On BitTorrent Media Distribution

David Erman
On BitTorrent Media Distribution

David Erman

Department of Telecommunication Systems
School of Engineering
Blekinge Institute of Technology
SWEDEN
Till minnet av Göran Erman.
Abstract

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 reactualised the research on multicast, with the consequence that many of the previous mechanisms and schemes are being re-evaluated.

This thesis provides a brief overview of several important techniques for media broadcasting and stream merging, as well as a discussion of traditional IP multicast and overlay multicast. Additionally, 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, as well as a detailed validation study of the BT simulator itself. Furthermore, we present a comprehensive set of BT models for several important traffic characteristics, at both session and message levels.
The work in this thesis was in part supported by the Swedish Agency for Innovation Systems (VINNOVA), Swedish Foundation for Internet Infrastructure (IIS), and the European Next Generation Internet (EuroNGI), and European Future Generation Internet (EuroFGI) Network of Excellences (NoE).

First, I would like to thank my advisor, Professor Adrian Popescu. I appreciate the trust placed in me as a student, colleague, and scientist. I would also like to thank Professor Arne Nilsson for accepting me as a graduate student and Dr. Markus Fiedler for interesting, rewarding, and heated discussions.

My fellow graduate students, present and past, in particular the other two D’s, Doru Constantinescu and Dragos Ilie, also deserve my gratitude: Doru, for nitpicking everything I write, and Dragos, for making me explain myself.

I also want to extend my gratitude to some of the Master students that have been part of the studies in this thesis. Daniel and José, you did a great job in helping out with the second set of measurements. Karel, your work on the simulator has been of enormous assistance. I thank you all.

I am grateful to my friends, for keeping my focus on what is important.

Last but not least, I want to thank my family for their support and encouragement, in particular my mother for reminding me of what that factory floor was like and my fiancée, Maria, for putting up with my endless comments regarding “heavy workload” and incessant working.

David Erman
Karlskrona, February 2008
# Contents

<table>
<thead>
<tr>
<th>1 Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Motivation</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Contributions</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Related Work</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Thesis Outline and Structure</td>
<td>8</td>
</tr>
</tbody>
</table>

## I Media Distribution

<table>
<thead>
<tr>
<th>2 Multicast Systems</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 IP Multicast</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Application Layer Multicast</td>
<td>25</td>
</tr>
<tr>
<td>2.3 Summary</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 Broadcasting Strategies</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Terminology</td>
<td>34</td>
</tr>
<tr>
<td>3.2 Conventional Broadcasting</td>
<td>35</td>
</tr>
<tr>
<td>3.3 Staggered Broadcasting</td>
<td>35</td>
</tr>
<tr>
<td>3.4 Pyramid Schemes</td>
<td>35</td>
</tr>
<tr>
<td>3.5 Staircase Schemes</td>
<td>37</td>
</tr>
<tr>
<td>3.6 Harmonic Schemes</td>
<td>38</td>
</tr>
<tr>
<td>3.7 Hybrid Schemes</td>
<td>39</td>
</tr>
</tbody>
</table>
### 3.8 Summary

#### 4 Stream Merging Strategies

4.1 Batching ............................... 42  
4.2 Piggybacking ............................ 43  
4.3 Patching ................................ 45  
4.4 Chaining ................................ 49  
4.5 Hierarchical and Hybrid Merging .................. 50  
4.6 Summary ............................... 51

#### 5 Caching Strategies

5.1 Replacement Policies .......................... 55  
5.2 Segment-based Caching ........................ 57  
5.3 Smoothing and Prefetching .......................... 58  
5.4 Summary ............................... 60

### II BitTorrent Streaming

#### 6 BitTorrent Streaming and Piece Selection

6.1 BitTorrent ............................... 63  
6.2 Piece Selection Algorithms for Static Content  ......... 66  
6.3 Piece Selection for Streaming Content ............. 70  
6.4 BitTorrent Extensions for Streaming .................. 76  
6.5 Summary ............................... 81

#### 7 BitTorrent Traffic Models

7.1 Measurements ............................... 84  
7.2 Modelling Methodology .......................... 85  
7.3 Session Models ............................... 89  
7.4 Message Models ............................... 98  
7.5 Summary ............................... 100
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Group communication.</td>
</tr>
<tr>
<td>2.2</td>
<td>Multicast architectures.</td>
</tr>
<tr>
<td>3.1</td>
<td>Stream parameters.</td>
</tr>
<tr>
<td>4.1</td>
<td>Batching methods for a single video object.</td>
</tr>
<tr>
<td>4.2</td>
<td>Piggybacking system state.</td>
</tr>
<tr>
<td>4.3</td>
<td>Chaining.</td>
</tr>
<tr>
<td>6.1</td>
<td>Reference client single piece picker policies.</td>
</tr>
<tr>
<td>6.2</td>
<td>Frames and pieces.</td>
</tr>
<tr>
<td>6.3</td>
<td>Azureus piece selection over time.</td>
</tr>
<tr>
<td>6.4</td>
<td>Piece ranges.</td>
</tr>
<tr>
<td>6.5</td>
<td>VBR video.</td>
</tr>
<tr>
<td>6.6</td>
<td>Illustration of the deadline-based scheme.</td>
</tr>
<tr>
<td>7.1</td>
<td>Session interarrival times.</td>
</tr>
<tr>
<td>7.2</td>
<td>Request rate CCDF and EPDF for measurement 11.</td>
</tr>
<tr>
<td>8.1</td>
<td>Dual Gaussian fit of average upstream request rates.</td>
</tr>
<tr>
<td>8.2</td>
<td>Average piece distribution.</td>
</tr>
<tr>
<td>8.3</td>
<td>Piece distribution illustration.</td>
</tr>
</tbody>
</table>
8.4 Average piece distribution without linear algorithm. . . . . . . . . . . . . . 117
8.5 Piece distributions for all algorithms and all runs, set 1. . . . . . . . 118
8.6 Server load results. ............................................................................. 120
8.7 Continuity index results. ................................................................. 122

A.1 Piece distribution with 95% confidence intervals, set 1. ........ 128
A.2 Piece distribution with 95% confidence intervals, set 2. ........ 129
A.3 Piece distribution with 95% confidence intervals, set 4. ........ 130
A.4 Piece distribution with 95% confidence intervals, set 4. ........ 131

B.1 Piece distributions for all algorithms and all runs, set 2. .... 134
B.2 Piece distributions for all algorithms and all runs, set 3. .... 135
B.3 Piece distributions for all algorithms and all runs, set 4. .... 136
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Group communication types.</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Pagoda segment-to-channel mapping.</td>
<td>39</td>
</tr>
<tr>
<td>6.1 Piece picking ranges.</td>
<td>77</td>
</tr>
<tr>
<td>7.1 Fitness quality boundaries.</td>
<td>88</td>
</tr>
<tr>
<td>7.2 Session summary for study 1.</td>
<td>90</td>
</tr>
<tr>
<td>7.3 Session interarrival parameters for study 1.</td>
<td>91</td>
</tr>
<tr>
<td>7.4 Session upstream size parameters for study 1.</td>
<td>93</td>
</tr>
<tr>
<td>7.5 Session duration parameters for study 1.</td>
<td>94</td>
</tr>
<tr>
<td>7.6 Session summary for study 2.</td>
<td>95</td>
</tr>
<tr>
<td>7.7 Session interarrival parameters for study 2.</td>
<td>96</td>
</tr>
<tr>
<td>7.8 Session duration parameters for study 2.</td>
<td>97</td>
</tr>
<tr>
<td>7.9 Session upstream size parameters for study 2.</td>
<td>97</td>
</tr>
<tr>
<td>7.10 Request rate modelling results.</td>
<td>99</td>
</tr>
<tr>
<td>8.1 Simulation parameters for validation scenario.</td>
<td>103</td>
</tr>
<tr>
<td>8.2 Meta-data for simulator validation scenario.</td>
<td>105</td>
</tr>
<tr>
<td>8.3 Fitted parameters for session interarrival times.</td>
<td>105</td>
</tr>
<tr>
<td>8.4 Log-normal parameters for simulated session duration.</td>
<td>106</td>
</tr>
<tr>
<td>8.5 Error percentage for piece selection.</td>
<td>108</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>8.6</td>
<td>Simulation parameters for experiment scenarios, set 1</td>
</tr>
<tr>
<td>8.7</td>
<td>Server load results</td>
</tr>
<tr>
<td>8.8</td>
<td>Continuity index results</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

One of the applications expected to become the next “killer application” on the Internet is large-scale multimedia distribution. One indicator of this is the development of the Internet Multimedia Subsystem (IMS). The IMS is a result of work done by the 3rd Generation Partnership Project (3GPP) and was first published as part of release 5 of the Universal Mobile Telecommunications System (UMTS) in March 2003 [14]. Multimedia is thus considered as being an integral part of the next generation telecommunication networks, and the Internet as the primary distribution channel for this media.

The IMS is not the first proposed media-related killer application for the Internet. A multitude of media applications were suggested in connection with the appearance of Internet Protocol Multicast (IPMC) [39, 40, 107]. IPMC provided a method for sending IP datagrams to several recipients without increasing the amount of bandwidth needed to do this. In effect, IPMC provided a service similar to that of the television broadcasting service, where clients choose to subscribe to a specific TV or multicast channel. Though IPMC was a promising technical solution, it also posed new and difficult problems that did not need to be considered in traditional unicast IP. For instance, there is no notion of a receiver group in unicast communication, and new mechanisms and protocols were needed to address issues of group management, such as the latency of joining and leaving a group, how to construct multicast trees, etc. Additionally,
the acknowledge-based congestion control algorithm used in unicast Transport Control Protocol (TCP) could not be used for multicast without modifications, as it would result in an overload of incoming acknowledgements to the source, effectively performing a distributed denial-of-service attack.

As IPMC was not natively implemented in most IP routers at the time, the Multicast Backbone (MBone) [110] was put forth as an interim solution until router manufacturers got around to implementing IPMC in their hardware. The MBone provides an overlay network which connects IPMC capable parts of the Internet via unicast links. However, connecting to the MBone requires administrative support, and not all Internet Service Providers (ISPs) allow access through their firewalls to provide MBone tunnelling. Thus, IPMC is still not deployed to a significant extent in the Internet. Additionally, as there were no real killer applications making use of IPMC, ISPs have been reluctant to increase their administrative burden in order to provide a service not requested by their customers.

An additional issue with IPMC is that it lacks native buffering capabilities. This becomes a significant problem when providing streaming services, and many solutions have been proposed to solve this problem. Patching (Section 4.3) and Chaining (Section 4.4) are examples of solutions using both application layer caching for buffering and IPMC for transmission. Another way is to move the functionality of the network layer to the application layer, thus forming overlay networks that can take into account more diverse parameters and provide more complex services, while at the same time simplifying deployment and removing the dependence on the underlying infrastructure.

One specific type of overlay network that has been gaining attention during the last few years is the Peer-to-Peer (P2P) network. Systems such as Napster [90], Gnutella [77], eDonkey [45], and BitTorrent (BT) [30] have been used for searching for or distributing files by millions of users. Additionally, much research is being done on implementing multicast as an overlay service, i.e., Overlay Multicast (OLMC). Systems such as End-System Multicast (ESM) [122] and PeerCast [99] are being used to stream video and audio to large subscriber groups. Furthermore, approaches such as Distributed Prefetching Protocol
for Asynchronous Multicast (dPAM) [115] and oStream [35] provide intelligent application layer multicast routing and caching services. Overlay systems based on Distributed Hash Tables (DHTs) have also been used to provide multicast services, e.g., Bayeux [136], Scribe [26], and Application Level Multicast Infrastructure (ALMI) [100].

1.1 Motivation

While there are several surveys of broadcasting mechanisms and stream merging mechanisms, e.g., [11, 25, 63], and a large amount of publications on Application Layer Multicast (ALM) and P2P systems intended for Video-on-Demand (VoD), there is little information available on applying the ideas and mechanisms from the former to the latter. Therefore, in Part I of the thesis, “Media Distribution”, we provide an overview of four related topics: multicast systems, broadcasting strategies, stream merging strategies, and caching mechanisms.

BT is currently one of the most popular P2P applications [2], and proposals for adapting it to provide streaming services have been put forth. While the original BT distribution model was designed for distributing large files in an efficient way, researchers have designed adaptations to the BT protocols and mechanisms so as to be able to use them as foundations for streaming systems [38, 128]. This is the primary motivation for Part II of the thesis, “BitTorrent Streaming”.

1.2 Contributions

The main questions we explore in the thesis are: Is BitTorrent a suitable system to base a streaming service upon? If so, what extensions and modifications are needed? To this end, we have performed work resulting in the following contributions:

- We present a comprehensive overview of conventional media distribution mechanisms and protocols. We discuss multicast, as this is the
technology that best fits large-scale media distribution. Broadcasting strategies are considered because of the scheduling aspects of multimedia transmissions. Stream merging strategies are discussed because of their bandwidth-conserving capability and relation to both broadcasting and caching. We also consider caching strategies, as these are important for decreasing bandwidth consumption as well as for ALM to perform well in comparison with IPMC.

- We present a set of BT models for several important traffic characteristics, at both session and message levels. The models have been obtained and verified in two different measurement studies.

- We propose a number of modifications and extensions to the BT system and core algorithms and mechanisms to make it suitable for use in providing a streaming video delivery service. Further, we implement parts of these in a simulator, developed by the Routing in Overlay Networks (ROVER) team at Blekinge Institute of Technology (BTH).

- We report on a simulation study of the implemented extensions to the BT system as well as a detailed validation study of the BT simulator itself. The validation study is done using the previously obtained BT traffic models.

1.3 Related Work

1.3.1 BitTorrent Streaming

Though the popularity of BT has resulted in a fair amount of research, many of the publications regarding BT are measurement studies or analytical performance studies [46, 67, 84, 102, 104, 124]. However, recent developments indicate that BT is becoming increasingly interesting as an alternative method for delivering not only large files, but also as a viable option for streaming media.

In [38], the authors present BitTorrent-Assisted Streaming System (BASS), a hybrid client–server/P2P streaming system. In BASS, BT is used to leverage a media server by downloading non-time critical parts of a stream for future
1.3. RELATED WORK

consumption. A BASS client downloads pieces sequentially (except pieces that have already been downloaded via BT) from the media server. The standard BT mechanisms are used for downloading the suffix of the stream, i.e., the part of the stream after the current playback position. The authors show, by using trace-driven simulation results, that the average required bandwidth on the media server can be decreased up to 34% when using BASS compared to the pure media server case. Additionally, the authors show that the average client waiting time is decreased by 27% compared to the pure media server case and by 10% compared to the pure BT case.

In contrast with BASS, BitTorrent Streaming (BiToS) [128] does not employ a central media server at all, but rather completely relies on BT functionality to provide a near-zero delay VoD service. However, like BASS, BiToS modifies the original BT piece selection algorithm to facilitate on-time delivery of data. The system is based on partitioning the non-downloaded pieces into three sets: the received pieces set, the remaining pieces set, and the high priority set. The high priority set contains those pieces that are close to being consumed by the peer playback device, while the remaining pieces set contains pieces that are not under such strict timing demands. The term received pieces is used to denote all the pieces that have been downloaded. The normal BT piece selection algorithm is augmented with selecting the high priority set with probability $p$ and with probability $(1 - p)$ from the remaining pieces set. Once a set has been selected, a modified version of the rarest-first algorithm is used within the set to select which piece to download. The modification consists of selecting the piece with the shortest time to consumption in the case of two pieces being equally rare. When a piece from the high priority set has been downloaded, the piece with the shortest deadline from the remaining pieces set is moved to the high priority set.

The probability $p$ is the most important parameter in the BiToS system. If $p$ is chosen poorly, the system performance degrades substantially. For instance, if $p = 1$, the peer is going to prioritise the pieces that are close to being consumed. This may lead to a situation in which the peer might not have any rare pieces to exchange with the swarm, thus decreasing the download rates from reciprocating peers. An adaptive mechanism for dynamically changing the value of $p$ with respect to the conditions in the swarm would therefore be beneficial. Using a
simulator developed in-house, the authors report results indicating that BiToS, using \( p = 0.8 \), outperforms the normal rarest-first algorithm, but only for certain scenario-specific cases. Furthermore, the reported results clearly indicate that both BiToS and the normal rarest-first algorithm outperform the sequential piece selection scheme.

In [118], the authors provide analytical models for the efficiency (i.e., the fraction of the total upload capacity available to the P2P application) of a BT system with regard to the number of peers in the swarm and the number of pieces available for sharing. The efficiency is defined in [104] as

\[
\eta = 1 - \sum_{n_i=0}^{N-1} \frac{1}{N} \left( \frac{N - n_i}{N(n_i + 1)} \right)^k,
\]

where \( N \) denotes the number of pieces in the swarm, and \( k \) represents the number of peers participating in the swarm. Eq. (1.1) is an approximation and is only valid for large values of \( N \). This is typically the case for a normal, non-streaming BT swarm, but for the scenario discussed in [118], i.e., live video streaming, this is not the case, as future pieces have not yet been created. However, the authors show that the efficiency is over 97.5\% with only 40 pieces available for sharing. Additionally, using more than 7-8 peers does not yield higher efficiency.

### 1.3.2 P2P Simulators

A brief overview of P2P simulators is presented below. For a more complete review, the work in [89] is more comprehensive.

An ambitious general framework is OverSim [12], which extends the OM-Net++ [125] simulation framework. OverSim provides a modular system for implementing both structured and unstructured P2P protocols as well as various types of underlays, i.e., transport protocols such as TCP or UDP. Depending on the selected underlay, simulations can be made on realistic scenarios when needed, but be run without considering, e.g., link delay or transport protocol details, when simulation speed is more important. The OverSim framework also supports connecting real-world applications to the simulation engine. Several
overlay protocols such as Chord, Pastry, and GIA are provided with the OverSim distribution, and several more are planned for implementation.

PeerSim [68] is a Java P2P simulator comprising two distinct simulation engines. One is an efficient cycle-based engine, which does not take into account many details of the protocol stack. The other is a more accurate event-based engine, and thus significantly slower, but allowing for more realistic simulations.

Like PeerSim, GPS [135] is another Java simulator. Also, like OMNeT++, it is an event-driven, message-oriented simulator. GPS incorporates a Graphical User Interface (GUI) and logging features in addition to the core simulation components. The simulator has been used to model the BT protocols, primarily the peer protocol, also known as the peer wire protocol. The authors also report on running the reference client in a small-scale LAN topology and compare measurements from this scenario with an identical simulation scenario. The results indicate some similarities between the measurement and simulation, but nothing conclusive. Additionally, the authors present a scalability analysis of the simulator, which indicates that the wall-clock simulation time increases exponentially with the number of simulated nodes.

One of the earliest attempts to simulate BT-like scenarios was the swarming simulator described in [78]. The simulator does not implement the full BT protocol, and development seems to have stopped. Additionally, the simulator abstracts the BT network entities in a rather unorthodox way, making extending the simulator more complex and difficult.

Another BT-specific simulator is the one used for the work presented in [15] and [16]. It is written in the C# language and implements the majority of the BT mechanisms. Being implemented in the C# language and making use of the Microsoft .NET runtime makes platform independence an issue, and, as with [78], development of the simulator seems to have largely stopped.
1.4 Thesis Outline and Structure

This thesis consists of two parts and nine chapters. This first chapter has presented the motivation for and main contributions of the thesis, as well as a short discussion on the current state of the art in BT streaming and simulation. Chapters 2–5 make up the first part, Part I, “Media Distribution”, of the thesis, beginning with Chapter 2, “Multicast Systems”, where we discuss two ways of implementing multicast: IP multicast and application layer, a.k.a overlay, multicast. In Chapter 3, “Broadcasting Strategies”, several broadcasting schemes for streaming video are presented. This is followed by Chapter 4, “Stream Merging Strategies”, where we present methods and mechanisms for merging temporally disjoint media streams. In Chapter 5, “Caching Strategies”, we discuss caching mechanisms and how caching of streaming objects relates to the caching of Web objects.

The second part of the thesis, Part II, “BitTorrent Streaming”, consists of Chapters 6–9, starting with Chapter 6, “BitTorrent Streaming and Piece Selection”, in which we discuss the BT system, its core algorithms, and extensions and modifications to these to make them suitable for streaming scenarios. In the following chapter, Chapter 7, we present a comprehensive set of BT traffic models. Chapter 8, “Experiments”, reports on the BT simulator validation study, as well as the experimental setup and results of our suggested BT modifications. Chapter 9 concludes the thesis with a brief discussion of possible future work.

1.4.1 Publications

This thesis is partly comprised of previously published material. In this section, we briefly reference the most important of these publications.

Parts of this chapter, together with Part I, “Media Distribution”, and parts of Chapter 6, “BitTorrent Streaming and Piece Selection”, were previously published as a part of “Replication Strategies for Streaming Media” [47].

Section 7.2 is to a large extent taken from my Licentiate thesis, “BitTorrent Traffic Measurements and Models” [46].
The material making up the foundation for Chapter 7, “BitTorrent Traffic Models”, was previously published in “BitTorrent Traffic Measurements and Models” [46], “BitTorrent Traffic Characteristics” [49], “BitTorrent Session Characteristics and Models” [48], “BitTorrent Request Message Models” [51], and “Validating BitTorrent Models” (accepted for publication in a special issue of the journal “Telecommunication Systems”, SpringerLink\(^1\)).

The material in Chapter 8.1 is based on “Simulating BitTorrent” [129].

\(^1\)http://www.springerlink.com/content/101753/
Part I

Media Distribution
Chapter 2

Multicast Systems

2.1 IP Multicast

Parts of this section were previously published in [32, 101].

Group communication as used by Internet users today is taken more or less for granted. Forums and special interest groups abound, and the term “social networking” has become a popular buzzword. These forums are typically formed as virtual meeting points for people with similar interests, that is, they act as focal points for social groups. In this section, we discuss the technical aspects of group communication as implemented by IPMC.

2.1.1 Group Communication

A group is defined as a set of zero or more hosts identified by a single destination address [40]. We differentiate between four types of group communication, ranging from groups containing only two nodes (one sender and one receiver – unicast and anycast) to groups containing multiple senders and multiple receivers (multicast and broadcast).
CHAPTER 2. MULTICAST SYSTEMS

Figure 2.1: Group communication (Grey circles denote members of the same multicast group).

Unicast

Unicast is the original Internet communication type. The destination address in the IP header refers to a single host interface, and no group semantics are needed or used. Unicast is thus a 1-to-1 communication scheme (Figure 2.1(a)).

Anycast

In anycast, a destination address refers to a group of hosts, but only one of the hosts in the group receives the datagram, i.e., a 1-to-(1-of-m) communication scheme. That is, an anycast address refers to a set of host interfaces, and a datagram gets delivered to the nearest interface with respect to the distance metric of the routing protocol used. There is no guarantee that the same datagram is not delivered to more than one interface. Protocols for joining and leaving the group are needed. The primary uses of anycast are for load balancing and server selection.

Broadcast

A broadcast address refers to all hosts in a given network or subnetwork. No group join and leave functionality is needed, as all hosts receive all datagrams
2.1. **IP MULTICAST**

sent to the broadcast address. Broadcast is a 1-to-m communication scheme as shown in Figure 2.1(b). Broadcast communication is typically used for service discovery.

**Multicast**

When using multicast addressing, a single destination address refers to a set of host interfaces, typically on different hosts. Multicast group relationships can be categorised as follows [105]:

1-to-m: Also known as “One-to-Many”, or 1toM. One host acts as source, sending data to the \( m \) recipients making up the multicast group. The source may or may not be a member of the group (Figure 2.1(c)).

n-to-m: Also known as “Many-to-Many”, or MtoM. Several sources send to the multicast group. Sources need not be group members. If all group members are both sources and recipients, the relationship is known as symmetric multicast (Figure 2.1(d)).

m-to-1: Also known as “Many-to-One”, or Mto1. As opposed to the two previous relationships, m-to-1 is not an actual multicast relationship, but rather an artificial classification to differentiate between applications. One can view it as the response path of requests sent in a 1-to-m multicast environment. Wittman and Zitterbart refer to this multicast type as concat, or concentration casting [106].

Table 2.1 summarises the various group relationships discussed above.

2.1.2 **Multicast Source Types**

In the original multicast proposal by Deering [40], hosts wishing to receive data in a given multicast group, \( G \), need only to join the multicast group to start
Table 2.1: Group communication types.

<table>
<thead>
<tr>
<th>Senders</th>
<th>Receivers 1</th>
<th>Receivers m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unicast / Anycast</td>
<td>Multicast / Broadcast</td>
</tr>
<tr>
<td>n</td>
<td>Concat</td>
<td>Multicast</td>
</tr>
</tbody>
</table>

receiving datagrams addressed to the group. The group members need not know anything about the datagram or service sources, and any Internet host (group member or not) can send datagrams to the group address. This model is known as Any-Source Multicast (ASM). Two additional functions that a host wishing to take part in a multicast network needs to implement are:

\[
\text{Join } (G, I) \rightarrow \text{ join the multicast group } G \text{ on interface } I.
\]

\[
\text{Leave } (G, I) \rightarrow \text{ leave the multicast group } G \text{ on interface } I.
\]

Beyond this, the IP forwarding mechanisms work the same as in the unicast case. However, there are several issues associated with the ASM model, most notably addressing, access control, and source handling [17].

### Addressing

The ASM multicast architecture does not provide any mechanism for avoiding address collisions among different multicast applications. There is no guarantee that the multicasted datagram a host receives is actually the one that the host is interested in.

---

1 Additional to the unicast host requirements defined in [19].
2.1. IP MULTICAST

Access Control

In the ASM model, it is not possible for a receiver to specify which sources it wishes to receive datagrams from, as any source can transmit to the group address. This is valid even if sources are allocated a specific multicast address. There are no mechanisms for enforcing that no other sources will not send to the same group address. By using appropriate address scoping\(^2\) and allocation schemes, these problems may be made less severe, but this requires more administrative support.

Source Handling

As any host may be a sender (n-to-m relationship) in an ASM network, the route computation algorithm makes use of a shared tree mechanism to compute a minimum cost tree within a given domain. The shared tree does not necessarily yield optimal paths from all senders to all receivers, and it may incur additional delays as well.

Source Specific Multicast (SSM) addresses the issues mentioned above by removing the requirement that any host should be able to act as a source [62]. Instead of referring to a multicast group \(G\), SSM uses the abstraction of a channel. A channel is comprised of a source, \(S\), and a multicast group \(G\), so that the tuple \((S, G)\) defines a channel. In addition to this, the \texttt{Join}(G) and \texttt{Leave}(G) functions are extended to:

\begin{align*}
\text{Subscribe} & \ (s, S, G, I) - \text{request for datagrams sent on the channel } (S, G) \\
& \quad \text{to be sent to interface } I \text{ and socket } s \text{ on the requesting host.} \\
\text{Unsubscribe} & \ (s, S, G, I) - \text{request for datagrams to no longer be received} \\
& \quad \text{from the channel } (S, G) \text{ to interface } I.
\end{align*}

\(^2\)An address scope refers to the area of a network in which an address is valid.
2.1.3 Multicast Addressing

IPMC addresses are allocated from the pool of class D addresses, i.e., with the high-order nibble\(^3\) set to 1110. This means that the address range reserved for IPMC is 224/24, i.e., 224.0.0.0 – 239.255.255.255. The 224/8 addresses are reserved for routing and topology discovery protocols, and the 232/8 address block is reserved for SSM. Additionally, the 239/24 range is defined as the administratively scoped address space [87]. There are also several other allocated ranges [9].

Address allocation

Multicast address allocation is performed in one of three ways [121]:

**Statically:** Statically allocated addresses are protocol-specific and typically permanent, i.e., they do not expire. They are valid in all scopes and need no protocol support for discovering or allocating addresses. These addresses are used for protocols that need well-known addresses to work.

**Scope-relative:** For every administrative scope (as defined in [87]), a number of offsets have been defined. Each offset is relative to the current scope, and together with the scope range it defines a complete address. These addresses are used for infrastructure protocols.

**Dynamically:** Dynamically allocated addresses are allocated on-demand and are valid for a specific amount of time. It is the recommended way to allocate addresses. To manage the allocation, the Internet Multicast Address Allocation Architecture (MALLOC) has been proposed [121]. MALLOC provides three layers of protocols:

\(^3\)A nibble is a bit sequence of four bits, or a half-byte.
2.1. IP MULTICAST

Layer 1 – Client–server: Protocols and mechanisms, such as Multicast Address Dynamic Client Allocation Protocol (MADCAP) [60], for multicast clients to request multicast addresses from a Multicast Address Allocation Server (MAAS).

Layer 2 – Intra-domain: Protocols and mechanisms to coordinate address allocations to avoid addressing clashes within a single administrative domain.

Layer 3 – Inter-domain: Protocols and mechanisms to allocate multicast address ranges to a Prefix Coordinator in each domain. A Prefix Coordinator is a central entity (either a router or a human administrator) responsible for an entire prefix of addresses. Individual addresses are then assigned within the domain by MAASs.

2.1.4 Multicast Routing

The major difference between traditional IP routing and IP multicast routing is that datagrams are routed to a group of receivers rather than a single receiver. Depending on the application, these groups have dynamic memberships, and this is important to consider when designing routing protocols for multicast environments.

Multicast Topologies

While IP unicast datagrams are routed along a single path, multicast datagrams are routed in a distribution tree or multicast tree. A unicast path selected for a datagram is the shortest path between sender and receiver. In the multicast case, the graph-theoretic problem of finding the shortest path between two vertices becomes the problem of finding a Shortest-path Tree (SPT), Minimum Spanning Tree (MST), or Steiner tree. An SPT minimises the sum of each source-destination path, while the MST and Steiner trees minimise the total tree cost. The MST
CHAPTER 2. MULTICAST SYSTEMS

and Steiner tree algorithms differ in that Steiner trees are allowed to add more vertices than are available in the original graph.

Typically, there are two categories of multicast trees: source-specific and group shared trees. A source-specific multicast tree contains only one sending node, while a group-shared tree allows every participating node to send data. These two tree types correspond to the 1-to-m and n-to-m models presented in Section 2.1.1, respectively. Regardless of which tree type a multicast environment makes use of, a good, i.e., well-performing, multicast tree should exhibit the following characteristics [58]:

**Low Cost:** A good multicast tree keeps the total link cost low.

**Low Delay:** A good multicast tree minimises the end-to-end (e2e) delay for every source–destination pair in the multicast group.

**Scalability:** A good tree should be able to handle large multicast groups, and the participating routers should be able to handle a large number of trees.

**Dynamic Group Support:** Nodes should be able to join and leave the tree seamlessly, and this should not adversely affect the rest of the tree.

**Survivability:** A good tree should survive multiple node and link failures.

**Fairness:** This requirement refers to the ability of a good tree to evenly distribute the datagram duplication effort among participating nodes.

**Routing Algorithms**

There are several types of routing algorithms for multicast environments. Some of the non-multicast-specific algorithms include flooding, improved flooding, and spanning trees. The flooding algorithms are more akin to pure broadcasting and tend to generate large amounts of network traffic. The spanning tree protocols are typically used in bridged networks and create distribution trees which ensure that all connected networks are reachable. Datagrams are then broadcasted on
2.1. **IP MULTICAST**

this distribution tree. Due to their group-agnostic nature, these algorithms are rarely used in multicast scenarios. However, there are exceptions, such as the Distance Vector Multicast Routing Protocol (DVMRP).

Multicast-specific algorithms include *source-based routing*, *Steiner trees*, and *rendezvous point trees* (also called *core-based trees*).

**Source-based Routing:** Source-based routing includes algorithms such as Reverse Path Forwarding (RPF), Reverse Path Broadcasting (RPB), Truncated Reverse Path Broadcasting (TRPB), and Reverse Path Multicasting (RPM) [70, 91]. Of these algorithms, only RPM specifically considers group membership in routing. The other algorithms represent slight incremental improvements of the RPF scheme in that they decrease the amount of datagram duplication in the distribution tree and avoid sending datagrams to subnetworks where no group members are registered. Examples of source-based protocols are the DVMRP, Multicast Extensions to Open Shortest Path First (MOSPF), Explicitly Requested Single-Source Multicast (EXPRESS), and Protocol Independent Multicast – Dense Mode (PIM-DM) protocols.

**Steiner trees:** As mentioned previously, the Steiner tree algorithms optimise the *total* tree cost. This is an NP-hard problem, making it computationally expensive and not very useful for topologies that change frequently. While Steiner trees provide the minimal *global* cost, specific paths may have a higher cost than those provided by non-global algorithms. The Steiner tree algorithms are sensitive to changes in the network, as the routing tables need to be recalculated for every change in the group membership or topology. In practice, some form of heuristic, such as the Kou, Markowski, and Berman (KMB) heuristic [81], is used to estimate the Steiner tree for a given multicast scenario.

**Rendezvous Point trees:** Unlike the two previous algorithms, these algorithms can handle multiple senders and receivers. This is done by appointing one node as a *Rendezvous Point (RP)* through which all datagrams are routed. A substantial drawback with this approach is that the RP becomes a single point...
of failure, and it may be overloaded with traffic if the number of senders is large. Examples of this type of protocol are the Core Based Tree (CBT), Protocol Independent Multicast – Sparse Mode (PIM-SM), and Simple Multicast (SM) protocols.

**IP Multicast Routing Protocols**

**DVMRP:** DVMRP [131] was created with the Routing Information Protocol (RIP) as a starting point and uses ideas from both the RIP and the TRPB [107] protocols. As opposed to RIP, however, DVMRP maintains the notion of a receiver–sender path (due to the RPF legacy of TRPB) rather than the sender–receiver path in RIP. DVMRP uses *poison reverse* and *graft/prune* mechanisms to maintain the multicast tree. Being a Distance Vector (DV) protocol, DVMRP suffers from similar problems as other DV protocols, e.g., slow convergence and flat network structure. The Hierarchical Distance Vector Multicast Routing Protocol (HDVMRP) [1] and Host Identity Protocol (HIP) [21] protocols address this issue by introducing hierarchical multicast routing.

**MOSPF:** MOSPF [88] is based on the Open Shortest Path First (OSPF) link state protocol. It uses Internet Group Management Protocol (IGMP) to monitor and maintain group memberships within the domain and OSPF link state advertisements to maintain a view on the topology within the domain. MOSPF builds a shortest-path tree rooted at the source and prunes those parts of the tree with no group members.

**PIM:** Protocol Independent Multicast (PIM) is actually a family of two protocols, or operation modes: PIM-SM [52] and PIM-DM [3]. The term *protocol independent* stems from the fact that the PIM protocols are not tied to any specific unicast routing protocol as DVMRP and MOSPF are tied to RIP and OSPF, respectively.

PIM-DM refers to a multicast environment in which many nodes are participating in a “dense” manner, i.e., a large part of the available nodes are participating and there is much bandwidth available. Typically, this implies
2.1. IP MULTICAST

that the nodes are not geographically spread out. Like DVMRP, PIM-DM uses RPF and grafting/pruning, but differs in that it needs a unicast routing protocol for unicast routing information and topology changes. PIM-DM assumes that all nodes in all subnetworks want to receive datagrams and uses explicit pruning for removing uninterested nodes.

In contrast to PIM-DM, PIM-SM initially assumes that no nodes are interested in receiving data. Group membership thus requires explicit joins. Each multicast group contains one active RP.

**CBT:** The CBT [10] protocol is conceptually similar to PIM-SM in that it uses RPs and has a single RP per tree. However, it differs in a few important aspects:

- CBT uses bidirectional links, while PIM-SM uses unidirectional links.
- CBT uses a lower amount of control traffic compared to PIM-SM. However, this comes at the cost of a more complex protocol.

**BGMP:** The protocols discussed so far are all Interior Gateway Protocols (IGPs). The Border Gateway Multicast Protocol (BGMP) [120] protocol is a proposal to provide inter-domain multicast routing. Like the Border Gateway Protocol (BGP), BGMP uses TCP as a transport protocol for communicating routing information, and it supports both the SSM and ASM multicast models. BGMP is built upon the same concepts as PIM-SM and CBT, but differing in that participating nodes are entire domains instead of individual routers. BGMP builds and maintains group shared trees with a single root domain and can optionally allow domains to create single-source branches if needed.

### 2.1.5 Challenges for Multicast Communication

An important aspect to consider when designing any communication network, multicast included, is the issue of scalability. It is imperative that the system does not “collapse under its own weight” as more nodes join the network. The
exact way of handling scalability issues is application and topology-dependent, such as can be seen in the dichotomy of PIM: PIM-DM uses one set of mechanisms for routing and maintaining the topology, while PIM-SM uses a different set. Additionally, if networks are allowed to grow to very large numbers of nodes (on the order of millions of nodes, as with the current Internet), routing tables may grow very large. Typically, scalability issues are addressed by introducing hierarchical constructs to the network.

Related to the scalability issue, there is the issue of being conservative in the control overhead that the protocol incurs. Regarding topology messages, this is more a problem for proactive or table-driven protocols that continuously transmit and receive routing update messages. On the other hand, reactive protocols pay the penalty in computational overhead, which may be prohibitively large if the rate at which nodes join and leave the multicast group (a.k.a. \textit{churn}) is high.

In addition to keeping topology control overhead low, multicast solutions should also consider the group management overhead. Every joining and leaving node will place load on the network, and it is important that rapid joins and leaves do not unnecessarily strain the system. At the same time, both joins and leaves should be performed expediently, \textit{i.e.}, nodes should not have to wait too long before joining or leaving a group.

Another important issue for RP-based protocols is the selection of an appropriate rendezvous point. As the RP becomes a traffic aggregation point and a single point of failure, it is also important to have a mechanism for quickly selecting a replacement RP in the case of failure. This is especially important for systems in which a physical router may act as RP for several groups simultaneously.

While there are many proposals for solutions to the problems and challenges mentioned above, none of them have been able to address what is perhaps the most important issue: wide scale deployment and use – IPMC just hasn’t taken off the way it was expected to. Whether this is due to a lack of applications that need a working infrastructure or the lack of a working infrastructure for application writers to use is still unclear.

Additional application-specific issues also appear, \textit{e.g.}, when deploying
services considered “typical” multicast services, such as media broadcasting and VoD. Since IPMC operates at the network layer, it is not possible for transit routers to cache a video stream that is transmitted through them. This caching would have to take place at the application layer instead. If two clients, A and B, try to access the same stream at different times, client A can not utilise the datagrams already received by B, but will have to wait until retransmission. This waiting time may be on the order of minutes or tens of minutes, depending on the broadcasting scheme used. Additionally, VCR-like functionality (fast forward, rewind, and pause) and other interactive features are difficult to provide.

2.2 Application Layer Multicast

As the lack of deployment of IPMC on a large scale makes the development of new algorithms and distribution mechanisms difficult, much research has been performed on Application Layer Multicast (ALM). In ALM systems, the typical network functions of routing, group membership, and addressing are performed by hosts on the edges of the network. This allows for more complex and intelligent mechanisms to be employed than is possible in the stateless, best-effort Internet. Additionally, since applications have the possibility of using information on link quality, they can also consider soft Quality of Service (QoS) guarantees and provide topology-aware routing without costly infrastructure support.

2.2.1 Issues with ALM

Though ALM is a promising alternative to network layer multicast, there are significant drawbacks associated with it. One issue is that using topology-awareness breaks the layering principle since network layer functionality is duplicated in the application layer. In addition, transport layer functionalities

---

4An alternative term is Overlay Multicast (OLMC), but, as this term is also used to denote a specific type of ALM, we will avoid using it here. However, the term overlay will still be used interchangeably when referring to application layer networks in general.
CHAPTER 2. MULTICAST SYSTEMS

such as congestion and error control are also duplicated in several ALM systems. Other serious problems are related to complexity and network resource usage. As the ALM system can take more parameters into account and contain larger and more complex policies, the systems themselves may become more complex. This places higher demands on both the programming skills of the system implementors as well as the routers (in this case edge hosts acting as ALM routers). Fortunately, modern edge hosts are quite capable of handling a fair amount of processing and furthermore do not have to handle as much traffic as, e.g., a core router. The resource usage problem is particularly notable in ALM systems when compared to unicast application layer systems. This is because ALM typically operates on top of unicast IP links, which makes it impossible to completely avoid packet duplication on these links. By using intelligent caching algorithms and other methods, it is possible to decrease the duplication, and achieve better resource usage than IPMC [35, 115]. However, these solutions are application specific, as opposed to the application-agnostic IPMC.

2.2.2 Performance Metrics

Several performance metrics have been defined to characterise the multicast communication service and its impacts on the network [28, 117]. The most important metrics are:

**Link stress, σ:** The link stress is a measure of how many times a given packet is duplicated across a link due to the overlay. It is defined as the number of identical copies of a packet transmitted across a specific physical link.

**Relative delay penalty, ρ:** The Relative Delay Penalty (RDP) is defined as the ratio of the delay between two hosts in the overlay to the delay of the shortest path unicast delay between the same two hosts.

**Link stretch, λ:** The link stretch is similar to the RDP, although it compares the distance between the hosts instead of the delay. It is defined as the ratio of the length of the overlay path to the length of the unicast shortest path between the two hosts.
2.2. Application Layer Multicast

Resource usage, ∇: This metric describes the system-wide resource usage of the overlay system. It is defined as the sum of the stress-RDP product of all hosts participating in the ALM system, i.e.,

\[ \nabla = \sum_{i=0}^{N} \sigma_i \rho_i, \]  

where \( N \) is the number of ALM links.

2.2.3 ALM Classification

There are several ways of designing and implementing ALM systems. One classification is provided in [101], which identifies three major categories of ALM systems: P2P ALM systems, OLMC systems and, Waypoint Multicast (WPMC) systems.

Peer-to-Peer ALM

In P2P ALM (Figure 2.2 (b)), participating end hosts are responsible for all forwarding, group management, and addressing. All hosts are equally responsible for these tasks, and no host is defined as providing a specific service or functionality by the system. Certain hosts may be more popular than others, and more load may be placed upon them, but this can be viewed as an emergent property rather than an intrinsic property of the system itself. This equality in function may also lead to very dynamic topologies as hosts tend to join and leave the system frequently (a phenomenon known as churn). Churn affects the network in a substantial way, and a good ALM solution must take this into consideration.

Overlay ALM

As opposed to the flat network of P2P ALM systems, OLMC systems (Figure 2.2 (c)) provide a service much akin to an overlay proxy system, where the proxies are
placed at strategic end hosts in the Internet. The overlay proxies can be organised to provide higher QoS with regard to bandwidth, delay, jitter, and improved accessibility. One example of this type of system are Content Distribution Networks (CDNs), such as Akamai [6], where several Points of Presence (PoPs) are responsible for clusters of surrogate servers. Each surrogate server maintains copies of the content and is responsible for delivering the content to requesting clients.
Waypoint ALM

A third ALM alternative is WPMC [53, 66] (Figure 2.2 (d)). A waypoint is an edge node that provides the same functionality as any other node in the ALM system, but does not consume the resource. For instance, in a streaming scenario, the waypoint host acts only as a router and not as a video viewer. Waypoints may be dynamically or statically provisioned and are used to provide more resources to the ALM system.

2.2.4 ALM Topologies

Network topologies in IPMC are by their nature tree topologies, rooted at either the source or an Rendezvous Point (RP). Topologies for ALM have no such constraints and may be implemented in several ways [59, 101].

Mesh-based overlays: In the mesh-based approach, hosts are organised in a flat mesh-based topology in which each host maintains a list of other hosts termed neighbours. One salient advantage with mesh topologies is that there are typically alternate paths between any given host pair, which means that this type of topology is less sensitive to node failure. Since alternate paths already exist, path reconstruction need not be performed when a path crashes due to, e.g., node failures. As opposed to the tree-based approaches, nodes in a mesh topology self-organise to form the network.

Tree-based overlays: Tree-based overlays apply an overlay-specific mechanism to construct a distribution tree. In [18], the authors present three tree topology types: linear, trees with outdegree $k$ ($Tree^k$), and a forest of parallel trees ($PTree^k$). In a linear tree, nodes are organised in a chain (see also Section 4.4, “Chaining”) where the first node is the only node connected to the content server. A $Tree^k$ topology has nodes organised so that each node serves $k$ other nodes. In a $PTree^k$ topology, the content is partitioned into $k$ parts, each of which is then distributed using a separate tree. The $PTree^k$ approach is the best
CHAPTER 2. MULTICAST SYSTEMS

performing of the three.

**Multiple tree/mesh overlays:** To address sender and receiver heterogeneity in single trees or meshes, an approach in which multiple trees or meshes are used has been suggested [53]. By employing multiple trees or meshes, several desirable properties are achieved, e.g., resiliency to network and group dynamics, increase in the available bandwidth, and address sender and receiver heterogeneity. Additionally, in [92], the authors use a layered codec (named Multiple Description Codec (MDC)), and transmit each layer of the encoded stream on a different distribution tree. The receiver then reconstructs the stream to a QoS-level corresponding to the number of stream layers it has received.

**Ring and multi-ring overlays:** One inherent problem with tree and mesh architectures is that of congestion and flow control. This is particularly notable when using traditional ACK-based congestion control. Also, for trees in particular, dynamic key management is very complex, in addition to the complexity of constructing completely disjoint backup trees for survivability. Ring and multi-ring topologies are less complex, making these problems less pronounced [133]. However, this simplicity comes at the cost of longer communication paths.

### 2.3 Summary

Multicast communication is a central component for efficient media distribution. In this chapter, we presented the two most common ways of implementing multicast: Internet Protocol (IP) Multicast and Application Layer Multicast (ALM). Both methods have inherent advantages and drawbacks, which were also discussed. IPMC can offer more efficient forwarding, but suffers from several problems. One problem is the lack of buffering, which becomes an issue for streaming transmissions started at different times. Another issue with IPMC is that it requires substantial infrastructural and administrative support. Furthermore, IPMC has not been deployed as widely as was originally hoped, and few applications thus take advantage of IPMC.
2.3. SUMMARY

ALM, on the other hand, is easier to deploy than IPMC and typically requires less infrastructure support. It is inherently less efficient when forwarding, since forwarding end hosts use unicast links for this. However, using intelligent caching algorithms, ALM systems can significantly decrease media server bandwidth requirements [35]. An additional issue for ALM is churn, where nodes frequently join or leave the system.

In addition to the specific issues of IPMC and ALM, there are more general issues with all media multicast systems. For instance, the issues of whether to allow ASM or only SSM and selection of RPs (in IPMC) and waypoints (in ALM) have significant impacts on scalability and performance. Furthermore, congestion and error control must be considered as well. In IPMC, an important issue is that an ACK-based mechanism may overload the system, while in an ALM system it is important to avoid duplicating congestion control functionality in the transport layer.

Regardless of at which layer multicast is implemented, scheduling and segmentation of the media objects to be transmitted must be considered as well. This is the topic of the following chapter, Chapter 3, “Broadcasting Strategies”.
When we use the terms “broadcast” and “broadcasting” in this chapter, we are referring to the colloquial usage of the terms, *i.e.*, “the transmission of audio and/or video to a set of subscribers”, as opposed to the networking term, *i.e.*, “to transmit a packet to all nodes in a network.”

Media broadcasting schemes can roughly be divided into two categories: *periodic broadcast* and *scheduled multicast*. In the periodic broadcasting case, object transmissions are initiated at fixed intervals. The time intervals at which the transmissions are started is the main parameter distinguishing the various periodic broadcasting schemes. In scheduled multicast, transmission start times are decided according to a specific server scheduling policy, *e.g.*, “start transmission when transmission queue contains three client requests” (see also Section 4.1, “Batching”).

In this chapter, we discuss various periodic broadcasting schemes. These schemes typically view a stream to be broadcast as a sequence of segments of varying size transmitted at different intervals. The segment sizes as well as the transmission rates and sizes, are the primary differentiating factors of the various schemes.
3.1 Terminology

We use the following terms in this chapter. An object, or video object, refers to an entire video clip, such as a movie or TV episode, stored in digital form. When an object is transmitted to a client that consumes it while receiving the object, we refer to this as streaming the object. A channel refers to a multicast group with an associated available bandwidth. Also, when referring to popular and unpopular objects, we use the terms “hot” and “cold”, respectively.

We denote the playout-rate (in units\(^1\) per second) of a given video stream by \(b\), the number of segments in a stream by \(S\), and the segment size by \(s\) (Figure 3.1). \(s_i\) refers to the size of segment \(i\). We denote the size of the entire video object by \(V\) and the duration (in time) of the object by \(L\), while \(l_i\) indicates the duration of segment \(i\). The number of allocated channels for each video object is denoted by \(C\); \(c_i\) refers to channel number \(i\), and \(B\) refers to the physical link bandwidth.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{stream_parameters.png}
\caption{Stream parameters.}
\end{figure}

\(^1\)For instance, frames, bits, or bytes.
3.2 Conventional Broadcasting

By *conventional* broadcasting, we refer to broadcasting strategies where video objects are transmitted in their entirety before allowing new clients to partake in the broadcast. These schemes are analogous to normal television broadcasting. The segment size in conventional broadcasting is $V$, and the maximum waiting time is $L$. Several channels can be used, but only one object is used per channel at any given time.

3.3 Staggered Broadcasting

In *staggered broadcasting* [8], the simplest non-naive broadcasting strategy, each video stream is allocated a single channel of bandwidth $b$. Clients can only access a stream at pre-defined timeslots, making client access latency on the order of minutes, depending on the sizes of the timeslots. A new channel is allocated only if there was a client request in the previous slot, and the entire video stream is re-transmitted on this channel. The segment size is thus $V$, and the maximum waiting time is $L/C$.

3.4 Pyramid Schemes

In *Pyramid Broadcasting (PB)* [127], the video object is divided into segments of increasing size, i.e., $s_1 < s_2 < \cdots < s_S$. The available server bandwidth is divided into $C$ channels, each with bandwidth $B/C$. One channel is allocated to each segment size, and the associated segments are continuously transmitted on its associated channel, i.e., segment $s_i$ is repeatedly transmitted on channel $c_i$.

The segment sizes are selected according to a geometric series with parameter $\alpha$, i.e., $s_{i+1} = \alpha s_i$ for $i = 1 \ldots S - 1$. The highest bandwidth efficiency is obtained for $\alpha = B/K$, where $K$ is the number of allocated channels and $B$ is the physical link bandwidth (given in multiples of $b$). At any given time, there are at most two consecutive segments being received by a client. The
first segment, $s_i$, is also being played back simultaneously, while the second segment is only being downloaded. This means that the download rate of the second segment must be at least $\alpha$ and that the entirety of segment $s_{i+1}$ must be buffered at the client. The authors conjecture that optimal client access times (i.e., the time when the client makes a request to the server) are achieved for $\alpha = e$, where $e$ is Euler’s constant.

If there is more than one video object being broadcasted, then these are multiplexed across each channel.

### 3.4.1 Permutation-based Schemes

In [5], the authors present an improvement to the pyramid scheme, called Permutation-based Pyramid Broadcasting (PPB). As with pyramid broadcasting, each channel carries only one segment, and each segment size is given by the same geometric series as the original pyramid scheme. In the permutated scheme, however, each transmission channel is divided into $p$ logical subchannels, each with bandwidth $B/pS$ (for the single video object case). Each subchannel carries the same bitstream, but shifted by $s_i/p$ bits. A client does not start the download of segment $s_{i+1}$ until segment $s_i$ has finished playing. This removes the need for large buffer space at the client, but introduces the risk for playback gaps between segments. This situation is handled by downloading a short part of segment $s_{i+1}$ while $s_i$ is playing to bridge the gap between the segments.

This scheme decreases the overall bandwidth required as well as the client disk storage and access requirements.

### 3.4.2 Skyscraper Schemes

The skyscraper scheme presented in [65] differs from pyramid broadcasting in the segment sizing algorithm. In skyscraper broadcasting, segment sizes are given by a recursive function. The resulting series of this function is

\[ [1, 2, 2, 5, 5, 12, 12, 25, 25, \ldots] \]
where each integer corresponds to a multiple of $s_1$. To avoid segments getting too large, a maximum segment size is used to limit the segment size.

A client receives the stream in odd or even transmission groups, i.e., consecutive segment sizes $(A, A, \ldots)$, e.g., $(2, 2)$ or $(25, 25)$. The algorithm requires disk storage, but less than the original pyramid scheme.

## 3.5 Staircase Schemes

In staircase data broadcasting [74], the entire video object is allocated $B$ channels, each of bandwidth $b$. The object is divided into $S = 2^B - 1$ equally sized segments.

For transmission, each channel $c_i$ is further divided into $2^i$ subchannels of bandwidth $b/2^i$, and $2^i$ contiguous segments $s_{2^i}, \ldots s_{2^i+1-1}$ are associated with $c_i$. Each segment in $c_i$ is also divided into $2^i$ subsegments. The subsegments are transmitted continuously on each associated subchannel.

The client begins downloading each segment $s_v$ from channel $c_i$ at time $t_0 + (v - 2^i) \delta$, where $i = \lceil \log_2 v \rceil$, $t_0$ is the download start time for the initial segment, and $\delta = S/(2^i - 1)$ is the period of $s_1$. Downloading of segment $s_v$ is stopped at $t_0 + v \delta$. The maximum initial waiting time is $\delta$, and the client buffer space is upper bounded by $V/4$. The client buffer requirements of the staircase scheme are lower than the original pyramid scheme, and, if $C < 10$, also lower than that of the permuted pyramid scheme.

A scheme similar to the staircase scheme, known as fast broadcasting, is presented in [76]. The fast scheme is designed for “hot”, i.e., popular, videos. It uses the same channel allocations and segment sizes as the staircase scheme, but does not use subsegments or subchannels. Instead, each set of segments $s_{2^i}, \ldots, s_{2^i+1-1}$ is periodically broadcasted on channel $c_i$. Clients download from all channels simultaneously. The main advantages of the fast broadcasting scheme are that it only needs to allocate four channels and that it can work without client buffering. However, in this case, the scheme suffers from the same problems as the PPB, i.e., subsequent segments on different channels need to be buffered to avoid playback gaps.
A problem with the fast scheme is that it always assumes that videos are “hot” and thus wastes bandwidth when there are few requests for a given video object. The adaptive fast broadcasting scheme [75] remedies this by allocating and releasing channels according to client requests.

### 3.6 Harmonic Schemes

In contrast to the varying sized segments in the pyramid and staircase schemes, Harmonic Broadcasting (HB) [73] divides the video object into equally sized segments, making each segment size $s_i = s/S$. Additionally, each $i$th segment is further divided into $i$ sub-segments, i.e., $s_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,i}\}$. Every segment $s_i$ is allocated a separate channel with bandwidth $b/i$, and sub-segments of $s_i$ are transmitted sequentially on the associated channel. A client subscribes to all $S$ channels simultaneously. The term harmonic broadcasting stems from the fact that the total bandwidth for the video object is given by $B = \sum_{i=1}^{S} \frac{b}{i}$, which can be written as $B = bH_S$, where $H_S$ is the $S$th harmonic number\(^2\). Suppose that a maximum delay before a client can begin consuming the video object is $L/30$; the required bandwidth is then $\approx 4b$, since $H_{30} \approx 4$.

In [96], the authors show that the harmonic scheme does not guarantee timely delivery of every segment. The authors suggest two solutions to this problem: Cautious Harmonic Broadcasting (CHB) and Quasi-Harmonic Broadcasting (QHB). CHB solves the timing problem of HB by using bandwidth $b$ for both the first and second transmission channels. Additionally, segments $s_2$ and $s_3$ are transmitted alternatingly on the second channel. The following $3 \ldots S-1$ channels transmit the remaining segments in the same manner as in HB. The CHB scheme requires $b/2$ higher available bandwidth compared to HB. QHB works by dividing each segment into $im - 1$ fragments and then dividing each timeslot (the time needed to transmit an entire segment) into $m$ subslots. The timing problem is solved by a clever scheduling of fragments. The additional bandwidth needed compared to HB is $\sum_{i=2}^{n} \frac{b}{i(m-1)}$. Another harmonic scheme proposed by the authors behind CHB and QHB, Polyharmonic

\(^2\)The $S$th harmonic number is given by $H_s = \sum_{i=1}^{s} \frac{1}{i}$. 

38
3.7. HYBRID SCHEMES

Broadcasting (PHB) is presented in [94]. The PHB scheme provides a maximum client waiting time while requiring less server bandwidth than the HB scheme.

3.7 Hybrid Schemes

As with the harmonic schemes, the Pagoda scheme [95] also divides the video object into equally sized segments, each of duration $d$. The segment duration is denoted a time slot. However, unlike the harmonic schemes, Pagoda does not divide the channel bandwidth, but allocates the bandwidth $b$ for each channel. The Pagoda scheme can thus be viewed as a hybrid between the Pyramid and Harmonic schemes.

The first channel periodically transmits segment $s_1$ with the period $d^{-1}$. The following channels transmit segments according to the segment-to-channel mappings and periodicities presented in Table 3.1.

<table>
<thead>
<tr>
<th>Segments</th>
<th>Channel</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_z$ to $s_{3z/2−1}$</td>
<td>$2k$</td>
<td>$(zd)^{-1}$</td>
</tr>
<tr>
<td>$s_{3z/2}$ to $s_{2z−1}$</td>
<td>$2k + 1$</td>
<td>$2(3zd)^{-1}$</td>
</tr>
<tr>
<td>$s_{2z}$−$s_{3z−1}$</td>
<td>$2k$</td>
<td>$(2zd)^{-1}$</td>
</tr>
<tr>
<td>$s_{3z}$−$s_{5z−1}$</td>
<td>$2k + 1$</td>
<td>$(3zd)^{-1}$</td>
</tr>
</tbody>
</table>

The authors show that Pagoda broadcasting is almost as efficient as the harmonic broadcasting schemes with respect to the maximum client waiting time. In [93], the authors present New Pagoda, which further improves the Pagoda scheme with 25% shorter initial delay than the original protocol. The New Pagoda scheme employs a more efficient segment-to-channel mapping scheme in order to achieve the improvements over the original protocol.
CHAPTER 3. BROADCASTING STRATEGIES

3.8 Summary

In this chapter, we described several important broadcasting strategies. A broadcasting strategy typically encompasses a scheduling algorithm, which states when and on what channel objects are transmitted, and a segmentation algorithm, which describes how a video object is partitioned into smaller pieces.

The main parameters in a broadcasting strategy are the maximum initial delay, i.e., how long a client must wait before receiving a stream after having made a request, and the number of channels used by the server for transmitting segments. Another important factor for the performance of a broadcasting strategy is the popularity of the video object. A popular, or “hot”, video requires a different strategy than an unpopular, or “cold”, video.

Two additional problems are that it is not possible to have zero-delay VoD using a broadcasting strategy alone, and clients that receive the same stream, but which started at different times, do not share the same multicast. These issues are addressed by various stream merging techniques, which are the topic of the following chapter.
Stream merging is used for mitigating the cost of temporally separated streams, i.e., streams that are started at different times. This implies both decreasing the initial delay for the client, as well as having different clients subscribing to the same server channels by using various mechanisms to “catch up” with an ongoing transmission. One of the first references to the term stream merging was by Eager, Vernon, and Zahorjan [41, 44].

The broadcasting mechanisms described in the previous chapter suffer from a common problem: they exhibit a non-negligible initial delay before playback can start. That is, clients must wait until a pre-defined time before the transmission of the video object starts. This means that true VoD, i.e., zero-delay streaming is not possible. Many of the stream merging schemes address this problem and provide near-instant playback\(^1\).

Also, while the broadcasting strategies in the previous chapter primarily are of the server–push type, i.e., transmissions are scheduled by the server, most stream merging schemes are of the client–pull type. Client–pull means that transmissions are initiated by client requests and channel allocations are done accordingly. In merging schemes, a stream is often viewed as being made up of two parts: the prefix and the suffix. The suffix is typically transmitted using one

\(^1\)Of course subject to bandwidth limitations and transmission times.
of the periodic broadcasting schemes from the previous chapter, and the prefix is the initial part of the entire stream that clients arriving late for a scheduled broadcast wish to catch up with or merge into.

This chapter presents an overview of the most common stream merging techniques. Not all of these techniques are unambiguous stream merging techniques as such, e.g., batching. However, we include them here as they fit well in this chapter.

### 4.1 Batching

In **batching** [113], clients are placed in queues, and when a server channel is available, a batch of pending client requests are served according to a queue selection policy. Examples of such selection policies are First Come First Served (FCFS), Maximum Queue Length (MQL), and Round Robin (RR) [113]. This means that clients must wait until a server channel is available, causing an initial delay before it is possible to view the video object.

Bandwidth utilisation and storage Input/Output (I/O) may be substantially decreased when using batching, in particular if the objects are “hot”, since several requests are likely to arrive within a short space of time. For “cold” objects, there is little to no bandwidth saving to be made, as requests are likely to be few and far between. This means that there is no one single optimal policy for batching, but rather that the algorithms must take object popularity into account [113]. Batching can be used in both periodic broadcast and scheduled multicast scenarios.

Figure 4.1 illustrates the two batching classes. The figure assumes that there are available channels. The periodic broadcasts start at times $t_0, \ldots, t_5$, regardless of how many clients are waiting to be served. The scheduled multicasts are started at times $s_0, \ldots, s_3$, according to the policy “start transmission when queue contains three outstanding client requests”.

42
4.2 Piggybacking

Adaptive piggybacking [4, 57] is a merging technique in which the actual playout rate of a video object is modified. If the playout rate is modified only slightly, the change is not detectable by the human eye. The authors of [57] state that as long as the playout rates are within $\pm 5\%$ of the nominal playout rate, the change is not perceivable by the viewer. The playout rate may be modified in two ways: online or offline. In the online case, the stream is time compressed on-the-fly, typically requiring specialised hardware, while in the offline case, the time compressed object is pre-compressed and stored on secondary storage. By changing the playout rate, the disk I/O streams for the object on the server are also changed. Each arriving client request for a specific video object causes a new I/O stream to be allocated. In essence, if a request for a video object arrives from client $A$ at time $t_0$, and another request from client $B$ arrives at time $t_1$, the display rate is increased for client $B$ and decreased for client $A$ until the I/O streams can be merged into a single stream.

Figure 4.2 illustrates several key parameters for the piggybacking algorithm. The display streams for clients $A$ and $B$ are denoted by $i$ and $j$, while $S_i$ and $S_j$ represent the display speeds for clients $A$ and $B$, respectively. The current playback positions in the streams are denoted by $p_i$ and $p_j$, while $p_m$ indicates the merge point, i.e., the video object frame at which streams $i$ and $j$ merge. Further, $d$ corresponds to the distance (in frames) between the playback positions of stream $i$ and stream $j$, while $d_m$ indicates the distance (in frames) between
CHAPTER 4. STREAM MERGING STRATEGIES

Figure 4.2: Piggybacking system state.

the playback position of stream $j$ and the merge point. Finally, $W_p(p_i)$ defines
the catch-up window for a policy $p$, i.e., the largest possible distance between
the playback positions of streams $i$ and $j$ so that merging the streams would be
beneficial. It is assumed that the playback position of $j$ is earlier than that of $i$,
and, for merging to be possible, that $S_i > S_j$.

In the original proposal of the piggybacking scheme, the authors presented
four merging policies: the baseline policy, the odd-even reduction policy, the
simple merging policy, and the greedy policy. When employing the baseline policy,
there is no modification of the display rate at all. Under the odd-even reduction
policy, consecutive requests are paired up for merging when possible. The simple
merging policy is similar to the odd-even policy, but instead of grouping requests
in pairs, requests are grouped if they arrive within a specific catch-up window
$W_{sm}(0)$. The greedy policy tries to merge requests as many times as possible
and defines a new catch-up window at every merge point.

Assume that there are eight streams $s_1, \ldots, s_8$ started at times $t_1, \ldots, t_8$,
where $t_1 < t_2 < \cdots < t_8$. Under the odd-even policy and greedy policy, $s_8$
merges with $s_7$, $s_6$ with $s_5$, $s_4$ with $s_3$, and $s_2$ with $s_1$. Under the greedy policy,
the merging continues by merging $s_7$ with $s_5$, $s_3$ with $s_1$, and lastly merging $s_5$
with $s_1$.

Under the baseline policy, the bandwidth demand is dependent on the number
of streams, denoted by $N$. This means that $BW_b = NC_N$, where $C_N$ denotes
the bandwidth demand for every stream (of the same object). The authors
of [57] show using simulations and analytic results that, under the assumption
of Poisson arrivals and for popular content (interarrival times of less than 30 s), the

44
greedy algorithm yields over 80% reduction in bandwidth utilisation compared to the baseline policy. The odd-even and simple merging policies achieve a reduction of about 50%.

In [4], the authors present two additional merging policies: the generalised simple merging policy and the snapshot algorithm. The generalised simple policy performs almost similar to the original greedy policy, and the snapshot algorithm outperforms all the previously discussed policies.

In [82], an additional grouping merging policy is presented: the equal-split algorithm. Under the equal-split algorithm, streams are grouped such that the largest distance between any two streams in a specific group is \( W \), the duration of the catch-up window. The equal-split algorithm outperforms both the greedy and odd-even policies and performs almost as well as the case of a-priori knowledge of all arrivals and evaluation of all possible merging trees (the offline brute-force algorithm).

4.3 Patching

Patching [23, 64] is a merging technique in which clients that arrive late for a multicast are allocated a new multicast channel, a patching channel, for the missing stream prefix. That is, the prefix is “patched” by the patching channel. When making their initial request, clients start receiving data on both the scheduled multicast channel and the patching channel. When the client has received enough data on the patching channel, i.e., has caught up with the data cached from the multicasted stream, the patching channel is released. Patching decreases the number of allocated regular multicast channels on the server at the expense of more, but shorter-lived, patching channels. Additionally, the storage requirements on the clients are increased, as they must receive two streams simultaneously and cache part of the video object.

The primary parameter in patching is the patching window for a video object \( v \), denoted by \( W(v) \). The patching window indicates the maximum skew, i.e., the time since the start of a normally started broadcast. The part of the stream
that needs patching is denoted by $v[\text{skew}]$. The size of the patching window is the most important parameter when using patching. If the patching window is too large, too many clients occupy patching channels instead of merging into the regularly scheduled multicasts. On the other hand, if the patching window is too small, too many clients will have to wait for a new scheduled multicast channel. If the patching window is set to be 0 minutes, patching degenerates into a normally scheduled multicast. If the patching window is set to the duration of the video object, the patching policy is called greedy patching. Under greedy patching, no new multicast channels are allocated as long as there is at least one channel already active. This means that even if a client arrives very late, a new patching channel will be allocated to it, which results in sub-optimal server bandwidth utilisation.

A second policy, grace patching (presented in [64]) addresses this issue by taking client buffer space into account. Under grace patching, a patching channel is only allocated if the client has enough buffer space to cache all of $v[\text{skew}]$. Otherwise, a new multicast channel is allocated. In [24], the authors show that, while grace patching typically performs significantly better than greedy patching, it degrades badly as the client buffer sizes grow. This is because the patching window grows accordingly, and a situation similar to that induced by greedy patching occurs when client buffers grow large. To avoid this situation, the authors derive a formula for the average server bandwidth requirement, depending not only on the client buffer size, but also on the client request rate.

$$B_s = b \cdot \frac{D_{[0,W(v)]}}{W(v)+\frac{1}{\lambda}}.$$  \hspace{1cm} (4.1)

In Eq. (4.1), $B_s$ is the required server bandwidth, $W(v)$ the length of the patching window, $\lambda$ the client request rate, and $D_{[0,W(v)]}$ represents the amount of data transmitted during the interval $[0,W(v)]$. The authors derive six different possible optimal patching window lengths, depending on the length of the video object and the size of the client buffers:
4.3. PATCHING

\begin{align}
W_1(v) &= 0 \quad (4.2) \\
W_2(v) &= \frac{\sqrt{2|v|\lambda^2 - \lambda + 1} - 1}{\lambda} \quad (4.3) \\
W_3(v) &= B \quad (4.4) \\
W_4(v) &= |v| - B \quad (4.5) \\
W_5(v) &= \frac{-1 + \sqrt{\lambda^2 (B - |v|) (|v| - B + 1) + \lambda (2D_{[0,|v|-B]} - 1) + 1}}{2\lambda} \quad (4.6) \\
W_6(v) &= |v|, \quad (4.7)
\end{align}

where $|v|$ represents the entire duration of the video object. The optimal patching window is the one that minimises Eq. (4.1), with $W(v)$ replaced by $W_i(v), i = 1, \ldots, 6$.

One issue that appears with the normal patching scheme is that as soon as a client has received the needed prefix, the patching channel is deallocated. In [22], the authors show that, by letting the patching channel exist for a few more time units, bandwidth savings of up to 50% can be achieved. This is because arriving clients have a larger probability of finding an already existing patching channel. The proposed technique is termed transition patching, and the extra data sent after the patching stream is called a transition stream. Accordingly, when the patching channel is finished, it becomes known as the transition channel.

### 4.3.1 Catching

*Catching* [56] is a merging scheme very similar to patching. The basic idea behind them is the same, *i.e.*, to subscribe to a regularly scheduled broadcast stream while simultaneously playing back the missing prefix of the stream by using an additional connection to the media server. However, catching is based on controlled multicast [55], which differs from the original patching scheme in that it employs a patching threshold to avoid patching windows from becoming too large, much like the optimal patching scheme in [24]. The authors base the catching mechanism on previous work reported in [54] and [55], employing both
CHAPTER 4. STREAM MERGING STRATEGIES

Greedy Disk-based Broadcast (GDB) and controlled multicast as foundations of catching. The analysis in [56] focuses on the optimal number of patching channels rather than on the size of the patching window (termed catch-up window in [56]). The authors show that the number of channels required by catching only differs by a constant factor from the minimum achievable using any given broadcasting scheme.

The normal catching scheme is most efficient for “hot” video streams, since this makes the duration (and thus also the number) of the catch-up channels shorter than for “cold” streams.

Selective catching is a modification of the original catching scheme, in which catching is used in conjunction with controlled multicast depending on whether the video stream is “hot” or “cold”. In selective catching, a video is considered “hot” if

\[
K_i^* + \frac{L_i\lambda_i}{\sum_{j=1}^{K_i^*} f(j)} > \sqrt{2L_i\lambda_i + 1} - 1, \tag{4.8}
\]

where \(K_i^*\) is the optimal number of channels needed for video delivery, \(L_i\) is the length of the video object, \(\lambda_i\) is the average request rate\(^2\), and \(f(j)\) is a partitioning function\(^3\) defined by extending GDB, given by

\[
f(n) = \begin{cases} 
1 & \text{if } n \leq 3 \\
2 & \text{if } 3 < n < 6 \\
5 & \text{if } 6 < n < 8 \\
12 & \text{if } 8 < n < 10 \\
5f(n-4) & \text{if } n > 9.
\end{cases} \tag{4.9}
\]

In addition to the two catching schemes, the authors also present a proxy-based solution based on selective catching that further decreases the number of

\(^2\)Assumed to be Poisson.

\(^3\)This term is analogous to the “segment sizing algorithm” used in Chapter 3.
channels required at the media server by storing the prefixes of video objects at proxies closer to the clients.

### 4.4 Chaining

In chaining \[116\], server network I/O is reduced by using the client workstations as buffers, and a stream is viewed as being made up of a number of retrieval blocks, or R-blocks. Each R-block is made up of a set of Group of Pictures (GOP)\[^4\]; a single R-block is the smallest data unit considered in the chaining scheme.

The primary parameters in a chaining setup are the amount of caching storage (in R-blocks) on a client, denoted by \(b_d\), and the playback distance between two clients, denoted by \(d(i, j) = j - i\), where \(i\) and \(j\) represent playback positions of two different clients\(^5\). Two clients can be served by the same video chain, \(i.e.,\) multicast channel, as long as the playback distance between them does not exceed \(b_d\). For instance, the distances in Figure 4.3 are:

\[
\begin{align*}
    d(b, a) &= 4; \\
    d(c, b) &= 0; \\
    d(d, c) &= 3; \\
    d(e, d) &= 4 \\
    d(f, e) &= 7; \\
    d(g, f) &= 2; \\
    d(h, g) &= 1.
\end{align*}
\]

The playback distances between two adjacent clients between \(a\) and \(e\) are all less than or equal to \(b_d = 4\), but the playback distance between client \(f\) and \(e\) is \(d(f, e) = 7\), which means that clients \(f\) to \(h\) can not join the chain started by \(a\), \(i.e.,\) Chain A in Figure 4.3. The first client in a chain (clients \(a\) and \(f\) in the scenario in Figure 4.3) is known as the head of the chain. The head of a chain is the only client that needs to receive the video stream from the server, and the head then further multicasts the stream to clients arriving later. Note that the head is only responsible for multicasting to clients that arrive within \(b_d\) R-blocks. Clients arriving later are responsible for the subsequently arriving

\[^4\]A GOP is a set of I-, P-, and B-frames making up a Motion Picture Experts Group (MPEG) video stream.

\[^5\]It is assumed that \(j > i\).
clients. In Figure 4.3, a is responsible for multicasting to b and c, c to d, d to e, f to g, and g to h.

This policy, in which clients are made part of the same chain if and only if the playback distance between them is smaller than $b_d$, is known as standard chaining. For popular video objects, this works fairly well, but when request interarrival times increase, the policy rapidly deteriorates, as less streams are merged. To remedy this, the authors propose a heuristic scheme, Heuristic Extended Chaining (HEC), in which clients are delayed before they are allowed to join the chain. By doing this, the chain is kept open for a longer period of time, thus allowing clients arriving later to join the chain. Note, however, that HEC best deals with temporary increases in client interarrival times and does not solve the problem with merging streams when the playback positions are large. In this case, the problem is quite similar to the one patching tries to address. However, while patching requires a single client to store the entire patching stream, chaining can amortise this storage cost over several clients.

### 4.5 Hierarchical and Hybrid Merging

Hierarchical Multicast Stream Merging (HMSM) [44] is a hybrid merging strategy which attempts to integrate the advantages of several broadcasting mechanisms
and stream merging schemes. The HMSM algorithm operates under the following assumptions:

1. Video streams are transmitted using multicast.
2. Clients can receive data at a higher rate than the playout rate. The server transmits streams at the playout rate.
3. Clients merge into larger groups when possible.
4. Clients in the same group receive on the same multicast channel.

As the name HMSM indicates, the scheme attempts to merge clients into larger and larger groups in order to achieve a higher degree of stream merging. This is similar to both the dynamic skyscraper scheme [43] and the greedy policy in piggybacking [4]. However, in contrast to piggybacking, but analogous to patching and chaining, patch streams are allocated dynamically. Additionally, under the HMSM scheme, the playout rates are not modified as they are in piggybacking.

The authors of [44] show by using simulation that an HMSM(2,1) scheme\(^6\) outperforms both the dynamic skyscraper and optimal patching schemes. A very similar scheme, Hierarchical Hybrid Multicast Stream Merging (HHMSM), is reported in [71].

### 4.6 Summary

In this chapter, we discussed several stream merging strategies. Stream merging schemes try to solve the initial delay problem of the broadcasting schemes presented in the previous chapter, while at the same time decreasing the required server bandwidth.

A common feature of stream merging schemes is that they require caching part of the stream in order to be able to join previously started transmissions. There are several issues to consider with regard to caching, in particular under

\(^6\)A scheme in which the server transmits at the playout rate, and clients can receive at twice that rate.
the strict timing constraints of streaming media. This is the topic of the following chapter, Chapter 5, “Caching Strategies”.

Caching has been an important part of the World Wide Web (WWW) since its inception. Caching, i.e., storing objects closer to the location where it is consumed, can be performed at two basic entities: at the client and at an intermediary caching proxy. Proxies are entities acting on behalf of clients, forwarding requests to the origin server and storing the results of these requests for future requests from the same or other clients. This decreases the load on the origin server as well as the access latency and jitter for clients. The functional requirements of a proxy are thus those of both a client and a server\textsuperscript{1} \cite{33}. When caching at the client, two caches are typically used: a small in-memory cache and a larger on-disk cache. The access latency of the cached objects grows when cache misses occur and the client must access objects from the on-disk cache or from the caching proxy.

When caching Web objects, the entire object is usually cached. Since typical Web objects (usually HyperText Markup Language (HTML) and image files) are rather small\textsuperscript{2} – in the kilo- or megabyte range – this is a feasible policy. However, when caching larger objects, such as high-definition video streams

\textsuperscript{1}Incidentally, this coincides with the functional description of a P2P peer, or servent.

\textsuperscript{2}Though research has shown that Web object sizes can be described by heavy-tail distributions, most objects are rather small \cite{34, 69, 79}.
CHAPTER 5. CACHING STRATEGIES

or live video\(^3\) (where object sizes may be in the double-digit gigabyte range) caching entire objects is not a realistic option. One way of solving this is to use Content Distribution Networks (CDNs) for proxying and caching these large objects, but CDNs may be very expensive. A more elegant way is to only cache parts of the objects, a mechanism known as \textit{partial caching} [27].

Partial caching entails partitioning the object into smaller, more manageable amounts of data than the original object. This partitioning can be done either by separating the object into disjoint byte ranges or by using specific encoding schemes that separate the object into objects with decreasing, but additive, quality. The first method is called \textit{segment-based}\(^4\) and can be viewed as a time-domain division of the object. The second method is called \textit{layer-based} and can be regarded as a quality-domain division. As previously mentioned, layer-based caching requires specific media codecs that can produce several streams (or layers) of differing quality of the same object, which can be assembled at the receiver to produce the original object. Often, the different layers are called \textit{base} and \textit{enhancement} layers. The quality\(^5\) of the video depends on the number of streams the client receives and re-assembles.

While segmentation mechanisms basically describe \textit{what} should be cached\(^6\), they do not consider \textit{when} caching should occur. Objects may be cached either \textit{on-demand}, \textit{i.e.}, as the direct result of a client request, or be \textit{prefetched}, \textit{i.e.}, be cached before a client requests the object. There is a tradeoff between caching early, \textit{i.e.}, prefetching, and caching late, \textit{i.e.}, on-demand. When the caching policy is on-demand, there is always an initial delay before the object can be consumed, regardless of the segmentation policy or segment size. This is because the object has to be retrieved from the origin server before being relayed to the client. The advantage with on-demand caching is that the cached content reflects the actual requests of the clients, whereas prefetching policies have to make some inference as to which object and which parts of the object should be cached. In highly dynamic environments, the access patterns of clients may be

\(^3\)The length of a live broadcast is not known in advance.
\(^4\)Several methods for this segmentation are discussed in 3, “Broadcasting Strategies”.
\(^5\)In terms of resolution, colour depth, etc.
\(^6\)In the single-object case.
difficult to predict, and proxies may cache content that is never requested thus wasting the bandwidth for downloading it.

Another important factor to consider when designing a caching system is where to locate content replicas. If a cache is placed close to the requesting client, the access latency is low, but the caching proxy will primarily act as a single-client cache or at least serve a very small group of clients. On the other hand, if the cache is placed closer to the server, it can serve a larger number of clients, but at the cost of higher storage requirements and larger client access latencies. If placed at an edge node, it is possible to avoid deploying costly infrastructure hardware, but this also makes management of the architecture more difficult. Additionally, while processing and network capabilities of edge nodes have increased dramatically during the last decade, they still can not provide the same performance as dedicated task-specific hardware such as IP routers.

5.1 Replacement Policies

Regardless of what type of object is being cached, the caching entity must make a decision of when to replace objects or object segments in its cache. To decide what object (or segment) to replace, a suitable policy must be used. If a poor policy is used, it will result in unnecessary requests to the server.

Some of the more common cache replacement policies are [79]:

**Least Recently Used (LRU):** under the LRU policy, the object unaccessed for the longest amount of time is selected for replacement.

**Least Frequently Used (LFU):** under the LFU policy, the object accessed the least is selected for replacement.

**Size of object:** the largest object in the cache is selected for replacement.

**Hyper-G:** the Hyper-G policy uses both the LRU, LFU and size policies, in that order. If there are tied objects under one policy, the next policy is used as a tie-breaker.
GreedyDual: under the GreedyDual policy, each object in the cache is assigned a utility value, and the object with the lowest utility is selected for replacement.

These replacement policies are also known as block-level policies. In the full-file scenario, these elements consider entire objects, *e.g.*, files, as the caching “blocks”. However, in the partial caching case, each object segment is considered a “block” without regard to which object it is part of.

Most replacement policies can be generalised to using a utility function. The notion of a utility function makes it possible to design more complex replacement policies. For instance, in [27], the authors design a utility function taking into account the following parameters:

- Average number of cache accesses.
- Average access duration.
- The size of the cached object.
- Predicted probability of future access.

A QoS-adapted utility function can, *e.g.*, take into account parameters such as video object compression ratio and codec. For instance, a Constant Bit-Rate (CBR) codec has significantly different statistical properties from a Variable Bit-Rate (VBR) codec. One crucial property of a replacement policy for streaming media that *must* be present is that it should provide for *continuous streaming*. A cache hit that occurs too late for playback is in effect a cache miss. Classical replacement policies such as LRU and LFU can not guarantee this, and a replacement strategy for streaming media must take the access patterns of the data into account. One such policy is the Generalized Interval Caching (GIC) policy, which additionally handles both very large and very small media objects well [37]. One of the reasons for this is that GIC takes into account the fact that segments have inter-object dependencies, *i.e.*, that segment $s_{i+1}$ of an object is more likely to be accessed than $s_{i-1}$ when $s_i$ has just been accessed. In Resource-Based Caching (RBC) [119], cache entities are characterised by three parameters: bandwidth requirement, storage requirement, and caching gain.
5.2. SEGMENT-BASED CACHING

Objects are classified as either continuous, i.e., streaming, or non-continuous, e.g., Web objects or media objects. The caching gain is then calculated in different ways depending on the class of the object to be cached as well as on the granularity of the object segments. A modification of RBC, pooled RBC [7], utilises some of the excess bandwidth that is unused under the original RBC policy.

5.2 Segment-based Caching

Segment-based caching is the most common form of stream caching. This is because it lends itself well to broadcasting schemes such as those presented in Chapter 3, “Broadcasting Strategies”, since these mechanisms already partition the objects into segments. For instance, in [42], the authors describe an architecture based on the dynamic skyscraper scheme (Section 3.4.2) that uses regional proxies to cache stream segments. Their method reduces delivery costs by up to 52%. They also note that storing stream prefixes for many streams tends to be more cost-effective than storing the entire object for fewer files. An example of an extension of a stream merging scheme is Self-Organizing Cooperative Caching Architecture (SOCCER) [61], which uses patching (Section 4.3) as its foundation. In SOCCER, caching is performed in one of two ways:

**Statically**: entire segments are stored and are replaced according to a specific replacement policy.

**Dynamically**: a ring buffer is used to mask the temporal differences between consuming clients.

Depending on the situation, the SOCCER system employs both caching methods. The caching is performed by a set of self-organizing proxies called *helpers* and which are similar in functionality to waypoints in WPMC.
5.2.1 Prefix Caching

Prefix caching [111] is a special case of the more general segment caching. As the name implies, prefix caching entails caching only the initial segments of objects. This decouples the prefix and suffix transmission schemes. Prefix caching is beneficial for several reasons:

• It conserves storage space at the proxy, as full objects need not be stored. Research indicates that 20% is enough to obtain a substantial saving in transmission cost [132].

• Disk access at the proxy may be decreased, as there is less random seeking in large files\(^7\).

• It allows the proxy to be more flexible in adapting the prefix transmission scheme to changing regional network conditions without changing the global transmission scheme [134].

• It can aid the proxy in performing smoothing of VBR streams, which can significantly decrease the variability of this type of video stream [109].

• Using prefix caching for streaming media as opposed to caching entire objects significantly lowers the transmission cost [132].

5.3 Smoothing and Prefetching

An important characteristic of VBR video objects is that, in many cases, the bit-rates of the objects exhibit Long-Range Dependence (LRD) properties [13, 80, 86]. The most important aspect of LRD in video objects is that their bitrates exhibit large burstiness on several timescales. This has significant impact on the bandwidth requirements for transmitting the object as well as on the buffer size requirements at clients and proxies.

\(^7\)Given that the proxy only stores prefixes.
5.3. SMOOTHING AND PREFETCHING

One way of addressing the problems with VBR video objects is by using a CBR codec when compressing the video instead of a VBR codec. However, these codecs are less efficient in utilising the available bandwidth and provide lower quality video for the same average bitrate [123] when compared to VBR codecs. A more elegant way is making the VBR video appear as a CBR video. This approach is known as smoothing [108, 109, 112]. The common trait for all smoothing schemes is that they prefetch (in the client and proxy cases) or pre-transmit (in the server case) video frames before a burst. This reduces the size of the burst, since parts of it have already been transmitted. Smoothing can be performed in several ways, e.g., online and offline, and at different network locations, e.g., at an intermediary proxy, at the media server, or at the client.

When performing offline smoothing, the frame sizes of the entire video object are known in advance. This is typically the case for pre-recorded material such as movies and television shows. The advantage with this is that it is possible to calculate an optimal transmission schedule for every video object. For instance, bandwidth savings of 200-400% are possible using the method described in [112].

However, for non-pre-recorded material such as video teleconferencing and live streaming video, frame sizes are not known in advance. In this case, online smoothing must be used. Online smoothing implies predicting frame sizes and adjusting transmission rates according to these predictions. Even so, depending on the length of the prediction interval as well as on the client and server buffer sizes, significant bandwidth savings can be made in the online case.

From the discussion of smoothing above, it is clear that there is a strong connection between smoothing and prefetching. Beyond smoothing, prefetching is typically used to decrease the initial playback delay or to mitigate available bandwidth fluctuation issues. In the ALM system dPAM [114, 115], a prefetch-and-relay scheme is used to decrease the server bandwidth requirement. The authors show that, even under churn, server bandwidth is reduced while client playback quality is increased.
CHAPTER 5. CACHING STRATEGIES

5.4 Summary

In this chapter, we discussed various issues pertaining to caching in general and stream caching in particular. We presented several cache replacement policies, as well as the stream caching mechanisms known as segment-based caching and smoothing.

Both segmentation and caching are vital components in a well-performing video streaming system. By designing a segmentation, caching, and prefetching scheme in a suitable way, it is possible to not only decrease the server bandwidth requirements, but also to manage heterogeneity and churn in ALM systems.
Part II

BitTorrent Streaming
Chapter 6

BitTorrent Streaming and Piece Selection

The previous part of the thesis discussed media, in particular video, replication and distribution from a conventional IP-based point of view. This first chapter of the second part continues the theme of video distribution, albeit from a P2P, or overlay, point of view.

In this chapter, we discuss BitTorrent (BT), a popular P2P content distribution system and modifications thereof to adapt it to be able to provide a streaming service.

6.1 BitTorrent

BT is a P2P system for content distribution and replication designed to quickly, efficiently, and fairly replicate data [20, 31]. It uses swarming, i.e., downloading partial content from different peers, to distribute load among participating peers. A BT network, also denoted as a swarm, does not provide any resource query, lookup, routing, or topology forming functionality. The sole purpose of a BT swarm is to efficiently disseminate data. Consequently, the system is primarily used to distribute large pieces of content, such as operating system distributions
or compressed video files.

A swarm consists of a number of peers and at least one tracker. The role of the tracker is to provide peers with the IP addresses of other peers; it does not participate in the content distribution. Peers are partitioned into seeds and leechers. A seed is a peer that is in possession of the entire content of the swarm, while a leecher is a peer that still needs to download parts of the content. An initial seed is necessary for a swarm to be successful, i.e., for non-seed peers to be able to download the entire content.

The content distributed in a BT swarm is divided into pieces. Data is requested among peers using piece numbers and byte ranges within these pieces as identifiers. These byte ranges are also known as subpieces, blocks, or chunks.

The leech phase of a BT peer may be partitioned into three different sub-phases. During the first sub-phase, the peer tries to connect to a pre-defined maximum number of peers. This means that the number of connected peers increases during this phase, thus increasing the number of outgoing piece requests as well. On entering the second phase, the peer has connected to enough peers. During this phase, the number of connected peers and outgoing messages fluctuates around an average value. The final phase is the end-game mode, which occurs when a peer only has a few pieces of the content left to download. To quickly obtain the last few pieces, requests are broadcasted to all connected peers, which results in a dramatic increase in the upstream request rate.

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 of the swarm tracker and a set of Secure Hash Algorithm One (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.

Fairness in the BT system is implemented using a scheme in which peers express a desire to download by sending interested-messages to peers that have needed pieces. The serving peers may allow or disallow download with unchoke or choke-messages, respectively. Data transfer takes place by the downloading peer issuing request-messages to randomly selected serving peers and the serving peers
responding with *piece*-messages. Each *request-piece*-message exchange refers to one subpiece. Once all blocks are downloaded, a *have*-message is transmitted to all connected peers.

### 6.1.1 Piece Selection and Reciprocation

The two most important algorithms in BT are the *choke algorithm*, *i.e.*, the peer selection algorithm, and the *rarest-first algorithm*, *i.e.*, the piece selection algorithm. In [85], the authors present a detailed study of both the choking algorithm and the rarest-first piece selection algorithm. The rarest-first algorithm uses four policies:

**Random first**: A peer selects pieces to download at random until it has downloaded four full pieces.

**Strict priority**: When a part of a piece (termed *block* or *chunk*) is downloaded, the remaining blocks of this piece are prioritised.

**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.

**Rarest first**: A peer keeps a list of what pieces every participating peer has and uses this information to maintain 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.

### The Choking Mechanism

The *choke/unchoke* and *interested/not interested* mechanisms provides fairness in the BT protocol. Since it is the transmitting peer that decides whether to
allow a download or not, peers not sharing content tend to be reciprocated in the same manner. In the reference BT implementation, clients maintain a rotating list of four peers unchoked at any given time. The list of peers that are unchoked is reassessed periodically.

To allow peers that have no content to join the swarm and start sharing, a mechanism called optimistic unchoking is employed. Optimistic unchoking means that from time to time, a peer with content allows even a non-sharing\(^1\) peer to download. This allows the non-sharing peer to share the small portions of data received so far and thus enter into a data exchange with other peers.

This means that while sharing resources is not strictly enforced, it is strongly encouraged. It also means that peers that have not been able to configure their firewalls and/or Network Address Translation (NAT) routers properly will only be able to download the pieces altruistically shared by peers through the optimistic unchoking scheme.

More detailed descriptions of the BT system are available in [30, 46, 85].

### 6.2 Piece Selection Algorithms for Static Content

Most BT implementations are used for downloading static files, i.e., files that are not consumed while downloading, such as game patches, disk images, and other large files. This significantly impacts the implementation choices made for the piece selection algorithms. However, there are some general requirements for a piece selection algorithm in order for a swarm to be healthy, i.e., have all pieces replicated in about the same amount. If there is an uneven distribution of pieces in the swarm, i.e., only a few peers have a specific piece or set of pieces, these peers can be overloaded with requests.

- The first requirement is that the algorithm should quickly provide a client with full pieces. This is because without pieces to share, the client is dependent on the optimistic unchoking of other peers, and the download

\(^1\)A new peer typically has no data to share.
rate is likely to be very low. The sooner a peer downloads all the blocks of a piece, the sooner it can send have-messages to the rest of the swarm and other peers can then request these pieces to engage in reciprocal up- and download.

- A second requirement is that the distribution, i.e., the number of peers in possession of a piece, of rare pieces in a swarm should be increased. This requirement is addressed by downloading the rarest pieces, i.e., employing a rarest-first policy. Another way of viewing this requirement is that the piece overlap between peers should be minimised to allow specific pieces to be downloaded from different peers.

Ideally, the notion of rarity should in this case be related to the popularity of the piece as well. For instance, if a specific part of the content is not as popular as other pieces, there is less need to replicate it than more popular content.

6.2.1 Reference Client

The piece selection algorithm implemented in the reference BT client\textsuperscript{2} employs three distinct policies for selecting which single piece to download next and a fourth policy that provides a set of pieces to download next. The algorithm is known as the rarest-first algorithm, and refers to the most important of the policies employed by the algorithm.

Policy 1: When a client enters a swarm and has no downloaded pieces, random pieces are selected for download, i.e., a random-first policy is employed.

Policy 2: As soon as a peer has downloaded a number of pieces (user-configurable, defaults to four pieces), pieces that are the least replicated in the swarm are selected for download, i.e., a rarest-first policy.

\textsuperscript{2}Version 4.0.0.
CHAPTER 6. BITTORRENT STREAMING AND PIECE SELECTION

Policy 3: The third policy overrides the previous two. As soon as a block has been downloaded from a piece, the corresponding piece is prioritised. This policy is called the strict priority policy [85].

Policy 4: The fourth policy is used once all pieces have been requested. It is called the end-game mode and overrides the three previous policies. During end-game mode, requests for the remaining blocks are sent to the entire active peer set. As soon as a remote peer unchokes the client, it cancels all the outstanding requests for the same piece.

Output: piece: number of next piece to download
if #started > 0 then Strict priority mode
   | piece ← random downloading piece#;
else if #downloaded < rarestFirstCutoff then
   Random first
   | piece ← random undownloaded piece#;
else Rarest first
   | piece ← rarest piece#;
return piece

Algorithm 1: Reference BT client piece picker algorithm.

Figure 6.1: Reference client single piece picker policies.

6.2.2 Azureus

The Azureus/Vuze client is one of the most popular BT clients and integrates a video distribution service under the name of Vuze [130]. The client is implemented in Java and incorporates advanced configuration options to control client behaviour. It also features advanced sharing and tracker functionality.

The latest version of the client, version 3, has support for streaming video
6.2. PIECE SELECTION ALGORITHMS FOR STATIC CONTENT

without first downloading the entire video object. Not all media offered by the Vuze service are available in streaming format, however.

The core piece selection algorithm in Azureus is quite clearly documented in the source code of the application. For any connected peer, as soon as a new request should be made, the following logic is applied\(^3\):

0. If there is a FORCED\_PIECE or reserved piece, that will be started/resumed if possible.
1. Scan all the active pieces and find the rarest piece (and highest priority among equally rarest) that can possibly be continued by this peer, if any.
2. While scanning the active pieces, develop a list of equally highest priority pieces (and equally rarest among those) as candidates for starting a new piece.
3. If it can not find any piece, this means all pieces are already downloaded/fully requested.
4. Returns int\[\] pieceNumber, blockNumber if a request to be made is found, or null if none could be found.

Note that item 4 in the comments from the source code does not correspond to the actual source code, as the algorithm returns a piece number only, and not a (piece,block)-tuple.

With only a cursory look, the algorithm employed by Azureus appears very similar to that used by the reference client. However, as item 0 in the comments indicates, the rest of the algorithm can be short-circuited if there are reserved or forced pieces.

The Azureus GUI allows the user to prioritise specific files, as well as to disallow specific files from being downloaded. Additionally, the algorithm always employs the rarest-first policy. The strict priority policy is modified to continue downloading the rarest active piece, as opposed to the reference client, which

\(^3\)Text from Azureus source code, file “PiecePickerImpl.java”, lines 1311–1322. Formatting changed from original source code.
picks a random, active piece to continue downloading. In Azureus, random pieces are only downloaded if they are equally rare and of equal priority.

### 6.3 Piece Selection for Streaming Content

While BT is very efficient in distributing content that has no requirements on timely delivery, the rarest-first algorithm can not provide a streaming service due to the random ordering of downloaded pieces. This, together with the temporal requirements of each piece, is the major problem that needs addressing in order for BT to become a viable option for streaming video.

In addition to the requirements mentioned in Section 6.2, a piece selection scheme for streaming must also address the following requirement:

**Keep the playback deadline.** This implies keeping a non-empty playback buffer, as well as maintaining deadlines for each piece. Maintaining per-piece deadlines also implies a-priori information on the playback duration of a piece. In the case of VBR video, this means that this information must be either pre-computed and transmitted with the torrent file or predicted from the already downloaded parts of the object. Since BT already distributes meta-data as part of the protocol, adding playout deadlines in the torrent file is a suitable solution. It also provides the possibility of smoothing the traffic when downloading the object.

We should note here that while there is a fairly large amount of research on characterising VBR video traffic, we doubt that these results will be significantly relevant to deadlines used for BT pieces. The reason for this is that since VBR video frames are of varying size, with varying inter-deadline times, models for these sizes or inter-deadline times are not directly applicable to the fixed-size BT pieces encapsulating the video frames.

For instance, in Figure 6.2, both pieces 1 and 2 would have to be downloaded for frame 2 to be able to be consumed, thus making the deadline the same for these pieces.
6.3. PIECE SELECTION FOR STREAMING CONTENT

There have been a few attempts to leverage existing video streaming methods with BT functionality and thus decrease the load on the media server. To the best of our knowledge, none of these have been deployed fully. We have discussed some of these systems in Section 1.3, but reiterate briefly below.

6.3.1 BitTorrent-Assisted Streaming

BitTorrent-Assisted Streaming System (BASS) [38] 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 pieces that are after the current playout point for download, storing them for later playback. Clients download pieces in a linear fashion from the media server, except for:

a) Pieces that have already been downloaded by BT.

b) Pieces that are being downloaded by BT and are expected to finish before the playout deadline.

The authors show using simulations 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 [38] 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. In previous work, we have observed that the arrival rates of BT peers, while being hyper-exponentially distributed, depend on the swarm content [48, 72].

6.3.2 BiToS

BitTorrent Streaming (BiToS) [128] is a modification of the BT piece selection algorithm. In BiToS, the piece set is divided into three disjoint subsets: the received pieces set, the high priority (HP) set, and the remaining pieces (RP) set. Pieces that are close to being 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 that 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 \( 1 - p \).

6.3.3 Azureus xStream

In May 2007, the Vuze service was extended with a beta version of a streaming service, called xStream. By inspecting the torrent files and data downloaded, some observations regarding the functionality of the service can be made.

The first observation is that extensions to the torrent file have been made. In addition to the standard fields, Vuze adds an application-specific dictionary named “azureus_properties” containing content-related information such as “publisher”, “title”, “content type” and a thumbnail of the media. In addition to these fields, QoS-related fields such as “QOS class” and “Progressive” have
6.3. PIECE SELECTION FOR STREAMING CONTENT

been added. These fields appear to occur in all media distributed by the Vuze service. However, in the torrent files used for streaming, another field, “Speed Bps” is included. Whether this is the average playback rate, necessary download rate or something similar is not known.

Figure 6.3: Azureus piece selection over time.

Figure 6.3 shows the difference in piece selection when using xStream compared to the normal Azureus piece selection scheme. Figure 6.3(a) displays the piece selection results obtained from the normal, non-streaming algorithm, while Figure 6.3(b) displays the corresponding results for the streaming algorithm. The same data was downloaded for both tests. The non-streaming algorithm quite clearly selects pieces more randomly than the streaming algorithm. The linear piece selection can clearly be observed in the streaming algorithm.

6.3.4 On the Suitability of BitTorrent for Streaming

BT has become one of the most popular P2P applications and is primarily being used as an efficient distribution system for large files. There are, however, few
CHAPTER 6. BITTORRENT STREAMING AND PIECE SELECTION

applications and systems that make use of BT as a streaming media solution. We have presented a few in the previous section, and while there are other proposals for using BitTorrent as a media delivery system, such as BitTorrent.com, these do not provide streaming services. Whether this is because of technical reasons or administrative reasons is difficult to know. However, we want to discuss the factors that affect the adoption of a BT-like system for streaming. By “BT-like,” we refer to systems that:

- Use some form of meta-data that describe the object to be streamed. Peers thus have full a-priori knowledge of the content to be streamed.
- Use swarming and thus implicitly segmentation.
- May have a central entity present. If present, it manages peers, but does not partake in the data transmission.

Positive Features

BT has several characteristics that makes it interesting as a possible solution for streaming media to a large subscriber group. The following is a discussion of these characteristics:

Segmentation: The BT system is based on segmentation by default. It has support for requesting segments of various sizes on a peer-to-peer basis. This means that there is no need for a media server to make compromises in the segmentation scheme to handle various client access patterns. Peers request the segments they need from other peers, regardless of any server segmentation scheme.

Meta-Data: To be able to join a BT swarm, a peer must first obtain the set of meta-data corresponding to the swarm. This provides several desirable features:

- Implicit access control. Unless a peer has the meta-data, it can not join the swarm, since the SHA1 of the meta-data is the identifier of the entire swarm.
6.3. PIECE SELECTION FOR STREAMING CONTENT

- **Implicit group management.** The swarm can be viewed as an ALM multicast group, but without the need for management protocols. The meta-data is the group identifier or address.

- **A-priori information about the stream is available.** The original version of the torrent-file format contains information about the data distributed in the swarm and SHA1 hashes to verify reception of the data. However, it is possible to extend this information with extra data, for instance regarding bitrates and bitdepths of a video object. This can facilitate more intelligent decisions regarding piece selection and also allow for optimal smoothing of VBR video objects.

**Negative Features**

As BT was not initially designed for streaming media, some mechanisms are not well-suited for streaming media. To design a streaming system with BT as a foundation, a few issues must be addressed:

**Piece selection:** The original piece selection worked according to the rarest-first algorithm (Section 6.1.1) which does not presuppose any ordering requirement of segments. A streaming solution has requirements on both ordering and timeliness, which means that the original piece selection algorithm must be modified.

**Peer selection:** The original peer selection algorithm favoured peers with high upload rates. Version 4.0.0 of the reference client changed this to favour peers that have been downloading for shorter amounts of time and used bandwidth as a tie-breaker. Transmission latency is not considered, except for the response delay of a peer. If a peer does not send any data in a certain amount of time, it is punished (*snubbed* in BT terminology) for not cooperating.
CHAPTER 6. BITTORRENT STREAMING AND PIECE SELECTION

6.4 BitTorrent Extensions for Streaming

In this section, we discuss a number of changes we suggest making to the core BT algorithms and mechanisms. These changes consists of meta-data additions to the torrent file, as well as changes to the piece selection algorithms. For brevity, we refer to these changes as BitTorrent Extensions for Streaming (BEST).

6.4.1 Torrent File Extensions

One of the main advantages of BT is the fact that it already has a mechanism for describing and distributing meta-data: the torrent file. This means that no new functionality needs to be added to BT clients to parse this meta-data. Also, new meta-data can be added to the torrent file, as well-behaving clients ignore unknown fields.

To facilitate offline smoothing, piece deadlines should be added to the torrent file. This requires a mapping of frame deadlines to piece deadlines. A piece may contain more than one frame or less than a full frame (Figure 6.2). The important thing is that each piece has an associated deadline by which it has to be downloaded for continuous playback. In addition to piece deadlines, other QoS and content-related information could also be added.

6.4.2 Piece Selection Ranges

Our modifications to the original BT piece selection algorithm require a slightly different view on which pieces are available for selection. In the original algorithm, pieces can be selected from the entire set of undownloaded pieces. For an algorithm that takes streaming playback into account, this is not the case. This is because:

1. Pieces that are prior, i.e., lower-numbered, than the currently playing piece are not relevant for download. (At least not until all other pieces are downloaded and then only for altruistic reasons.)
6.4. BITTORRENT EXTENSIONS FOR STREAMING

2. Some pieces are more important than others. Pieces that are close to consumption, \textit{i.e.}, will soon be played back, are more critical to download than pieces that are farther from the playback point.

In Figure 6.4, we illustrate these arguments by dividing the entire piece set into three disjoint ranges: a \textit{seeding range}, a \textit{playback buffer}, and a \textit{picking range}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{piece_ranges}
\caption{Piece ranges.}
\end{figure}

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, \textit{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. Table 6.1 provides an enumeration of the three disjoint piece ranges.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Pieces} & \textbf{Range} \\
\hline
\{1, \ldots, p\} & seeding range \\
\{p + 1, \ldots, p + s\} & playback buffer \\
\{p + s + 1, \ldots, n\} & picking range \\
\hline
\end{tabular}
\caption{Piece picking ranges.}
\end{table}
In Table 6.1, \( p \) refers to the currently playing piece, \( i.e., \) the playout-point, \( s \) denotes the size of the playback buffer, and \( n \) represents the total number of pieces in the piece set.

### 6.4.3 Piece Selection Algorithms

Once the playback buffer is filled, \( i.e., \) the pieces \( p+1, \ldots, s \) are either downloaded or downloading, the pieces in the picking range are available for selection. We test several piece selection algorithms:

- **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.

- **Smoothing:** Select pieces that contribute most to network traffic load, \( i.e., \) the pieces that have the shortest associated deadlines.

- **Rarest-first:** The default BT selection algorithm. Pick the pieces that are least replicated in the swarm.

- **Hybrid:** Various combinations of the above.

### Linear download

We have implemented the linear download algorithm primarily as a way of relating the BT sharing scheme in a streaming scenario to conventional Web streaming and do not expect this algorithm to perform well with respect to server load and sharing efficiency.

### Smoothing / most disruptive first

Smoothing [109] is used to decrease the variability of VBR video by prefetching data when the link is under-utilised. This can be done either *online*, where future video frame sizes are predicted and *offline*, where they are known in advance. In
the BT scenario, offline smoothing can be used, as the frame sizes can be placed in the torrent file. In Figure 6.5, we illustrate this mechanism.

![Figure 6.5: VBR video.](image)

A general algorithm description is as follows: if the current bandwidth utilisation is less than a particular threshold value, \( \bar{b} \), (i.e., in the light grey area in Figure 6.5), prefetch the piece that increases the variability of the traffic the most, i.e., a piece from the dark grey area. In our implementation, a list of deadlines is maintained, and the piece with the shortest deadline is returned from the smoothing algorithm.

**Rarest-first Hybrids**

By rarest-first hybrids, we refer to piece selection algorithms that, at least in part, employ the original BT rarest-first algorithm. In our case, we have opted to implement combinations of the rarest-first and smoothing algorithms by implementing and testing two different variants of this scheme:

**Probabilistic:** This is a very simple scheme, in which pieces are selected using the smoothing algorithm with a probability \( p \) and selected using the rarest-first algorithm with probability \( 1 - p \). For \( p = 1 \), pure smoothing is performed, i.e., the piece with the shortest associated deadline is selected. Consequently, for \( p = 0 \), the original rarest-first algorithm is used.
Deadline-based: The deadline-based scheme uses the smoothing algorithm to implement something more akin to conventional smoothing, i.e., only perform smoothing when the link is under-utilised. The following is a description of the algorithm:

1. Pick a piece using rarest-first, \( p_r \), with inter-deadline time \( t(p_r) \), and associated required download bandwidth \( b(p_r) = 1/t(p_r) \).
2. If \( b(p_r) < \bar{b} \), pick a piece \( p_s \) for smoothing that minimises \( t(p_r) + t(p_s) = t(p_r + 1) \). 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.

![Time Deadlines](chart.png)

**Figure 6.6:** Illustration of the deadline-based scheme.

In Figure 6.6, we indicate the remaining time until the next deadline by the grey areas. The deadlines in Figure 6.6 are drawn to be deterministic for clarity.

6.4.4 Algorithm Overview

The previous sections have described a number of suggested modifications to the piece selection algorithms of BT in order to make it suitable for streaming services. In Algorithm 2, we provide an overview of the general functionality of the BEST modifications. Comparing Algorithm 2 with Algorithm 1 (page 68), we point out that the first mode, i.e., strict priority mode, must be maintained, but with the modification that currently downloading pieces are not selected
randomly for continued download. Instead, the pieces with outstanding blocks
that have the shortest deadlines should be selected.

Output: piece: number of next piece to download
if \( \#_{\text{started}} > 0 \) then
  Strict priority mode
  piece \leftarrow \text{piece with shortest deadline currently downloading};
else if playback buffer not full then
  Fill playback buffer
  piece \leftarrow \text{lowest undownloaded piece within} \{p + 1, \ldots, p + s\};
else
  Current BEST algorithms
  piece \leftarrow \text{piece picked by the currently used BEST algorithm, i.e., the
    linear, probabilistic or hybrid algorithms};
return piece

Algorithm 2: General BEST algorithm overview.

6.5 Summary

In this chapter, we briefly introduced the BitTorrent system and its associated
core algorithms and mechanisms. In particular, we discussed the piece selection
algorithm of both the reference client and one of the most popular alternative
clients.

Furthermore, we presented a set of modifications, termed BitTorrent Extensions for Streaming (BEST), to the core BT algorithms intended to make BT a
suitable alternative for streaming video.
This chapter presents a collection of models and characteristics of key BT traffic characteristics. We provide models at both the session and message levels.

While the models we present have been obtained in normal, i.e., non-streaming BT swarms, we believe the results to still be valid and useful for streaming scenarios as well. For instance, in [46], we noted that the session size of a typical BT peer is limited by the size of the content distributed in the swarm. Peers do not download data unnecessarily. While the measurements we have made cannot provide seeding times, i.e., the time a peer stays in a swarm after having downloaded the full content, assuming this time to be exponentially distributed appears to be an accepted assumption [104]. The tendency of peers to defect as soon as they have downloaded the content is further indicated by the fact that many torrent sites implement ratio systems to entice peers to stay in the system. A similar situation is probable in a streaming scenario, as peers that have finished watching a video have no direct incentive to remain in the system.
CHAPTER 7. BITTORRENT TRAFFIC MODELS

7.1 Measurements

The models we present in this chapter were obtained by analysing data from two distinct measurement studies temporally spaced by about 16 months. The details for these measurements have previously been reported in [46] and [72]. We therefore only provide a brief overview here.

7.1.1 Study 1

For this study, two sets of BT measurements were performed. The first set used the instrumented version of the reference BT client as the main measurement tool, with partial packet capture to determine timestamp accuracy. The second set involved full packet capture and stream reassembly in addition to application logging.

The traffic for the first set of measurements was collected at two different locations over a three-week time period starting on 3 May, 2004. One location was the BTH networking lab, and the other was at a local ISP with a 5 Mbps link. The measurements represent 12 different measurement runs (with lengths of 2 to 12 days) of the instrumented client, 3 of which were run as the only active application. This was done so as to establish a point of reference without applications competing for available bandwidth. To measure more realistic scenarios, the rest of the measurement runs were done with some temporal overlap [50]. A total of 20 GB of uncompressed XML logs were collected in the first set of measurements. After post-processing, the amount of logs was over 25 GB. The logs contain approximately 100 million protocol messages from almost 300,000 individual sessions. The BT log files contain a list of client software states, e.g., tracker announcements, new connections, choke, unchoke, interested, and uninterested, along with the timestamps indicating when the state change took place.

The second set of traces were collected as tcpdump traces at the BTH networking lab during one week, starting 4 June, 2004. A single instance of the reference client was run as the only application on the measurement node. The set
7.2. MODELLING METHODOLOGY

contains 150 GB of data, out of which 143 GB are tcpdump traces. The rest of the data are application logs and post-processed logs. Approximately 22 million messages were transmitted in 53,000 sessions during the second measurement set.

7.1.2 Study 2

For this study, 10 measurements were performed at the networking lab at BTH. The measurements span two weeks in late November 2005. Open source GNU/Linux distributions as well as files under Creative Commons licence were selected as the content for our measurements. Files were specifically selected based on their popularity, size, and content type in order to model a wide range of contents.

The measurements were performed using full packet capture and stream reassembly, as well as tracker scraping and application logging. Three of the measurements were discarded due to hardware failure, disk space issues, and failure to pass self-consistency tests\(^1\).

Around 180 GB of tcpdump traces were collected. In total, over 25 million messages were transmitted in almost 120,000 sessions.

7.2 Modelling Methodology

7.2.1 Distribution Selection

The first activity in a characterisation study is determining a suitable distribution for the parameter under study. For each parameter to be modelled, distribution selection is an exploratory process of observing Empirical Probability Density Function (EPDF) and Complementary Cumulative Distribution Function (CCDF) plots. For inter-arrival and inter-departure times, EPDF plots are primarily used, as these show the body of the distribution more clearly. For message rates,

\(^1\) Pieces missing from the parsed traces.
the CCDF is the preferred method, as it focuses on the tail of the distribution. The CCDF is thus better suited for detecting potential long-range dependent behaviour. This is important to detect in the case of rates, sizes, and durations, as underestimating these characteristics may have an adverse effect on the network.

7.2.2 Parameter Estimation

Based on the candidate distributions selected for modelling, Maximum Likelihood Estimation (MLE) is used to obtain parameter estimates. With the number of observations available for the measurements, the obtained parameter estimates are assumed to be accurate enough to consider the associated distribution fully specified, given that the confidence intervals for the estimated parameters are within acceptable boundaries.

In the case of single and mixture distributions, parameter estimation is a straightforward procedure, and estimates are obtained from the complete set of data. In the censored mixture model case, successive right censoring as employed in [69], together with an error percentage assessment (described in the following section), is used to find out the cutoff points for the mixture model. The censored models are deprecated, however, in favour of more tractable mixture models, as censored models are less convenient to use in a simulation environment.

Once ML estimates are available, the error percentage presented in the following section is further used to optimise the parameters. We have found that this procedure often provides better results than directly accepting the ML estimates.

7.2.3 Fitness Assessment

To determine whether a distribution is representative of the observed data, visual procedures, formal hypothesis tests (to a certain extent), and an error percentage assessment are used.

To assess the quality of the estimated distributions in a more quantitative
7.2. MODELLING METHODOLOGY

manner, a method similar to the Empirical Distribution Function (EDF) test that does not suffer as much with increasing number of observations is employed.

The method is based on the EDF test for a fully specified distribution, as described in [36]:

1. Obtain the order statistics \(X_1 < X_2 < \cdots < X_n\) from the measured data.

2. Transform the original data by using the Probability Integral Transform (PIT) method and the selected distribution and estimated parameters. If the samples \(X_1, \cdots, X_n\) are Independent and Identically Distributed (IID) samples from a distribution \(F\), then \(\hat{U}_i = F(X_i; \hat{\Theta})\), where \(i = 1, 2 \ldots n\), are uniformly IID on \([0, 1]\), and \(\hat{\Theta}\) is a set of estimated parameters to the distribution.

3. Obtain the error percentage by using the following expression:

\[
E_{\%} = \frac{100}{n E_{max}} \sum_{i=1}^{n} |U_i - \hat{U}_i|, \quad (7.1)
\]

where \(E_{max}\) is defined as

\[
\int_{0}^{1} \sup \{U(x), 1 - U(x)\} \, dx = \int_{0}^{\frac{1}{2}} 1 - U(x) \, dx + \int_{\frac{1}{2}}^{1} U(x) \, dx = \frac{3}{4}, \quad (7.2)
\]

or, in plain terms, the maximum discrepancy from a true \(U[0, 1]\) distribution that may occur.

4. Accept or discard the estimated distribution as “good enough” according to pre-defined criteria. For the purposes of this thesis, \(E_{\%} \approx 5\) is chosen as an upper limit for not discarding the estimated distribution. It is important to mention that this is not a statistical significance level, but rather an acceptable margin of error. The informal degrees of fitness quality presented in Table 7.1 are used.
CHAPTER 7. BITTORRENT TRAFFIC MODELS

Table 7.1: Fitness quality boundaries.

<table>
<thead>
<tr>
<th>$E_%$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree</td>
<td>excellent</td>
<td>very good</td>
<td>good</td>
<td>fair</td>
<td>poor</td>
</tr>
</tbody>
</table>

7.2.4 Distributions

In this section, for convenience and completeness, we present the definitions of the Probability Density Functions (PDFs) of the four distributions used for the modelling results reported in Sections 7.3 and 7.4. These are: the binary hyper-exponential, dual Gaussian, log-normal, and Generalised Pareto distributions.

The binary hyper-exponential PDF is defined as:

$$f(x) = p\lambda_1 e^{-\lambda_1 x} + (1 - p)\lambda_2 e^{-\lambda_2 x}, \quad (7.3)$$

where $\lambda_1$ and $\lambda_2$ are the arrival rates for the two exponentials and $p$ is the mixing weight.

The binary, or dual, Gaussian PDF is defined as:

$$f(x) = p\Phi(\mu_1, \sigma_1) + (1 - p)\Phi(\mu_2, \sigma_2), \quad (7.4)$$

where $\Phi$ represents a Gaussian distribution with mean $\mu_x$ and standard deviation $\sigma_x$ and $p$ is the mixing weight.

The log-normal PDF is given in Eq. (7.5), where $\mu$ is the mean and $\sigma_{LN}$ is the standard deviation.

$$f(x) = \begin{cases} \frac{1}{x\sqrt{2\pi\sigma_{LN}^2}} e^{-\left(\ln x - \mu\right)^2/2\sigma_{LN}^2} & x > 0, \sigma > 0 \\ 0 & \text{otherwise} \end{cases} \quad (7.5)$$
The Generalised Pareto PDF is defined as:

\[ f(x) = \begin{cases} 
\frac{1}{\beta} \left( \frac{x - \mu}{\beta} \right)^{-\alpha - 1} + 1 & \text{if } \alpha \geq 0 \\
\mu \leq x \leq \mu - \frac{\beta}{\alpha} & \text{if } \alpha < 0 \\
0 & \text{otherwise}
\end{cases} \quad \text{if } \alpha \geq 0
\]

where \( \beta, \alpha, \text{ and } \mu \) are the scale, shape, and location parameters, respectively.

### 7.3 Session Models

#### 7.3.1 Study 1

In this section, we report the modelling results for the distributions of session interarrival times, upstream session sizes, and durations for our first study. Table 7.2 provides a summary of the number of sessions in each of the thirteen measurements, except for number 9, which was lost due to hardware failure. We observe that measurement 6 is different with regard to both mean session size and mean session duration. Further, the maximum session size for this measurement is more than twice that of any other measurement. The mean session size is observed to be about twice that of the corresponding measurement of the same content (measurement 10). As measurements 6 and 10 have large session sizes, it is likely that the session size in this case is related to the total content size (4.3 GB).

The minimum session durations were all 0 s, indicating that all of them are shorter than the accuracy provided by the application logs. These very short sessions are also reflected in the minimum session size, which is 73 bytes for all the measurements. These sizes correspond to a session containing only a handshake or an interrupted handshake. More detailed information is available in [46].
Table 7.2: Session summary for study 1.

<table>
<thead>
<tr>
<th></th>
<th>Sessions</th>
<th>Session duration (s)</th>
<th>Session size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Max</td>
<td>Std</td>
</tr>
<tr>
<td>1</td>
<td>29712</td>
<td>343</td>
<td>98991</td>
</tr>
<tr>
<td>2</td>
<td>46022</td>
<td>233</td>
<td>117605</td>
</tr>
<tr>
<td>3</td>
<td>28687</td>
<td>465</td>
<td>171074</td>
</tr>
<tr>
<td>4</td>
<td>13493</td>
<td>750</td>
<td>143707</td>
</tr>
<tr>
<td>5</td>
<td>12354</td>
<td>910</td>
<td>180298</td>
</tr>
<tr>
<td>6</td>
<td>10685</td>
<td>1207</td>
<td>223235</td>
</tr>
<tr>
<td>7</td>
<td>4444</td>
<td>218</td>
<td>46478</td>
</tr>
<tr>
<td>8</td>
<td>17287</td>
<td>231</td>
<td>87026</td>
</tr>
<tr>
<td>10</td>
<td>9701</td>
<td>652</td>
<td>267497</td>
</tr>
<tr>
<td>11</td>
<td>43939</td>
<td>448</td>
<td>141509</td>
</tr>
<tr>
<td>12</td>
<td>68288</td>
<td>197</td>
<td>292241</td>
</tr>
<tr>
<td>13</td>
<td>52833</td>
<td>465</td>
<td>483996</td>
</tr>
</tbody>
</table>

Session interarrival times

The reported distributions refer to interarrival times for remotely initiated sessions during the seeding phase of our measurement peer. We do not consider the leech phase, partly because it is short compared to the seed phase and the number of non-locally initiated sessions is fairly low, and partly because the peer is more active during this phase than during the seed phase. The combination of active peer status and low number of samples that is present during the leech phase (e.g., only 10–20 sessions) makes the analysis more difficult.

We have modelled the session interarrival times by using a binary hyper-exponential distribution. Parameter estimates for each of the measurements have been obtained by using a maximum likelihood estimator implemented in
the R language [103]. The estimation procedure is part of the MASS package for R [126]. It uses the built-in optimisation function of R and is based on a gradient algorithm. Table 7.3 reports the parameter estimates and the associated standard deviations obtained in the fitting procedure. Also presented are the $E_{\%}$ value and the resulting fitness degree.

**Table 7.3:** Session interarrival parameters for study 1.

<table>
<thead>
<tr>
<th>#</th>
<th>$\hat{\lambda}<em>1 \pm \hat{\sigma}</em>{\lambda_1}$</th>
<th>$\hat{\lambda}<em>2 \pm \hat{\sigma}</em>{\lambda_2}$</th>
<th>$\hat{p} \pm \hat{\sigma}_p$</th>
<th>$E_{%}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0593 ± 0.0046</td>
<td>0.1696 ± 0.0085</td>
<td>0.2215 ± 0.0467</td>
<td>2.07367</td>
</tr>
<tr>
<td>2</td>
<td>0.1158 ± 0.0009</td>
<td>0.7556 ± 0.0279</td>
<td>0.7936 ± 0.0066</td>
<td>0.41535</td>
</tr>
<tr>
<td>3</td>
<td>0.0566 ± 0.0006</td>
<td>0.3653 ± 0.0099</td>
<td>0.6575 ± 0.0077</td>
<td>0.49009</td>
</tr>
<tr>
<td>4</td>
<td>0.5372 ± 0.0178</td>
<td>0.0168 ± 0.0002</td>
<td>0.2533 ± 0.0052</td>
<td>2.79455</td>
</tr>
<tr>
<td>5</td>
<td>0.5538 ± 0.0212</td>
<td>0.0162 ± 0.0002</td>
<td>0.2156 ± 0.0052</td>
<td>2.79722</td>
</tr>
<tr>
<td>6</td>
<td>0.4798 ± 0.0174</td>
<td>0.0127 ± 0.0002</td>
<td>0.2879 ± 0.0060</td>
<td>3.93588</td>
</tr>
<tr>
<td>7</td>
<td>0.4188 ± 0.0143</td>
<td>0.0052 ± 0.0001</td>
<td>0.3014 ± 0.0076</td>
<td>2.05403</td>
</tr>
<tr>
<td>8</td>
<td>0.5142 ± 0.0113</td>
<td>0.0168 ± 0.0002</td>
<td>0.4252 ± 0.0050</td>
<td>2.79291</td>
</tr>
<tr>
<td>10</td>
<td>0.5581 ± 0.0205</td>
<td>0.0128 ± 0.0002</td>
<td>0.3276 ± 0.0064</td>
<td>3.76412</td>
</tr>
<tr>
<td>11</td>
<td>0.0140 ± 0.0009</td>
<td>0.0802 ± 0.0005</td>
<td>0.0219 ± 0.0024</td>
<td>2.20763</td>
</tr>
<tr>
<td>12</td>
<td>0.0935 ± 0.0004</td>
<td>5.8224 ± 0.1380</td>
<td>0.8252 ± 0.0021</td>
<td>3.84606</td>
</tr>
<tr>
<td>13</td>
<td>0.0563 ± 0.0004</td>
<td>0.4175 ± 0.0065</td>
<td>0.5897 ± 0.0048</td>
<td>1.87389</td>
</tr>
</tbody>
</table>

Summarising the results for session interarrival times during the seeding phase, we observe that all measurements pass according to the selected error criteria. Furthermore, it is observed that measurements 2 and 3 have low $E_{\%}$ values, and they show significance levels of $\approx 0.005$ when using the Anderson-Darling (AD) test. This is an indication for good quality of fitting for the selected distributions. Figure 7.1 shows the EPDF and CCDF for measurement 3 together with the fitted parameters overlaid.

The appearance of a hyper-exponential model for session interarrival times is
interesting, though not very surprising as Paxson and Floyd showed that network user session interarrival times are exponentially distributed [98]. A BT session arrival process is in effect filtered through the tracker, since a new peer first needs to contact the tracker to obtain a subset of the total number of peers in the swarm. This provides one hint as to why the hyper-exponential model fits. An additional reason could be that the arrival rates vary slightly with time. As the model applies to several measurements with different content type, size, and popularity, we expect this model to apply to BT in general, regardless of content characteristics.

### Session duration and size

In this section, we report the modelling results for the size and duration of remotely initiated peer sessions. For reasons similar to those considered for session interarrival times, we consider for modelling the following:

- Measurements with more than 20,000 sessions.
• Sessions initiated after the start of the seeding phase.

• Sessions that request and receive at least one piece.

The models for session sizes and durations are reported in Tables 7.4 and 7.5, respectively. Only the sessions that receive data have been modelled. Log-normal distributions with parameters \(\mu\) and \(\sigma_{LN}\) have been used for modelling.

The second to fifth columns show the estimated parameters together with the associated estimated standard deviations, for which the best value of \(E_\%\) was obtained. The value of \(E_\%\) is given in column 8. The sixth column indicates the tail probability mass for which the fitting passed the 5\% fitness limit of \(E_\%\), while the seventh column shows the tail probability mass for which the best value was obtained.

Since the number of samples is substantially smaller than for the hyper-exponential models shown in section 7.3.1, we also calculate the AD statistic for the fitted distribution. The last column shows the significance levels obtained in the AD test under the assumption that the parameter estimates are good enough to assume a fully specified distribution.

Table 7.4: Session upstream size parameters for study 1.

<table>
<thead>
<tr>
<th>#</th>
<th>(\hat{\mu})</th>
<th>(\hat{\sigma})</th>
<th>(\hat{\sigma}_{LN})</th>
<th>(\hat{\sigma})</th>
<th>Pass</th>
<th>Tail mass</th>
<th>(E_%)</th>
<th>AD sign.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18.7</td>
<td>0.04</td>
<td>0.62</td>
<td>0.02</td>
<td>0.45</td>
<td>0.21</td>
<td>2.1</td>
<td>&gt; 0.25</td>
</tr>
<tr>
<td>2</td>
<td>17.8</td>
<td>0.04</td>
<td>0.99</td>
<td>0.03</td>
<td>1</td>
<td>0.4</td>
<td>2.9</td>
<td>&gt; 0.025</td>
</tr>
<tr>
<td>3</td>
<td>18.4</td>
<td>0.04</td>
<td>0.60</td>
<td>0.02</td>
<td>1</td>
<td>0.24</td>
<td>3.3</td>
<td>&gt; 0.05</td>
</tr>
<tr>
<td>11</td>
<td>14.1</td>
<td>0.06</td>
<td>2.44</td>
<td>0.04</td>
<td>1</td>
<td>0.99</td>
<td>2.4</td>
<td>(\approx) 0.001</td>
</tr>
<tr>
<td>12</td>
<td>13.6</td>
<td>0.05</td>
<td>2.36</td>
<td>0.04</td>
<td>0.86</td>
<td>0.74</td>
<td>3.4</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>13</td>
<td>19.0</td>
<td>0.03</td>
<td>0.69</td>
<td>0.02</td>
<td>1</td>
<td>0.17</td>
<td>3.0</td>
<td>&gt; 0.025</td>
</tr>
</tbody>
</table>

The models obtained for session sizes and durations largely reflect those reported by Paxson in [97] in that they are in general well-described by a log-
normal distribution. In [97], the distributions for the sizes of bulk transfers such as FTP, SMTP, and NNTP are shown to closely resemble a log-normal distribution. Since BT is a bulk transfer application, the similarity of results is not surprising. Since the models are slightly poorer than the corresponding interarrival time models, we believe that the session sizes and durations are more dependent on the content and swarm characteristics than on individual user behaviours.

### 7.3.2 Study 2

In this section, we report on the modelling of the same session characteristics as in Section 7.3.1, i.e., session interarrival times, sizes and durations. We present summary statistics of the sessions from which the modelling was made in Table 7.6. As with study 1, the minimum session duration for this study was 0s, while the minimum session size was 68 bytes. This is a 5 byte difference compared to study 1. As these small and short sessions do not contribute significantly to either the session durations or session sizes, we did not investigate this slight discrepancy further.
7.3. SESSION MODELS

Table 7.6: Session summary for study 2.

<table>
<thead>
<tr>
<th>#</th>
<th>Number of sessions</th>
<th>Session length (s)</th>
<th>Session size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean</td>
<td>Max</td>
</tr>
<tr>
<td>1</td>
<td>47090</td>
<td>136</td>
<td>84947</td>
</tr>
<tr>
<td>3</td>
<td>12273</td>
<td>141</td>
<td>37759</td>
</tr>
<tr>
<td>4</td>
<td>23474</td>
<td>185</td>
<td>59220</td>
</tr>
<tr>
<td>5</td>
<td>997</td>
<td>333</td>
<td>43510</td>
</tr>
<tr>
<td>6</td>
<td>1635</td>
<td>267</td>
<td>40519</td>
</tr>
<tr>
<td>7</td>
<td>26491</td>
<td>57</td>
<td>63041</td>
</tr>
<tr>
<td>8</td>
<td>7113</td>
<td>104</td>
<td>30839</td>
</tr>
</tbody>
</table>

**Session interarrival times**

The session interarrival times for study 2 were also modelled using a second-order hyper-exponential distribution. However, except for measurements 3 and 4, the modelling was done on censored versions of the session data. The censoring consisted of discarding the upper 1% percentile. Since large interarrival times do not add to the network load as shorter interarrival times do, we do not consider this to be an issue. Additionally, during measurement 8, the tracker was offline for 7 hours, during which time the interarrival times became uncharacteristically large. We therefore discarded sessions initiated during this time.

In Table 7.7, we report the results of the modelling procedure for session interarrival times. Again, as with study 1, the hyper-exponential distribution describes the measured data rather well. All the measurements pass according to our criteria from Table 7.1 with at least a grade of poor, but we also note that three of the measurements pass with a grade of excellent.
Table 7.7: Session interarrival parameters for study 2.

<table>
<thead>
<tr>
<th>#</th>
<th>$\hat{\lambda}<em>1 \pm \hat{\sigma}</em>{\lambda_1}$</th>
<th>$\hat{\lambda}<em>2 \pm \hat{\sigma}</em>{\lambda_2}$</th>
<th>$\hat{p} \pm \hat{\sigma}_p$</th>
<th>$E_{%}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2625 $\pm$ 0.0020</td>
<td>1.1953 $\pm$ 0.0735</td>
<td>0.8947 $\pm$ 0.0079</td>
<td>0.5759</td>
</tr>
<tr>
<td>3</td>
<td>0.0667 $\pm$ 0.0185</td>
<td>0.3104 $\pm$ 0.0031</td>
<td>0.0034 $\pm$ 0.0017</td>
<td>2.4940</td>
</tr>
<tr>
<td>4</td>
<td>0.2982 $\pm$ 0.0026</td>
<td>1.7022 $\pm$ 0.3389</td>
<td>0.9740 $\pm$ 0.0072</td>
<td>0.8918</td>
</tr>
<tr>
<td>5</td>
<td>0.0102 $\pm$ 0.0006</td>
<td>7.8385 $\pm$ 0.4598</td>
<td>0.5150 $\pm$ 0.0198</td>
<td>3.1503</td>
</tr>
<tr>
<td>6</td>
<td>0.0082 $\pm$ 0.0003</td>
<td>6.5876 $\pm$ 0.3176</td>
<td>0.4855 $\pm$ 0.0151</td>
<td>4.8206</td>
</tr>
<tr>
<td>7</td>
<td>0.1792 $\pm$ 0.0015</td>
<td>2.0132 $\pm$ 0.5672</td>
<td>0.9666 $\pm$ 0.0071</td>
<td>0.2994</td>
</tr>
<tr>
<td>8</td>
<td>0.0233 $\pm$ 0.0004</td>
<td>10.9804 $\pm$ 0.9083</td>
<td>0.8872 $\pm$ 0.0058</td>
<td>4.7376</td>
</tr>
</tbody>
</table>

Session duration and size

As with study 1, only sessions sending or receiving at least one piece are modelled. Due to this censoring, measurements 5 and 6 were discarded as they only contained 5 and 12 sessions respectively, which was not enough to properly model them.

However, in contrast with study 1, the session durations of the second study did not reveal any clear correspondence with a single distribution for the session duration characteristic, and mixture models were therefore used to model it. The mixture models consisted of using a log-normal distribution for the body and a Pareto distribution for the tail. To find the optimum cutoff point between the two distributions, we employed minimisation of a weighted error percentage defined as:

$$E_W = P_c E_{\text{body}} + (1 - P_c) E_{\text{tail}},$$

where $P_c$ is the cutoff point probability value, and

$$E_{\text{body}} = E | \hat{U}_i = \hat{F}(X_i \leq c; \hat{\Theta}),$$

$$E_{\text{tail}} = E | \hat{U}_i = \hat{F}(X_i > c; \hat{\Theta}),$$

where $\hat{F}(\cdot)$ represents the probability distribution function, with the estimated parameters $\hat{\Theta}$, the measured samples $X_i$ while $c$ is the cutoff point value.
7.3. SESSION MODELS

Table 7.8: Fitted censored mixture log-normal and Pareto parameters for session duration for study 2.

<table>
<thead>
<tr>
<th>#</th>
<th>Log-normal</th>
<th>Cutoff Point</th>
<th>Generalised Pareto</th>
<th>$E_W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\hat{\mu} \pm \hat{\sigma}_\mu$</td>
<td>$\hat{\sigma} \pm \hat{\sigma}_\sigma$</td>
<td>$\hat{\alpha} \pm \hat{\sigma}_\alpha$</td>
<td>$\hat{\beta} \pm \hat{\sigma}_\beta$</td>
</tr>
<tr>
<td>1</td>
<td>6.26±0.01</td>
<td>0.35±0.02</td>
<td>0.80±0.02</td>
<td>0.71±0.01</td>
</tr>
<tr>
<td>2</td>
<td>6.16±0.01</td>
<td>0.33±0.01</td>
<td>0.74±0.01</td>
<td>0.62±0.15</td>
</tr>
<tr>
<td>3</td>
<td>6.24±0.01</td>
<td>0.36±0.02</td>
<td>0.85±0.01</td>
<td>0.55±0.14</td>
</tr>
<tr>
<td>4</td>
<td>6.82±0.01</td>
<td>1.37±0.02</td>
<td>-</td>
<td>0.47±0.07</td>
</tr>
<tr>
<td>5</td>
<td>3.82±0.01</td>
<td>1.09±0.01</td>
<td>0.73±0.02</td>
<td>0.37±0.06</td>
</tr>
</tbody>
</table>

The results of the modelling using these mixture models are presented in Table 7.8. Again, we note that all the measurements pass for $E_W$. However, the log-normal fit of the distribution body fails for measurement 8, while it passes for the corresponding mixture model. Also, for measurement 7, both a single log-normal as well as a single Pareto distribution pass according to the fitness criteria. The models provided in Table 7.8 for this measurement represent these single distribution parameters, and the $E_W$ values represent the errors for the single distributions.

In contrast with the session durations, the session sizes of the second study were not well-modelled by a mixture model. However, both the log-normal and the Pareto distributions fit the data, though not very well in either case. The results of the fitting procedure are reported in Table 7.9

Table 7.9: Fitted single log-normal and Pareto parameters for session upstream size for study 2.

<table>
<thead>
<tr>
<th>#</th>
<th>Log-normal</th>
<th>Generalised Pareto</th>
<th>$E_W$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\hat{\mu} \pm \hat{\sigma}_\mu$</td>
<td>$\hat{\sigma} \pm \hat{\sigma}_\sigma$</td>
<td>$\hat{\alpha} \pm \hat{\sigma}_\alpha$</td>
</tr>
<tr>
<td>1</td>
<td>12.61±0.03</td>
<td>1.44±0.02</td>
<td>3.11</td>
</tr>
<tr>
<td>2</td>
<td>13.25±0.08</td>
<td>1.93±0.06</td>
<td>2.33</td>
</tr>
<tr>
<td>3</td>
<td>12.93±0.06</td>
<td>2.04±0.04</td>
<td>4.39</td>
</tr>
<tr>
<td>4</td>
<td>13.89±0.11</td>
<td>2.43±0.08</td>
<td>3.37</td>
</tr>
<tr>
<td>5</td>
<td>13.52±0.13</td>
<td>2.19±0.09</td>
<td>5.66</td>
</tr>
</tbody>
</table>

97
CHAPTER 7. BITTORRENT TRAFFIC MODELS

7.4 Message Models

7.4.1 Upstream Leech Phase Request Rate for Study 1

The request-messages and their responses are the major bandwidth contributors in a BitTorrent session. It is therefore important to model the request behaviour of an individual peer. The models we present in this section are obtained from the first study.

A subset of the data was used for modelling the upstream leech phase request rates to ensure that the end-game mode and startup-phase of the peer do not skew the model. The following censoring procedure was used to locate the steady-state portion of the request rate time series. The upper limit is set at 99% of the time series, thus excluding the end-game mode requests [46]. To locate the lower limit, we calculate a smoothed version of the time series using a Lowess smoother [29]. The lower limit is then given by the location of the first minimum of the smoothed time series.

The resolution used for modelling the message rates is one second. Also, the number of back-to-back messages has not been modelled, i.e., messages with interdeparture times of 0 s are excluded from the models.

Table 7.10 reports the results of the request rate modelling. In certain cases, such as for measurement 11 (Figure 7.2(b)), the bi-modality of the dual Gaussian distribution is more pronounced than for the other measurements. The modes correspond to a changing number of outgoing requests due to an increase or decrease in the number of connected peers. The reason that standard deviations of the estimated parameters are missing from Table 7.10 is that for most of the parameters, the optimisation software failed to supply these values. We therefore opted to not include any of them at all.
Table 7.10: Request rate modelling results. The model for measurement 13a refers to the application layer traces, while 13b refers to the model obtained from Ethernet traces.

<table>
<thead>
<tr>
<th>#</th>
<th>$\mu_1$</th>
<th>$\sigma_1$</th>
<th>$\mu_2$</th>
<th>$\sigma_2$</th>
<th>$p$</th>
<th>$% E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>87.3</td>
<td>27.8</td>
<td>88.4</td>
<td>7.51</td>
<td>0.16</td>
<td>0.59%</td>
</tr>
<tr>
<td>2</td>
<td>76.7</td>
<td>25.6</td>
<td>78.5</td>
<td>8.24</td>
<td>0.28</td>
<td>0.44%</td>
</tr>
<tr>
<td>3</td>
<td>95.2</td>
<td>36.2</td>
<td>112</td>
<td>13.9</td>
<td>0.35</td>
<td>0.88%</td>
</tr>
<tr>
<td>4</td>
<td>16.0</td>
<td>3.45</td>
<td>20.6</td>
<td>4.49</td>
<td>0.68</td>
<td>0.44%</td>
</tr>
<tr>
<td>5</td>
<td>13.2</td>
<td>2.88</td>
<td>17.7</td>
<td>4.43</td>
<td>0.74</td>
<td>0.43%</td>
</tr>
<tr>
<td>6</td>
<td>9.71</td>
<td>2.73</td>
<td>14.8</td>
<td>4.12</td>
<td>0.35</td>
<td>0.45%</td>
</tr>
<tr>
<td>7</td>
<td>34.1</td>
<td>16.5</td>
<td>43.7</td>
<td>10.4</td>
<td>0.39</td>
<td>1.58%</td>
</tr>
<tr>
<td>8</td>
<td>7.78</td>
<td>2.51</td>
<td>20.0</td>
<td>11.2</td>
<td>0.93</td>
<td>0.84%</td>
</tr>
<tr>
<td>10</td>
<td>10.0</td>
<td>3.69</td>
<td>22.9</td>
<td>8.07</td>
<td>0.84</td>
<td>0.81%</td>
</tr>
<tr>
<td>11</td>
<td>6.58</td>
<td>2.28</td>
<td>17.9</td>
<td>6.89</td>
<td>0.26</td>
<td>0.31%</td>
</tr>
<tr>
<td>12</td>
<td>11.1</td>
<td>3.81</td>
<td>29.1</td>
<td>8.85</td>
<td>0.68</td>
<td>0.80%</td>
</tr>
<tr>
<td>13a</td>
<td>48.2</td>
<td>12.4</td>
<td>53.8</td>
<td>5.83</td>
<td>0.38</td>
<td>0.57%</td>
</tr>
<tr>
<td>13b</td>
<td>33.8</td>
<td>8.03</td>
<td>38.0</td>
<td>4.68</td>
<td>0.47</td>
<td>0.36%</td>
</tr>
</tbody>
</table>

Figure 7.2: Request rate CCDF and EPDF for measurement 11.
7.5 Summary

In this chapter, we have presented a set of BT models in the form of probability distributions fitted to two sets of measurement data collected at two different locations with 16 months between measurements.

We have identified the following characteristics:

- Session interarrival times are well-modeled by a binary hyper-exponential distribution in both measurement sets. We note that other studies have also indicated this characteristic [38].

- Session durations are modelled by a log-normal distribution for the first measurement set and a mixture of the log-normal and Pareto distributions for the second set. For the second set of measurements, the body is typically well-modelled by the log-normal distribution, while the tail is more accurately characterised by the Pareto distribution.

- Session sizes are modelled by the log-normal distribution for both measurement sets. However, for the second set, the Pareto distribution also fits the measurement data acceptably well.

- Upstream request rates are well-modelled by a dual Gaussian distribution for both measurement sets.
Chapter 8

Experiments

One of the main contributions of this thesis is a set of extensions suggested to the BT core algorithms. We evaluate these extensions by simulation for several reasons. It would certainly be possible to use real BT clients, either by modifying already existing ones, using existing libraries, or by implementing a new one from scratch. While using the reference mainline client would relieve us of the need to validate the functionality of the rarest-first piece selection algorithm, using this approach would suffer from the same problems as would using any real client. For instance, the time to run the experiments would be significantly longer than in a simulated environment for the same scenario parameters, i.e., 1000 nodes and 40 repetitions of each experiment. Also, setting up a controlled environment of this size would be a challenge.

If deployed in a real live network connected to the Internet, controlling the experiment would be even more difficult, if not impossible. Furthermore, as the algorithms we want to test may result in adverse behaviour with respect to both the specific BT swarm and the Internet itself, great care must be taken to not disrupt either. For these reasons, we have opted to use simulations to test our algorithms before attempting to implement and deploy them in a laboratory environment or live network.
8.1 Simulator Implementation and Validation

8.1.1 Simulator Implementation

The simulator used for the experiments presented in this thesis was designed and implemented by the ROVER team at the department of Telecommunication Systems at BTH. It is written in C++ using the OMNeT++ simulation framework [125].

The BT simulator is made up of four modules: TorrentFile, ClientAssigner, BTClient, and BTTracker. The TorrentFile module is responsible for loading and parsing a real torrent file, making the file and piece size available to the other modules. The ClientAssigner assigns the initial states (leecher or seeder, start time and seeding time) to each BT client. The remaining two modules, BTClient and BTTracker, represent the corresponding BT entities.

The implementation of the BT protocol is based on the mainline BT client, version 4.0.2, as described in [83]. A more detailed description of the simulator is available in [129].

8.1.2 Simulator Validation

To validate the functionality of the simulator implementation of the original BT algorithms, we have run a number of validation simulations. We then compare the results from these simulations with the measurement results presented in Chapter 7. We focus on three main parameters for this validation: per-peer session interarrival times, session duration, and per-peer upstream request rates, as we have previously obtained well-fitting models for these parameters in the measurement studies presented in Chapter 7. We think that these parameters can adequately capture BT swarm behaviour.
Simulation parameters

Many of the parameters of a BT swarm have not been previously measured. This forces us to make a few assumptions regarding the input distributions of these parameters. Nevertheless, some of these assumptions have also been used in previous studies, such as the exponential seeding time used in the work conducted by Qiu and Srikant [104]. The parameters used for the validation simulation scenario are documented in Table 8.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeds</td>
<td>1</td>
</tr>
<tr>
<td>Peers</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum number of connections per peer</td>
<td>80</td>
</tr>
<tr>
<td>Number of peers requested from tracker</td>
<td>50</td>
</tr>
<tr>
<td>Swarm interarrival time</td>
<td>exponential, $\mu = 12.395\text{ s}$</td>
</tr>
<tr>
<td>Link delay</td>
<td>Uniform, [50, 400] ms</td>
</tr>
<tr>
<td>Link bandwidth</td>
<td>Uniform, [0.2, 10] Mbps</td>
</tr>
<tr>
<td>Asymmetric links</td>
<td>yes</td>
</tr>
<tr>
<td>Initial piece distribution</td>
<td>Uniform, [0, 50] %</td>
</tr>
<tr>
<td>Request waiting time</td>
<td>exponential, $\mu = 100\text{ ms}$</td>
</tr>
<tr>
<td>Block size</td>
<td>$2^{15}$ (32768) bytes</td>
</tr>
<tr>
<td>Simulation runs</td>
<td>40</td>
</tr>
</tbody>
</table>

The reason for selecting a single seed is primarily 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.

The maximum number of connections per peer and the number of peers requested from the tracker were both selected to be the same as in the default
client. We should note that the maximum number of connections includes the tracker connection, thus making the maximum number of peer connections 79.

The parameter value for the mean swarm interarrival time was selected at random from the results reported in Chapter 7.1. The value selected, $\mu = 12.395$ s, is one of the component parameters for measurement 13.

The distribution parameters for link delay and bandwidth were selected to cover a large spread of possible link types, as was the asymmetry of the links. We acknowledge the fact that the link delay and bandwidth values used in the simulation are not typical of an Internet-like topology. However, while we have opted for a rather simplified topology model, using it, we observe both session and message dynamics that correspond to real-world measurements. We report these results in the following sections.

The initial piece distribution of each peer was chosen to be uniformly distributed, as no models for this were known at the time. The parameter represents the ratio of pieces available at a joining peer when it enters the swarm (the initial seed always has all pieces.) Which specific pieces are available is selected randomly. This was done to reflect the fact that the measurements reported in Chapter 7 were performed on swarms that already contained active peers already in possession of pieces of the content.

Due to various factors, such as scheduling of requests of a peer, snubbing, bandwidth limiting, and other delay-generating processes, an exponential response delay with mean 100 ms was added to account for these factors.

The block size of 32768 bytes was selected, as it is the default size recommended by the BT specification [30].

In addition to the simulation parameters themselves, the simulator uses an associated torrent file for the meta-data required to join a swarm. As the simulation does not take into account any information below the application layer, such as host names, IP addresses, or port numbers, only the meta-data in Table 8.2 is used.
8.1. SIMULATOR IMPLEMENTATION AND VALIDATION

Table 8.2: Meta-data for simulator validation scenario.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of pieces</td>
<td>1205</td>
</tr>
<tr>
<td>Piece size</td>
<td>62144 bytes</td>
</tr>
<tr>
<td>Content size</td>
<td>74825472 bytes</td>
</tr>
</tbody>
</table>

Session interarrival Times

In both studies presented in Section 7.3, session interarrival times measured at a single peer were characterised as 2nd order hyper-exponential distributions (Tables 7.3 and 7.7 on pages 91 and 96).

For the validation simulation, we characterised the session interarrival times at the initial seed to get the maximum number of samples for fitting a distribution. We fitted both a single exponential distribution as well as a hyper-exponential distribution. Table 8.3 shows the results of the fitting procedure for both distributions.

Table 8.3: Fitted parameters and 95% confidence intervals for validation simulation run session interarrival times for initial seed.

<table>
<thead>
<tr>
<th></th>
<th>Exponential</th>
<th>Hyper-exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>$0.081 \pm 0.001$</td>
<td>$0.069 \pm 0.005$</td>
</tr>
<tr>
<td>$E_{95}$</td>
<td>$1.000 \pm 0.102$</td>
<td>$0.173 \pm 0.073$</td>
</tr>
<tr>
<td>$\lambda_1$</td>
<td></td>
<td>$0.381 \pm 0.085$</td>
</tr>
<tr>
<td>$\lambda_2$</td>
<td></td>
<td>$0.895 \pm 0.094$</td>
</tr>
<tr>
<td>$p$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It should be noted that the data from one of the 40 simulation runs failed the MLE estimation procedure for the hyper-exponential fit and was thus removed from the data set. We note that the hyper-exponential distribution is, on the average, a better match for the simulated data than the single exponential. This is in accordance with the results presented in Chapter 7.
CHAPTER 8. EXPERIMENTS

Session Duration

While the studies reported in Section 7.3 did not conclusively characterise session durations as being equidistributed, both studies did indicate that a log-normal distribution is an acceptable choice for modelling BT session durations. The second study indicated that the log-normal distribution tends to underestimate the tail of the distribution, as reported in Table 7.8.

For the validation simulation scenario, we measured the duration of all the sessions in each simulation run and fitted a log-normal distribution to these sessions. The resulting fitted parameters were then averaged over the 40 simulation runs to produce the results reported in Table 8.4. The values in Table 8.4 are thus the average fitted mean ($\bar{\mu}_{LN}$), variance ($\bar{\sigma}_{LN}$), and error percentage, together with the associated 95% confidence intervals.

Table 8.4: Log-normal parameters and 95% confidence intervals for simulated session duration.

<table>
<thead>
<tr>
<th>$\bar{\mu}_{LN}$</th>
<th>$\bar{\sigma}_{LN}$</th>
<th>$E%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4037 ± 0.0033</td>
<td>0.6076 ± 0.0035</td>
<td>1.5986 ± 0.1076</td>
</tr>
</tbody>
</table>

We note that the level of fitness is almost on par with the fitness of the session interarrival times. We consider this good enough to accept the simulator as validated with regard to session duration.

Leech state upstream request rates

The characterisation of upstream request rates during the leech phase of a single BT peer presented in Section 7.4.1 (page 98) indicates that the request rates can be well-modelled by a dual Gaussian distribution.

The validation simulation data was aggregated to count the number of requests per second and censored to discard the first and last 5% of the aggregated data. The first part was censored to ascertain that the client had connected to enough peers, while the second part was discarded to avoid the effects of the end-game
8.1. SIMULATOR IMPLEMENTATION AND VALIDATION

Request rates were calculated on a per-peer basis and aggregated on a time scale of one second. A dual Gaussian finite mixture distribution was fitted to each session, and an error percentage was calculated for each fit. These error percentages were then averaged, and confidence intervals were calculated.

![Upstream leech request rates](image)

**Figure 8.1:** Error percentages and 95% confidence intervals for dual Gaussian fit of the average upstream request rates.

Figure 8.1 shows the error percentages and associated 95% confidence intervals for all 40 simulation runs. We observe that the error percentages are clustered between 3.4% and 3.7%. While these error percentages are rather high, they are still within a passing degree. We attribute these errors to the fact that most sessions request and download all pieces rapidly. This means that the aggregation of the number of pieces for calculating the rates results in a low number of samples. The average number of samples was 75, and the maximum was 247. With this in mind, we consider the simulator to represent the request behaviour of a real-life BT peer in a satisfactory way with reference to the results.
reported in Section 7.4.1.

**Piece Distribution**

One important aspect of the original piece selection scheme is that in steady state, pieces are uniformly distributed in the swarm, \textit{i.e.}, every piece is equally popular. To verify this functionality of the simulator, we count the total number of requests for each piece during an entire simulation run and then fit a uniform distribution to these data. The error percentages of these fitted distributions are then averaged, and confidence intervals are calculated.

| $E_{\%}$ | $0.0591 \pm 0.0042$ |

Table 8.5 reports the average error percentage and associated 95\% confidence intervals for the piece distribution of the validation simulation scenario. The fitness level is very high, and we consider the piece selection scheme of the simulator to adequately represent the original BT scheme.

### 8.2 Experiment Design

#### 8.2.1 Assumptions

In addition to the simulation input parameters, we make further assumptions regarding the simulated swarms, as detailed below.

- At least one seed is always present. More may be available, but in the discussion below “initial seed” refers to one or more seeds deployed by the media distributor.
8.2. EXPERIMENT DESIGN

- Peers tend to leave rather soon as they have stopped consuming a video object, \textit{i.e.}, there is little or no altruism. As in the validation simulation scenario, we therefore assume exponential seeding times. In the streaming case, this means that a peer stays online for an exponentially distributed time after having viewed the entire stream.

- All pieces are received without error, \textit{i.e.}, retransmissions are not considered at the transport, nor at the application layer.

- Peers start out without any data. In a conventional BT file-sharing scenario, stopping and restarting a download is a rather common occurrence due to network outages, dropped connections, and other similar issues. In a streaming scenario, it is unlikely that a joining peer has previously downloaded a random subset of the piece set.

8.2.2 Evaluation Parameters

The experiments we perform 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, in our case a streaming BT VoD solution. 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, server load, and a continuity index. A description of these metrics is as follows:

**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 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 $p_r$. However, this metric is primarily useful for clients implementing
the BT protocol and not for evaluating the efficiency of a specific piece selection strategy, in particular not a streaming one. A more useful metric would be using the average number of copies of each piece and their variances together with $p_r$, e.g., the tuple $(p_r, m, \sigma^2)$.

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 AD or Cramér-von Mises statistic. For our work, we opt for a visual measure and use distribution graphs to assess the quality of the piece distribution.

**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. In [85], the authors denote pieces that are only available at the initial seed as rare pieces. 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, 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.

**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 is a system-centric metric, 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.

### 8.2.3 Simulation Parameters

In this section, we discuss the various parameter values and distributions used in our experiments. Several of the parameters in the experiment simulations are the same as the simulation parameters for the validation run described in Table 8.1 (page 103). Therefore, in Table 8.6 we only present the parameters that differ from Table 8.1.

We have performed four sets of simulations to study the effects of various parameter changes. The parameters reported in Table 8.6 are the ones used for set 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of connections per peer</td>
<td>10</td>
</tr>
<tr>
<td>Number of peers requested from tracker</td>
<td>10</td>
</tr>
<tr>
<td>Link delay</td>
<td>Uniform, [5, 400] ms</td>
</tr>
<tr>
<td>Link bandwidth</td>
<td>Uniform, [2, 10] Mbps</td>
</tr>
<tr>
<td>Asymmetric links</td>
<td>no</td>
</tr>
<tr>
<td>Initial piece distribution</td>
<td>0 %</td>
</tr>
<tr>
<td>Piece inter-deadline time</td>
<td>Truncated Gaussian, ( \mu = 0.3 \text{ s}, \sigma = 0.5 )</td>
</tr>
<tr>
<td>Playback buffer size</td>
<td>10 pieces</td>
</tr>
</tbody>
</table>

The number of connections and maximum number of peers requested from the tracker are decreased compared to the validation scenario. Also, since the largest connection bandwidth for the experiment scenarios is 10 Mbps, the total bandwidth (up- and downstream) for any given peer is 100 Mbps. Again,
one connection per peer is reserved for a connection to the tracker. Also, 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 more typical video broadcasting behaviour. Typically, a video is not likely to be partially prefetched in advance. Another important difference from the validation scenario is that a playback buffer of 10 pieces is employed. 10 pieces is equivalent to 621,440 bytes, or, on the average, 3 s of a video stream as represented in the following subsection.

The remaining simulation sets use the same parameters as in Table 8.6, but with the following changes:

**Set 2:** The maximum number of connections per peer is increased to 80. This was done to observe the effect of having a larger set of peers available to the choking algorithm.

**Set 3:** The playback buffer is increased to 20 pieces. This was primarily done in order to observe the effects on the piece distribution of a larger playback buffer.

**Set 4:** The maximum link delay is decreased to 100 ms, and the mean of the exponential request waiting time is decreased to 10 ms. This was done to observe the effect on the CI metric.

**Piece Interdeadline Time**

The piece interdeadline time is the representation of a video stream in the simulations. Modelling compressed VBR video streams is a highly complex process, as a salient characteristic of them is that their bandwidth requirements are highly correlated and bursty. Additionally, many compression schemes exploit this, further adding to the complexity of the traffic patterns generated when streaming the objects. [80, 86].

Another factor that complicates the issue in the specific case of using BT as the transport is that the interdeadline times now refer to pieces instead of frames. This means that any model designed for frame sizes and deadlines would
8.3. EXPERIMENT RESULTS

have to be adapted for use with piece deadlines. One of the initial requirements
of the simulator was to be able to parse actual torrent files, and, as mentioned in
Section 6.4.1, one of the extensions we suggest making to the BT system is adding
piece deadlines to the torrent file. Once such deadlines are in place, the simulator
is easily extended to take these deadlines into account. As this functionality is
not yet implemented, we opted to use a simple truncated Gaussian model for
the piece deadlines. We are aware of this shortcoming in our input model, and
it is highly prioritised in our future work to implement a video stream analyser
to provide empirical models for various video codecs.

8.3 Experiment Results

We have configured six simulation scenarios according to the parameters in
Table 8.6 and Table 8.1, only changing the piece selection algorithm used in
each scenario. The following scenarios have been configured and simulated:
rarest-first, linear, hybrid rarest-first/smoothing with probabilities 0.1 and 0.5,
pure smoothing, and the hybrid rarest-first/deadline-based algorithm. We refer
to these scenarios as the linear, rarest, prob-0.1, prob-0.5, prob-1.0, and deadline,
respectively.

8.3.1 Piece Distribution

One key aspect of employing BT for streaming video is that both BEST and
non-BEST peers should be able to collaborate in a swarm without adversely
affecting each other. For BEST peers, this primarily means that they should still
contribute positively to the swarm piece distribution, i.e., increase the number
of rare pieces. In this section, we report on the piece distribution results for our
simulation scenarios.

Figure 8.2 shows the average number of replication per piece for all of the
40 simulation runs. The data for the graph was obtained as follows. For each
run and piece, we count the number of available replicas each time a seed leaves
the swarm, thus yielding a series of 1000 values for each run. It also means
that the samples are not equidistant, since each peer has a different download and seeding time. We then average the number of replicas for each piece. We also calculate 95% confidence intervals, but omit these from Figure 8.2 as the graph then becomes very difficult to read. We provide graphs with corresponding confidence intervals of all scenarios and sets in Appendix A.

We observe in Figure 8.2 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 \textit{rarest}, \textit{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 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. In the case of the scenarios where only the playback buffer is downloaded sequentially, pieces outside of the playback buffer are downloaded randomly, and pieces closer to the playback buffer are not necessarily replicated more often.

In Figure 8.3, we illustrate the effect of the linear selection. The light grey pieces are selected using the playback buffer, and the dark grey pieces are selected using the currently chosen selection algorithm. Assume further that peer A has a higher download rate and thus finishes its download prior to peer B. Peer B starts its download after peer A. Using linear selection, pieces 1−7 are replicated twice prior to peer A leaving the swarm after downloading piece 10. Using one of the other piece selection schemes, the set of pieces replicated twice is disjoint, consisting of pieces \{1 − 4, 7, 9, 10\}.

The reason that the replication of pieces decreases with increasing piece number is that once the playback buffer is full, pieces are downloaded from the remaining pieces using a different algorithm. Once the playback buffer filling algorithm notices that a piece is already downloaded, it is immediately placed
in the playback buffer, and a new piece is selected using the currently chosen selection algorithm.

Including the linear scenario in Figure 8.2 makes the remaining scenarios difficult to observe. We therefore repeat the graph in Figure 8.4, but without the linear scenario. The main conclusion that can be made is that all scenarios and the corresponding algorithms, except the pure smoothing (\textit{prob-1.0}) algorithm, yield very similar results. Another observation is that the variance of the piece replication is slightly higher for the \textit{prob-1.0} scenario. Again, we mention the fact that the pure smoothing algorithm is the only one not employing the rarest-first algorithm at all, and we attribute the difference in appearance to this. In effect, pieces in the \textit{prob-1.0} scenario are selected according to a sorted list of numbers drawn from a truncated Gaussian distribution, as given in Table 8.6 (page 111).

Another salient feature observed in both Figure 8.2 and Figure 8.4 is the fact that the first piece is underrepresented compared to the following few pieces. We believe that this is related to the fact that the linear mode is not used, as the same feature does not appear in the linear scenarios.

Further, in Figures 8.4(a)–(b) we also observe that increasing the size of the playback buffer (set 3) does not increase the average replication of a piece. However, having a larger amount of peers to choose from (set 2) does result in an increase, as does shorter transmission and response delays. Hence, combining a large number of peers with low delay will likely increase the average replication per piece. A typical scenario would be a metropolitan-sized or smaller network with a large number of participating peers.

The erratic behaviour of the smoothing algorithm is further illustrated in Figure 8.5. By comparing Figures 8.5(b)–8.5(e), we observe an increase in noise (indicated by less smooth vertical lines) as the algorithm behaves more as smoothing and less as rarest-first.

Another interesting observation is the similarity of Figures 8.5(d) and 8.5(f). As pieces in the deadline scenario are selected alternatingly using the rarest-first algorithm and a variant of the smoothing algorithm, this is not very surprising.

Similar behaviour can be observed in simulation sets 2–4 (Figures B.1–B.3
8.3. EXPERIMENT RESULTS

![Graphs showing average piece distribution per piece and scenario without linear algorithm.](image)

Figure 8.4: Average piece distribution per piece and scenario without linear algorithm.

In Appendix B, though less pronounced in the levelplot for set 4.

Summarising the results for piece distribution, we draw the conclusion that all the algorithms that use the *rarest-first* algorithm perform adequately. Though the piece distribution is not uniform, the fact that the first 25% or so of the piece set is overrepresented in the swarm is not a problem. As these pieces are needed first when streaming video, it is actually beneficial to have these slightly overrepresented in the swarm. If the swarm consists of both BEST and non-BEST peers, the impact of the skewed piece distribution will be decreased as well.
CHAPTER 8. EXPERIMENTS

Figure 8.5: Piece distributions for all algorithms and all runs, set 1.
8.3. EXPERIMENT RESULTS

8.3.2 Server Load

We next turn our attention to the server load metric. Recall that we define the server load as the number of pieces sent by the initial seed, divided by the total number of pieces in the piece set. For reference, we also point out that for a conventional, non-BT streaming service with the same number of nodes as our simulation scenarios, the corresponding server load would be 999.

We calculate the server load by counting the number of pieces sent by the initial seed for each run. We divide this number by the number of pieces in the piece set, which yields the server load for each run, i.e., 40 samples per scenario. We then calculate the average server load and 95% confidence intervals for each scenario.

We present the results for the server load metric in Figure 8.6 and report the numerical values in Table 8.7. 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 the conventional non-cooperative streaming solution by using only 2.08% of the bandwidth for simulation set 1.

Table 8.7: Server load results.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>20.83 ± 0.45</td>
<td>21.02 ± 0.45</td>
<td>20.25 ± 0.42</td>
<td>19.11 ± 0.49</td>
</tr>
<tr>
<td>rarest-first</td>
<td>16.99 ± 0.52</td>
<td>18.42 ± 0.39</td>
<td>16.57 ± 0.53</td>
<td>17.11 ± 0.50</td>
</tr>
<tr>
<td>prob 0.1</td>
<td>17.17 ± 0.45</td>
<td>18.86 ± 0.37</td>
<td>16.67 ± 0.39</td>
<td>17.27 ± 0.48</td>
</tr>
<tr>
<td>prob 0.5</td>
<td>17.35 ± 0.47</td>
<td>19.63 ± 0.49</td>
<td>17.32 ± 0.55</td>
<td>17.42 ± 0.48</td>
</tr>
<tr>
<td>prob 1</td>
<td>19.38 ± 0.40</td>
<td>20.86 ± 0.44</td>
<td>19.43 ± 0.43</td>
<td>19.02 ± 0.57</td>
</tr>
<tr>
<td>deadline</td>
<td>17.40 ± 0.43</td>
<td>19.34 ± 0.38</td>
<td>17.32 ± 0.42</td>
<td>17.43 ± 0.43</td>
</tr>
</tbody>
</table>

Similar to the results for the piece distribution, we 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 a 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 \textit{prob-1.0} and \textit{linear} algorithms are similar since the pieces are actually picked deterministically. This also gives an explanation as to why the \textit{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.

![Server load results.](image1)

(a) Set 1.

![Server load results.](image2)

(b) Set 2.

![Server load results.](image3)

(c) Set 3.

![Server load results.](image4)

(d) Set 4.

\textbf{Figure 8.6: Server load results.}

In all four simulation sets, the \textit{deadline} and \textit{prob-0.5} algorithms have very similar performance. We attribute this to the same reason for the similarity of the corresponding piece distribution results, \textit{i.e.}, that in both cases, about 50\% of all pieces are picked using a smoothing algorithm. We interpret this as meaning
that, unless there are advantages not captured by our evaluation metrics, there is no benefit in using the deadline algorithm over the prob-0.5, except for in the case of a large number of available peers (set 2). Further, increasing the playback buffer size yields little or no improvement in all simulation sets.

Summarising the results for the server load, we conclude that the best performance is achieved using the rarest-first algorithm. This is an expected result, as this algorithm minimises the distribution of rare pieces, thus increasing the probability of a piece being available on peers other than the initial seed.

8.3.3 Continuity Index

The Continuity Index (CI) metric partially captures the quality of the consumption of a video object. It is defined as the ratio of pieces meeting their deadlines to the number of pieces in the piece set. For each scenario, we calculate the CI by counting each piece received on time for every peer and dividing by the total number of pieces, yielding 999 per-peer CI samples. For each run, we then average these values and calculate the 95% confidence intervals.

In Figure 8.7, we show a graph of the resulting per-run averages and confidence intervals. We point out that in our simulations, one missed deadline is equivalent to a CI of $1 - 1/1205 = 0.99917$, and two missed deadlines is equivalent to a CI of 0.99834. We also point out that missing a single piece is more disrupting than missing a single frame, as a piece is likely to carry more than one frame and a single frame may span the end of one piece and the beginning of the next. The solid lines in Figure 8.7 represent the average CI for all runs for the corresponding scenario. The values and 95% confidence intervals for these averages are reported in Table 8.8 (page 123).

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.

Another feature observed in Figure 8.7 is the results for the prob-1.0 scenario. First, the average CI for that scenario is the only one that represent missing more than one piece deadline, as can also be seen in Table 8.8. Second, what is
CHAPTER 8. EXPERIMENTS

not conveyed very well in the table, is the large confidence intervals evident on a per-run basis for the prob-1.0 scenario.

Figure 8.7: Continuity index results.

Again, as with the other evaluation metrics, we observe that increasing the playback buffer size (set 3) does not improve the CI significantly. The most obvious improvement is having access to peers with lower delay. However, having access to more peers (set 2) does not noticeably improve the performance and can, as in the case of the pure smoothing scenario, actually decrease performance considerably.
### Table 8.8: Continuity index results.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>0.999848 ± 0.00000021</td>
<td>0.999851 ± 0.00000018</td>
<td>0.999952 ± 0.00000008</td>
<td>1.000000 ± 0.00000010</td>
</tr>
<tr>
<td>rarest-first</td>
<td>0.999427 ± 0.00001289</td>
<td>0.999360 ± 0.00003500</td>
<td>0.999585 ± 0.00002900</td>
<td>0.999063 ± 0.00000550</td>
</tr>
<tr>
<td>prob 0.1</td>
<td>0.999724 ± 0.00001412</td>
<td>0.999725 ± 0.00003570</td>
<td>0.999960 ± 0.00003760</td>
<td>0.999971 ± 0.00000515</td>
</tr>
<tr>
<td>prob 0.5</td>
<td>0.999595 ± 0.00002111</td>
<td>0.999550 ± 0.00003741</td>
<td>0.999941 ± 0.00000311</td>
<td>0.999950 ± 0.00000519</td>
</tr>
<tr>
<td>prob 1</td>
<td>0.999015 ± 0.00002711</td>
<td>0.998310 ± 0.00007609</td>
<td>0.999860 ± 0.00000510</td>
<td>0.999894 ± 0.00000514</td>
</tr>
<tr>
<td>deadline</td>
<td>0.999572 ± 0.00002251</td>
<td>0.999568 ± 0.00002262</td>
<td>0.999958 ± 0.00000269</td>
<td>0.999965 ± 0.00000024</td>
</tr>
</tbody>
</table>
Summarising the results for the CI metric, we conclude that the main improvement is achieved by having access to peers with low delay. In this case, it matters little which piece selection algorithm is used. However, when delays are longer, choosing a proper piece selection algorithm becomes paramount. The best results are achieved by using the linear algorithm, but the probabilistic hybrid rarest-first algorithm also performs adequately.

8.4 Summary

In this chapter, we have presented a validation study of a BT simulator developed at BTH. Our results show that, for the characteristics we use for the validation, i.e., session interarrival times, session duration, upstream request rates, and piece distribution, the simulator yields results similar to the results reported in Chapter 7.

Further, we have also reported on a simulation study of the modifications described in Chapter 6 using this simulator. The main findings from this study are:

- The addition of a small buffer, which we call a playback buffer, close to the playout point, which downloads pieces linearly is the primary change needed to make BT a suitable system for providing a streaming service.

- The piece distribution performances of all modified algorithms are similar, except for linear piece selection, in the sense of the algorithms succeeding in behaving similar to a normal BT peer. The main difference is that our modifications increase the replication of low-numbered pieces, a phenomenon we attribute to the use of a playback-buffer.

- The original rarest-first algorithm outperforms our modifications with regard to the load on the initial seed.

- The performance of the CI, i.e., how many pieces arrive on time, is primarily related to the link delay (and the available bandwidth).
Chapter 9

Conclusions and Future Work

The questions we set out to explore, and hopefully answer, in Chapter 1 were: Is BitTorrent a suitable system to base a streaming service upon? If so, what extensions and modifications are needed?. The first part of this question is answered with an unequivocal “yes”, as both the actual deployment of the Azureus streaming service and our own results indicate. The bandwidth used at the content server is significantly decreased using the swarming functionality of BT.

Regarding the second part of the question, our answer is that the main change that needs to be made to the BT system is adding a linear selection of the pieces that are close to consumption. With this simple change, the original BT rarest-first algorithm can be used for the non-critical pieces without affecting the performance significantly. Also, as this change is performed client-side, no major infrastructural changes are needed.

There are a number of improvements and additions that would be useful to make to the simulator used to obtain the results reported in this thesis. Primarily, these improvements regard the models used for various input parameters used for the simulations. Two of the most important improvements would be using more realistic models for the piece deadlines and the topology. We consider it likely that the performance of smoothing algorithms would improve as the
burstiness of the input video stream increases.

Another improvement, not related to the input parameters of the simulator, would be implementing retransmissions due to failed downloads. This would, however, require modelling of the piece verification failure rate for different swarms and contents.

While the results reported in this thesis are certainly positive indications of the possibility of using BT as the transport mechanism for a streaming service, it is important to point out that to test the suggested BT modifications, a real implementation would be necessary.
Appendix A

Confidence Interval Graphs
Figure A.1: Piece distribution with 95% confidence intervals, set 1.
Figure A.2: Piece distribution with 95% confidence intervals, set 2.
APPENDIX A. CONFIDENCE INTERVAL GRAPHS

Figure A.3: Piece distribution with 95% confidence intervals, set 4.
Figure A.4: Piece distribution with 95% confidence intervals, set 4.
APPENDIX A. CONFIDENCE INTERVAL GRAPHS
Appendix B

Piece Distribution Levelplots
Figure B.1: Piece distributions for all algorithms and all runs, set 2.
Figure B.2: Piece distributions for all algorithms and all runs, set 3.
Figure B.3: Piece distributions for all algorithms and all runs, set 4.
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>3rd Generation Partnership Project</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>AD</td>
<td>Anderson-Darling</td>
</tr>
<tr>
<td>ALMI</td>
<td>Application Level Multicast Infrastructure</td>
</tr>
<tr>
<td>ALM</td>
<td>Application Layer Multicast</td>
</tr>
<tr>
<td>ASM</td>
<td>Any-Source Multicast</td>
</tr>
<tr>
<td>BASS</td>
<td>BitTorrent-Assisted Streaming System</td>
</tr>
<tr>
<td>BEST</td>
<td>BitTorrent Extensions for Streaming</td>
</tr>
<tr>
<td>BGMP</td>
<td>Border Gateway Multicast Protocol</td>
</tr>
<tr>
<td>BGP</td>
<td>Border Gateway Protocol</td>
</tr>
<tr>
<td>BT</td>
<td>BitTorrent</td>
</tr>
<tr>
<td>BiToS</td>
<td>BitTorrent Streaming</td>
</tr>
<tr>
<td>BTH</td>
<td>Blekinge Institute of Technology</td>
</tr>
<tr>
<td>CBR</td>
<td>Constant Bit-Rate</td>
</tr>
<tr>
<td>CBT</td>
<td>Core Based Tree</td>
</tr>
<tr>
<td>CCDF</td>
<td>Complementary Cumulative Distribution Function</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Distribution Network</td>
</tr>
<tr>
<td>CHB</td>
<td>Cautious Harmonic Broadcasting</td>
</tr>
<tr>
<td>CI</td>
<td>Continuity Index</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash Table</td>
</tr>
<tr>
<td>dPAM</td>
<td>Distributed Prefetching Protocol for Asynchronous Multicast</td>
</tr>
<tr>
<td>DVMRP</td>
<td>Distance Vector Multicast Routing Protocol</td>
</tr>
<tr>
<td>DV</td>
<td>Distance Vector</td>
</tr>
<tr>
<td>EDF</td>
<td>Empirical Distribution Function</td>
</tr>
<tr>
<td>EPDF</td>
<td>Empirical Probability Density Function</td>
</tr>
<tr>
<td>ESM</td>
<td>End-System Multicast</td>
</tr>
<tr>
<td>EuroFGI</td>
<td>European Future Generation Internet</td>
</tr>
<tr>
<td>EuroNGI</td>
<td>European Next Generation Internet</td>
</tr>
<tr>
<td>EXPRESS</td>
<td>Explicitly Requested Single-Source Multicast</td>
</tr>
<tr>
<td>FCFS</td>
<td>First Come First Served</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GDB</td>
<td>Greedy Disk-based Broadcast</td>
</tr>
<tr>
<td>GIC</td>
<td>Generalized Interval Caching</td>
</tr>
<tr>
<td>GOP</td>
<td>Group of Pictures</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HB</td>
<td>Harmonic Broadcasting</td>
</tr>
<tr>
<td>HDVMRP</td>
<td>Hierarchical Distance Vector Multicast Routing Protocol</td>
</tr>
<tr>
<td>HEC</td>
<td>Heuristic Extended Chaining</td>
</tr>
<tr>
<td>HIP</td>
<td>Host Identity Protocol</td>
</tr>
<tr>
<td>HMSM</td>
<td>Hierarchical Multicast Stream Merging</td>
</tr>
<tr>
<td>HHMSM</td>
<td>Hierarchical Hybrid Multicast Stream Merging</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transport Protocol</td>
</tr>
<tr>
<td>HP</td>
<td>high priority</td>
</tr>
<tr>
<td>IGMP</td>
<td>Internet Group Management Protocol</td>
</tr>
<tr>
<td>IGP</td>
<td>Interior Gateway Protocol</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>IID</td>
<td>Independent and Identically Distributed</td>
</tr>
<tr>
<td>IIS</td>
<td>Swedish Foundation for Internet Infrastructure</td>
</tr>
<tr>
<td>IMS</td>
<td>Internet Multimedia Subsystem</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPMC</td>
<td>Internet Protocol Multicast</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>KMB</td>
<td>Kou, Markowski, and Berman</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LFU</td>
<td>Least Frequently Used</td>
</tr>
<tr>
<td>LRD</td>
<td>Long-Range Dependence</td>
</tr>
<tr>
<td>LRU</td>
<td>Least Recently Used</td>
</tr>
<tr>
<td>MAAS</td>
<td>Multicast Address Allocation Server</td>
</tr>
<tr>
<td>MADCAP</td>
<td>Multicast Address Dynamic Client Allocation Protocol</td>
</tr>
<tr>
<td>MALLOC</td>
<td>Internet Multicast Address Allocation Architecture</td>
</tr>
<tr>
<td>MBone</td>
<td>Multicast Backbone</td>
</tr>
<tr>
<td>MDC</td>
<td>Multiple Description Codec</td>
</tr>
<tr>
<td>MLE</td>
<td>Maximum Likelihood Estimation</td>
</tr>
<tr>
<td>ML</td>
<td>Maximum-Likelihood</td>
</tr>
<tr>
<td>MOSPF</td>
<td>Multicast Extensions to Open Shortest Path First</td>
</tr>
<tr>
<td>MPEG</td>
<td>Motion Picture Experts Group</td>
</tr>
<tr>
<td>MQL</td>
<td>Maximum Queue Length</td>
</tr>
<tr>
<td>MST</td>
<td>Minimum Spanning Tree</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NNTP</td>
<td>Network News Transfer Protocol</td>
</tr>
<tr>
<td>NoE</td>
<td>Network of Excellence</td>
</tr>
<tr>
<td>OLMC</td>
<td>Overlay Multicast</td>
</tr>
<tr>
<td>OSPF</td>
<td>Open Shortest Path First</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PB</td>
<td>Pyramid Broadcasting</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>PHB</td>
<td>Polyharmonic Broadcasting</td>
</tr>
<tr>
<td>PIM-DM</td>
<td>Protocol Independent Multicast – Dense Mode</td>
</tr>
<tr>
<td>PIM-SM</td>
<td>Protocol Independent Multicast – Sparse Mode</td>
</tr>
<tr>
<td>PIT</td>
<td>Probability Integral Transform</td>
</tr>
<tr>
<td>PoP</td>
<td>Point of Presence</td>
</tr>
<tr>
<td>PPB</td>
<td>Permutation-based Pyramid Broadcasting</td>
</tr>
<tr>
<td>QHB</td>
<td>Quasi-Harmonic Broadcasting</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RBC</td>
<td>Resource-Based Caching</td>
</tr>
<tr>
<td>RDP</td>
<td>Relative Delay Penalty</td>
</tr>
<tr>
<td>RIP</td>
<td>Routing Information Protocol</td>
</tr>
<tr>
<td>ROVER</td>
<td>Routing in Overlay Networks</td>
</tr>
<tr>
<td>RPB</td>
<td>Reverse Path Broadcasting</td>
</tr>
<tr>
<td>RPF</td>
<td>Reverse Path Forwarding</td>
</tr>
<tr>
<td>RPM</td>
<td>Reverse Path Multicasting</td>
</tr>
<tr>
<td>RP</td>
<td>Rendezvous Point</td>
</tr>
<tr>
<td>RR</td>
<td>Round Robin</td>
</tr>
<tr>
<td>SHA1</td>
<td>Secure Hash Algorithm One</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SM</td>
<td>Simple Multicast</td>
</tr>
<tr>
<td>SOCCER</td>
<td>Self-Organizing Cooperative Caching Architecture</td>
</tr>
<tr>
<td>SPT</td>
<td>Shortest-path Tree</td>
</tr>
<tr>
<td>SSM</td>
<td>Source Specific Multicast</td>
</tr>
<tr>
<td>TCP</td>
<td>Transport Control Protocol</td>
</tr>
<tr>
<td>TRPB</td>
<td>Truncated Reverse Path Broadcasting</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>VBR</td>
<td>Variable Bit-Rate</td>
</tr>
<tr>
<td>VINNOVA</td>
<td>Swedish Agency for Innovation Systems</td>
</tr>
<tr>
<td>VoD</td>
<td>Video-on-Demand</td>
</tr>
<tr>
<td>WPMC</td>
<td>Waypoint Multicast</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>e2e</td>
<td>end-to-end</td>
</tr>
<tr>
<td>Reference</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>


147


