5 Endgame mode

The endgame mode addresses the problem of slow transfers for the last data blocks of an exchange, since at that stage most pieces have been downloaded; therefore the degree of parallelization is low. In this mode the client sends REQUEST messages for each missing block to all peers that are not choking it, as opposed to a single peer. Another unclarified aspect of the endgame mode regards the entry condition. In this implementation, the client enters this mode when the number of missing blocks equals the number of requested blocks, meaning that all missing blocks have been requested. This feature can be turned on/off using the END GAME MODE configuration parameter (see Table 1).

Parameter	Default Value
file size (MB)	700
piece size (KB)	256
block size (KB)	16
DHT port	-1
pstr	BitTorrent protocol
pstrlen	19
keep alive (sec)	120
have supression	true
choking interval (sec)	10
downloaders	4
optUnchokedPeers	1
optUnchoking interval (sec)	30
seederDownloaders	4
seederOptUnchokedPeers	1
rarest list size	5
minNumConnections	30
maxNumConnections	55
timeToSeed (sec)	0
request queue length	5
super seed mode	false
end game mode	true
maxNumEmptyTrackerResponses	5
newlyConnectedOptUnchokeProb	0.75
downloadRateSamplingDuration (sec)	20

Table 1: Peer-wire protocol parameters

3. Video Streaming

3.1 ***The Uses and Benefits*** [5]

The Internet provides users many ways to access video files online. The traditional method involves downloading a video file just like a user would with a regular document or a picture file. The user must wait for the download to finish, then open and view the video. Streaming videos, however, let users view a video as it is being downloaded from the Internet.

Streaming video technology is useful for real-time and on-demand requests. A common use for streaming videos is for viewing movie clips. Many movie studios provide previews or trailers of their upcoming film features through streaming video clips. Also from the entertainment business, music labels use this technology to stream music videos for the public.

Streaming videos are also used in the field of education. Pre-recorded lectures are available from many university and educational websites. This allows students to view and listen to lectures over and over again.

For all the advantages of streaming videos, there are also limitations and disadvantages. For videos to be delivered quickly to a user's desktop for viewing, two things have to be considered: the size of the video and the Internet bandwidth available to the user. Compressing videos can normally sacrifice their quality, and a small bandwidth normally results in choppy video playback. Another drawback to the availability of streaming videos is the increase in network or Internet traffic.

3.2 ***Video Streaming using BitTorrent*** [6]

Although multimedia streaming networks have similarities with BitTorrent swarms, BitTorrent in itself is not suitable for multimedia streaming since it does not account for the real-time needs of streaming applications. While highly successful in large-scale content distribution [16, 17, 18], traditional BitTorrent [19] has one major drawback when it comes to video-on-demand (VoD): the viewer has to wait for the whole video to download before he or she can start watching the video, because BitTorrent splits the file to be downloaded into pieces that are downloaded in a non-sequential rarest-first order [10]. Also, the tit-for-tat policy forces too many peers to wait for too long before joining the swarm.

To alleviate this problem, research [4, 7] has suggested applying the rarest-first piece selection only within a small window that slides through the file to be downloaded. In order to implement this kind of “windowing”, only minor changes to BitTorrent clients are needed, and the modified clients are compatible enough to be used for downloading data from standard BitTorrent swarms.

3.3 ***Related work***

Two different approaches have been proposed for utilizing BitTorrent in multimedia streaming and video-on-demand. The server-assisted systems such as BASS [8] and PONDER [20] combine BitTorrent with a traditional server in order to alleviate the server load. Another approach is modifying the BitTorrent piece selection strategy. BiToS [4] utilizes a piece selection algorithm in which a sliding window called “the high priority set” slides

through the file to be downloaded. Shah and Pâris [7] have also proposed using a similar sliding window. Finally, Savolainen, Raatikainen, and Tarkoma [6] suggested an adaptive window solution, or, as they call it, stretching window.

The purpose of this thesis does not include adding a streaming server, so I followed the second approach by changing the piece selection algorithm (see Section 2.2.3). By comparing the three available propositions, I came up with the most suitable algorithm to implement for the BitTorrent Module for Omnet++.

➢ Shah and Paris [7] propose a *sliding window* algorithm, which is depicted in Fig. 1.

- Fixed size window: contains the next w chunks to be consumed by a client
  - window size remains unchanged
- Chunks that arrive after their scheduled playback deadline are dropped
- Downloading pieces located ahead of the current window is not allowed
  - all window pieces must be downloaded before the window can move
- Using rarest-first policy to select pieces from the sliding window

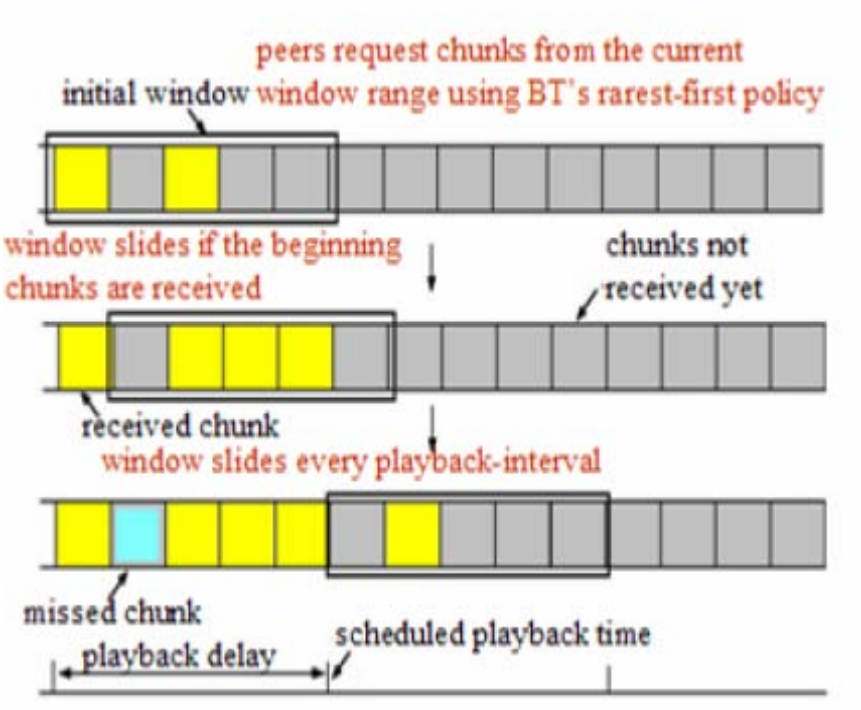

Fig. 1. Sliding window algorithm

➢ BiToS [4] introduces the term *High Priority Set* (HPS), which works similarly to the sliding window described above and is illustrated in Fig. 2. The main differences from the sliding window algorithm are that in BiToS the window moves when a single window piece is downloaded and that pieces are not selected only from the HPS, but requests are made for pieces outside the window as well.

- The peer chooses with some probability p to download a piece of the video stream, which is contained in the High Priority Set and with probability 1– p a piece outside HPS.

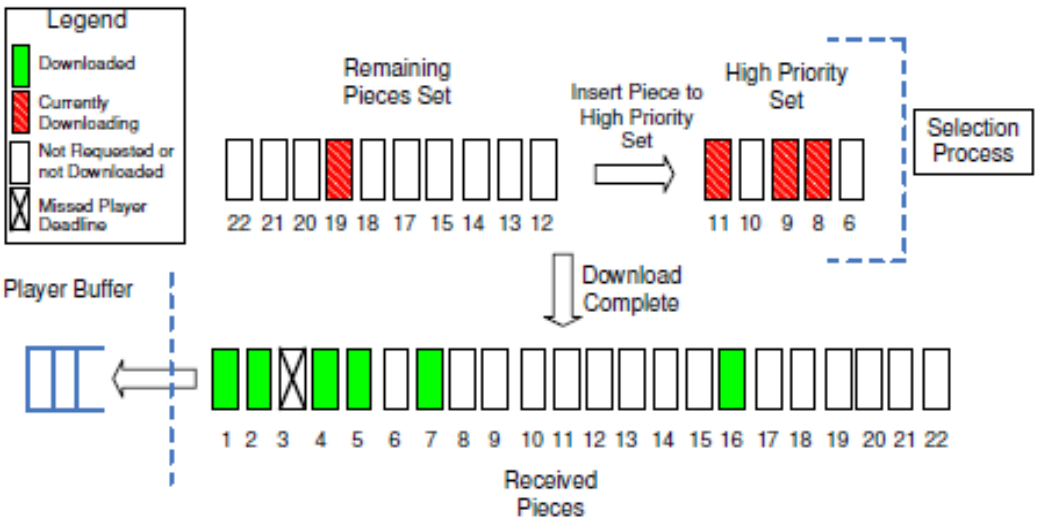

Fig. 2. BiToS approach for supporting streaming in BitTorrent

In both of the previous algorithms, the beginning point of the window is defined as the first piece that has not yet been downloaded but has not missed its playback deadline. Both solutions also define a window size, measured in number of pieces. The difference in window definition between the two is that, when calculating which pieces are within the window, the fixed-size window algorithm counts both the arrived and non-arrived pieces, whereas BiToS only takes into account the non-arrived pieces, which means that the distance of the first and the last piece in the BiToS window can grow large in certain circumstances.

- The *stretching window* algorithm [6] has a critical difference from the sliding window: it is not of fixed size. The three propositions are illustrated in Fig. 3.

  - Adaptive size of the window: the bound *b* constrains the distance between the first and the last piece in the window, so the window adapts its size according to *b*.
    - The stretching window algorithm has two different maximum sizes that are enforced simultaneously: the BiToS-like “maximum number of non-arrived pieces in the window” and the fixed-window-like “maximum absolute distance between the first and the last piece in the window”.
  - In the stretching window algorithm pieces are always requested from within the window.

WINDOWING ALGORITHMS

Algorithm	Maximum Window Size	Probability of Requesting from within the Window
Fixed-size Window	*w* pieces	1
BiToS	*w* non-arrived pieces	*p* (typically 0.8)
Stretching Window	*w* non-arrived pieces or *b* pieces	1

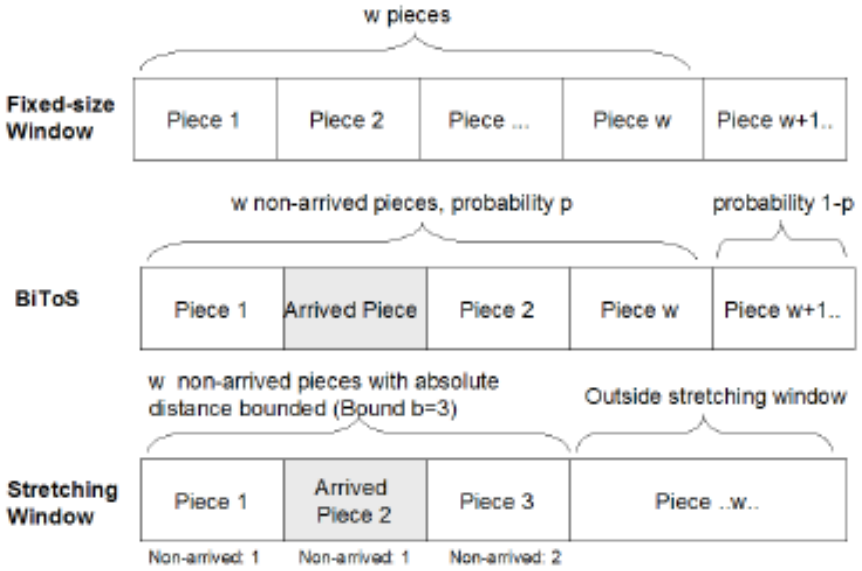

Fig. 3. Windowing Algorithms

BitTorrent is based on Tit-for-Tat policy; each peer chooses to upload to its peer as long as it takes something in return. If the neighbor peer behaves selfishly the Choking mechanism is invoked and the peer stops uploading to its neighboring peer. As a result, each peer must have some rare pieces to exchange to avoid being choked. For that reason, I believe that BiToS [4] approach is more suitable for BitTorrent streaming, because it does not constrain the piece selection inside the window and that leads to high piece diversity. In conclusion, my implementation of streaming for the BitTorrent Module for Omnet++ is very similar to the BiToS approach.

4. BitTorrent Module modifications

4.1 ***Piece selection strategy for video streaming***

As mentioned before, the selection strategy for video streaming is very similar to the BiToS approach [4], which is illustrated in Fig. 2.

In the implementation there is an entity called **High Priority Set** (HPS), which contains the pieces of the Video Stream that have not been downloaded yet, are not *missed* and are soon to be reproduced by the player. Thus, these pieces have higher priority to be requested over the remaining pieces. This set has a fixed size of pieces and this size is a system parameter. A piece in this set can be in the following states: Not-Requested or Currently-Downloading.

In the selection process, the peer chooses with some probability p to download a piece of the video stream, which is contained in the *High Priority Set* and with probability 1 – p, a piece from the remaining – and not missed – ones. It is actually a *coin toss* with probability p for HPS and probability 1–p for non-HPS.

After a piece is downloaded, the piece is removed from its current set and joins the received pieces (Download Complete function in Fig. 2). At the same time, if the piece was in the High Priority Set, the next in order piece is added to the High Priority Set from the remaining pieces. For example in Fig. 2, piece *12* will move to the High Priority Set if any of the *currently-downloading* pieces becomes *downloaded*. In this way the cardinality of the High Priority Set remains fixed. The pieces within the sets do not have to be consecutive since the pieces are not requested in order.

4.2 ***Buffering time***

An initial buffering delay is needed for proper streaming. The player waits until the first pieces of the file are downloaded and then starts playing the video, while the rest of the pieces continue the download process. In my implementation, the parameter notifying the player on how many pieces must wait to be downloaded, is set to five (5), but can be changed by the user in the simulation .ini files.

By increasing the initial buffering time we can achieve a significant decrease of piece loss and therefore better streaming performance.

4.3 ***Video Player***

The implemented entity that acts as a video player; waits for the buffering time to end and then starts playing the pieces one by one in sequential order. The player is triggered by an event every *x* seconds and asks for the next piece to play. The parameter x can be set by the user in the .ini files and it actually represents the streaming bit rate (e.g. how many seconds does a 256KB piece need to be played).

While the player keeps asking for the next piece, if a requested piece is not available (downloaded), then the piece stops downloading (if already requested) and joins the **missed pieces set**. By counting the size of this particular set, for every peer participating in the simulation scenarios, we acquire very important knowledge about streaming performance.

5. Simulation Parameters

5.1 ***Swarm size***

The number of peers participating in the swarm is a crucial parameter for the overall performance of the BitTorrent protocol. In the simulation scenarios I tried different swarm size values in order to see if the protocol would work as expected: more peers → better performance.

5.2 ***Piece size***

BitTorrent allows piece sizes as small as 32 KB, the default size being 256 KB. Table 2 shows the values used for the simulation scenarios. A smaller piece size leads to a shorter initial buffering time. Smaller piece size means that the window size in kilobytes is smaller, and thus a smaller amount of data needs to be downloaded before the first piece in the window has arrived and the window can move forward. However, a small piece size also leads to larger .torrent files, because the .torrent file includes a hash for each piece. Another concern is that the number of requests for pieces and announcements of received pieces increases, causing an increase in overhead. A decrease in throughput can also be seen when moving to smaller piece sizes. On the other hand, by using a larger piece size the overhead and the .torrent file size are decreased, but this could lead in huge piece loss rate, which is far worse.

5.3 ***HPS size***

The size of the High Priority Set is crucial for streaming in BitTorrent. In the BiToS approach [4], as well as in my implementation, the HPS size is expressed as a percentage of the total number of pieces that the file is split to.

By using a really small size of the High Priority Set, peers do not increase the diversity of the pieces because they tend to download the same pieces due to the small size of the set. This results in low parallelism in downloading, which stalls the downloading process and results in low streaming performance. On the other hand when the size of the list is large (>20%), the peer downloads pieces based on their rareness, without considering their deadline, so we are closer to the original BitTorrent, which is not suitable for streaming video. In other words, the optimal size of the High Priority Set must capture the pieces that will be needed soon for the playback and at the same time is large enough for the rarest first piece selection mechanism to work properly. Table 2 shows the values of HPS size used for the simulation scenarios.

5.4 ***HPS probability p***

The probability p can have an important impact on the performance of the Streaming. Large values of p guarantee that the pieces that will be reproduced soon, will be requested for downloading earlier than the rest of the pieces of the video stream. On the other hand, this could lead to a situation in which the peer chooses to download pieces that most of the peers have. Therefore, the peer wouldn't have any rare pieces to exchange and consequently would be choked by most of the peers according to the Incentive mechanism of BT. Apart from this, rare pieces that are currently available might not be available in the future. For example, peers that have these pieces might leave the network or fail. Hence, by acquiring these rare pieces before they become extinct we can increase the QoS. [4]

5.5 ***Video bit rate***

The issue for video bit rate is choosing an acceptable compromise between two opposing goals: high bit rate offers better *video quality* but might cause significant *piece loss* issues and on the other hand, low bit rate can guarantee a smooth streaming process but no guarantees about the video quality.

Table 2 shows the values for bit rate that were used for the simulation scenarios. As mentioned before the player is triggered by an event in every x seconds and plays the next piece in order; the player actually plays one piece of the file in x seconds. By choosing x = 7 sec with piece size of 256 KB, the bit rate is about 300 Kbps. Furthermore, by using x = 7 sec with piece size of 128 KB the bit rate is about 150 Kbps and so on.

Parameter	Value
file size (MB)	200
swarm size	60, 120, 180
piece size (KB)	64, 128, 256
block size (KB)	8, 16
HPS size (%)	2, 3, 8, 20
HPS probability	0.8, 0.9
video bit rate (kbps)	150, 300
timeToSeed (sec)	360

Table 2: Simulation Parameters

6. Simulation Results

6.1 ***Simulation metrics***

The most crucial parameters for video streaming were used as metrics: **Buffering delay** and **Piece loss**. The initial buffering delay was common in most of the scenarios; except for the first 2-3 peers entering the system, the rest had to wait from 7 till 27 seconds. The first peers entering the swarm cannot find many seeders so, naturally, they end up with delay time from 230 to 260 seconds. Piece loss as a metric is more interesting because it cannot be predicted so easily and will be discussed in detail in the simulation scenarios below.

6.2 ***Scenarios – Graph description***

From a total of 56 simulation scenarios, I chose to introduce the most interesting ones, grouping them together so as to point out the most interesting conclusions. The following graphs show the best working simulation parameters for video streaming in the BitTorrent Module for Omnet++ [3]. Every scenario is described in the CDF graphs by the following:

- **Swarm size**, expressed by the number of participating peers
- The parameter **x** (see Section 4.3) for the seconds needed for a piece to be played by the player (or in other words, the streaming **bit rate**)
- **HPS size** expressed as a percentage of the total number of pieces (e.g. 2hps → 2% HPS size)
- **Piece size**, expressed in Kilobytes (e.g. 128piece → 128 KB piece size)
- **Block size**, expressed in Kilobytes (e.g. 16block → 16 KB block size)
- **BitTorrent** indication means that the scenario ran with original BitTorrent approach (no HPS, no coin toss, just rarest-first BitTorrent protocol)
- **Time indication** in the end of the description means that additional buffering delay was added in the particular scenario (e.g. 360sec → initial buffering delay + 360 seconds)

The CDF graphs show the **peers percentage** on the vertical axis and the **piece loss percentage** on the horizontal axis. Since the swarm size had 3 different values depending on the simulation scenario, peers percentage was the more appropriate approach. Piece loss (%) indicates the number of missed pieces as a percentage of the total pieces of the file.

6.3 ***Piece size***

- For large values (256KB) the window is not flexible enough, missed pieces occur more often
- For very small values (64KB) the increased overhead reduces the number of *successful* pieces

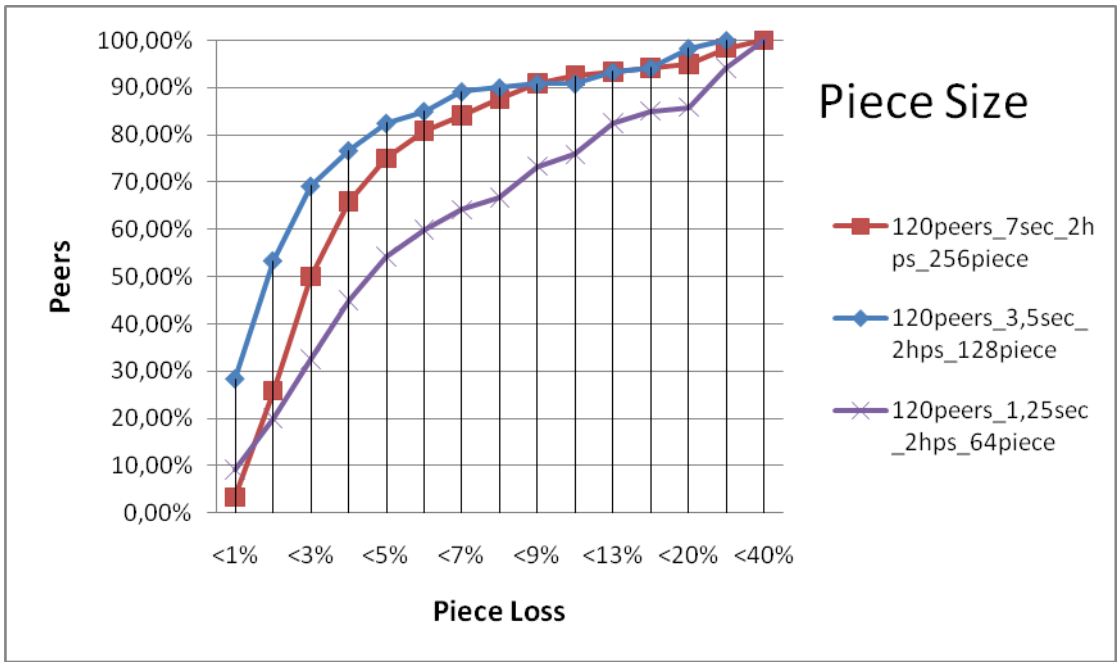

Fig. 4. Piece Size scenarios

6.4 ***Block size***

Pieces are divided into blocks; each piece gets downloaded block by block; as a result block size can affect the download process. In our case (Fig. 5) there is not much difference, when the block size is 8 KB or 16 KB. There is a slightly decrease of missed pieces when using 8 KB block size.

Fig. 5. Block Size scenarios

*6.5 **Swarm size***

As we can see in Fig. 6, the swarm size does not seem to affect much the piece loss rate. This probably has to do with the fact that peers do not join and leave the swarm as they would do in a *flash crowd scenario*. In the simulation scenarios, peers were entering and leaving the system with a *steady rate* and that is probably the explanation to what the CDF graph on Fig. 6 is showing. A slight decrement of missed pieces is noticeable as the swarm size gets bigger.

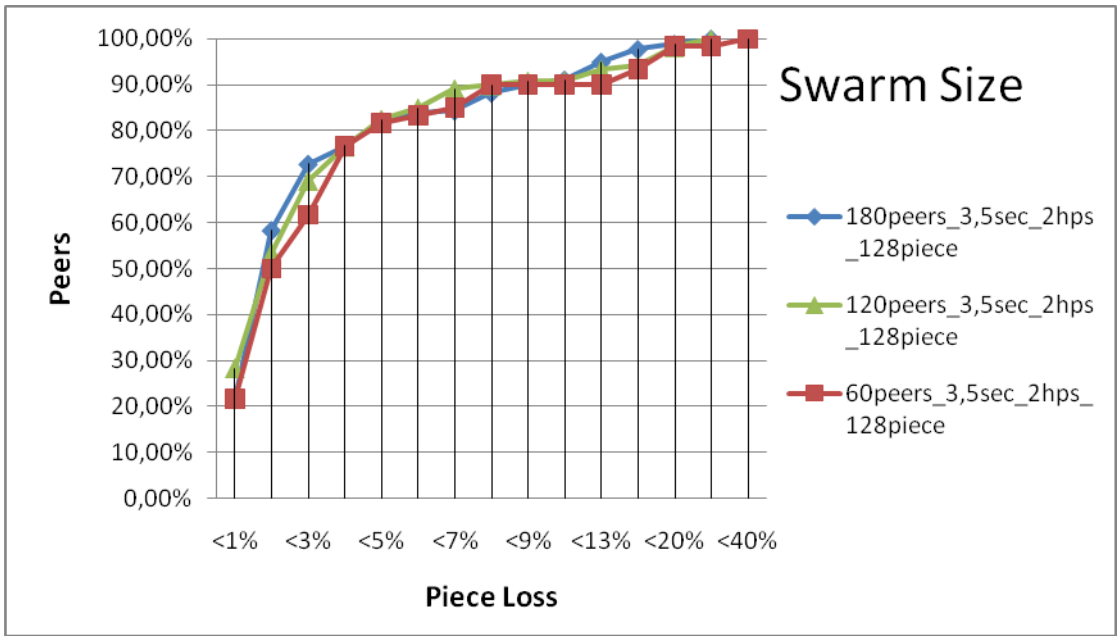

Fig. 6. Swarm Size scenarios

*6.6 **HPS size***

The CDF graph on Fig. 7 leads us to some really interesting observations about the HPS size and streaming performance relation:

- The original BitTorrent is unacceptable for video streaming
- A large HPS size (20% of the file size) is the worst case scenario, close to original BitTorrent
- The BiToS [4] recommended HPS size (8% of the file size) is far away from the optimal
- A small HPS size (2%) is obviously the best choice when dealing with large files

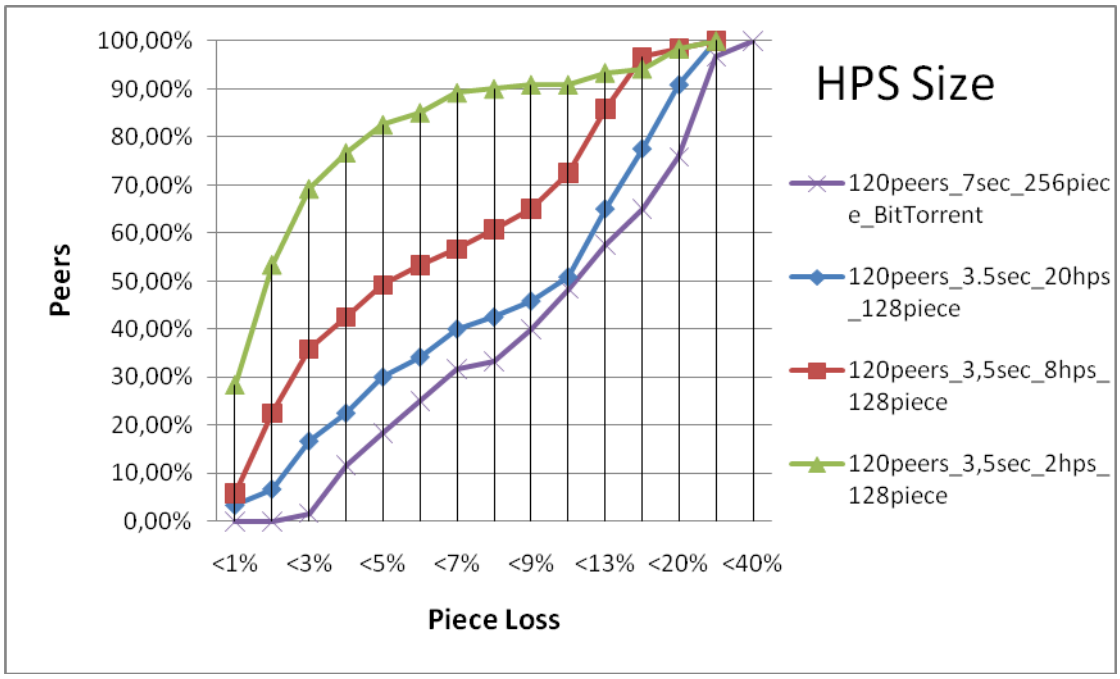

Fig. 7. HPS Size scenarios

6.7 ***Buffering delay***

As mentioned before, by adding an initial buffering delay we can significantly improve the streaming performance. Figure 8 confirms the above: the bigger the initial buffering delay, the better for the system's performance.

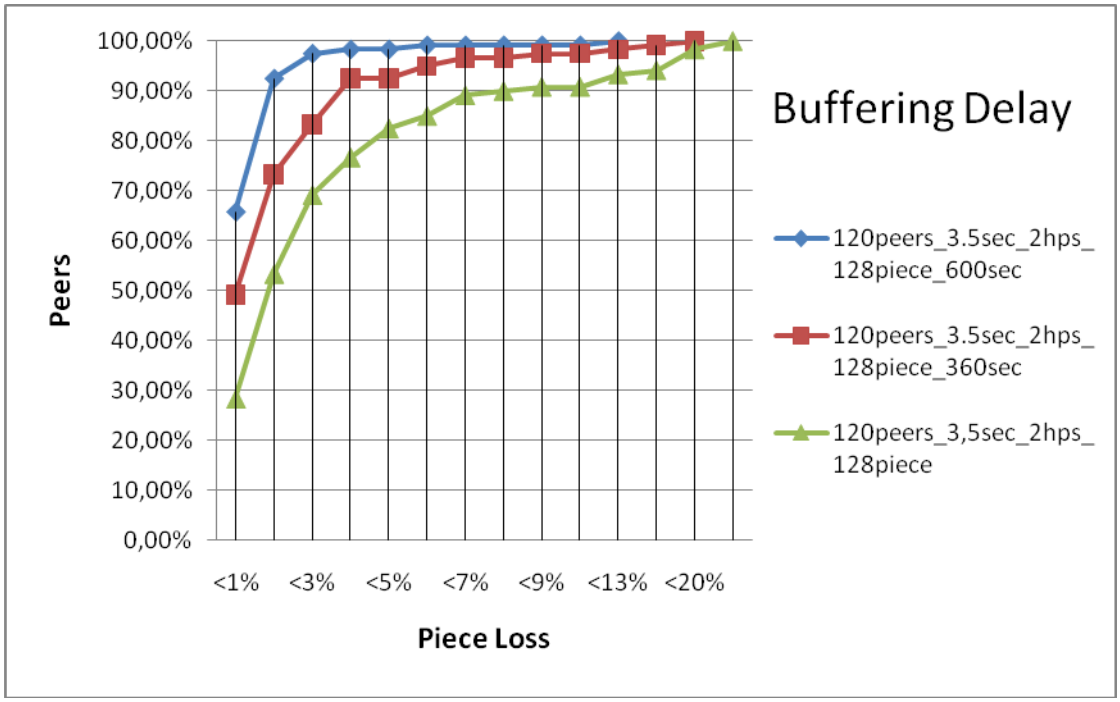

Fig. 8. Additional buffering delay scenarios

6.8 ***Video bit rate***

Figure 9 depicts the minimal piece loss when reducing the bit rate by half. The quality of the video though gets significantly lower as we can see in Figures 10-11. The low bit rate scenario can be used on time-sensitive applications, which do not demand high video quality.

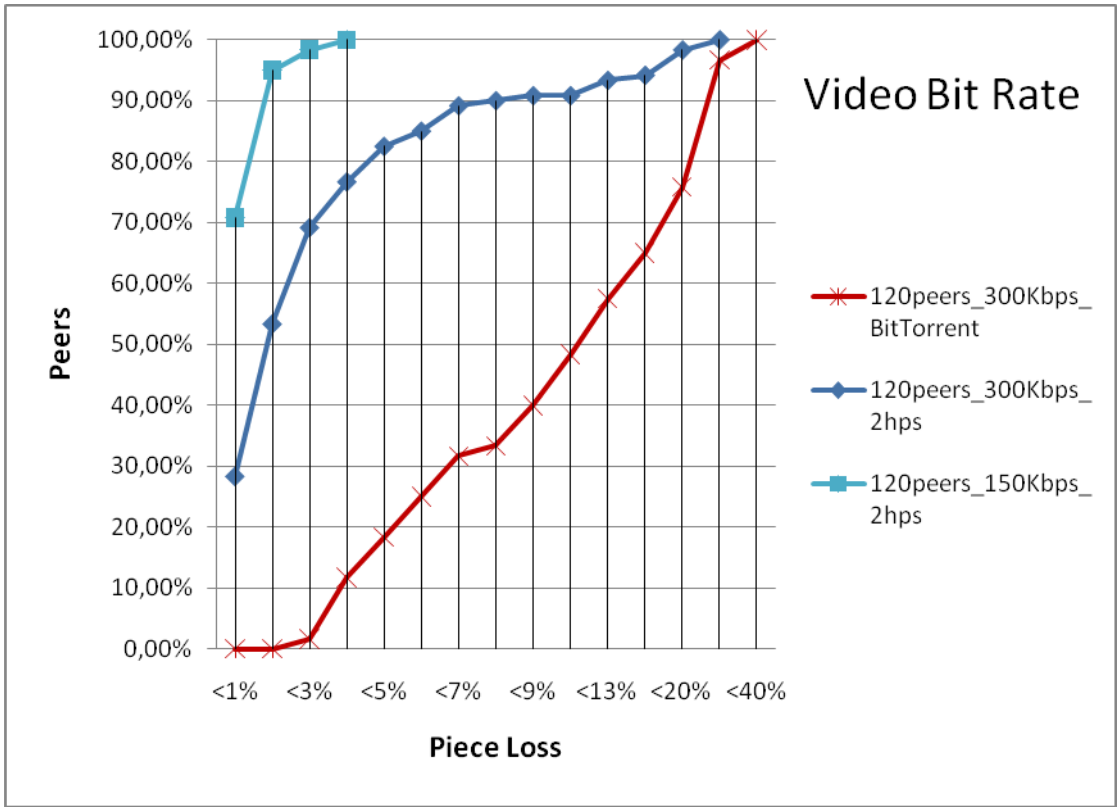

Fig. 9. Decreasing bit rate scenario

Fig. 10. 300 Kbps bit rate snapshot

Fig. 11. 150 Kbps bit rate snapshot

6.9 ***Overview***

Figure 12 shows an overview of the most interesting simulation scenarios.

- Depending on file size, modifications can increase streaming quality
- The original BitTorrent mechanism is clearly unsuitable for streaming
- For large files, a small HPS (on the order of 2% of the file) works best
- Increasing the buffering delay leads to lower loss rates
- Reducing the bit rate of the content leads to lower loss rates

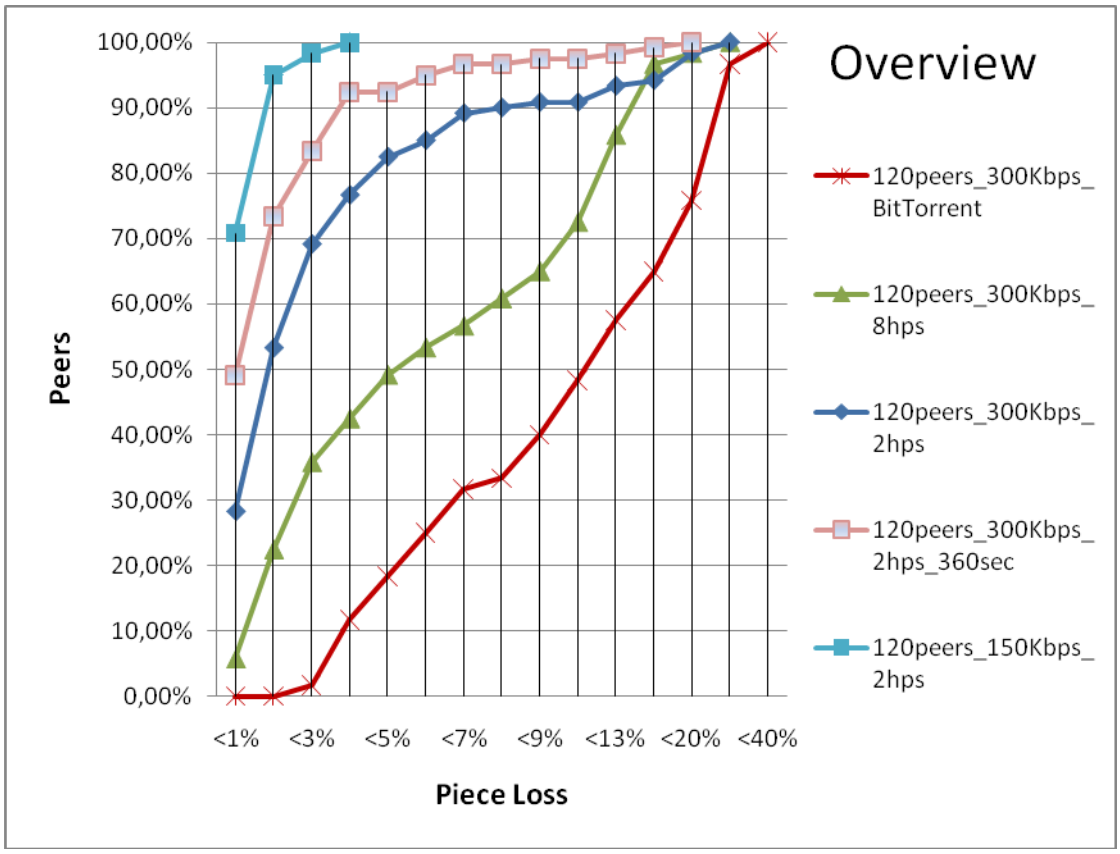

Fig. 12. Simulation scenarios overview

7. Comparison with related work

Shah and Pâris [7] in their paper *P2P Multimedia Streaming using BitTorrent*, used similar parameter values and metrics with these described above. Comparing their results with mine can lead to useful conclusions about the video streaming performance of the BitTorrent module for Omnet++.

Success ratio: This metric represents the playback continuity defined as the number of pieces that arrive before a scheduled playback deadline over the total number of pieces in the video file. Success ratio is used to quantify the performance of the P2P multimedia streaming network.

Peer-to-Peer Multimedia Streaming Using BitTorrent	
150MB file size, 256KB piece size, 100 swarm size, 60sec delay	
Piece Selection Policy	**Success Ratio**
Rarest-first policy (original BitTorrent)	23.6%
Sequential policy	8.1%
Sliding window and rarest-first policy	83.3%

Table 3: P2P Multimedia streaming using BitTorrent results

BitTorrent Multimedia Streaming Module for Omnet++	
200MB file size, 120 swarm size, 7-27sec delay	
Piece Selection Policy	**Success Ratio**
HPS (8% of the file size) – 256KB piece size	91.59%
HPS (2% of the file size) – 256KB piece size	93.88%
HPS (2% of the file size) – 128KB piece size	95.64%

Table 4: Video streaming for BitTorrent module for Omnet++ results

- **Peer-to-Peer Multimedia Streaming Using BitTorrent**
  - Best case scenario success ratio (average): 83.3%

- **BitTorrent Streaming Module for Omnet++**
  - Best case scenario success ratio (average): 95.64%
  - Could reach 99% for 150Kbps streaming, or 97.84% by adding 360sec buffering delay

8. Conclusions

In this thesis we show that with minimal changes BitTorrent can support streaming. The presented findings showed that, by carefully optimizing streaming parameters, a decent level of performance can be achieved, while leaving the original BitTorrent tit-for-tat mechanism intact. The parameters described below are optimized for streaming large video files (>100 MB):

- Video Streaming operates more efficiently with 128KB *piece size* and 8KB *block size*
- The recommended *HPS size* is 2% of the file size
- The possibility *p* to choose a piece from the HPS, should be near 0.8
- We could significantly increase streaming performance by adding an initiative *buffering delay* or by decreasing the streaming *bit rate*
- Larger *swarm size* increases streaming performance

In comparison with Shah and Pâris, "Peer-to-peer multimedia streaming using BitTorrent" [7] while using similar parameters, it is shown that this approach works significantly better. An important reason for this performance difference is the use of the featured and extensible BitTorrent module for Omnet++ [3], designed and implemented in the Mobile Multimedia Laboratory of Athens University of Economics and Business, which was extended to make this thesis possible.

This research, in contrast with others, is not altruism-dependent. Although peer altruism, in terms of seeding, is desirable and can significantly improve streaming performance, it is not needed for proper streaming. The simulation results have shown that seeding while watching the video is enough for this approach to have a decent streaming performance.

9. Future work

9.1 ***Adaptation of HPS probability p***

The adaptation of the probability p can be triggered by events, such as a deadline miss. For example, a miss of a piece's deadline while there are many pieces unplayed inside the Received Pieces Buffer indicates that the probability p should be increased in order to give higher priority to the pieces that have shorter deadlines. On the other hand, if we miss many deadlines and there are no other received pieces and the download rate is small, this could indicate that the peer is chocked by most of its peers, because it doesn't have pieces to exchange. Therefore, the decrease of the value of the probability can be helpful in order to acquire some rare pieces that the peer can use as leverage.

9.2 ***Missed pieces decision***

Missing pieces is the most crucial factor affecting streaming performance. We could reduce piece loss rate by choosing not to "throw away" pieces that have been downloaded in a large proportion. When a piece meets its deadline and it is not fully downloaded we could check how many of its blocks we already have and decide if the piece can be played or not.

10. References

[1] Wikipedia - BitTorrent

[2] BitTorrent Still King of P2P Traffic

[3] K. Katsaros, V. P. Kemerlis, C. Stais and G. Xylomenos, "A BitTorrent Module for the OMNeT++ Simulator," Proc. 17th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), London, Great Britain, September 2009 [PDF]

[4] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications", in Global Internet Workshop in conjunction with IEEE INFOCOM 2006, April 2006 [PDF]

[5] Streaming Video, By Damian Sofsian

[6] P. Savolainen, N. Raatikainen, and S. Tarkoma, "Windowing BitTorrent for video-on-demand: Not all is lost with tit-for-tat," in *Proc. of IEEE GLOBECOM, 2008.* [PDF]

[7] P. Shah and J.-F. Pâris, "Peer-to-peer multimedia streaming using BitTorrent," in *Proceedings of the 26th IEEE International Performance, Computing, and Communications Conference (IPCCC '07)*, pp. 340–347, New Orleans, La, USA, April 2007. [PDF]

[8] C. Dana, D. Li, D. Harrison, and C. N. Chuah, "BASS: BitTorrent Assisted Streaming System for Video-on-Demand," in IEEE International Workshop on Multimedia Signal Processing (MMSP), October 2005. [PDF]

[9] R. LaFortune, C. D. Carothers, W. D. Simth, J. Czechowski and X. Wang, "Simulating Large-Scale P2P Assisted Video Streaming", In Proceedings of the Hawaii International Conference on System Sciences (HICSS-42), Waikoloa, Big Island, Hawaii, January 2009. [PDF]

[10] BitTorrent Development Community. BitTorrent Protocol Specification v1.0
http://wiki.theory.org/BitTorrentSpecification

[11] András Varga OMNeT++ network simulator homepage
http://www.omnetpp.org

[12] Estimating Self-Sustainability in Peer-to-Peer Swarming Systems by D. Menasche, A. Rocha, E. de Souza e Silva, R. M. Leao, D. Towsley, A. Venkataramani

[13] A. Legout, G. Urvoy-Keller and P. Michiardi, "Rarest first and choke algorithms are enough", In Proc. ACM SIGCOMM'06. [PDF]

[14] T. Berners-Lee, L. Masinter, and M. M. (eds). Uniform Resource Locators (URL). Internet Request for Comments, December 1994. RFC 1738.

[15] A. Bharambe, C. Herley, and V. Padmanabhan, *"Analyzing and improving BitTorrent performance"*, Technical Report MSR-TR-2005-03, Microsoft Research, 2005. [PDF]

[16] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, L. Garces-Erice, *"Dissecting BitTorrent: Five months in a torrent's lifetime"*, In Proceedings of the 5th Passive and Active Measurement Workshop, Antibes Juan-les-Pins, France, April 2004. [PDF]

[17] J. Pouwelse, P. Garbacki, D. Epema, H. Sips, *"The Bittorrent P2P File-Sharing System: Measurements and Analysis"*, In IPTPS 2005, Ithaca, USA, February 2005. [PDF]

[18] Yang X. de Veciana G, *"Service Capacity of Peer to Peer Networks"*, In Proceedings of IEEE INFOCOM 2004. [PDF]

[19] B. Cohen, *"Incentives Build Robustness in BitTorrent"*, In Workshop on Economics of Peer-to-Peer Systems, Berkeley, USA, May 2003. [PDF]

[20] Guo,Y. Mathur, S. Ramaswamy, K . Yu, S. Patel, B. PONDER, *"Performance Aware P2P Video-on-Demand Service"*, In IEEE GLOBECOM, Washingston, DC, USA. November 2007. [PDF]