On-demand Television combined with non-real-time Peer-to-Peer Content Delivery for Television Content Providers

DARIO VODOPIVEC

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2010

TRITA-ICT-EX-2010:211

On-demand Television combined with non-real-time Peer-to-Peer Content Delivery for Television Content Providers

Dario Vodopivec
dariov@kth.se

Masters Thesis
Royal Institute of Technology (KTH)

24 August 2010

Gerald Q Maguire Jr.
KTH Supervisor

Henric Persson
Kanal5 Supervisor

Abstract

With the expansion of the Internet and an increasing fraction of consumers having broadband connections, more and more content is finding its way on-line. Video content is becoming one of the most popular types of media content on the Internet. Traditional media content providers, such as television networks, are placing their content on the Internet in order to gain a broader audience. On-demand websites such as kanal5play.se allow viewers to view the multimedia content they want at the time of their choice. While this gives viewers flexibility in their viewing, it creates resource problems for content providers.

Statistics from Kanal5 AB show that, even with individual viewers requesting content when they want, there are still patterns in which multiple viewers watch the same content at the same time. This means there are correlations in the demand for content. With unicast distribution this leads to spikes in requirements for bandwidth to the viewers. These peaks lead to high costs for network and server resources to deliver the requested content, but these resources have low average utilization. This thesis project investigates how a content provider can make use of each viewer's own resources to deliver content to other users using peer-to-peer techniques. The thesis evaluates what methods can be used in order to reduce the content provider's resource requirements during peak hours by exploiting copies of contents that have already been delivered to viewers who requested this same content earlier.

A prototype was made to evaluate the suggested design using Java Remote Method Invocation (RMI), which is built on top of the Transfer Control Protocol (TCP). Experiments show that an initial delay of several seconds is reached on a network with a simulated delay of 100ms, while a minimal initial delay was observed on a network with low delay, i.e. ideal conditions. The throughput results of the prototype show that the suggested solution is adequate for delivering on-demand content supplied by Kanal5 AB. However, the relatively poor startup performance of this solution argues for tuning the application to better work with the TCP protocol or to utilize another transport protocol - especially if the round-trip delay is large as TCP's 3-way handshake and flow control algorithm limit the performance of the prototype system.

Sammanfattning

Med utbyggnaden av Internet och en ökande andel konsumenter med bredband, mer och mer innehåll hittar sin väg på nätet. Video innehåll blir en av de mest populära typer av media på Internet. Innehållsleverantörer som använder sig av traditionella medier, exempelvis tv-nät, lägger sitt innehåll på Internet för att nå en bredare publik. On-demand webbplatser som kanal5play.se låter tittarna se multimediainnehållet de vill, när de vill. Även om detta ger tittarna flexibilitet i sitt tittande så skapar det resursproblem för innehållsleverantörer.

Statistik från Kanal5 AB visar att även med enskilda tittare som begär innehåll när de vill så finns det fortfarande mönster där flera tittare tittar på samma innehåll på samma gång. Detta innebär att det finns samband i efterfrågan på innehåll. Med unicast distribution leder detta till sprikar i krav på bandbredd till tittarna. Dessa toppar leda till höga kostnader för nät-och server för att leverera det efterfrågade innehållet, men dessa resurser har låga genomsnittliga utnyttjanden. Detta examensarbete undersöker hur en innehållsleverantör kan använda sig av varje tittares egna resurser för att leverera innehåll till andra användare med hjälp av peer-to-peer-teknik. Avhandlingen utvärderar vilka metoder kan användas för att minska innehållsleverantörens resursbehov under rusningstid genom att utnyttja kopior av innehåll som redan har levererats till tittarna som begärde samma innehåll tidigare.

En prototyp gjordes för att utvärdera den föreslagna konstruktionen med Java Remote Method Invocation (RMI), som är byggd ovanpå Transfer Control Protocol (TCP). Experiment visar en uppstartsfördröjning på flera sekunder på ett nätverk med en simulerad fördröjning på 100 ms, samtidigt som en minimal uppstartsfördöjning observerades på ett nätverk med låg fördröjning, dvs idealiska förhållanden. Resultaten för genomströmningshastigheten hos prototypen visar att den föreslagna lösningen är tillräcklig för att leverera on-demand innehåll som tillhandahålls av Kanal5 AB. De relativt dåliga uppstartsresultaten för denna lösning säger dock att förbättringar bör göras i applikationen så att den kan arbeta bättre med TCP protokollet, eller att ett annat protokoll används - särskilt om nätverksfördröjningen är stor, då TCP:s 3-vägs handskakning och flödeskontroll algoritm begränsar prestandan hos det föreslagna systemet.

Acknowledgments

First and foremost I would like to thank my academic advisor and examiner Professor Gerald Q. Maguire Jr. for his knowledge, constructive criticism, and advice which have helped me throughout the entire thesis. Next I would like to thank my industrial advisor at Kanal5 AB, Henric Persson, for his invaluable advice, programming expertise, feedback, and support. This thesis would not have been possible without this help from both of them.

Further acknowledgments go to Daniel Cedercrona for his thoughts and ideas regarding the direction of the thesis, for lending his industrial connections, and for giving me the opportunity to do my thesis project at Kanal5 AB.

Special thanks to George Younan and Anders Orrevad for their advice and support in time of confusion, to Rajna, Anders & Emil Lundqvist for their couch and support, and to all other family members and friends who supported me before, during, and after my thesis.

Thank you,

/Dario

Contents

Abstract i

Acknowledgments iii

List of Figures vii

List of Tables ix

List of Abbreviations x

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Goals of this master's thesis project . . . 2
- 1.3 What this thesis project does not cover . . . 3

2 Related Work 4
- 2.1 IPTV . . . 4
  - 2.1.1 Video on-demand . . . 5
  - 2.1.2 The nature of Video on-demand . . . 5
- 2.2 Content Distribution Networks . . . 6
- 2.3 Peer-to-peer . . . 7
  - 2.3.1 P2P explained . . . 8
  - 2.3.2 MPS Broadband . . . 10
  - 2.3.3 Gnutella . . . 10
  - 2.3.4 Daedalus . . . 11
  - 2.3.5 Joost . . . 12
  - 2.3.6 Spotify . . . 12
- 2.4 Additional relevant background information . . . 13
  - 2.4.1 Swedish home network speeds . . . 13
  - 2.4.2 IPv6 . . . 14
  - 2.4.3 Testing of P2P systems . . . 14
  - 2.4.4 Episodes . . . 14
  - 2.4.5 CODECs . . . 15

3 Loree: A system designed for On-demand P2P television 16
3.1 Loree overview . . . 16
3.1.1 Resources used for development . . . 16
3.1.2 Prototype implementation using Java RMI . . . 17
3.2 Key features of Loree . . . 18
3.3 Client and Content Server . . . 20
3.3.1 Client operation . . . 21
3.3.2 Client implementation . . . 22
3.3.3 Content Server . . . 22
3.4 Tracker . . . 23
3.4.1 Tracker implementation . . . 24
3.4.2 Monitor . . . 25
3.5 Modules . . . 25
3.5.1 Sender module . . . 26
3.5.2 Downloader . . . 27
3.5.3 Tracker module . . . 27
3.6 Classes in Loree . . . 28
3.6.1 Peer class . . . 29
3.6.2 Video class . . . 29
3.6.3 PartOfVideo and PartOfVideoInfo classes . . . 29
3.6.3.1 Size of PartOfVideo . . . 30
3.6.3.2 Order of downloading content . . . 31
3.7 Downloading other relevant episodes . . . 32

4 Measurements 34
4.1 Test environment . . . 34
4.2 Delays . . . 35
4.3 Initial delay . . . 37
4.3.1 Ideal conditions . . . 38
4.3.1.1 Peer retrieving content from a Content Server . . . 38
4.3.1.2 Peer requesting content from another Peer . . . 39
4.3.2 With added network delay . . . 41
4.4 Throughput . . . 42
4.4.1 Ideal conditions . . . 43
4.4.1.1 Peer requesting content from a Content server . . . 43
4.4.1.2 Peer requesting content from another Peer . . . 44
4.4.2 With added network delay . . . 46
4.5 Control overhead . . . 48

5 Evaluation and Recommendations 51
5.1 Initial delay and Throughput . . . 51
5.2 Playout buffer . . . 52
5.3 Overhead communications . . . 52
5.4 The Achillies' heel . . . 53

6 Conclusions and Future Work **54**
6.1 Conclusions 54
6.2 Future work 55
6.2.1 Future implementation work 55
6.2.2 Future research work 57

Bibliography **59**

Appendix A Viewing patterns of Kanal5play viewers **64**

List of Figures

2.1 Classic client-server content distribution. Server sends content directly to each user. 6
2.2 CDN network where content servers are strategically placed close to users. 7
2.3 Tree structured Peer-to-Peer system 9
2.4 Unstructured (mesh) Peer-to-Peer system 9

3.1 1. A client connects to the tracker's registry and retrieves the remote tracker module which it will then use to invoke functions. 2. Clients connect to other clients' registries and retrieve the remote Sender module object which is used to transfer files. 18
3.2 Example of a tree structure in the Loree system. Note that the peers can be organized in different ways and that they can retrieve content from multiple peers for different pieces of content. There can be multiple content servers in the network and the tracker is on a separate location and communicates with all peers and content servers. 20
3.3 Internal components of the tracker. 25
3.4 The Client and Server are put together using the different modules. Note, however, that they are essentially identical since they use the same modules. The difference is that the content server has all the content available from start. 26
3.5 The timeline of a TV-episode. 1 represents the point of time where a viewer might choose to start viewing, and some content is buffered after that, 2 & 3 represent other points in the episode which are buffered in advanced because they are predicted to be skipped to by most viewers and t=n is the end of the episode. Note that the above figure is strictly for illustration purposes as these patterns have not yet been investigated. 31
3.6 The figure shows how the amount of content sent by a server during peak hour could look like (compare with figure A.1 in appendix A). The red line represents how the throughput over time can look like when all viewers download the content from the server. The black line, on the other hand, represents how the peak could be reduced and the average throughput increased if peers download content in advanced and new peers joining the network download the content from them. 33

4.1 The network setup for testing the Loree system. Three PCs interconnected using a 100Mbps switch. 35
4.2 Time line diagram showing the traffic between a client, tracker, and another peer in the prototype implementation. 37
4.3 The percentage breakdown of traffic sent in bytes over the Loree network when transferring a file of 323MB in 256kB chunk sizes. Note that Peer messages consist of traffic sent between client and peer which includes both content and control traffic, thus this figure primarily shows the percentage of control traffic to and from the tracker. 49
4.4 The percentage breakdown of traffic sent in bytes over the Loree network when transferring a file of 323MB in 512kB chunk sizes. Note that Peer messages consist of traffic sent between client and peer which includes both content and control traffic, thus this figure primarily shows the percentage of control traffic to and from the tracker. 49
4.5 The percentage breakdown of traffic sent in bytes over the Loree network when transferring a file of 323MB in 1024kB chunk sizes. Note that Peer messages consist of traffic sent between client and peer which includes both content and control traffic, thus this figure primarily shows the percentage of control traffic to and from the tracker. 50

A.1 Number of page loads for the TV show "Ballar av stål" during the 21th of March 2010. The episode was shown on TV during the time interval 21:00-21:55. These statistics show that most viewers accessed the play site just after the show was broadcast on TV (with the majority of accesses taking place between 22:00-24:00). 65
A.2 Pages entered related to the show "Ballar av stål" on the day episode two of season two was shown (s02e02 in the above list). 65

List of Tables

3.1 The Sender Modules remote interface functions. 27
3.2 The Trackers remote interface functions. 28

4.1 Performance of the Loree prototype, measuring initial delay in milliseconds (ms) when a client downloads content of 323MB from a content server. 38
4.2 Performance test of the Loree prototype, measuring initial delay in ms when a client downloads content of 735MB from a content server. 39
4.3 Performance test of the Loree prototype, measuring the initial delay in ms when a client downloads content of 323MB from a peer. 40
4.4 Performance test of the Loree prototype, measuring the initial delay in ms when a client downloads content of 735MB from a peer. 41
4.5 Performance test of the Loree prototype, measuring the initial delay in ms of a client downloading content of 323MB from a peer. The client has a simulated delay of 50ms on inbound and outbound packets (total simulated round-trip time of 100ms). 42
4.6 Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 323MB from a content server. . . 43
4.7 Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 735MB from a content server. . . 44
4.8 Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 323MB from a peer. 45
4.9 Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 735MB from a peer. 46
4.10 Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 323MB from a peer in the network. The client has a simulated delay of 50ms on inbound and outbound packets (total simulated round-trip time of 100ms). 47

List of Abbreviations

AB	Aktie Bolag (Swedish for: Joint Stock Company)
ACK	Acknowledgment
API	Application programming interface
CDN	Content Distribution Networks
CODEC	Compressor-decompressor
GB	Gigabyte
Gbps	Gigabit per second
GHz	Gigahertz
H.264	H.264/MPEG-4 Part 10 or Advanced Video CODEC
HD-DVD	High Definition-Digital Video Disc
HP	Hewlett Packard
HTTP	Hypertext Transfer Protocol
ID	Identification
IPTV	Internet Protocol Television
ISP	Internet Service Provider
kB	Kilobyte
kBps	Kilobyte per second
kbps	kilobit per second
MB	Megabyte
Mbit	Megabit
Mbps	Megabit per second

ms	Millisecond
MSS	Maximum Segment Size
OS	Operating System
P2P	Peer-to-Peer
Play site	On-demand website
QoS	Quality of Service
RMI	Remote Method Invocation
RTT	Round-trip time
SPCs	Strategically placed servers
TV	Television
VM	(Java) Virtual Machine
VoD	Video on-demand

Chapter 1

Introduction

This chapter explains the goals of this thesis project and the scenario which the thesis project builds upon.

1.1 Background

The explosion of Internet usage has lead to a situation where a very large amount of multi-media content is transfered via the Internet. This multi-media content includes video clips and much of the material that was previously broadcast by traditional television (TV) networks. The Internet enables users to get the media content they want through "video on-demand" (VoD) services and/or through file sharing systems.

Internet growth has occurred both in numbers of users and in the amount of traffic being transferred (measured in both bytes and numbers of packets). The types of information that are being transfered is changing, with more and more traffic being video content.

Since today's youth spend more time in front of their computers, traditional TV content providers and aggregators (such as existing TV channels) are interested in finding new ways to reach these potential viewers. One potential way of reaching them is through on-demand content, as this enables viewers to view their choice of show or movie whenever they want. One of the technical questions that occurs is how to deliver the desired content to these Internet viewers, while at the same time keep the costs down and maintaining or increasing profits. Sending streams to individual viewers (via unicast) requires very large amounts of bandwidth (roughly proportional to the number of viewers). The success of peer-to-peer (P2P) file sharing and other P2P services suggest that one method to lower the costs of distributing content to users is to utilize the viewer's own resources to help other viewers. Earlier thesis projects have shown that splitting traffic over multiple download sites can decrease the time required to deliver content to a user [1] and that if the peers are located in the same fixed access network that the network access provider can more effectively deliver traffic within their network rather than having to transfer all traffic via peering points [2].

Peer-to-Peer (P2P) communication is a well established technology. It has been widely used for file sharing, in particular for software distribution and audio content, but video streaming P2P techniques are not yet widely utilized. There are a number of different services available for live streaming [3], e.g. TVants [4]. There are some on-demand video services, such as Joost [5]. However, tests show that Joost directly supplies 2/3 of the traffic to individual users, thus only 1/3 of the traffic is delivered by P2P. In a "pure" P2P service, almost all of the content should be supplied by other users (see section 2.3). However, saving even 1/3 of the network resources that would have been required to provide all of the content is still 1/3 savings. For a site providing content that is popular with a large number of users this could represent a substantial costs savings (both of capital expenses and operating expenses), hence even this level of savings might make introducing P2P distribution worthwhile.

Today there are some practical problems that may affect the operation of a P2P network. For example, for the large numbers of Internet subscribers who access the Internet via Assymetric Digital Subscriber Line (DSL) and Very high-rate DSL there is a substantial assymetry in the bandwidth of their broadband connections. Cao Wei Qiu showed that this asymmetry can be exploited by utilizing multiple sources from which a peer simultaneously downloads different portions of a file - thus exploiting the high downlink bandwidth by supplying content from multiple peers with more limited uplink bandwidth [1]. The asymmetry in bandwidth of many users means that we must examine how to use more than one source to provide delayed content, specifically exploiting a set of sources that each provide different parts of content.

Numbers from a Swedish TV channel show that many viewers who miss part of an episode, or the full episode, from a popular show will later go to the channel's on-demand website to watch this episode (see section 2.1.1). For one channel's statistics see appendix A. These statistics (and statistics for other days) show that there is a peak hour during which many viewers watch the same on-demand content, but not all viewers watch the program at the time it is first shown[1]. This thesis project examines how the traffic content that is delivered to viewers during the peak hour could be exploited to provide delayed delivery of this content to viewers who wish to view the content later. The focus of this thesis will be on non-real-time P2P delivery for TV content providers (e.g. for television networks), in contrast to real-time P2P delivery as previously described in [6].

1.2 Goals of this master's thesis project

P2P delivery may not make a huge difference in the network load on the content provider with respect to on-demand video, due to uncorrelated requests, i.e., when people watch different things at different times. However, for popular content there are peak hours when there are many viewers who want to watch the same content and there are substantial numbers of viewers who want to watch this content with a delay of

[1]Note that the detailed statistics of requests for content are not important to this thesis, only the fact that there are many requests for 1) the content which has been recently requested and 2) related content. It is simply the fact that there is a correlation in the content requested by multiple users which is important.

several hours. This thesis will explore the use of P2P technology to save resources when delivering this delayed content when there is some correlation in requests.

The goal of this thesis is to investigate how a content provider can make use of each viewer's own resources to deliver content to other users using P2P techniques.

The thesis will be organizes as follows:

- Chapter 1 explains the goals of this thesis project.
- Chapter 2 provides basic background information the reader needs in order to understand the rest of this thesis and surveys related work.
- Chapter 3 describes the system that has been designed and implemented to solve the problem described in Chapter 1.
- Chapter 4 describes a set of measurements made of this solution while analysis are made in Chapter 5.
- Chapter 6 states the conclusions made and suggests future work.

The reader is expected to understand the principles of internetworking along with knowledge of internetworking protocols.

1.3 What this thesis project does not cover

This thesis project will not focus on any questions regarding subscription based content or free content, nor will it discuss advertising or commercials which can be broadcast together with the on-demand content. It will not discuss issues regarding Network Address Translation (NAT) boxes or firewalls. In addition, this thesis will not consider optimal algorithms for content distribution. This later area is left for future work - although it should be noted that there has already been extensive work in this area such as [7].

Chapter 2

Related Work

This chapter reviews existing technologies and provides background information that will provide the reader with a basis for understanding the rest of this thesis.

Silverston and Fourmaux classify P2P video streaming into two main categories: video on-demand and live video. The difference is that in on-demand video the content is a pre-recorded file of a known size; whereas in live video, the total amount of data is not know in advanced and can be unbounded [8]. This thesis will focus on on-demand video, thus the size of each file of media content is known in advance. Later this thesis will explore how knowing the size of each file can be exploited to divide content into manageable chunks to be distributed over a set of peers.

2.1 IPTV

Internet Protocol Television (IPTV) distributes digital video and audio content (digital television) through packet switched networks (e.g. the Internet) instead of traditional broadcast distribution through terrestrial broadcasts, satellite broadcasts, or via a cable TV distribution network. IPTV has a number of advantages over traditional broadcast. Among these advantages are:

- Better control of who the content is delivered to.
- As the content distributor can know who the content is distributed to, they can communicate with the consumer of this content. For example, they can ask questions regarding the content or ask viewers to vote upon what will happen in a certain episode.
- IPTV viewers can access a virtually unlimited number of channels, without being limited to the specific channels that have been allocated cable/satellite broadcast resources.

Gilbert Held [9] divides IPTV into: Homeowner entertainment, On-demand video, Business TV to the desktop, Distance learning, Corporate communications, Mobile phone television, and Video chat. All of these are types of services than can be provided by IPTV.

There are also disadvantages of IPTV, such as sensitivity to packet loss. As a result, users need to have at least a certain minimum available bandwidth otherwise they might not receive the content with acceptable quality. Depending on the scenario and content, there can be a trade-off between increasing delay and increasing quality when the available bandwidth is limited.

2.1.1 Video on-demand

Video on-demand (VoD) refers to delivering video content that a user can view whenever he or she wants following a request for specific content[1]. In recent years many media companies have begun offering viewers the ability to watch TV episodes on-line through their website. Examples of these services are Sveriges Television (SVT) Play [10], TV4 Play [11], and Kanal5 Play [12]. These services are mostly focused on TV series, while there are other content providers who focus on VoD for movies - for example Voddler [13].

2.1.2 The nature of Video on-demand

VoD has the advantage of allowing viewers to watch what they want, when they want. This changes how video content is viewed in comparison to traditional TV - as viewers are no longer bound to follow schedules which are determined by TV content providers nor does each user have to record the broadcast in order to time shift it for viewing later.

Since VoD provides users with flexibility, it is unlikely that all users will watch the same content at the same time. There are different types of on-demand video: content provided by the viewers themselves (e.g. Youtube [14], Google Video [15], etc.) and content provided by media production companies (such as movie studios) and distributed by media distribution companies (e.g., Joost [5] and Voddler [13]).

Despite this flexibility, there are still patterns in the times when users watch content provided by television networks. Statistics provided by Kanal5 AB (Aktie Bolag) based upon their on-demand website [12] show that many users watch episodes shortly after they have been broadcast. This leads to the concept of *on-demand television* (in contrast to video on-demand); where television broadcasts affects the patterns of what viewers watch and when they watch it. In classical broadcast television, editors influence what viewers watch by promoting certain shows or episodes (via advertising - which could be in any media). In the same way, editors of a TV network can influence viewers to access a website to view specific episodes (or other content) at certain hours.

However, editors are not the only reason for this correlation. Many viewers view the content after it was shown on broadcast television because they want to see it again or they missed some part of the broadcast. Other factors may also contribute to the correlation in viewing (for example, social networks and a desire to have a shared

[1]This content can be subscription based or free.

context). The result of this correlation in demand is that there are clear peak hours for on-demand television in contrast to the lower correlation of demand for video on-demand content.

2.2 Content Distribution Networks

For many years content distribution networks (CDNs) have been used for unicasting/multicasting content. Unicasting video streams means sending the same stream separately to each viewer. While this may sound intuitive and simple, its not very efficient. See Figure 2.1. Unicasting a 1 Megabit per second (Mbps) video stream to 1000 users simultaneously would require a broadband connection of 1 Gigabit per second (Gbps). A television network such as Kanal5 of Sweden can have thousands of viewers viewing a program at the same time, thus unicasting does not scale well. As Cao Wei Qui states in his thesis: ”One solution is to increase the link capacity, but this is not an optimal solution because this will lead to potential bandwidth wasting when the peak hours passed” [1, section 1.3.1].

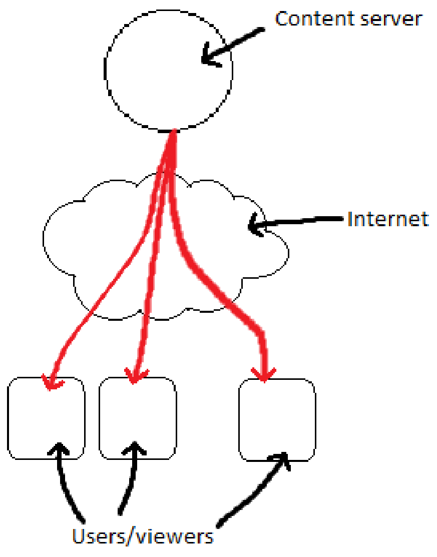


Figure 2.1: Classic client-server content distribution. Server sends content directly to each user.

Multicast can be used to send a stream to many receivers. In the example above, the content provider would only need a 1Mbps connection from their server to send the same content to 1000 users at the same time. However, not all routers and firewalls allow multicast traffic to pass through them - hence multicasting is often limited to multicasting within a given Internet Service Provider’s (ISPs) access network. This is due to multicast lacking reliable packet dissemination, security, flow control and scalability in number of groups, as discussed by Vigfusson et.al. in [16].

An alternative to multicasting, is that rather than having one server send the full stream to many users, multiple distributed servers can be used to unicast the traffic to the users. Thus the individual server load is reduced, but the aggregated load is still high. This approach scales well if there is a mix of content served by the CDN and if the requests for content are **not** correlated.

A CDN distributes the content from one main server containing the content to multiple strategically placed servers (SPCs) from which the content can be redistributed to the users. This reduces the workload on the central server and reduces the bandwidth needed by this server by simply dividing the workload over a set of distributed servers. If these SPCs are placed in good locations, they also reduce the latency between the individual user clients and the SPC from which each client is getting content. See Figure 2.2. Although this reduces the bandwidth and computing power needed for a single server, the sum of the workload and bandwidth is still the same. Cao Wei Qiu provides a deeper analysis of CDN networks in his thesis [1]. This leads to the question: Can we reduce the resources required by the content distributor without reducing the Quality of Service (QoS) provided to users? One alternative is to enable the users to help provide the resources!

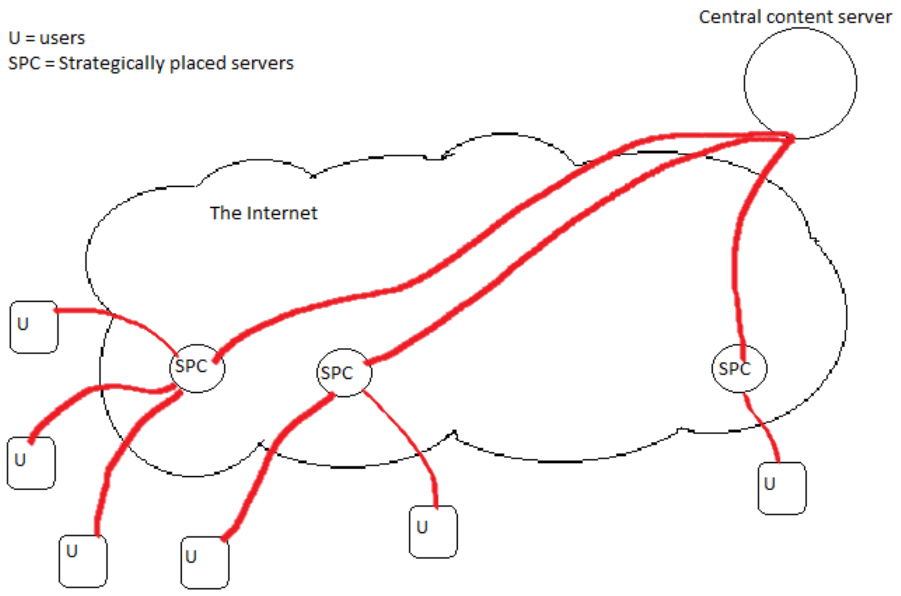


Figure 2.2: CDN network where content servers are strategically placed close to users.

2.3 Peer-to-peer

Peer-to-peer (P2P) distribution of content enables the users to supply the resources for distributing the content themselves. This section will describe how P2P technology can be used for this distribution of content.

P2P is mostly (in)famous for being used for downloading of illegal content, however, the technology itself is very useful since it uses its users own resources to provide content to other users. There are many different implementations, but the basic idea is that if one or more users has a file, then another user can download a copy of this file from *any* user that has this file instead of going to a central server. This is in contrast to client-server based systems where all of the files would be kept in a central server which must supply all of the storage space for the content and have sufficient total bandwidth to satisfy all of its users. Using P2P, the content provider shifts the provisioning of resources to the users.

Section 2.3.1 reviews how P2P works and describes some of the different alternatives. For more information on P2P systems, please refer to [17].

2.3.1 P2P explained

According to Androutsellis-Theotokis and Spinellis, the two defining characteristics of P2P architectures are:

- ”The sharing of computer resources by direct exchange, rather than requiring the intermediation of a centralized server. Centralized servers can sometimes be used for specific tasks (system bootstrapping, adding new nodes to the network, obtain global keys for data encryption)” [17] and
- ”Their ability to treat instability and variable connectivity as the norm, automatically adapting to failures in both network connections and computers, as well as to a transient population of nodes.” [17].

These characteristics can be summarized by saying: Instead of having a centralized server (like the classic client-server architecture described in section 2.2) users connect to each other in order to receive and send content.

"Pure” P2P refers to as a system which has no supervision or central server whatsoever, i.e. where all peers are equal and handle the network creation and maintenance and transfer content among themselves. In such pure P2P systems, searches and network control messages are sent via peers to other peers. The network is self-managing and supports peers joining and leaving the network. An example of a pure P2P network is Gnutella (see section 2.3.3). Gnutella uses a distributed query approach to find content. In this approach a peer sends a message to its neighbors and they, in turn, forward the request to their neighbors if they do not have the content. This continues for up to 7 steps.

Napster [18] and BitTorrent [19] are examples of *hybrid* P2P networks. Napster uses a central server which keeps track of every file existing in the network, hence users connect to the server to ask where the file is located and then download it from the specified location. BitTorrent, on the other hand, uses a tracker which keeps information *about* files. Users connect to the tracker and receive meta-information about a file (e.g. where parts of the file can be downloaded from) and then the users download parts of the file from the different sources.

P2P systems can be categorized based on their network structure. In *structured* P2P networks, the peers build a topology of trees or graphs in order to optimize the downloading rates and speed up queries for files (see figure 2.3). However, a structured network is hard to maintain due to the fact that peers can join and leave at unexpected times. When either of these changes happen, a new topology should be rebuilt in order to optimize the graph (this is especially true when a peer high up in a tree hierarchy leaves - as this might disrupt network operations severely). A properly structured P2P network can provide very high performance. Companies such as MPS Broadband AB (see section 2.3.2) specialize their P2P solutions for live viewing where many viewers watch the same content at the same time with few viewers leaving or joining at unexpected times.

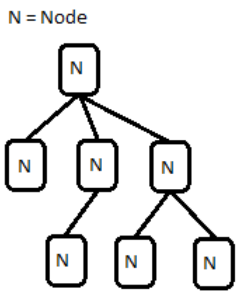


Figure 2.3: Tree structured Peer-to-Peer system

Unstructured P2P systems represent another category of P2P systems (see figure 2.4). Here peers connect arbitrarily to other peers and simply build a mesh network. The advantage is that when a peer leaves unexpectedly, it does not significantly change the network topology since peers are not dependent on other peers higher up in a hierarchy. However, large overheads (in terms of traffic) are incurred in order for the peers to know about the existence of other peers and when searching for content. Gnutella is an example of an unstructured P2P network (section 2.3.3).

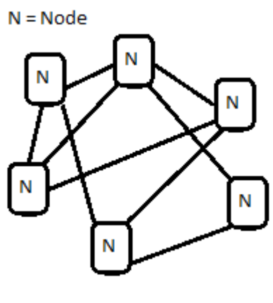


Figure 2.4: Unstructured (mesh) Peer-to-Peer system

If "pure" P2P systems wish to maintain information regarding their users (i.e. what users exist, how stable they are, what level in the hierarchy should they be in, etc.)

they can implement a reputation system (for an example of a reputation system see [20]). Having a central server keeping track of all users and the hierarchy (for structured systems) simplifies the work of the peers. In pure P2P systems the peers communicate with each other and maintain this information amongst themselves (as a distributed task). This is not an easy task since nodes can come and go as they please, and if one of these nodes leaves unexpectedly then important information could be lost unless it is replicated elsewhere. When peers frequently connect and disconnect, much traffic needs to be sent in order to update the status of the nodes of the network.

The following sections, will describe some existing P2P systems and companies that are using P2P technology.

2.3.2 MPS Broadband

MPS Broadband AB[21] uses a tracker system (i.e. their live P2P service is based on a torrent architecture) developed by Peerialism AB [22]. This tracker system keeps track of all users in their network. Then, using a patented algorithm, they consider all users every three seconds and compare them one to another and optimize the user hierarchy so that the users with the best broadband connections are placed higher in the hierarchy of users (the higher in this hierarchy a user is the closer the user is to the source server). Additionally, to make ISPs happier, they organize users within the same ISP's network to send more often to each other. This reduces the amount of traffic send to/from other ISPs (thus reducing the ISP's costs).

They claim that, up to 98% of the traffic sent in the network comes from other users [23]. However, in on-demand streaming the same numbers are not likely to be possible since most users will probably not watch the same thing with in a short period of time.

They claim that the algorithm they use to organize the peers is mathematically proven to be optimal [23].

2.3.3 Gnutella

Gnutella was originally developed by Nullsoft as an "enhanced Napster clone" [24][2]. The idea was to create a protocol which would enable users to download music in the same way Napster had, but without a centralized server which could be blocked by network administrators (or legal action). The other, and more important, advantage Gnutella had was that because it had not centralized server, it became very hard for the Gnutella network to be shut down since the content is distributed amongst users in the entire network. This meant that it could evolve and continue working where Napster had been stopped due to court decisions.

The Gnutella network works in such a way that each node connects to at least one peer and builds a small ring of peers. When a node wants to perform a search it sends a request to each node it is connected to, these nodes will in turn send this request to

[2]For information regarding the history of Napster, please refer to `http://en.wikipedia.org/wiki/Napster`.

each of the nodes that *it* is connected to, up to a maximum of 7 ”hops”[3]. This means that a query would quickly reach around 8,000 computers in a simple and quick way[6, Section 2.2]. In version 0.4 of the Gnutella protocol, the number of actively connected peers for each node was around 5, in version 0.6 the terms *leaf nodes* and *ultra nodes* were introduced. Differentiation into these two types of nodes were introduced in order to reduce the network overhead caused by the flooding of messages between nodes. These nodes also increased stability since nodes might leave the network at any time and thus search requests would not reach so far into the network if nodes along the way are not there. An ultra node, or *ultrapeer*, is a powerful node which maintains many connections to non-ultrapeer nodes (also known as leaf nodes) and some connections to other ultrapeers [25]. Instead of all nodes being considered equal, nodes entering into the network were kept at the ’edge’ of the network as a leaf, hence they are not responsible for any routing of queries. Ultrapeers, in turn, were nodes which were considered stable, thus they were entrusted with accepting leaf connections and forwarding searches and network maintenance messages. Leaf nodes connect to a small number of ultrapeers (approximately 3) while ultrapeers connect to many leaf nodes and to many ultrapeers (more then 32). This shields leafs from messages and increases the stability of the network while reducing the number of hops messages need to travel since nodes are connected to many more peers and could thus reach many more nodes.

After a user receives responses to the search query, the node can initiate a Hypertext Transfer Protocol (HTTP) request to the host holding the file in order to download the file.

2.3.4 Daedalus

Daedalus is a P2P based architecture for media content delivery on large scale networks [6]. However, it is not available in any commercial form at the time of writing. The project was conducted internally at Ericsson [26]. It provides the network with channels of content. A user in the network can choose which channel he or she wants to receive (just like choosing television channels). The initial user(s) will receive the data from a ”media injector”, which is basically a streaming server. Then, subsequent users will receive the media from peers already watching the stream. Thus, a P2P network is created to distribute (near) real-time streamed content.

What differentiates Daedalus from other P2P streaming networks is the fact that it has multiple channels in its network. Thus it differs from on-demand video where users choose what episode/movie they want to watch, but it is similar to a TV network where multiple TV channels of broadcast television are transmitted by a network. The peers communicate with each other by sending messages and if a peer fails, the peers which were receiving data from it will discover this failure and try to find another peer. This also means that the peers do not have to communicate with a central server, but can rely on each other. The system only consists of two types of components: the media

[3]The maximum number of hops were later reduced to 4 with the introduction of leaf nodes and ultrapeers.

injector and peers[4].

2.3.5 Joost

Joost is a P2P service for sending (standard) television-quality video on-demand. Development started in 2006 and was created by the authors of Skype and KaZaA [6, p. 25][5].

The procedure followed when a peer wants to watch Joost content is similar to that of Skype and KaZaA. The client contacts a super node, which handles control traffic and directs clients to peers. The client attempts to download the content from the peers. However, according to Hall, et al. [27], the majority of video traffic comes from Joost's own servers, at least two-thirds of the content. This means that Joost is a partial P2P system. However, the reason for this is due to the fact that it is a video *on-demand* system. This means that the number of users watching a specific video may be few, as viewers are spread over all of the available content. As is the nature of on-demand content, people want to look at what they prefer *when* they prefer it and that is what separates on-demand viewing from ”regular” television.

2.3.6 Spotify

Spotify is a music streaming service developed by the Swedish company Spotify AB [28]. Spotify allows users to stream unlimited amounts of selected music and is a legal alternative for listening to music [29]. Spotify comes in different versions which are currently available in 7 different countries: a free version which contains advertisements and a premium version which is free of advertisements and supplies content at higher bitrates. One of the features of Spotify is that it has virtually no buffering delay [30]. Spotify is available on computers and for premium users also on mobile phones.

Spotify uses a proprietary network protocol which streams audio of 96-320 kilobit per second (kbps) bitrates [31]. The first piece of the stream is downloaded from Spotify servers (which are placed in a CDN) while P2P is used for the rest. The client switches back and forth between Spotify servers and peers as needed, the focus being to reduce delay, where the initial delay is considered to be the most critical - however, if the client is running out of content and the peers are not providing it the Spotify servers or CDN are used. Clients keep a persistent TCP connection to all current servers (both Spotify's servers and peers) while logged in as this enables the servers to constantly see if the users are connected to the network. (Note that the mobile client does not use P2P, but downloads all content directly from Spotify servers.)

According to Kreitz [31], the goals of the Spotify P2P architecture is to minimize bandwidth charges, hardware costs, latency, and to avoid shutter. The P2P network is built as a unstructured network (see section 2.3.1) where all peers are equals (e.g. it has

[4]However, in the Daedalus prototype [6] there was also a Master Node which would maintain information about all the peers within the P2P network which, in turn, it would relay to newly joined members. Matris and Strikos claim that it is not needed in an actual system based on the Daedalus architecture

no supernodes, unlike Gnutella) and clients only download data they need. In contrast to BitTorrent, Spotify has only one P2P network for all tracks [32]. Peers find each other using a partial central index (making it a *hybrid* P2P system) where a small number of peers (approximately 20) per track are maintained. Clients also broadcast queries in a small neighborhood and perform limited broadcasts for local peer discovery (in Local Area Networks).

Using a partial central index limits the number of peers participating in the distribution of the content. The advantages are that not all peers expend their bandwidth for distributing content and the tracker does not need to keep a (potentially large) list of peers. Another advantage is that since it is an unstructured network, the tracker does not try to organize the peers in a hierarchy, thus saving tracker resources. The disadvantages are that the peers will not be organized in any optimal order based on available bandwidth or content. Spotify has performed sufficiently well that it has amassed a large user base in two years (7 million users as of 2010 [33]), hence it apparently does not need such optimizations. However, for video streaming services - which are the target of this thesis (unlike the audio only Spotify service) - much more bandwidth is required which means a wider peer base may be required in order to efficiently supply enough content for all requests.

Clients connect to an Access Point which will handle authentication and encryption for clients and forward requests to backend servers. Spotify clients are quite clever when downloading content. They ask for the most urgent pieces first and if a peer is slow, then it will re-request the pieces from new peers. If a client has low playout buffers (less than 3 seconds worth of data), it will download the missing content from the central server as well, and if the buffers are very low then it will stop uploading in order to preserve bandwidth for making requests and acknowledging download traffic.

2.4 Additional relevant background information

This section describes other relevant background information which this thesis will utilize.

2.4.1 Swedish home network speeds

When building a streaming technology, it is important to know the speed with which users can potentially receive data. This is relevant for both a streaming and P2P technology.

A quick check on the Swedish website Bredbandskollen.se [5] shows that out of 146,000 users who have used the site's self-measurement tool, the average connection speeds are roughly 5.6Mbps uplink and 15.3Mbps downlink [34]. This value is derived from the users testing their connection speed on the site during February 2010.

Assuming that a video stream requires 1Mbps, the average user should be able to redistribute content to 5 other destinations.

[5] An independent organization for Swedish consumers to evaluate their Internet connection

2.4.2 IPv6

IPv6 is the new version of the Internet Protocol [35]. It is designed to replace IPv4 [36] by overcoming limitations in the protocol. One improvement (and probably the most important one) is the extended addressing capabilities: addresses in IPv6 are 128 bits long, compared to 32 bits in IPv4, which means a total of $3.4 * 10^{38}$ addresses in IPv6 compared to $4.3 * 10^{9}$ in IPv4. Other improvements include eliminating some obsolete and unused header fields, improved support for extensions and options in the protocol, flow labeling capabilities for labeling traffic and security capabilities to support authentication, data integrity and confidentiality.

As of February 2010, Geoff Huston's daily IPv4 report predicts that the last of the IPv4 addresses will be depleted by September 2011 [37]. However, studies show that the total world usage of IPv6 is still less then 1%[38]. This reveals both a problem and an advantage of using IPv6. The problem is that the majority of Internet users still do not utilize IPv6, but the advantage is that using IPv6 in new technology could produce a new killer application which would increase the usage of IPv6.

In Java, IPv6 has been implemented transparently and is part its networking stack. The Java networking stack will first check if IPv6 is supported by the underlying OS and if it is it will try to use it. Even legacy systems built on IPv4 can use the IPv6 automatically without any porting needed [39].

2.4.3 Testing of P2P systems

Testing of P2P systems can be done in different ways, but the important requirement is to have a large user base to test on. Thomas Silverston and Olivier Fourmaux compared different P2P live streaming players by during the International Federation of Association Football world cup [3, 8].

By placing computers of average power in different locations (on the university campus with a powerful Internet connection and a couple average home connections) they were able to collect a lot of data on how several different P2P programs behaved and how much traffic is coming from which source. For their surveys they used tcpdump [40] to collect time-stamped packets. In addition to its own capture file format, Wireshark can deal with a number of other file formats including that used by tcpdump.

Wireshark is a network protocol analyzer which enables users to deeply analyze the network traffic generated and received by their computer [41]. This tool can be used to identify transmission sequences along with overhead traffic.

2.4.4 Episodes

This section will give an overview of the content provided by the Kanal5 video on-demand service [12].

The majority of the content is encoded using a H.264/MPEG-4 Part 10 or Advanced Video CODEC (henceforth referred to as H.264) compressor-decompressor (CODEC) while some of the older content is encoded using On2's TrueMotion VP6 (VP6) (see section 2.4.5). Free content is provided at at up to 800kbps video + 96kbps sound while the subscription based content is provided at up to 2200kbps video + 128kbps sound. A free 45-minute episode is approximately 300 megabyte (MB) in size while a comparable subscription based is around 780MB. The majority of episodes are around 45 minutes long[6].

2.4.5 CODECs

The majority of content provided by the Kanal5's video on-demand service is provided in H.264 format including all new content, thus this section will only cover the H.264 CODEC and not the older, less used VP6. For more information on VP6 (developed by On2 Technologies [42], acquired by Google Inc. [43]) see [44].

H.264 is a block-oriented motion-compensation-based CODEC standard developed by the Joint Video Team [45]. This CODEC is commonly used in applications such as High Definition-Digital Video Disc (HD-DVD), Blu-Ray Disc, videos from Youtube and the iTunes store, etc. This CODEC exploits the fact that the difference between two consecutive frames in most movies is the result of either the camera moving or an object in the frame moving. This means that the actual difference between two frames, in terms of the information stored, is quite small. This fact can be exploited by storing the full information of a frame every, for example, forth frame (also known as a key frame), and by having the frames in between only store the differences between the two adjacent frames (e.g. the information needed to transform one frame into the next frame). With this optimization, the H.264 standard can match the quality of the Moving Picture Experts Group-2 standard (used in Digital Video Discs) using half the frame rate [46].

[6]Aside from full length episodes, kanal5play.se also contains short clips which are only 2-5 minutes in duration, however these are not relevant to this thesis.

Chapter 3

Loree: A system designed for On-demand P2P television

This chapter describes the design of Loree, a P2P system designed for TV content providers' on-demand websites. This P2P system is designed to handle peak hour patterns as described earlier in section 2.1.2, i.e., where viewers access on-demand websites to watch TV episodes shortly after they have been broadcast on TV and released on-line. Statistics from Kanal5 show that on the day an episode is released, users also watch previous episodes from the same show (see appendix A). Based on this information, a system has been built to reduce the load on the content providers' servers.

3.1 Loree overview

The Loree system is designed so that users themselves need to do as little as possible (in fact they only need to select the content of their choice). Loree is not meant for live streaming, but rather for content which is pre-recorded (complete episodes). Loree is also designed to reduce the required server bandwidth resources.

3.1.1 Resources used for development

The Loree prototype was developed in Java. Java was chosen as a development environment since the author was already familiar with it. The following resources were used during the development of Loree:

- Subversion - a version control system for source code management [47]
- Java Developer Kit 1.6.0_18 [48]
- Java RMI [49]
- Java Eclipse - An Integrated Development Environment [50]

An advantage to using Java for the development is that its networking stack has transparent support for IPv6 (see section 2.4.2), meaning that the Loree system will not need to be ported or rewritten to support the "new" technology.

3.1.2 Prototype implementation using Java RMI

For the implementation of the prototype, this thesis project uses Java Remote Method Invocation (RMI). RMI was incorporated so that each client, content server, and tracker has its own registry server to which the other components can connect. A registry server is a component in the Java RMI application programming interface (API) which keeps track of objects which are available remotely from Java virtual machines (JVMs, abbreivated simply as VMs in this thesis).

The reason for using Java RMI in the implementation is that it simplifies the communication between clients and tracker thus avoiding the need for direct low-level socket communication. Instead, clients communicate with the Tracker and each other by using remote method calls and send objects instead of packets. The advantage of this simplicity comes at a cost, namely the communication overhead involved in Java RMI (which makes the remote invocations of procedures transparent) reduces performance [51]. This means that in future work, the implementation could be optimized by using socket programming [52, 53].

When a Loree client connects to the tracker; the client connects to the tracker's registry server on a predefined port and looks up a remote object using the remote reference "tracker". The registry server will return this tracker as a remote object (see figure 3.1). The object contains an API which can be used to perform tracker operations. (Note that the functions listed in following chapters are not the only functions available in the remote objects. These other functions provide thread synchronization, Java specific operations, and other internal functionality and are not described unless specifically relevant.) This tracker API includes the following functions:

- registerPeer(Peer peer) - registers the peer as a client in the Tracker user list;
- unregisterPeer(Peer peer) - tells the tracker that the peer is leaving the network; and
- whereCanIFind(String videoName, int partNumber) - asks the tracker for a peer from which to download a specific part of a video.

The parameters and returned objects sent must be serializable. The serializing interface encodes Java objects, enabling these objects to be saved outside the VM as files or to be sent through sockets just as any other data [54].

More detailed information regarding Java RMI and its functionality can be found in [49].

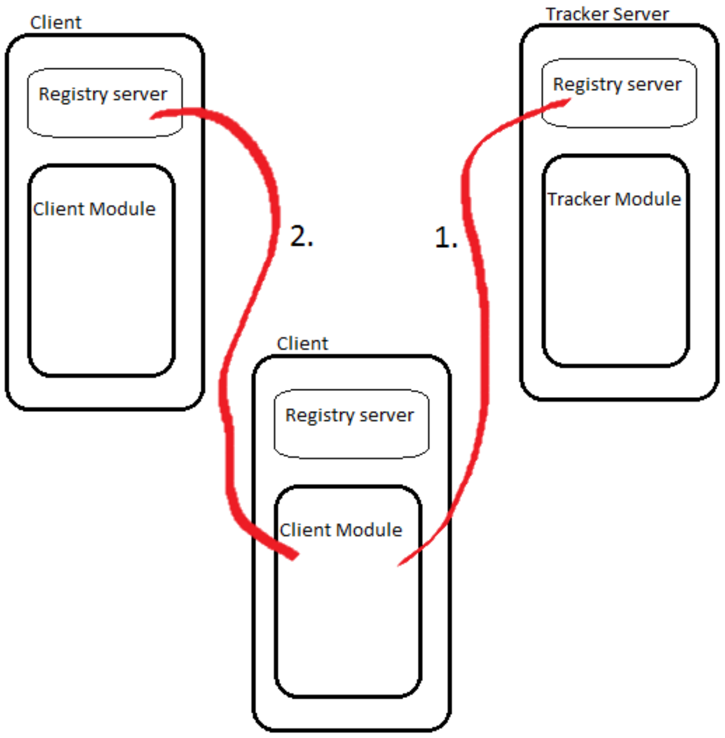


Figure 3.1: 1. A client connects to the tracker's registry and retrieves the remote tracker module which it will then use to invoke functions. 2. Clients connect to other clients' registries and retrieve the remote Sender module object which is used to transfer files.

3.2 Key features of Loree

The main goal was to design a P2P system which could distribute on-demand television content while reducing the load on the content providers server(s) during peak hours. This leads to a prototype called Loree. The following paragraphs describe some of the key features of Loree. An example setup of the Loree network with its components can be seen in figure 3.2.

- *Structured tree hierarchy hybrid P2P network*
  As described in section 2.3, there are different types of P2P architectures. Even though pure P2P systems avoid the need for a central monitoring server, they prove challenging when trying to optimize the flow of traffic. Building a structured, hybrid P2P system enables the content provider to control how the traffic traverses the network and thus enables peers with the highest available bandwidth to be placed higher up in the hierarchy (with the content server at the top of this hierarchy). Structured P2P systems work well for live streaming when it is known in advanced that there will be many viewers watching the same content at the same

time [3]. That said, the main drawback of a hierarchical system is the uncertainty of what people will watch and when they will watch it. Since viewers can choose to watch their choice of content when they want, it is not certain that each viewer will stay for the entire episode. In a hierarchy, peers depend on each other and if parent peers leave this can cause QoS problems for their children. For these reasons in this thesis project we assume that this problem can be solved by suitable communication between the peers.

Due to the advantages of an hierarchical system, Loree will use a tree hierarchy to distribute content in order to reduce the content server's traffic load during peak hours (as discussed in section 2.1.2).

- *One Tracker to rule them all*
  For a hierarchical system to work, there needs to be some sort of organization. In section 2.3.1 there were examples of different solutions for this. However, viewers of on-demand content are unpredictable in terms of how long they stay in the system, if they watch the same content as others, etc. This is of course in contrast to live content viewers who all have to watch the content at the same time. Due to the potential for clients to enter and leave the system independently, a system where the clients organize the hierarchy themselves will not be sustainable nor scalable. Therefore, Loree uses a Tracker whose task is to organize the hierarchy in an optimal way with regard to the available bandwidth & content and to communicate frequently with clients - so that if a client leaves, those clients who depend upon this client will not experience reduced QoS.

- *Content Server*
  The content server is the source of media content in the Loree system. The content server is nearly identical to a client with the exception that the content server is labeled as a content server and it has all of its content loaded in advanced. This provides flexibility since it is possible to have multiple content servers in the same network that operate completely independently. The tracker can choose to utilize these content servers however it sees fit (based on parameters such as available bandwidth). These content servers do not need to have the same content available on all of them, thus they can more flexibly be organized in the network hierarchy (in order to serve the expected requests).

- *Clients*
  Clients are the core components in any P2P system. The difference between Loree and pure P2P systems is that Loree's clients ask a tracker for instructions about where they should download their content from. A client has two main separate threads: one acts as a server (e.g. distributing content) and one downloads the desired content. The download part must keep track of which part of the episode being downloaded the client wants to watch next and ensure that it downloads the relevant part of this episode (thus the next expected portion of the episode has the highest priority for downloading).

Part of the client's functionality is to ask for additional content to download after the episode requested has been completely downloaded. Doing so reduces the load on the content servers since other clients will have multiple sources to download content from. The extra content that is downloaded should be content which is predicted to be requested in the near term (by this client or other clients).

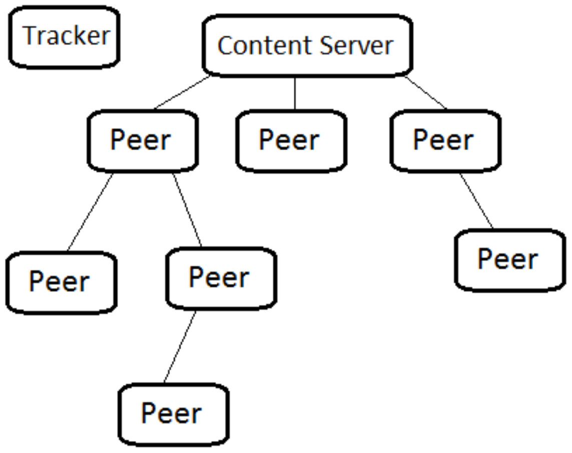


Figure 3.2: Example of a tree structure in the Loree system. Note that the peers can be organized in different ways and that they can retrieve content from multiple peers for different pieces of content. There can be multiple content servers in the network and the tracker is on a separate location and communicates with all peers and content servers.

3.3 Client and Content Server

The client's server and the content server are essentially the same. Thus the tracker can view the content server as a regular client that has a very large portion of the content already available. As a result, the tracker places the content server at the top of the hierarchy (see section 3.4). Advantages of sharing this server code include simplicity in constructing the content server and simplifying for the tracker the use of multiple content servers (with perhaps different content).

In the current prototype, each client only downloads one episode at a time. However, the client should be able to download multiple episodes if these episodes are predicted to have a high demand in the near term. This gives a two-fold advantage: (a) if one could predict that a viewer watching a certain episode will also watch the following episode then the viewer will experience a smoother transition between watching the episodes (e.g. he or she will not have to wait for downloading the next episode since it is already downloaded) and (b) If an episode is experiencing high demand then downloading another episode will distribute the bandwidth load from the servers to the peers. The load distribution is also true for case (a). How to determine which episodes will be most likely to be requested during which hours will be part of future work (see discussion in section 3.7) as this is affected by editors and when episodes are broadcast on television.

3.3.1 Client operation

The basic client operation is as follows:

- Register with the Tracker using the viewer's identification (ID)[1].
- Start a sender module (see section 3.5.1) which will keep track of the content that is downloaded and redistribute this content.
- Ask the Tracker for an episode. The Tracker will reply with a Video object containing information regarding the episode's size, number of parts, and other relevant information (for more information about video objects see section 3.6).
- The client will ask the tracker where to download individual parts of the episode from. Note that a client does not have to start by requesting the first part since the viewer might want to start watching from a later point in the episode. This also enables the client to download content in different order, see section 3.6.3.2.
- For each part of the episode, the client will connect to the target peer's Sender module (see section 3.5.1) and call the getVideoPart() function which will return the requested content (e.g. download it).
- After the full episode has been downloaded, the client will ask the tracker for more content to download. The content which should be downloaded is that which is predicted to be in high demand (see section 3.7). Note that this is not implemented in the prototype, but is meant as a problem for future work in order to reduce load on the content servers.

Also note that in parallel with the above steps, the content already downloaded can be viewed by the viewer.

If a target peer is not available to download content from (e.g. it suffers unexpected problems which make it unavailable) the peer requesting the content will to ask the tracker for a new peer to download from. In this case, each client would receive a reference (supplied by the tracker) to the content server providing the desired content. If a peer were to fail, the client can then ask the content server directly for the part it was trying to download, in order to avoid waiting for a reply from the tracker before continuing to download the content. This would reduce the chance of buffer under-run (i.e., not having content to play) which would ruin the user's viewing experience. This also means that the clients keep references to relevant content servers in case of such emergencies.

[1]Each viewer has some way of identifying them. In the current prototype version the ID is a command-line argument where each peer selects a unique name. However, there is no verification stopping multiple clients from using the same ID. In future work, this could be implemented using viewer's identification to kanal5play.se as this would give each user an unique ID.

3.3.2 Client implementation

Each client starts a Java RMI registry server (TCP port 5011 by default). This server enables peers to connect to this peer when they want to access remote objects such as the Sender. Next the client connects to the Tracker registry and retrieves a remote Tracker object (see section 3.1.2) and registers with the Tracker. After the client has registered with the Tracker, it will create a Sender thread (encapsulating the Sender module, see section 3.5.1) which will keep track of downloaded content. The SenderModule is also an RMI remote object, thus when peers want to download something from this peer they will retrieve the remote Sender object and use it to retrieve content by calling a get-function. Next, the client asks for the video content it specifically wants from the Tracker by using the remote Tracker object it retrieved earlier. The Tracker returns a Video object which contains information about the video (e.g. how many parts it has, length of the video, etc., see 3.6.2). Using this video object, the client knows how many parts it needs to download and can decide which part it should download first and which ones can be downloaded later (i.e. if the viewer wants to watch from the middle of the episode, then the client will download the appropriate part(s)). The client will download all of the parts of an episode with the priority being the parts following the part the viewer is currently watching. As parts are downloaded the sender module keeps track of these parts and makes them available for redistribution to other clients.

When the client has finished downloading the desired episode, it will ask the tracker for which other episode is most likely to be requested at this time and download it. By doing this, other clients arriving later will be able to download this episode from this client without the client experiencing poor QoS while the content server(s) will not need to download the content to these new clients, hence reducing their load.

3.3.3 Content Server

As previously mentioned, the content server is very similar to the client. The main similarity is that both clients and content servers use the Sender module to distribute content. The difference is that the content server does not necessarily download any content via the Loree network. It loads the content in advanced and registers this content in it's Sender (see section 3.5.1). After the content is registered with the Sender, the content server will register with the tracker and notify it of the content that it has available for distribution and the Sender module will do the rest. Basically, the content server is a slim-down version of the client. Note that this content server can be replicated and need not be running on only a single physical computer.

Each content server loads just one piece of content (i.e. one episode) which it distributes. However, due to the design decision that each content server is simply another peer in the system it is possible to have multiple content servers for each episode. These content servers can share the same server hardware but each uses its own ports.

In the prototype the content is loaded from disk and divided into evenly sized pieces based on a pre-defined number and put into PartOfVideo objects which are then loaded into the Sender module (see section 3.6.3). The content server's RMI server operates by default on TCP port 5001, but is changeable with command-line arguments.

What is interesting is that clients can easily be used as content servers. Instead of needing to have content saved on disk, it is possible to distribute the load between (geographically) separated servers by having a "client" join the network and download the content, then continue to operate as a dedicated server. For example, an ISP might operate a set of such "clients" in order to reduce the cross-ISP interchange traffic by supplying peers in their own network with copies of the content from one of these "clients".

3.4 Tracker

The tracker's role in the system is to keep the peers organized in terms of where they receive their content from. The tracker maintains a list of all peers in the system and what content they have available in the form of PartOfVideoInfo objects (see section 3.6.3). The tracker also contacts each client frequently to see if the client is still operational. Instead of the tracker informing each peer of how to handle their traffic, each peer individually asks for the content it wants from the tracker which replies to each individual request[2]. Simply put, the tracker is primarily a database and much of the computing is distributed to the peers, thus the trackers computing resources are dedicated to simply tracking where instances of PartOfVideoInfo objects are.

The tracker internally organizes the peers in a tree hierarchy where the content servers are placed at the top of the hierarchy, followed by the peers with the most amount available bandwidth (and the specific content) available (see figure 3.2). Since the Loree system is focused on on-demand content, the tracker needs to build a tree for each episode available in the system. This is in contrast to live content where most often there will be only one stream of content available in a network.

Having a tree hierarchy which contains every node in the system could potentially lead to extreme loads on the tracker who must sort the tree. One alternative, as implemented by Spotify (see section 2.3.6), is to have partial central index where the tracker only maintains a limited number of peers in its hierarchy. In future work, this needs to be tested on a large user base to verify if this is required in order to maintain a high throughput and QoS, or if the tracker can handle potentially large trees.

The tree structure could be further optimized based on parameters such as location and which ISP's network the peers are located in. Basing the structure on location could reduce delay when downloading the parts. Which ISP's network peers are connected to, on the other hand, can make a difference for ISPs who could benefit from reduced costs (MPS Broadband AB partly base their algorithm on this factor, see section 2.3.2).

Having the tracker separate from the content servers gives the advantage that if content servers or peers experience unexpected problems such as unavailability, the tracker can continue to keep the network of peers operational even if a content server is

[2]This may generate a lot of overhead traffic in terms of messages to the tracker which could be reduced by having each peer ask for multiple parts of content at once.

unavailable. This is a great advantage in terms of server maintenance, availability, and reliability.

The prototype system only works with one piece of content at a time and, in order to complete this thesis project on time, the author chose to implement a simple version of the tracker's organization algorithm in the Loree prototype. The tracker will, for a request of content from peer A, return information about a peer B; where peer B is the peer with the highest number of available upload slots amongst the peers which have the specific content available. Therefore, no explicit tree hierarchy is built, but instead the tracker's list of peers is simply traversed each time a request is processed. Note that this is suboptimal but simple to implement and that it will be replaced in future work.

3.4.1 Tracker implementation

The tracker implementation consist of three parts (see figure 3.3): the RMI server, the tracker module which is the remote object the RMI server allows peers access to, and a separate monitor thread which monitors all peers and makes sure they have not left the network unexpectedly. The RMI server is by default initialized on TCP port 5000[3], through it peers reach the remote tracker object where different functions are available (see section 3.5.3). In parallel to the tracker, a monitor thread is initialized which goes through the tracker's list of peers every second and tries to connect to each peer in the list to verify that they are still operational and in the system.

Spotify uses an alternative approach in which each client maintains a persistent TCP connection to the tracker (see section 2.3.6). In their approach, when a client leaves the network (for any reason) the tracker will notice it quickly (due to their use of a short time between TCP keep alive messages). The major difference (in regards to this) between the Loree and Spotify trackers is that the Spotify tracker does not organize clients in a hierarchy, thus it has more available resources to maintain connections, while the Loree tracker organizes clients to gain throughput in the network. In order for the Loree tracker to see if clients are still in the system, a Monitor is used which frequently verifies that clients are operational by connecting to them (see section 3.4.2). The downside of using a monitor is that the monitor will not notice clients leaving the network as quickly as the Spotify tracker. It is not obvious which alternative is better, as one must consider the large amount of bandwidth required for video content in contrast to audio content. As part of future work, a comparison needs to be made between these two alternatives to see which is most useful for on-demand television and to determine how the choice of method affects the throughput and stability of the network.

[3]The default port for the content server's RMI server is TCP port 5001, which means the tracker and at least one content server can be run on the same machine.

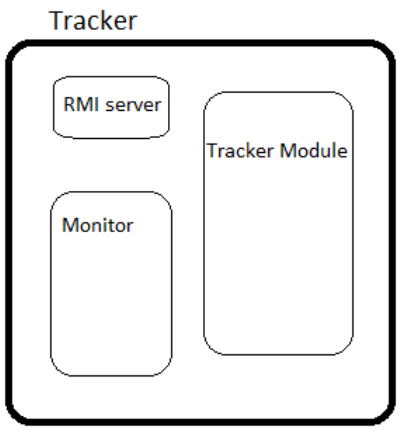


Figure 3.3: Internal components of the tracker.

3.4.2 Monitor

The monitor is a separate thread in the tracker which verifies that all peers are operational. In the prototype, this is done once every second. The monitor retrieves the tracker's peer list and connects to each peer's RMI server. When it connects to it, it tries to retrieve the client's sender module to verify that it is operational. If an error occurs during this step, then it is safe to assume that the client is malfunctioning - in which case the monitor unregisters the peer from the tracker.

The reason the author chose this implementation is that an ICMP ping (see ICMP echo message, also known as ping operation [55]) would be successful as long as a computer at the target IP address is turned on and connected to the Internet, but this response would not indicate if the Loree application is running. To verify that the target client is fully operational, its RMI server must be working and its sender module must be available. The downside to this approach is that the amount of overhead data sent over the network due to RMI operations will increase as the number of peers in the system grow.

3.5 Modules

One of the main design features of the Loree system is its modular architecture. The idea behind using a modular design is to build standardized modules which can dynamically be re-used in different components (e.g. client and server). In figure 3.4 the construction of the client and server are shown. As the figure shows, the content server is almost identical to the client since it uses the same modules. However in contrast to the client, the content server has all of the content loaded in advanced. The tracker stands out as its only modules are an external interface for verifying that the clients are operational (the Monitor, see section 3.4.2) and an algorithm module for organizing the clients.

The following sections will describe in further detail how the different modules are built up and also describe some key functions available in the modules. Information regarding the object types can be found in section 3.6.

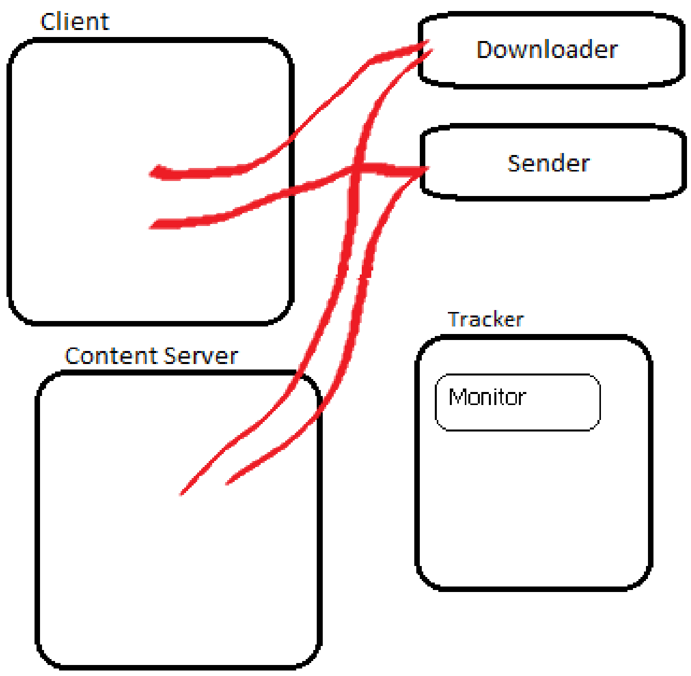


Figure 3.4: The Client and Server are put together using the different modules. Note, however, that they are essentially identical since they use the same modules. The difference is that the content server has all the content available from start.

3.5.1 Sender module

The client and content server share a module named Sender which acts as a server for peers. The Sender module is a remote object whose primary function for inter-peer communication is to send content. When peer A connects to peer B, peer A will retrieve B's remote Sender object and use it to download the content it specifically wants. Table 3.1 shows some of the key functions available in the Sender module.

Table 3.1: The Sender Modules remote interface functions.

Function name	Function operation
getVideoPart(String string, int partNumber)	Used by peers to download a specific part of a video. Returns the requested PartOfVideo. If the specific part is not available, it will return null.
addContents(PartOfVideo video)	This is used by the Downloader module internally within the peer to add downloaded Content to the sender so that it can redistribute it.

3.5.2 Downloader

The downloader module was designed to isolate partial download of video parts and in order to enable downloading multiple parts in parallel. This module is implemented as a separate thread which is created and run with the sole purpose of downloading one PartOfVideo (see section 3.6.3). By separating the download of each part into threads, the order of downloads can be more flexibly controlled and organized.

The downloader operates in three simple steps: it connects to the peer specified by input variables and retrieves its remote sender module and downloads the PartOfVideo object requested, it loads the remote sender of the local client and saves the downloaded content in the sender object, and finally it registers the downloaded content with the tracker indicating that the local client now has this content available for redistribution.

In the prototype implementation of the client a maximum of 10 downloader threads are operational at any point and these are run in a sequential order for simplicity (this number can be modified in order to optimize speed vs. Central Processing Unit (CPU) power and memory consumption, but this is not relevant at this point since the downloading should be done following patterns in future implementations, see section 3.6.3.2).

3.5.3 Tracker module

The tracker is implemented as a remote RMI object. When peers retrieve the tracker object, they will have access to some specific functions (as can be seen in table 3.2) which will give them the information needed to download the content they want.

When a peer invokes the whereCanIFind() function, the tracker will find the most suitable peer from a dynamically built tree of peers and return the Peer object associated with the most optimal peer to the requesting peer. The tree needs to be rebuilt frequently so that when the whereCanIFind() function is called, the information will be up to date.

Note that instead of the tracker sending out updates to all clients, the clients will ask the tracker for the location of content when they want. The main advantage of this approach is that this reduces the workload on the tracker since it does not need to

Table 3.2: The Trackers remote interface functions.

whereCanIFind(String videoName, int videoPart)	Returns a peer object to download a specific part of a video from
registerPeer(Peer peer)	Registers a peer to the tracker
unregisterPeer(Peer peer)	Unregisters a peer from the tracker (e.g. a ”nice” departure from the network)
tellMeAbout(String videoName)	Returns a Video object with information about the size of the video, numbers of parts, etc.
registerContent(Peer peer, PartOfVideo vidPart)	Registers with the tracker that the peer has vidPart available for distribution
addContent(Video video)	Registers a new episode to the tracker
whichOtherVideoShouldIDownload()	Ask the tracker for which other episode should be downloaded for helping the P2P network after the requested episode has been downloaded completely. (Note that this function is not implemented in the Loree prototype)

update all peers with new targets to download from, hence the tracker can simply reply to requests and save resources for calculating the client tree hierarchy.

The algorithm for optimizing the hierarchy of the system is based on multiple parameters (e.g. available bandwidth, quantity of content, etc.). Finding an optimal solution is outside the scope of this thesis project. There already exists some solutions for this, such as the solution designed by MPS Broadband AB as noted in section 2.3.2. However, alternative algorithms should be covered in future work. In the prototype, the tracker returns the peer with the most available bandwidth out of the peers which have the content available. Note that this is sub-optimal since this calculation occurs each time a peer calls the whereCanIFind() function of the tracker. For the prototype a more optimized approach was unnecessary as tests were performed with a small user base and thus a pre-calculated hierarchy was not necessary (as can be seen in chapter 4).

3.6 Classes in Loree

The different classes used in the Loree system are discussed in this section, specifically what they contain and why and in what situations they are used. Since these classes are all remote objects sent between peers, they all implement the serializable interface. As described previously in section 3.1.2, serializing a Java object enables it to be saved as a file or sent through sockets.

3.6.1 Peer class

The tracker keeps track of all peers currently in the system, and each peer is identified using the Peer class. This class contains specific peer information such as the peer's name, IP address and port, content they have available, upload bandwidth and if this peer is a content server or not. Note that content servers are also registered as regular peers in the system, as this simplifies how the tracker handles organization of peers and also the communication between tracker and peer since the tracker can send a peer object as its response. Thus, the peers receiving a peer object does not need to account for whether the peer they are downloading from is a content server or not.

In order to reduce the amount of traffic sent in the system, the prototype implementation has two separate peer classes: *MiniPeer* and *Peer*. The Peer class inherits from MiniPeer. The difference is that MiniPeer contains only the basic information needed for peers to be able to connect to other peers. Thus the Peer class, in addition to the information in MiniPeer class, contains a list of content this peer has available in the form of PartOfVideoInfo objects (see section 3.6.3). This reduces the amount of overhead traffic sent in the network, specifically between tracker and peers. In addition, this reduces the download delay when a peer initializes a download.

When a peer receives a MiniPeer object from the tracker, it contains the IP address of the peer and the port its RMI server is running on. This is all of the information that this peer needs in order to download a requested piece of content. The larger *Peer* class is used by the tracker to calculate the most appropriate MiniPeer object to send to a peer requesting a piece of content.

3.6.2 Video class

The Video class is a general object *describing* the content. In the prototype implementation, the class simply contains the name of the content and the number of parts the content was divided into by the content server providing the content.

This class is used by the tracker to keep track of all the videos in the system, but it is primarily used to inform client peers of the information regarding content. This includes how many parts the content is divided in to, so that the peers know how many parts to download. This class can be extended with further information such as individual video download patterns (see section 3.6.3.2) or related videos in order to prepare clients with the likely next episode that tier user might watch - for example, multiple episodes in a series might be viewed directly one after another.

3.6.3 PartOfVideo and PartOfVideoInfo classes

PartOfVideo objects are created when content is loaded into the Loree system. When a content server loads content, the media content is divided into suitably sized pieces (see section 3.6.3.1) and each piece is placed into a PartOfVideo object. This class inherits from PartOfVideoInfo which contains all the relevant information about this video part, such as the name of the content and which part of the video this object contains. PartOfVideo, in turn, contains the actual content in a binary format (native Java byte array). Both classes implement the serializable interface [54] which enables the objects to be transmitted.

PartOfVideoInfo is used primarily by the tracker. The tracker keeps a list of all peers in the system and for each peer it maintains a list of PartOfVideoInfo objects so that it can keep track of what content peers have. When a peer asks the tracker for content to download, the tracker will examine the lists of locations and return an appropriate Peer object to the peer (see section 3.6.1).

When a peer wants to download a piece of content from another peer, it will connect to the peer and ask for the piece using a function call (see section 3.3.1). The function will return the piece of content in the form of a PartOfVideo object.

3.6.3.1 Size of PartOfVideo

To decide upon the size of each content part one must first consider the effects of using different sizes for these pieces of content. Assuming there is no limitation on bandwidth, the fastest way to transmit content between point A and point B is to transmit the entire content at once. Because peers have different upload and download bandwidth, in order for a P2P system to perform well the content needs to be divided up into multiple parts and these parts distributed amongst peers. This allows peers to download pieces from different peers (including content servers) in parallel. In order to minimize the initial delay between when a user requests certain content until he or she can start to watch it, smaller parts are better since they can be downloaded and played by a media player faster than having to wait for the download of large pieces. However, having too small pieces leads to high overhead in terms of requests and replies sent between tracker and peers.

BitTorrent [19] is one of the most common file sharing P2P protocols and is similar to Loree in several ways. According to the BitTorrent specifications [56], the most common chunk size for BitTorrent is 512 kilobyte (kB). Other common sizes are 256kB and 1024kB. The author used these same values as default values for the Loree system (later in section 4.4 the author examines the performance as a function of the size of the piece of video).

A possible extension to the Loree system is to identify patterns of how viewers choose to watch content, e.g. directly skip to a certain part of an episode, as discussed in section 3.6.3.2. For such scenarios, it may be relevant to have variable sized PartOfVideo objects based on what part of the content is (to be) viewed. If a certain part of an episode is known in advanced to be popular to be viewed first then the PartOfVideo objects in that part of the episode could have a smaller size (i.e. 256kB instead of 1024kB) in order to reduce the delay before viewers can watch the part of the episode that they ask for.

Assuming an H.264 CODEC (see section 2.4.5) is used for the media content, one could correlate the PartOfVideo sizes with the time between key frames. However, this discussion is outside the scope of this thesis project and will be left for future work.

3.6.3.2 Order of downloading content

A common way to download streamed content is to download everything from a certain point in the stream and onward. This occurs because a viewer might choose to watch an episode from an arbitrary point in the episode until the end of the episode, in this case everything after the selected starting point in time will have a higher download priority than the content in the episode before this point. No matter where the viewer chooses to start watching, the content with the highest download-priority must contain the portion starting from where the viewer has requested to start viewing, followed by the content that follows. However, after a certain amount of the content is buffered and ready to be rendered, other parts of the content could be downloaded. This is important to note, since viewers may choose to skip to some part of the episode and if this content has not yet been downloaded the viewer will experience a delay waiting for the download of this new content before they can continue watching.

With such non-linear viewing of an episode it is important to investigate just how viewers watch episodes and determine if there are patterns which could be used to predict future requests, hence changing the order in which content should be downloaded. Exploiting this knowledge would be useful to viewers because if they want to jump to these different places of an episode in the expected order, then they would not have to wait for missing content and could watch the episode without interruption(s). The viewer might even choose to jump backwards to an earlier point in the episode (that has not been downloaded yet), so the client could also download parts of the episode from different points in time, based on predictions of what the viewer will actually want to watch (see figure 3.5).

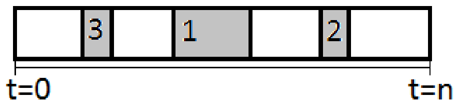


Figure 3.5: The timeline of a TV-episode. 1 represents the point of time where a viewer might choose to start viewing, and some content is buffered after that, 2 & 3 represent other points in the episode which are buffered in advanced because they are predicted to be skipped to by most viewers and t=n is the end of the episode. Note that the above figure is strictly for illustration purposes as these patterns have not yet been investigated.

In the Loree prototype there is no explicit limitation regarding the order in which content should be downloaded (although in the prototype content is downloaded sequentially for simplicity). This means that the prototype can easily be extended to download an episode in an order based on pre-defined patterns. Patterns for how episodes should be downloaded could be derived in the client, either by pre-defining a general pattern for all episodes or by having each client learn its user's viewing pattern to optimize the order in which it makes requests for content in the future. The patterns could also be stored in the Video objects (see section 3.6) based upon overall patterns of access by many clients. This would enable the patterns to be customized for individual episodes.

To find these patterns, a system could track viewer's patterns, then these patterns could be extracted and associated with individual episodes. Program editors could also define in advanced which parts are most likely to be watched. However, the optimal way to extract and utilize patterns is outside the scope of this thesis project and is left for future work.

3.7 Downloading other relevant episodes

After a user has downloaded the content he or she wants, the user's down-link capacity will be available for other purposes. If another episode is expected to have a peak hour shortly (before this user is expected to have finished watching the just downloaded episode) then this peer could download the episode that is most likely to be requested in preparation for the peak hour. By doing this, peers watching episodes other than the episode that is expected to be heavily demanded can help reduce the amount of traffic on the content servers by downloading and supplying the most frequently demanded content themselves. Another scenario is when a user chooses to watch the first episode of a series, their client might assume that the user will watch the following episode as well and download the following episode in advance of the user's request. Note that this feature may increase the average bandwidth utilization of the server(s), but it will help reduce the peak in throughput which occurs during the peak of requests, as shown in appendix A (for comparison, see figure 3.6).

Editors have a huge influence on what content will be requested in the near term. They could explicitly provide this information to the system so that peers can download this content when they are not busy downloading the content requested by their user. Specifications for how they could provide this information and how the clients could use this information remain questions for future work.

However, the advantage of clients downloading other content then the content requested by their users comes with a downside: if a client downloads content in preparation for peak hours or downloads the following episode in a series, but the expected behavior does not occur then this download was a waste of tracker and client resources. Investigations and answers for how to do this optimally are outside the scope of this thesis project. However these questions are interesting with regard to future work on the Loree system.

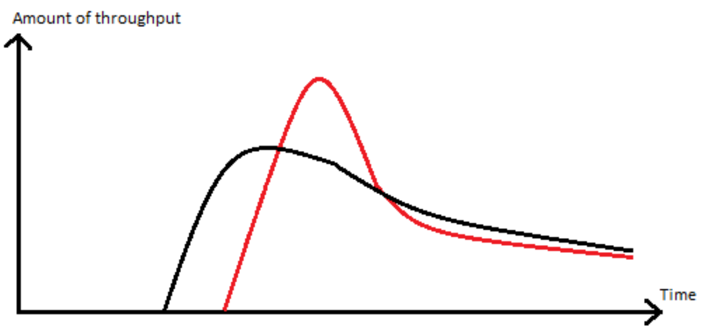


Figure 3.6: The figure shows how the amount of content sent by a server during peak hour could look like (compare with figure A.1 in appendix A). The red line represents how the throughput over time can look like when all viewers download the content from the server. The black line, on the other hand, represents how the peak could be reduced and the average throughput increased if peers download content in advanced and new peers joining the network download the content from them.

Chapter 4

Measurements

In order to evaluate the Loree prototype, the author designed and performed a set of experiments to evaluate Loree's network performance and the behavior of the various peers. In this chapter, these experiments along with the environment in which they were carried out are described.

Before presenting the experiments, it should be noted that the Loree prototype was designed to send chunks of data with a specified size, rather than media streams. This was a fundamental design decision. This decision was made to simplify both the implementation and experiments. Specializing the implementation to support any specific type of data (such as streaming content) or varying size chunks of data is left as future work. It is also worth mentioning that all communication and transfers are handled with TCP, as this is the default transport protocol for Java RMI.

4.1 Test environment

The experiments were performed under ideal conditions (e.g. the computers were directly connected by a switch) with one computer acting as both tracker and content server (this machine will henceforth be referred to as the server) and two other computers which were used as peers (see figure 4.1). The server was run on a Hewlett Packard (HP) Compaq dc7900 Ultra-slim Desktop computer with an Intel Core2 Duo E8400 CPU running at 2.99 Gigahertz (GHz), 3.46 Gigabyte (GB) of RAM, a Intel 82567LM-3 Gigabit network interface, and running a 32-bit version of Microsoft's Windows XP (Service Pack 3) operating system (OS). The peers (peer A and peer B in figure 4.1) were run on two HP Compaq dc7800p Small Form Factor computors with Intel Core2 Duo E6750 CPUs running at 2.66 GHz, 3.48 GB of RAM, Intel 82566DM Gigabit network interfaces, and running a 32-bit version of Microsoft's Windows XP (Service Pack 3) OS. The computers were interconnected using an Netgear Prosafe 8 port 10/100 Mbps full duplex switch model FS108 v2.

Most experiments were performed using three different sizes for each PartOfVideo object: *256kB*, *512kB*, and *1024kB*. The reason for considering these chunk sizes was described in section 3.6.3. These sizes of PartOfVideo objects will henceforth be referred to as *chunks*.

The experiments were performed using two episodes provided by kanal5play.se: a media file of 323MB and one of 735MB (see section 2.4.4). These files are representative of the content that would be used in a live implementation of Loree and are *currently* used on kanal5play.se. Additionally, the Monitor thread in the tracker (see section 3.4.2) was set to query peers every second.

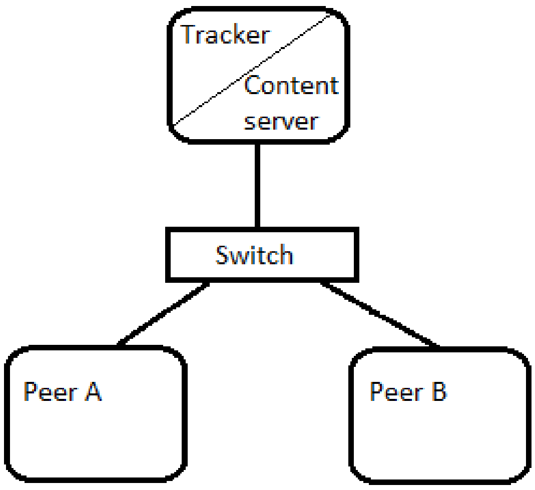


Figure 4.1: The network setup for testing the Loree system. Three PCs interconnected using a 100Mbps switch.

4.2 Delays

The overall delay can be decomposed into the parts shown in figure 4.2. The figure shows how the Loree implementation operates from a programming perspective. The actual transmissions on network packet level look different since Java RMI is built on top of TCP which generates overhead traffic such as 3-way handshakes. How this affects the Loree prototype is examined in sections 4.3.2, 4.4.2, and 4.5.

Some of these delays will be constant and some can even be approximated to zero. t1-t0 is an example of both since the only operations in this time period are some configuration parameters being loaded and set. The following delays will be constant as the operations performed in these time intervals are the same each time, even if the content differs: t3-t2, t8-t6, t13-t11, and t19-t18. The aforementioned delays are only affected by hardware (such as CPU power and available memory). Other delays are affected by underlying network variables such as packet loss, network delay, and bandwidth. Retrieving remote objects (t2-t1 and t14-t13) and sending requests and responses are directly affected the network delay. In our experiments t2-t1 can be approximated as t2-t1 = round-trip time (RTT, ICMP echo) between client and tracker, as the actual processing time on the tracker to grant the request of the remote object can be approximated as zero.

Delays when communicating with the tracker (t4-t3, t6-t5, t9-t8, t11-t10, t20-t19, and t22-t21) should be quite low (approximately the same as the network delay between the nodes) since messages sent to and from the tracker are small - as they only contain a minimal amount of information (see PartOfVideoInfo objects in section 3.6.3 and Video objects in section 3.6.2). In a production environment the tracker will be placed on high-performance servers with large amounts of available bandwidth. This means that theoretically the delays which affect the total delay the most are the interactions between peers (which are not content servers) as peers can be placed in random locations with variable computer resources and available bandwidths. The most time consuming delay should be retrieving a chunk (t18-t15) (even if the peer is acting as a content server). In the following section, test results are presented which show how the initial delay is affected by the chunk size.

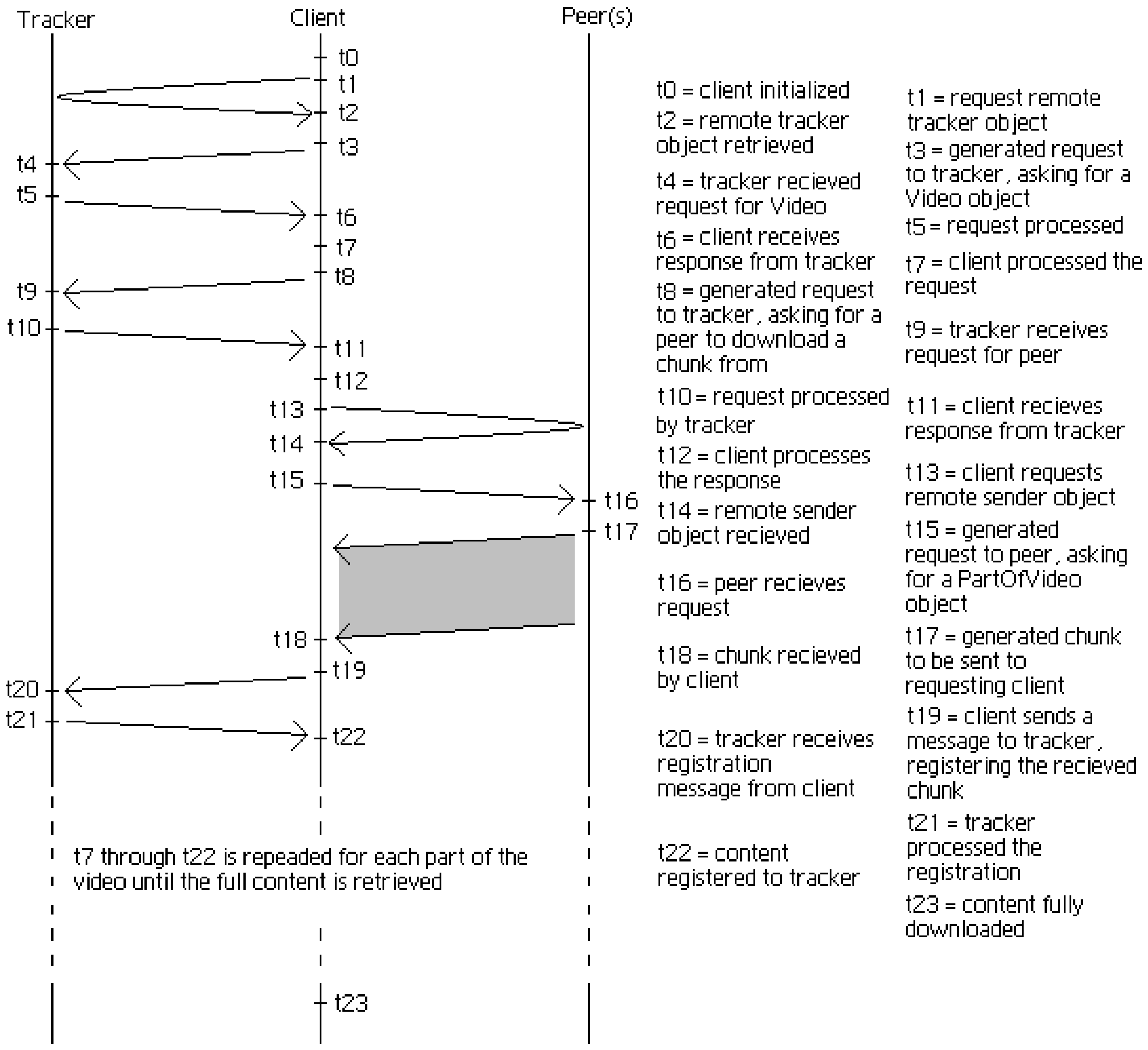

Tracker
Client
Peer(s)
t0
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13
t14
t15
t16
t17
t18
t19
t20
t21
t22
t7 through t22 is repeaded for each part of the video until the full content is retrieved
t23
t0 = client initialized
t1 = request remote tracker object
t2 = remote tracker object retrieved
t3 = generated request to tracker, asking for a Video object
t4 = tracker recieved request for Video
t5 = request processed
t6 = client receives response from tracker
t7 = client processed the request
t8 = generated request to tracker, asking for a peer to download a chunk from
t9 = tracker receives request for peer
t10 = request processed by tracker
t11 = client recieves response from tracker
t12 = client processes the response
t13 = client requests remote sender object
t14 = remote sender object recieved
t15 = generated request to peer, asking for a PartOfVideo object
t16 = peer recieves request
t17 = generated chunk to be sent to requesting client
t18 = chunk recieved by client
t19 = client sends a message to tracker, registering the recieved chunk
t20 = tracker receives registration message from client
t21 = tracker processed the registration
t22 = content registered to tracker
t23 = content fully downloaded

Figure 4.2: Time line diagram showing the traffic between a client, tracker, and another peer in the prototype implementation.

4.3 Initial delay

The *initial delay* is defined as the time from when a peer is started to the time the first requested part of the content has been downloaded and registered with the tracker (*initial delay* $= t22 - t0$). This time is measured starting from when a user request is made for specific content and ends when the client has received the entire first part of the requested content (i.e. the first chunk) and the client computer can start rendering the content in order to allow the user to watch the requested content[1]. Although this metric is not *as* critical in on-demand video as in live streaming, it is still an important aspect of the user's experience and should have an acceptable bounded value. The measured values for this initial delay is described for each experiment below.

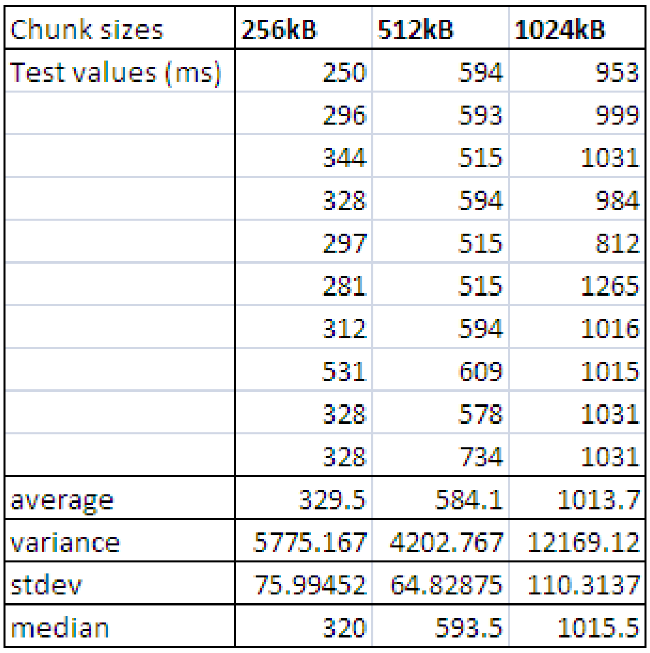
4.3.1 Ideal conditions

The following experiments were performed in ideal network conditions, according to the setup described in section 4.1 **without** any additional simulated delays (such as will be described in section 4.3.2).

4.3.1.1 Peer retrieving content from a Content Server

The first experiment was performed using a client (peer A in figure 4.1) to download a file of 323MB directly from the content server, with no other peers in the system. The experiment was repeated 10 times in order to get representative values. As can be seen in table 4.1, the median initial delays $\pm$ standard deviation for the three different chunk sizes were $320 \pm 76.0ms$, $593.5 \pm 64.8ms$, and $1015.5 \pm 110.3ms$.

Table 4.1: Performance of the Loree prototype, measuring initial delay in milliseconds (ms) when a client downloads content of 323MB from a content server.

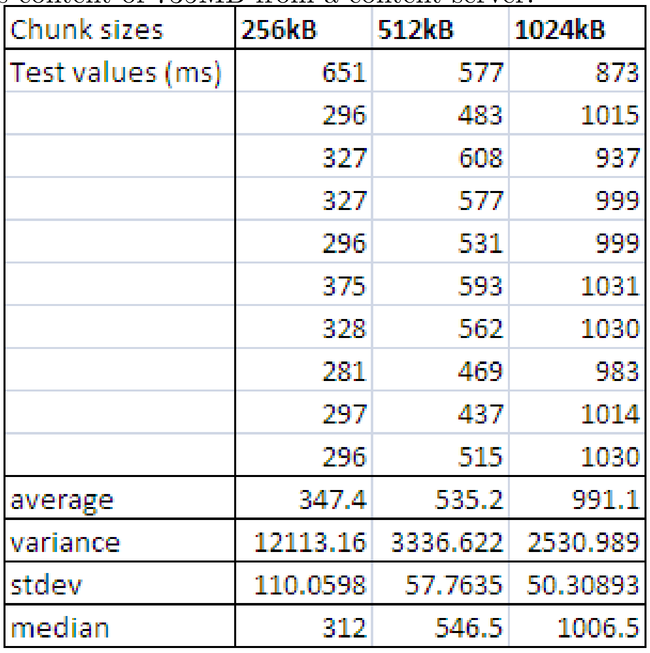


Chunk sizes	256kB	512kB	1024kB
Test values (ms)	250	594	953
	296	593	999
	344	515	1031
	328	594	984
	297	515	812
	281	515	1265
	312	594	1016
	531	609	1015
	328	578	1031
	328	734	1031
average	329.5	584.1	1013.7
variance	5775.167	4202.767	12169.12
stdev	75.99452	64.82875	110.3137
median	320	593.5	1015.5

[1]Of course the client can start to render the content before the end of the chunk has been received, but this introduces a risk that there might be an interruption in the playout of content if the remainder of the chunk is not received in time.

The same experiment was then performed using an episode of 735MB. The results from this experiment are presented in table 4.2. The median initial delays $\pm$ standard deviation for the three different chunk sizes were $312 \pm 110.1ms$, $546.5 \pm 57.8ms$, and $1006.5 \pm 50.3ms$. One can observe that the initial delays when downloading content of 323MB and content of 735MB are statistically equal. This means the initial delay is not affected by the size of the content nor the total number of chunks to be transferred. However, the initial delay is directly related to the chunk size, since we have defined the delay to include the time required to download the entire first chunk.

Table 4.2: Performance test of the Loree prototype, measuring initial delay in ms when a client downloads content of 735MB from a content server.



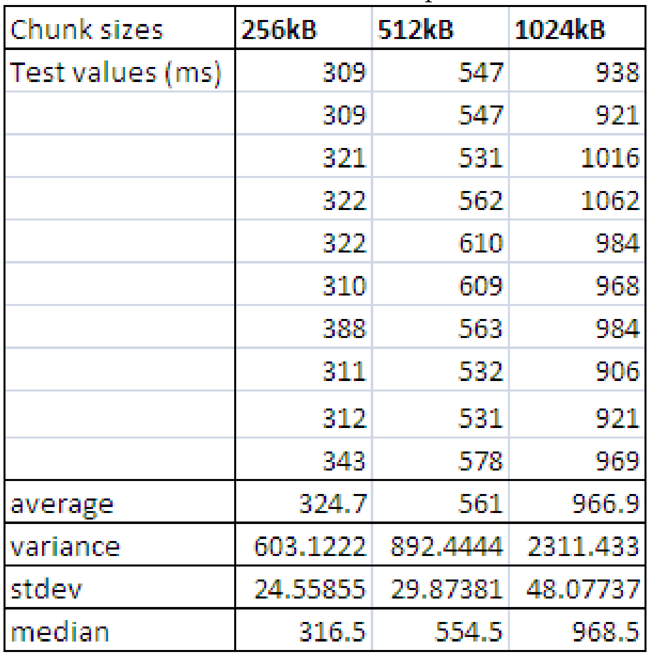
Chunk sizes	256kB	512kB	1024kB
Test values (ms)	651	577	873
	296	483	1015
	327	608	937
	327	577	999
	296	531	999
	375	593	1031
	328	562	1030
	281	469	983
	297	437	1014
	296	515	1030
average	347.4	535.2	991.1
variance	12113.16	3336.622	2530.989
stdev	110.0598	57.7635	50.30893
median	312	546.5	1006.5

4.3.1.2 Peer requesting content from another Peer

Another experiment involved having peer B download content from peer A after peer A downloaded the content from the content server (see figure 4.1). That means there are two peers in the tracker's list of peers. However, for the sake of this experiment peer A has been tweaked to be the optimal choice of the tracker when any peer requests content, thus for every request from peer B the tracker will return a reference to peer A (see discussion on Peer objects in section 3.6.1). These measurements were conducted in the same way as the experiments in section 4.3.1.1.

The results when downloading content of 323MB are shown in table 4.3. The median initial delays $\pm$ standard deviation for the different chunk sizes were: $316.5 \pm 24.6ms$, $554.5 \pm 29.9ms$, and $968.5 \pm 48.1ms$.

Table 4.3: Performance test of the Loree prototype, measuring the initial delay in ms when a client downloads content of 323MB from a peer.

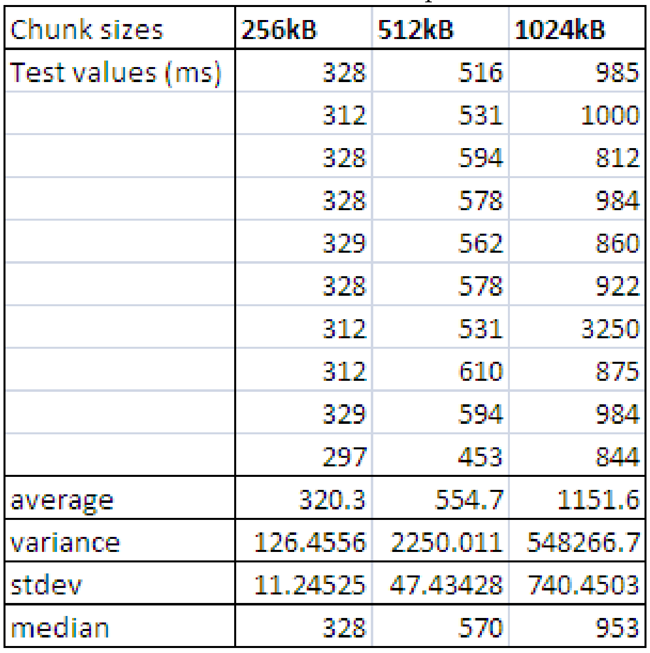


Chunk sizes	**256kB**	**512kB**	**1024kB**
Test values (ms)	309	547	938
	309	547	921
	321	531	1016
	322	562	1062
	322	610	984
	310	609	968
	388	563	984
	311	532	906
	312	531	921
	343	578	969
average	324.7	561	966.9
variance	603.1222	892.4444	2311.433
stdev	24.55855	29.87381	48.07737
median	316.5	554.5	968.5

The same experiment was performed using the 735MB episode. The results are presented in table 4.4. The median initial delays $\pm$ standard deviation for the three chunk sizes were $328 \pm 11.2ms$, $570 \pm 47.4ms$, and $953 \pm 740.5ms$. One of the values in table 4.4 for chunk sizes of 1024kB deviated strongly from the rest (3250ms). The reason for this deviation is *not* due to the Loree prototype implementation, but because during that experiment other processes on the computer were running and thus occupied greater CPU and memory resources than for other experimental runs - thus leading to greatly increased delay.

One can still observe that the median $\pm$ standard deviation when downloading content of 323MB and 735MB are statistically the same, thus concluding that the initial delay when downloading from a peer is not affected by the size of the content.

Table 4.4: Performance test of the Loree prototype, measuring the initial delay in ms when a client downloads content of 735MB from a peer.



Chunk sizes	**256kB**	**512kB**	**1024kB**
Test values (ms)	328	516	985
	312	531	1000
	328	594	812
	328	578	984
	329	562	860
	328	578	922
	312	531	3250
	312	610	875
	329	594	984
	297	453	844
average	320.3	554.7	1151.6
variance	126.4556	2250.011	548266.7
stdev	11.24525	47.43428	740.4503
median	328	570	953

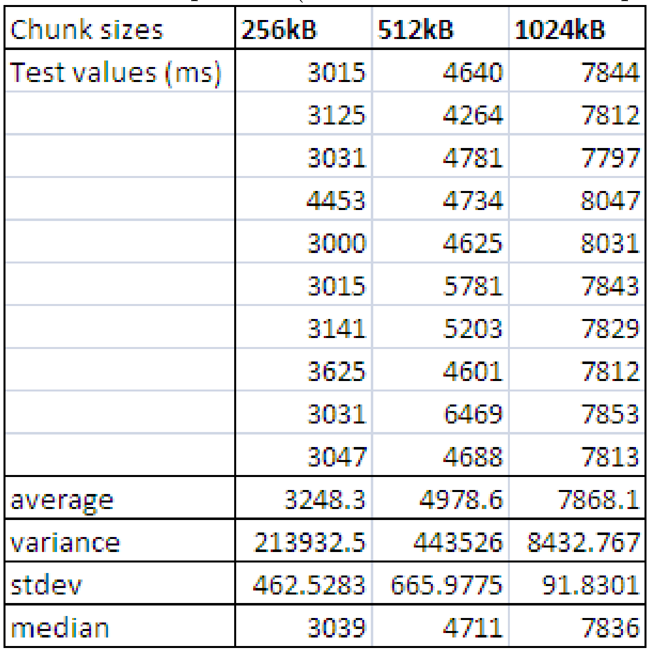
The values observed when downloading from a peer and from a content server (see section 4.3.1.1) are essentially identical since their medians ± standard deviations overlap. This means there is statistically no difference in initial delay between downloading content from a peer in contrast to downloading from a content server. Note that as there were no other peers active, we can not say what would happen if the load on the content server were to increase due to it providing content to multiple requestors. Nor can we indicate what the performance would be if there were multiple requestors making requests of a single peer that was providing content to several peers.

4.3.2 With added network delay

In this section, the initial delay was measured using a simulated network delay of 50ms on inbound and outbound packets to and from the client (yielding a total round-trip time of 100ms). This delay was simulated using Dummynet, a tool designed for testing network protocols [57]. In this experiment, peer B downloads content from peer A. The download process for downloading content from a content server and a peer is the same (and was also shown to be statistically identical in section 4.3.1.2), thus only downloading from a peer was tested in this experiment.

The results from this experiment are presented in table 4.5. The median initial delays ± standard deviation for the different chunk sizes were $3039 \pm 462.5ms$, $4711 \pm 666.0ms$, and $7836 \pm 91.8ms$. One can clearly note that the initial delay has increased by tenfold for each of the different chunk sizes. The initial delay has not fallen *as* much for the larger chunk sizes since this is due to fewer chunks (factor of 2) being transfered. Thus the amount of time spent fetching remote objects is reduced by a factor of 2.

Table 4.5: Performance test of the Loree prototype, measuring the initial delay in ms of a client downloading content of 323MB from a peer. The client has a simulated delay of 50ms on inbound and outbound packets (total simulated round-trip time of 100ms).



Chunk sizes	**256kB**	**512kB**	**1024kB**
Test values (ms)	3015	4640	7844
	3125	4264	7812
	3031	4781	7797
	4453	4734	8047
	3000	4625	8031
	3015	5781	7843
	3141	5203	7829
	3625	4601	7812
	3031	6469	7853
	3047	4688	7813
average	3248.3	4978.6	7868.1
variance	213932.5	443526	8432.767
stdev	462.5283	665.9775	91.8301
median	3039	4711	7836

However, the reason that the delay increases by such high values (compared to the case without additional delay) is that Java RMI is implemented on TCP, which has a substantial control overhead in comparison to UDP [58, 59]. An example is TCP's 3-way handshake which, during the test, was observed to take 100ms before the last acknowledgment (ACK) was sent to initiate the TCP connection. This was followed by Java RMI protocol initialization which took another 100ms (i.e., a single round-trip time. In addition, the throughput for downloading the first chunk is limited by the TCP slow-start algorithm, which is now increased by the RTT [60]. The point being that the network delay increases the Loree system's delay substantially due to the control overhead required by the TCP protocol. Experiments about how the network delay affects the throughput of the network are presented in section 4.4.2.

4.4 Throughput

Throughput is measured as the average rate of successful message delivery over a communication link. In these experiments, throughput is measured in kilobytes per second (kBps) and calculated based upon the time it takes for a peer to fully transfer a predefined sized file and to update the tracker with information about the content now available from this peer (for these measurements we exclude peer initialization time, t12-t0).

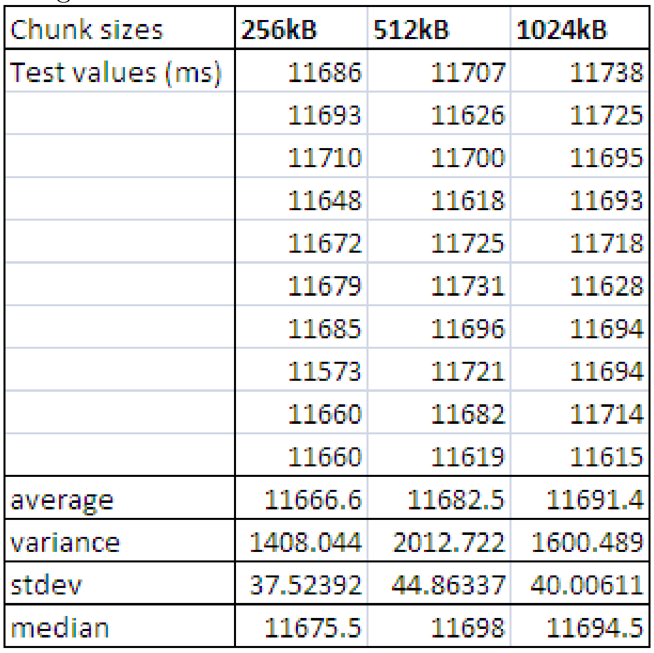
4.4.1 Ideal conditions

The experiments described in this section were performed under ideal conditions, i.e., **without** any simulated network delays (as will be examined in section 4.4.2).

4.4.1.1 Peer requesting content from a Content server

In the first throughput experiment a peer joins a server and only a single content server is available in the peer network (i.e., the network consists only of peer A and the server from figure 4.1). The results for downloading the 323MB file can be seen in table 4.6. The median download throughput ± standard deviation for the different chunk sizes were $11675.5 \pm 37.5 kBps$, $11698 \pm 44.9 kBps$, and $11694.5 \pm 40.0 kBps$. Comparing these results we see that the throughput is roughly constant and independent of the chunk size.

Table 4.6: Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 323MB from a content server.

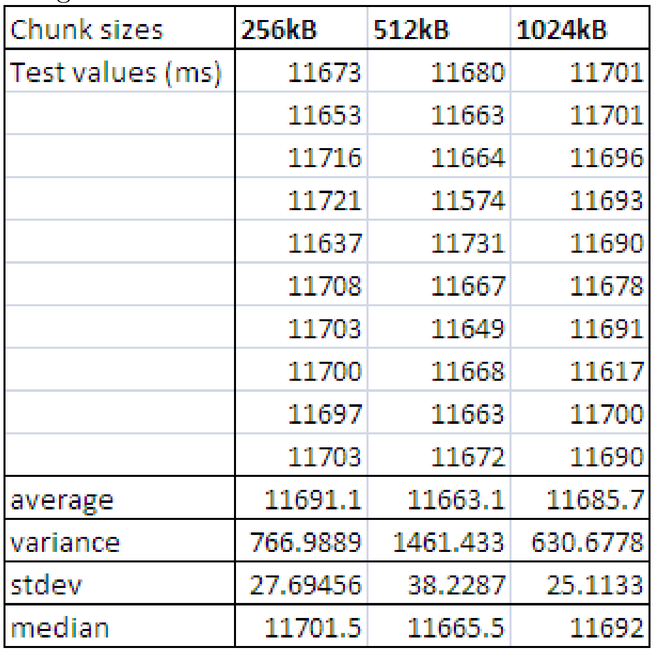


Chunk sizes	256kB	512kB	1024kB
Test values (ms)	11686	11707	11738
	11693	11626	11725
	11710	11700	11695
	11648	11618	11693
	11672	11725	11718
	11679	11731	11628
	11685	11696	11694
	11573	11721	11694
	11660	11682	11714
	11660	11619	11615
average	11666.6	11682.5	11691.4
variance	1408.044	2012.722	1600.489
stdev	37.52392	44.86337	40.00611
median	11675.5	11698	11694.5

The same experiment was performed using the 735MB episode. The results are presented in table 4.7. The median download throughput ± standard deviation for the different chunk sizes were $11701.5 \pm 27.7 kBps$, $11665.5 \pm 38.2 kBps$, and $11692 \pm 25.1 kBps$. Again, the results show that the throughput is roughly constant and independent of the chunk size.

Comparing these results to downloading content of 323MB, one can observe that the median ± standard deviation are statistically the same, thus concluding that the throughput when downloading content from a server is not affected by the size of the content.

Table 4.7: Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 735MB from a content server.



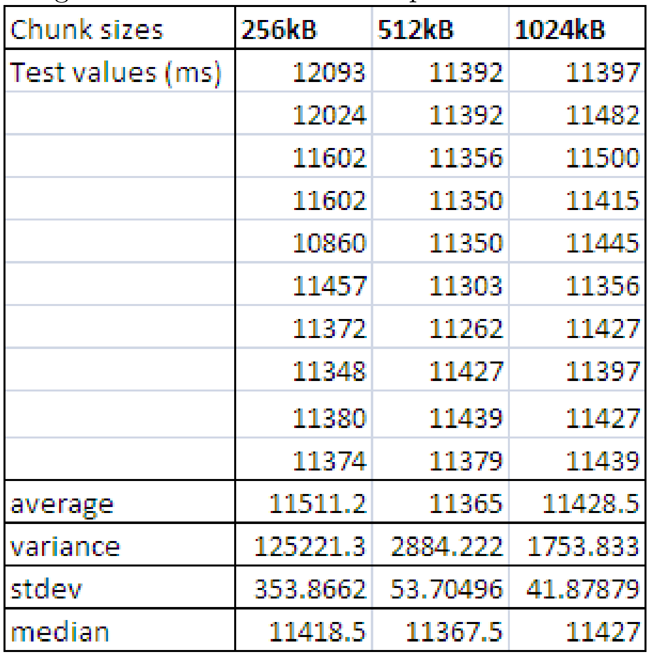
Chunk sizes	256kB	512kB	1024kB
Test values (ms)	11673	11680	11701
	11653	11663	11701
	11716	11664	11696
	11721	11574	11693
	11637	11731	11690
	11708	11667	11678
	11703	11649	11691
	11700	11668	11617
	11697	11663	11700
	11703	11672	11690
average	11691.1	11663.1	11685.7
variance	766.9889	1461.433	630.6778
stdev	27.69456	38.2287	25.1133
median	11701.5	11665.5	11692

4.4.1.2 Peer requesting content from another Peer

In the next throughput experiment, a peer B joins the network to download the content from peer A after peer A has finished downloading the content from the content server (see figure 4.1). This means there are two possible peers to download the content from, but peer A has been tweaked to always be the most feasible peer to download from, thus the tracker will, in this experiment, always respond indicating that the peer to download content from is peer A for each request for content made to the tracker.

The test results can be seen in table 4.8. The median download throughput ± standard deviation for the different chunk sizes were $11418.5 \pm 353.9kBps$, $11367.5 \pm 53.7kBps$, and $11427 \pm 41.9kBps$.

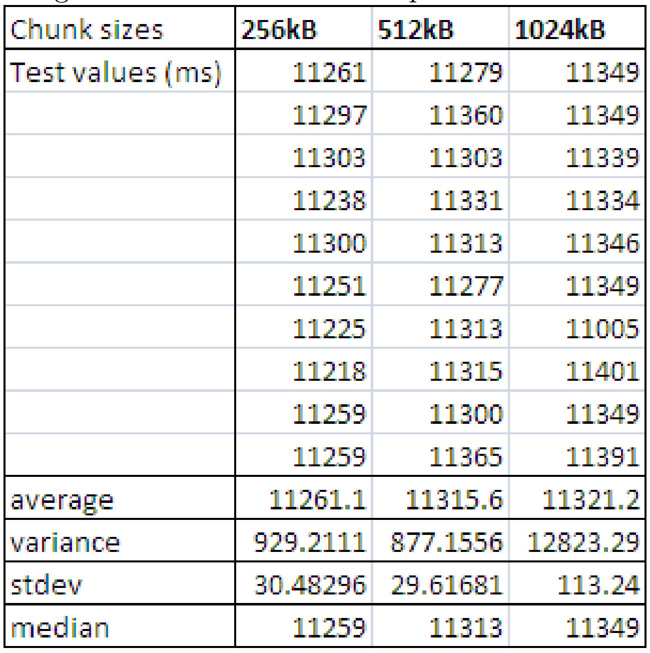
Table 4.8: Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 323MB from a peer.



Chunk sizes	256kB	512kB	1024kB
Test values (ms)	12093	11392	11397
	12024	11392	11482
	11602	11356	11500
	11602	11350	11415
	10860	11350	11445
	11457	11303	11356
	11372	11262	11427
	11348	11427	11397
	11380	11439	11427
	11374	11379	11439
average	11511.2	11365	11428.5
variance	125221.3	2884.222	1753.833
stdev	353.8662	53.70496	41.87879
median	11418.5	11367.5	11427

The same experiment was performed using the 735MB episode. The results are presented in table 4.9. The median download throughput ± standard deviation for the different chunk sizes were $11259 \pm 30.5 kBps$, $11313 \pm 29.6 kBps$, and $11349 \pm 113.2 kBps$. Comparing these numbers to the results presented in section 4.4.1.1 one can observe that there is statistically no difference between downloading from a peer compared to downloading from a content server, and that the size of the content does not affect the throughput.

Table 4.9: Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 735MB from a peer.

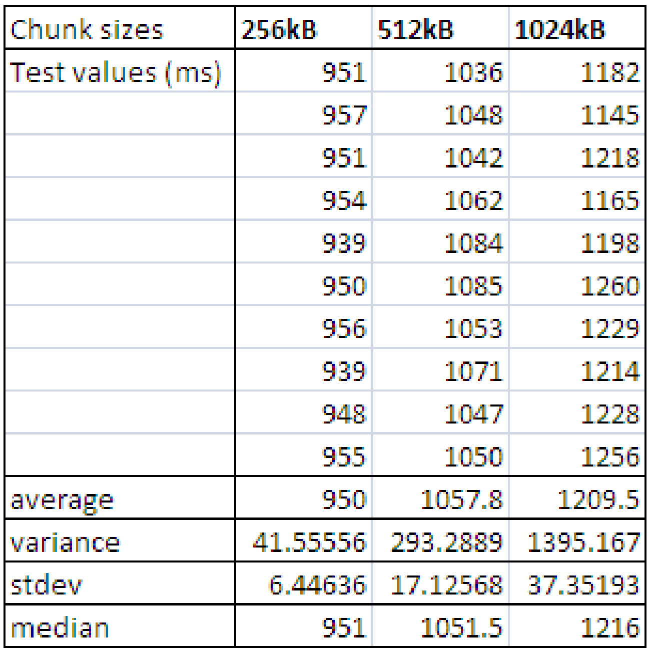


Chunk sizes	256kB	512kB	1024kB
Test values (ms)	11261	11279	11349
	11297	11360	11349
	11303	11303	11339
	11238	11331	11334
	11300	11313	11346
	11251	11277	11349
	11225	11313	11005
	11218	11315	11401
	11259	11300	11349
	11259	11365	11391
average	11261.1	11315.6	11321.2
variance	929.2111	877.1556	12823.29
stdev	30.48296	29.61681	113.24
median	11259	11313	11349

4.4.2 With added network delay

In the following throughput experiment, the throughput was measured using a simulated network delay of 50ms (in and out). In this experiment peer B downloads the 323MB episode from peer A, with the exact same setup as in section 4.3.2. The results are presented in table 4.10. The median download throughput $\pm$ standard deviation for the different chunk sizes when a client downloads content from a peer were $951 \pm 6.4 kBps$, $1051.5 \pm 17.1 kBps$, and $1216 \pm 37.4 kBps$. Here one can observe that the total throughput has fallen by up to twelve-fold in comparison to the results observed in section 4.4.1.2, and that the throughput difference when using different chunk sizes now differ from each other (**unlike** the case in ideal network conditions, see section 4.4.1.2).

Table 4.10: Performance test of the Loree prototype, measuring the throughput in kBps of a client downloading content of 323MB from a peer in the network. The client has a simulated delay of 50ms on inbound and outbound packets (total simulated round-trip time of 100ms).



Chunk sizes	256kB	512kB	1024kB
Test values (ms)	951	1036	1182
	957	1048	1145
	951	1042	1218
	954	1062	1165
	939	1084	1198
	950	1085	1260
	956	1053	1229
	939	1071	1214
	948	1047	1228
	955	1050	1256
average	950	1057.8	1209.5
variance	41.55556	293.2889	1395.167
stdev	6.44636	17.12568	37.35193
median	951	1051.5	1216

As mentioned earlier in section 4.3.2, Java RMI utilizes TCP as its transport protocol which adds additional overhead in the form of control traffic and additional delay due to the specifics of the protocol. More specifically, TCP implements a slow-start congestion control strategy to avoid sending more data than the network is capable of successfully transporting to the final destination. This leads advantages in terms of reliability as this reduces the chance of packets being lost in the network (in scenarios such as when an intermediate router runs out of queue space) and thus reduces re-transmissions [60]. However, with a longer network delay this will lead to a slower start than would be the case for a smaller round-trip delay as the time delay between each increase in window size (and hence throughput) will increase with larger network delay.

This exposes a serious design flaw in the prototype implementation of Loree: The throughput is limited by TCP flow control. This flow control limits the throughput in order to reduce congestion and thus packet-loss. The expected $throughput = \frac{1.22*MSS}{RTT*\sqrt{L}}$, where MSS is the Maximum Segment Size, RTT is the round-trip time between the hosts, and L is the loss rate [61, page 316]. During the experiments the MSS was 1460 bytes, the RTT was 100ms, and L could be approximated to 10^{-4}, thus resulting in a maximal of $\frac{1.22*1460}{10^{-1}*\sqrt{10^{-4}}} * 8 = 1781200 * 8 \approx 14Mbps$. The median throughput when downloading from a peer with an added network delay of 100ms round-trip for the different chunk sizes are approximately 7.8Mbit, 8.4Mbit, and 9.7Mbit. In contrast, the median throughput for the chunk sizes in a network without delay are approximately

90.5Mbit, based on the values in section 4.4.1.2. This means that TCP flow control is a big limitation for the throughput of the Loree network, but it also means that there are a few Mbit missing in the throughput tests. This throughput is “lost” when the client, peer, and tracker marshal the data into and out of the RMI encoding. It is clear, from the results, that the smallest chunk size has the greatest loss. This is due to the smallest chunk size having the largest traffic overhead. The main source of overhead comes from the implementation decision to have the client make a new request to the tracker, and fetch a new remote object for each chunk.

4.5 Control overhead

The *control overhead* metric represents the percentage of total traffic produced between a Loree client and tracker that is *not* content data, in contrast to the total traffic. This traffic includes requests for peers, content updates to tracker, monitor pings, and overhead generated by Java RMI.

An experiment was performed under ideal network conditions (as specified in section 4.1) to measure the amount of network traffic sent between a client and tracker. The same setup was used as in sections 4.3.1.2 and 4.4.1.2 where peer B downloads the 323MB sized content from peer A. Two instances of Wireshark (see section 2.4.3) were run on peer B with one instance listening to the network traffic between peer B and server, and the second instance listening to the network traffic between peer A and peer B. The results are stated in numbers of bytes. The percentage of overhead traffic sent to and from the tracker, based on the three different chunk sizes, are shown in figures 4.3, 4.4, and 4.5.

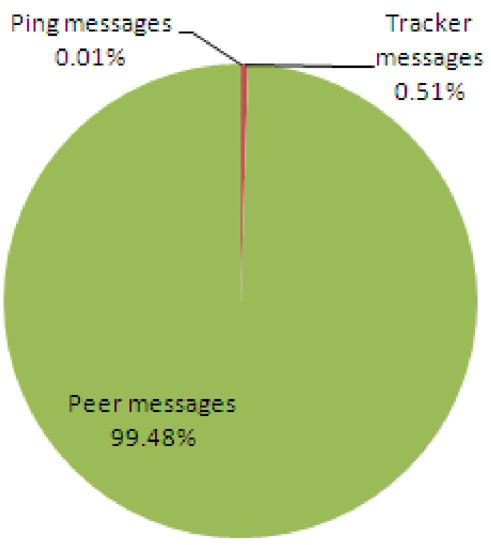


Figure 4.3: The percentage breakdown of traffic sent in bytes over the Loree network when transferring a file of 323MB in 256kB chunk sizes. Note that Peer messages consist of traffic sent between client and peer which includes both content and control traffic, thus this figure primarily shows the percentage of control traffic to and from the tracker.

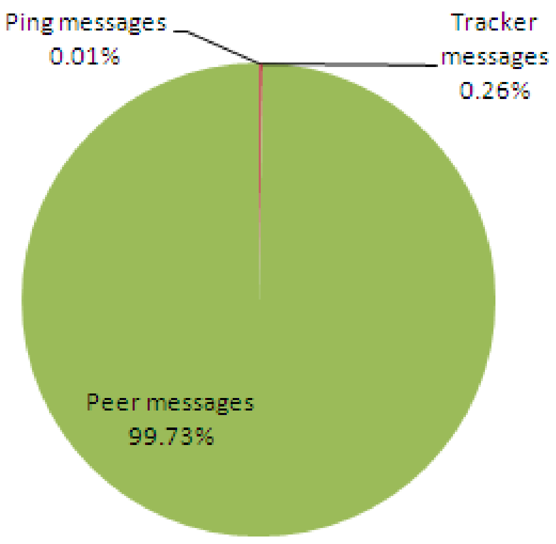


Figure 4.4: The percentage breakdown of traffic sent in bytes over the Loree network when transferring a file of 323MB in 512kB chunk sizes. Note that Peer messages consist of traffic sent between client and peer which includes both content and control traffic, thus this figure primarily shows the percentage of control traffic to and from the tracker.

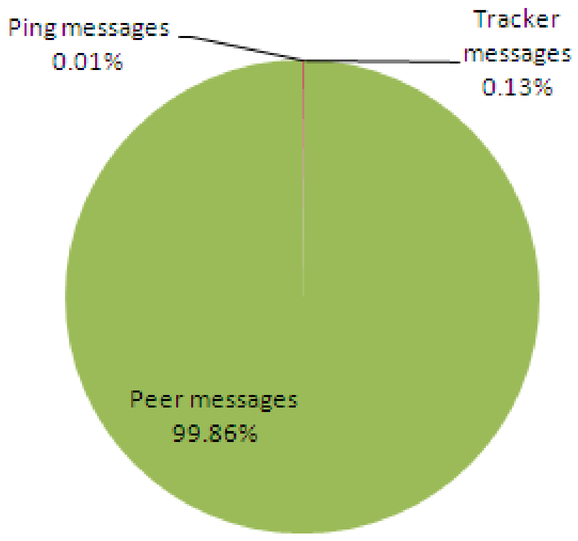


Figure 4.5: The percentage breakdown of traffic sent in bytes over the Loree network when transferring a file of 323MB in 1024kB chunk sizes. Note that Peer messages consist of traffic sent between client and peer which includes both content and control traffic, thus this figure primarily shows the percentage of control traffic to and from the tracker.

One can observe that the number of requests sent from peer B to the tracker is directly related to the number of chunks the content consists of. The number of requests made to a tracker (Y) is directly related to the number of chunks an episode is divided into (X), as specified in equation (4.1). One extra request is made to the tracker during initialization when the client initially requests a remote tracker object.

$$Y = X + 1 \tag{4.1}$$

The figures also show the fraction of network traffic generated by the Monitor (Loree query operations). During each of these query operations the Monitor (located on the tracker) requests the local peer's Sender object in order to verify that it is operational. As can be seen in the figures, the Monitor generates minimal traffic (0.01%) during the time it takes for peer A to download a file of 323MB from peer B under ideal network conditions. It should be noted that the total amount of traffic generated by the tracker will increase as the network delay increases. When the time for downloading content increases, because the Monitor still queries peers every second, more query operations will be generated during the increased download period. However, the amount of traffic will still be minimal in comparison to the rest of the Loree traffic.

Chapter 5

Evaluation and Recommendations

The evaluation of the Loree prototype is presented in this chapter. Here I analyze and discuss the measured results and recommend how the solution can be used.

5.1 Initial delay and Throughput

I begin by evaluating the initial delay when transferring content divided in chunk sizes of 256kB, 512kB, and 1024kB in ideal conditions, as described in section 4.3.1. The values show that the prototype system is unaffected by the size of the content and the quantity of chunks. The average of the median initial delays under ideal conditions when downloading from peer, content server, and downloading content of different sizes was measured to 319.1 ± 6.7ms, 566.1 ± 20.7ms, and 985.9 ± 29.9ms. I believe these are acceptable values for the initial delay, especially for the 256kB chunk size. It should be noted that these values do not include the time to load the in-memory content to a media player. Verifying that the total delay (including the time to load the content to a media player) is acceptable is left for future work.

We can see, however, by the above results that the initial delay is directly affected by the chunk size. This shows that dividing the content into variable sized parts can help improve the user experience by improving performance such as initial delay. If the start of an episode is divided into 256kB chunks then the client can start loading the content to a media player with greatly *decreased* delay. The throughput results presented in section 4.4.2 show that higher throughput can be achieved when using chunks of larger size. Ergo, when a client has buffered a certain amount of smaller chunks it can continue downloading larger chunks in order to reduce overhead traffic and increase throughput.

The results presented in section 4.4.1 show the network performance in ideal conditions. If the tracker is optimized to group peers within geographical regions and/or ISP's access networks, RTT times above 100ms are not relevant since peers will be close to each other. This would mean that the throughput of the Loree client would lie between 14Mbps and 90.5Mbps in a production environment, which is more than acceptable for the majority of the content used on e.g., kanal5play.se (as described in section 2.4.4). The main limitation for the throughput is the TCP flow control algorithm, as discussed in section 4.4.2. This limitation hinders the scaling of the content quality sent in the system. Thus, if more throughput is required in the future, the transport protocol for RMI (TCP) should be reviewed or another transport protocol should be used. One option is to increase TCP's send & recieve windows by using the TCP window scale option, thus increasing throughput for high bandwidth & high latency connections [62]. It must be noted that the average connection speed of Swedish Internet users (as presented in section 2.4.1) is 15.3Mbps downlink, meaning that the throughput shown in the above results will not be limited by the viewer's connection speeds.

However, in section 4.3.2 I present results which show that the initial delay is heavily affected by the RTT between the client and peer. The median of the results reach values of several seconds on a network with 100ms RTT, even for the smallest chunk size. I believe that this is **not** acceptable for on-demand viewing. In comparison, Matris and Striko's P2P architecture, Daedalus, is built for live streaming with specific focus on reducing initial delay determines an average delay "between when a user presses a button to watch a specific channel until full video quality appears on the user's screen" of around *650ms* in ideal conditions[1] [6].

5.2 Playout buffer

A Spotify client with less than 3 seconds worth of data in it's playout buffer is considered to have a low amount of data and will switch to downloading content directly from content servers (see section 2.3.6). By these standards, a 3 second playout buffer for video + sound content of 800kbps + 96kbps would require $\frac{896kbps*3seconds}{8} = 336kB$ of downloaded content, which is 80kB less than the smallest chunk size in Loree. One 256kB chunk would result in $\frac{256kB}{\frac{896kbps}{8}} \approx 2.3seconds$. Thus, we believe that one 256kB chunk of content is enough to satisfy a playout buffer before switching to downloading larger chunks. In a production environment, we can assume that most of the time a client will have higher throughput from a content server than from a peer[2]. With this assumption, and the performance results presented when higher RTT is introduced, I suggest that the second chunk to be downloaded be of 1024kB size and also downloaded from a content server, as this will add another $\approx 9.1seconds$ of playout buffer, before safely being able to start downloading content from peers.

5.3 Overhead communications

The amount of overhead in bytes generated when communicating with the tracker is presented in section 4.5. While the results show small effects in contrast to the traffic sent between client and peer, it is still a time consuming process since it will add a delay of $2 * RTT_{tracker}+$ processing time (t12-t16 plus t22-t19 from figure 4.2) per download

[1]Note that this time *includes* the time to load the in-memory content to a media player.

[2]As is presented in the test results in the previous chapter, a higher throughput involves a lower RTT between client and server.

thread[3]. Due to clients not keeping remote Sender objects in a local cache, a $RTT_{peer}+$ processing time is added to retrieve these objects. This becomes a total delay of $2 * RTT_{tracker} + RTT_{peer} + \alpha$, where α is the aggregated processing time required by the client, peer, and tracker to process these requests. If α is approximated to zero, this still leaves three RTTs between each chunk download. Although the results in section 4.4.2 show that the throughput in a 50ms delay network (100ms RTT) is adequate for most of the content provided by kanal5play.se, it can still be improved by adding a few simple optimizations. I recommend the following: Request multiple chunks in *one* request to tracker, save remote peer Sender objects in a local cache so that they can be reused if the tracker specifies the same peer to download from multiple times, and register multiple chunks in one request to the tracker. These optimizations can reduce the delay of up to $2 * RTT_{tracker} + RTT_{peer} + \alpha$ for one chunk download.

5.4 The Achillies' heel

However, the initial delay proves to be the Achillies' heel of the Loree design. While the prototype performs well in ideal conditions, the slow-start algorithm together with the 3-way handshake lowers the performance of the system severely when additional network delay is introduced. One optimization can be made in the client initialization process: When the client asks the tracker for a Video object, the tracker replies with a suitable content server to download the first chunk(s) from in addition to the requested Video object. This optimization can save at least a $RTT_{tracker}$. However, as long as TCP is used as transport protocol, a large fraction of this delay will remain. To solve this problem (and problems with throughput) I recommend tweaking TCP options by building a custom socket factory for RMI. While this alternative may give up some of the core features of TCP, such as reliability, it can increase throughput and reduce delays - with our key focus being on reducing delay. In addition, building a custom socket factory enables using different types of sockets for different connections [63]. A option is to use sockets which result in low delay when downloading the initial chunks (such as an UDP socket) and switching to TCP sockets once enough content has been buffered[4] in order to increase reliability.

[3]However, since multiple downloads are run in parallel this delay is slightly reduced.

[4]Enough content to avoid risking interruptions in the playout.

Chapter 6

Conclusions and Future Work

This chapter begins by stating some of the conclusions that have been made based upon the analysis in the previous chapter. This chapter concludes with some suggestions for future work.

6.1 Conclusions

In this masters thesis, a new approach for on-demand television based on P2P technology was presented focusing on the peak hours of viewing of on-demand content.

In Chapter 1, an introduction was given of the problem, together with the goals of the project. Following this, in Chapter 2 background was given regarding IPTV and P2P systems. The nature of on-demand viewing was discussed and the concept of *on-demand television* was introduced. Additionally, related P2P systems were presented, together with some of their advantages and disadvantages. The chapter concluded by providing additional information which would be useful to the reader.

Chapter 3 described the proposed solution called Loree. First the overall solution was presented, together with the technology it uses, followed by a functional overview of how different nodes in the system operate and a more in-depth description of the nodes' specifications. Finally, some of the classes used in the Loree system and reduction of delay by predicting what viewers will watch were presented.

Chapter 4 discusses the measurements that were performed in order to evaluate the design. This discussion began with a description of the environment used for the experiments, then continues with a description of the metrics chosen to evaluate the system along with the results of these experiments.

Chapter 5 discusses the results of the measurements with a focus on the possibilities and limitations of the design and recommendations were made to improve the performance.

While the peak hours of viewing specific content are lengthened (and the peak rate of demand reduced) when on-demand viewing is possible, there are still patterns in the times when users watch content provided by television networks. As discussed in Chapter 2 concerning the concept of *on-demand television*, television broadcasts continue to affect the patterns of what viewers watch and when they watch it. For instance, editors of a TV network can influence viewers to access a website to view specific episodes (or other content) at certain hours.

With these peak hours in mind, Loree was designed to reduce the resources needed during these peak hours in terms of the required bandwidth needed by the media content provider's server(s). By building a tree-hierarchy of peers, based on available content and bandwidth, Loree can be used to lower bandwidth costs for media content providers by taking advantage of these peers resources in order to distribute content to viewers.

While this master thesis report shows that the prototype implementation has potential and that it can perform adequately in Swedish homes in terms of throughput, it still has room for improvement - especially for the user experience in terms of initial delay. Improvements need to be made and some key features need to be re-implemented before this system could be run in a production environment. Based on the measurements and analysis made in previous chapters, Loree's integration with the TCP protocol needs to be improved in order to supply a better user experience before other features are implemented.

Notice that while this report covered many of the possible features of Loree, many were not implemented or tested. The reason is that Loree as a complete system is fairly large and complicated, and needs features such as optimal tree-building algorithms which are a master's thesis project themselves. A full implementation of all features and extensive testing could not be conducted in the time frame available for this masters thesis project. Therefore, in the following section the author presents potential features and required enhancements that need to be implemented before Loree would be suitable for use in a production environment.

6.2 Future work

Future work is divided into two parts: one part is a list of unimplemented features or features that have not been tested in the prototype and the second part is a list requiring further in-depth research and development (i.e. a list of possible follow-up thesis projects).

6.2.1 Future implementation work

In order to make the Loree system fully operational and suitable for testing/running with actual customers, the following should be implemented.

- A tree algorithm needs to be implemented according to the discussion in section 3.4.
- Today the client is implemented as a program which needs to be downloaded and run separately on each viewer's machine. However, in order to minimize the amount of work needed by viewers and to reduce the amount of places things can go wrong, the client should be integrated in a Flash player [64] (preferably the

existing kanal5play player and interface) and run completely in the background so that viewers do not have to interact with the Loree system. This should be done in a seam-less way so that regular viewers of the play sites (e.g. www.kanal5play.se) do not notice the transition to a new download system. The viewers should be able to continue watching the content they choose without having to focus on technical details.

- Tests should be performed on larger scales, preferably with actual viewers, in order to insure the stability and scalability of the system.
- Kanal5play.se precede their on-demand content with commercials [12]. Loree could take advantage of the time between when a commercial is fully downloaded until the playout of the commercial is completed. In this time-gap, Loree could potentially buffer several chunks of content. The by far greatest advantage with this is that viewers would not notice the poor initial delay the Loree system has in high delay networks due to TCP's 3-way handshake (as discussed in chapter 5).
- To simplify maintenance and operations, the tracker and content servers should be equipped with a simple graphical extension. The tracker should be equipped with a interface which shows the status of the network, what content is available and by whom, tree structures in the network per content, number of viewers in the system, etc. For the content server the interface should offer a simple way of loading content into it and removing it, also this interface should show the current load on the server in terms of how many peers are downloading the content and at what speeds.
- The client needs to be optimized by keeping peers' remote objects locally instead of retrieving them for each request, thus saving time between downloads and increasing throughput.
- The client should be optimized by requesting multiple parts from the same peer in one request, thus reducing the number of requests sent to the tracker and the total overhead traffic generated.
- Editors and other maintainers should be able to define which content will be under high demand during which hours so that the system can distribute content to viewers in the system that have their connection bandwidth available after finishing downloading their own content. It should be investigated how and when clients can download other content in order to help reduce load on the content servers and to determine if it is worth having other peers download the content at all.
- Implement unique IDs for each client. This could be connected to the viewer's kanal5play.se ID in order to insure unique names.
- Implement varying sized chunks of data. This is specifically for when the client loads its content, as currently the clients are not aware of the chunk sizes.

- Use variable-quality content in the chunks. For instance, have the first chunk(s) contain content of lower bit-rate (i.e., lower quality) while the rest of the chunks contain content with normal quality. Thus a larger playout buffer can be made and/or the initial delay reduced.
- Connect the Loree system to a media player (such as VLC Media Player [65]) and verify that the initial delay + the time to load the content to the player is within acceptable bounds.

6.2.2 Future research work

Although the following could be accomplished in relatively simple ways, the author believes each are important enough to be researched more deeply before being implemented in the system in order to find more optimal solutions.

- Define a framework or operation for defining what parts of a specific item content have a higher chance of being viewed immediately by viewers. See the discussion in section 3.6.3.2. This might be based upon monitoring how viewers choose to watch certain episodes (e.g. if they jump to a later point in the episode or if they start from the start) and from this create patterns which clients can use to modify the order in which they download content.
- Specify how editors can provide Loree with information about which content is expected to be under high demand and at what time.
- Find out if it is possible to correlate PartOfVideo sizes with time between key frames in H.264 CODEC.
- Compare performance with other popular P2P on-demand networks such as Joost.
- Port to Google App Engine or some other existing platform [66].
- Compare using persistent TCP connections and using a Monitor between Clients and Tracker. The comparison should compare how the stability of the network (i.e. tree architecture) and throughput are affected and how CPU power and memory resources are consumed (see the discussion in section 3.4.1).
- Investigate if the initial delay can be reduced by implementing a custom socket factory in the Java RMI [63].
- Investigate if it is worth having a client which is installed as a background process on viewer's computers which will save downloaded content for longer periods of time (e.g. a week or two) and redistribute it to peers even if the owner of the computer is not watching anything at the moment. This can be compared to the Voddler client [13].

- Investigate when (and if) content should be downloaded by clients with available bandwidth resources in order to further help in the distribution of content in the network and help divert load from the content servers. This can also be used for preemptive purposes where it might be predicted that the viewer will want to watch multiple episodes in a series. At which point his or her client will download the content in advanced. Note that this should be done *after* the client has finished downloading the content which was requested by its own viewer! See discussion in 3.3.1.
- Optimize the tracker to group clients within geographical regions and/or operator's (ISP's) access networks in order to reduce delay between peers.

Bibliography

[1] Cao Wei Qiu. A new Content Distribution Network architecture - PlentyCast. Master's thesis, Masters thesis, Royal Institute of Technology (KTH), School of Microelectronics and Information Technology, IMIT/LCN 2004-05, March 2004. `http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/040430-Cao_Wei_Qiu-with-cover.pdf`.

[2] Ayodele Damola. Peer to peer networking in Ethernet broadband access networks. Master's thesis, Masters thesis, Royal Institute of Technology (KTH), School of Microelectronics and Information Technology, IMIT/LCN 2005-10, May 2005. `http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/050529-Ayodele_Damola-with-cover.pdf`.

[3] Thomas Silverston and Olivier Fourmaux. P2P IPTV Measurement: A Comparison Study. *CoRR*, abs/cs/0610133, October 2006.

[4] TVants. TVants. `http://tvants.en.softonic.com/`. (last visited: April 2010).

[5] Joost. Joost. `http://www.joost.com`. (last visited: March 2010).

[6] Athanasios Makris and Andreas Strikos. Daedalus: A media agnostic peer-to-peer architecture for IPTV distribution. Master's thesis, Royal Institute of Technology (KTH), School of Information and Communication, COS/CCS, 2008-11, June 2008.

[7] Israel Cidon, Shay Kutten, and Ran Soffer. Optimal allocation of electronic content. *Computer Networks*, 40:205â€“218, 2002.

[8] Thomas Silverston and Olivier Fourmaux. P2P IPTV measurement: a case study of TVants. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–2, New York, NY, USA, December 2006. ACM.

[9] Gilbert Held. *Understanding IPTV*. Auerbach Publications, 1 edition, October 2006. ISBN-13: 978-0849374159.

[10] Sveriges television. SVT Play - Sveriges Television. `http://svtplay.se/`. (last visited: March 2010).

[11] Tv 4. TV4 Play - TV när du vill. `http://www.tv4play.se/`. (last visited: March 2010).

[12] Kanal5. Kanal5Play. `http://www.kanal5play.se/`. (last visited: March 2010).

[13] Voddler. Voddler Beta Home - Welcome to the Magical World of Movies. `http://voddler.com/`. (last visited: March 2010).

[14] YouTube. YouTube. `http://www.youtube.com/`. (last visited: March 2010).

[15] Google. Google Videos. `http://video.google.com/`. (last visited: March 2010).

[16] Ymir Vigfusson, Hussam Abu-Libdeh, Mahesh Balakrishnan, Ken Birman, Robert Burgess, Gregory Chockler, Haoyuan Li, and Yoav Tock. Dr. multicast: Rx for data center communication scalability. In *EuroSys '10: Proceedings of the 5th European conference on Computer systems*, pages 349–362, New York, NY, USA, 2010. ACM.

[17] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. Survey, Athens University of Economics and Business, December 2004.

[18] Napster LLC. Napster Free–Listen to free streaming music online. `http://free.napster.com/`. (last visited: March 2010).

[19] Bram Cohen. BitTorrent.org. `http://www.bittorrent.org/`. (last visited: March 2010).

[20] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A Reputation System for Peer-to-Peer Networks. Technical report, College of Computing, Georgia Institute of Technology, June 2003.

[21] MPS Broadband AB. Broadband video solution provider. `http://www.mpsbroadband.com/`. (last visited: March 2010).

[22] MPS Broadband AB. MPS launches revolutionary Peer-to-peer technology for distributing video online. `http://www.mpsbroadband.com/about_news.asp`, June 2009. News (last visited: June 2010).

[23] Pontus Eklöf, Sales Director US MPS Broadband AB. Interview on 2010.03.12.

[24] D. Ian Hopper. Open source Napster-like product disappears after release. `http://archives.cnn.com/2000/TECH/ptech/03/15/gnutella/index.html`, March 2000. (last visited: March 2010).

[25] Marcus Bergner. Improving performance of modern Peer-to-Peer services. Master's thesis, Umeå University, Department of Computing Science, June 2003. `http://www8.cs.umu.se/~bergner/thesis/thesis.pdf`.

[26] Ericsson. `http://www.ericsson.com/`. (last visited: March 2010).

[27] Yensy James Hall, Patrick Piemonte, and Matt Weyant. Joost: A Measurement Study. `http://www.patrickpiemonte.com/15744-Joost.pdf`, May 2007. School of Computer Science Carnegie Mellon University.

[28] Spotify AB. Spotify. http://www.spotify.com/. (last visited: July 2010).

[29] Maria Ringborg. Musiktjänsten Spotify lanseras. *Dagens Nyheter*, October 2008. http://www.dn.se/kultur-noje/musik/musiktjansten-spotify-lanseras-1.631484, (last visited:July 2010).

[30] Chris Salmon. Welcome to nirvana. *guardian.co.uk*, January 2009. http://www.guardian.co.uk/music/2009/jan/16/downloading-music-spotify, (last visited:July 2010).

[31] Gunnar Kreitz. Spotify — Behind the Scenes. http://www.nada.kth.se/~gkreitz/spotify/kreitz-spotify-kth_ict10.pdf, May 2010. Set of slides shown at the Spotify presentation at KTH, ICT 2010.

[32] Wikipedia. BitTorrent Tracker. http://en.wikipedia.org/wiki/BitTorrent_tracker. Wiki page modified 19 August 2010 at 20:31, (last visited: August 2010).

[33] Maggie Strömberg. Spotify öppnar för alla. *Sydsvenskan*, May 2010. http://www.sydsvenskan.se/kultur-och-nojen/article876807/Spotify-oppnar-for-alla.html, (last visited:July 2010).

[34] Bredbandskollen. Bredbandskollen TPTEST. http://www.bredbandskollen.se/statistik/?section=1&isp=0&region=0&month=022010. (last visited: February 2010).

[35] S. Deering and R. Hinden. Internet protocol, version 6 (IPv6) specification. RFC 2460, Internet Engineering Task Force, December 1998.

[36] J. Postel. Internet protocol. RFC 791, Internet Engineering Task Force, September 1981.

[37] Geoff Huston and Takashi Arano. IPv4 address report. http://www.potaroo.net/tools/ipv4/index.html. (last visited: March 2010).

[38] Steinar H. Gunderson. Global IPv6 statistics - Measuring the current state of IPv6 for ordinary users. http://www.ripe.net/ripe/meetings/ripe-57/presentations/Colitti-Global_IPv6_statistics_-_Measuring_the_current_state_of_IPv6_for_ordinary_users_.7gzD.pdf, October 2008. Set of slides shown at the RIPE57 meeting in October 2008.

[39] Oracle. Networking IPv6 User Guide for JDK/JRE 5.0. http://download.oracle.com/docs/cd/E17409_01/javase/6/docs/technotes/guides/net/ipv6_guide/index.html. (last visited: July 2010).

[40] TCPDUMP/LIBCAP public repository. http://www.tcpdump.org/. (last visited: March 2010).

[41] Wireshark Foundation. Wireshark. http://www.wireshark.org/. (last visited: June 2010).

[42] Google. On2 Technologies. `http://www.on2.com/`. (last visited: July 2010).

[43] Google Inc. Google closes on2 technologies acquisition. *Google investor relations*, February 2010. `http://investor.google.com/releases/2010/0219.html`.

[44] Wikipedia. VP6. `http://en.wikipedia.org/wiki/VP6`. Wiki page modified 16 July 2010 at 18:04, (last visited: July 2010).

[45] Wikipedia. H.264/MPEG-4 AVC. `http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC`. Wiki page modified 9 July 2010 at 18:04, (last visited: July 2010).

[46] Apple Inc. H.264 Frequently Asked Questions. `http://www.apple.com/br/quicktime/technologies/h264/faq.html`. (last visited: July 2010).

[47] Apache. Apache Subversion. `http://subversion.apache.org/`. (last visited: May 2010).

[48] Oracle. Java SE at a Glance. `http://java.sun.com/javase/index.jsp`. (last visited: May 2010).

[49] Oracle. Remote Method Invocation Home. `http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp`. (last visited: May 2010).

[50] The Eclipse Foundation. Eclipse.org home. `http://www.eclipse.org/`. (last visited: May 2010).

[51] Qusay H. Mahmoud. Advanced socket programming. *Oracle, Sun Developer Network*, December 2001. Section 'RMI vs. Socket and Object Serialization'.

[52] SeungJun Bang and JinHo Ahn. Implementation and performance evaluation of socket and rmi based java message passing systems. *Software Engineering Research, Management and Applications, ACIS International Conference on*, 0:153–159, 2007.

[53] Sun Developer Network. Lesson: All About Sockets. `http://java.sun.com/docs/books/tutorial/networking/sockets/index.html`. (last visited: May 2010).

[54] Sun Microsystems. Serializable (Java 2 Platform SE v1.4.2). `http://java.sun.com/j2se/1.4.2/docs/api/java/io/Serializable.html`. (last visited: May 2010).

[55] J. Postel. Internet Control Message Protocol. RFC 792 (Standard), September 1981. Updated by RFCs 950, 4884.

[56] Bittorrent Protocol Specification v1.0. `http://wiki.theory.org/BitTorrentSpecification#Notes`. Wiki page, revision 158 (last visited: 21 June 2010).

[57] Luigi Rizzo. Dummynet home page. `http://info.iet.unipi.it/~luigi/dummynet/`. (last visited: August 2010).

[58] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[59] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[60] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard), January 1997. Obsoleted by RFC 2581.

[61] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, USA, 5th edition, 2009.

[62] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992.

[63] Oracle. Creating a Custom RMI Socket Factory. `http://download-llnw.oracle.com/javase/1.3/docs/guide/rmi/rmisocketfactory.doc.html`. (last visited: August 2010).

[64] Adobe. Adobe - Adobe Flash Player. `http://get.adobe.com/se/flashplayer/`. (last visited: August 2010).

[65] Volunteers. VideoLAN, Free streaming and multimedia solutions for all OS! `http://www.videolan.org/`. (last visited: March 2010).

[66] Google Inc. Google App Engine. `http://code.google.com/intl/sv-SE/appengine/`. (last visited: July 2010).

[67] Kanal5 AB. kanal5.se - Underhållningsnyheter, TV-tablåer och WebbTV. `http://kanal5.se/`. (last visited: March 2010).

Appendix A

Viewing patterns of Kanal5play viewers

The following data was received internally from Kanal5 AB [67]. The statistics were analyzed to find a pattern of how viewers watch television and use the network's on-demand website kanal5play.se [12], i.e. to see if there is a explicit pattern for peak hours on the on-demand website.

An episode of "Ballar av stål" was shown 21:00 on the 21th of March 2010 on Kanal5 (the television channel). After the episode ended on TV, the episode was published on Kanal5play[1]. A pattern can clearly be seen in Figure A.1 where viewers watch the episode on-line just after it was broadcast on television. This clearly shows a peak hour of episodes after they are viewed. Interestingly, during the same day even earlier episodes of the show are viewed, as can be seen in Figure A.2.

Both figures clearly show that viewers access the on-demand website to view the same episode shortly after it is released and broadcast on television.

[1]`http://www.kanal5play.se/program/play/ballaravstal-s02e02`

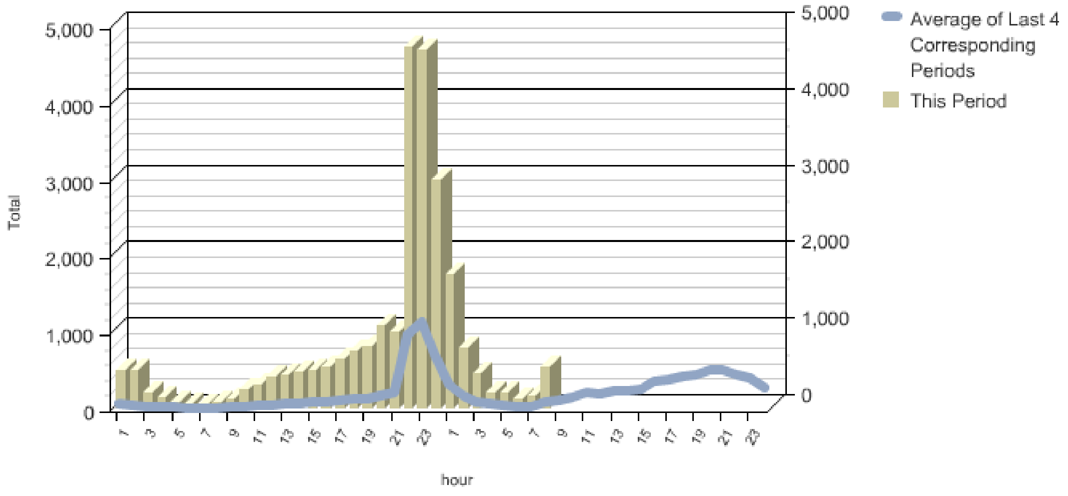
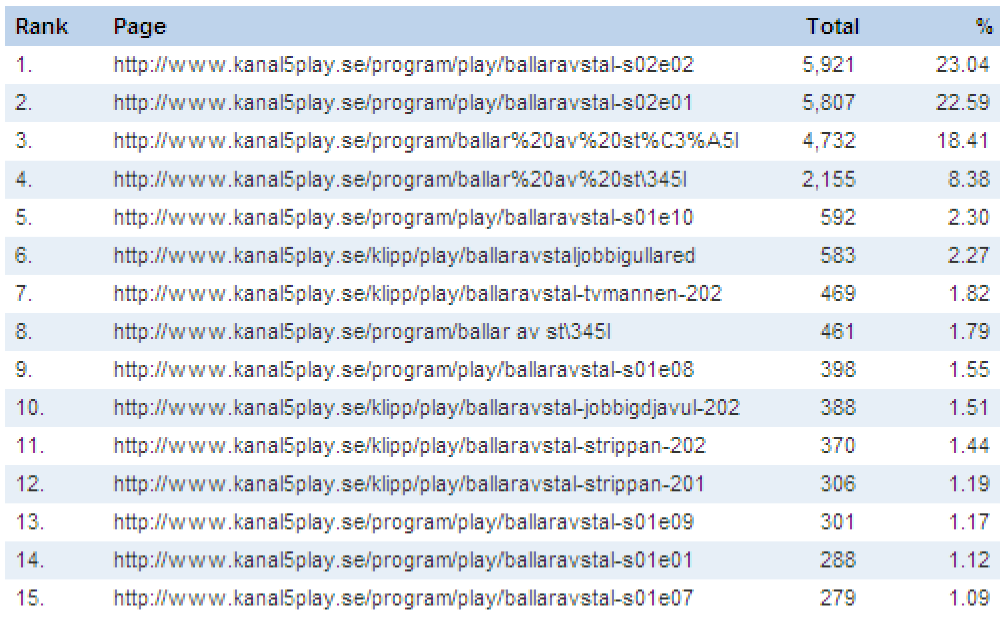


Figure A.1: Number of page loads for the TV show "Ballar av stål" during the 21th of March 2010. The episode was shown on TV during the time interval 21:00-21:55. These statistics show that most viewers accessed the play site just after the show was broadcast on TV (with the majority of accesses taking place between 22:00-24:00).



Rank	Page	Total	%
1.	http://www.kanal5play.se/program/play/ballaravstal-s02e02	5,921	23.04
2.	http://www.kanal5play.se/program/play/ballaravstal-s02e01	5,807	22.59
3.	http://www.kanal5play.se/program/ballar%20av%20st%C3%A5l	4,732	18.41
4.	http://www.kanal5play.se/program/ballar%20av%20st\345l	2,155	8.38
5.	http://www.kanal5play.se/program/play/ballaravstal-s01e10	592	2.30
6.	http://www.kanal5play.se/klipp/play/ballaravstaljobbigullared	583	2.27
7.	http://www.kanal5play.se/klipp/play/ballaravstal-tvmannen-202	469	1.82
8.	http://www.kanal5play.se/program/ballar av st\345l	461	1.79
9.	http://www.kanal5play.se/program/play/ballaravstal-s01e08	398	1.55
10.	http://www.kanal5play.se/klipp/play/ballaravstal-jobbigdjavul-202	388	1.51
11.	http://www.kanal5play.se/klipp/play/ballaravstal-strippan-202	370	1.44
12.	http://www.kanal5play.se/klipp/play/ballaravstal-strippan-201	306	1.19
13.	http://www.kanal5play.se/program/play/ballaravstal-s01e09	301	1.17
14.	http://www.kanal5play.se/program/play/ballaravstal-s01e01	288	1.12
15.	http://www.kanal5play.se/program/play/ballaravstal-s01e07	279	1.09

Figure A.2: Pages entered related to the show "Ballar av stål" on the day episode two of season two was shown (s02e02 in the above list).

TRITA-ICT-EX-2010:211

www.kth.se