Peer Assisted Live Video Streaming in Web Browsers using WebRTC

Jimmy Zöger
Marcus Wallstersson

Master of Science Thesis

Software and Computer systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

June 10, 2014

Examiner: Professor Mihhail Matskin
Abstract

This thesis presents a solution for peer assisted live video streaming in web browsers. The motivation behind the solution is that content providers, which need to allocate large amounts of server resources and bandwidth to support their services, could benefit from letting their viewers assist in distributing the video. Essential to this is the fact that live video streaming typically have relaxed time constraints, i.e. there is often a buffer of tens of seconds to allow for a smooth playback. The peer assistance is done with peer-to-peer connections that is natively supported in WebRTC-enabled web browsers. Peers cooperate by downloading different segments from the server and subsequently sharing this between themselves. For efficient utilization of the network, peers do also have a notion of the network topology and choose to cooperate with nearby peers. It is shown that server resources and bandwidth can be reduced by enabling peer assistance for suitable peers.

**Keywords:** Peer-to-peer, Live, Video, Streaming, WebRTC, HTTP based streaming, ISP-friendly P2P design, dash.js, PeerSim
Sammanfattning

Contents

1 Introduction ........................................ 1
   1.1 Background ........................................ 1
   1.2 Problem ........................................... 2
   1.3 Purpose ........................................... 2
   1.4 Goals ............................................... 3
   1.5 Methodology and methods ......................... 3
   1.6 Delimitations ..................................... 4
   1.7 Outline ............................................ 4

2 Extended background ................................ 5
   2.1 Video streaming ................................... 5
      2.1.1 Difference between live and on demand ....... 5
      2.1.2 Protocols ....................................... 6
      2.1.3 HTTP based protocols ......................... 6
         2.1.3.1 Adaptive Video Coding .................... 7
         2.1.3.2 Proprietary streaming protocols ......... 7
         2.1.3.3 MPEG-DASH ................................. 7
         2.1.3.4 SVC ........................................ 8
   2.2 WebRTC ........................................... 8
      2.2.1 NAT and firewall traversal in WebRTC ....... 9
      2.2.2 Signaling ....................................... 10
      2.2.3 Security in WebRTC ............................ 10
   2.3 Peer-to-peer networks ............................. 10
      2.3.1 Centralized .................................... 11
      2.3.2 Decentralized .................................. 11
      2.3.3 Overlays ....................................... 12
         2.3.3.1 Structured overlays ....................... 12
         2.3.3.2 Unstructured overlays .................... 13
   2.4 Peer selection .................................... 13
   2.5 Related Work ...................................... 14
      2.5.1 Peer-to-peer assisted CDN .................... 14
2.5.2 Streaming using peer-to-peer ........................................ 14
  2.5.2.1 Clive .................................................. 15
  2.5.2.2 SmoothCache .............................................. 15

3 Method ................................................................. 17
  3.1 Research approach .................................................. 17
  3.2 Data collection .................................................... 18
  3.3 Data analysis ...................................................... 19
  3.4 Ethical considerations and sustainable engineering ............... 19

4 Design ................................................................. 21
  4.1 Requirements ...................................................... 21
  4.2 Components ...................................................... 21
    4.2.1 Membership management: Cyclon .............................. 22
    4.2.2 Synthetic network coordinates: Vivaldi ..................... 24
      4.2.2.1 Vivaldi in detail ........................................ 24
    4.2.3 Peer selection: Close peer exploration ....................... 25
    4.2.4 Responsibility coordination of segments .................... 26
      4.2.4.1 The protocol ............................................ 27

5 Implementation ....................................................... 29
  5.1 PeerSim simulations ............................................... 29
    5.1.1 Vivaldi simulation .......................................... 30
    5.1.2 Close Peer Exploration simulation .......................... 31
    5.1.3 Responsibility coordination simulation ...................... 31
  5.2 Implementation of peer assisted live streaming ................... 31
    5.2.1 Prototype player ............................................. 32
      5.2.1.1 Timing of segment requests ............................. 32
    5.2.2 Peer connections ............................................. 35

6 Results ............................................................... 37
  6.1 Vivaldi simulations ............................................... 37
  6.2 Close peer simulations ........................................... 39
  6.3 Responsibility coordination simulation .......................... 40
  6.4 Peer assisted live streaming player ............................. 42

7 Analysis ............................................................. 45
  7.1 Vivaldi ......................................................... 45
  7.2 Close Peer Exploration ........................................... 45
  7.3 Responsibility coordination ....................................... 46
  7.4 Peer assisted live streaming player ............................. 48
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Relationships between the four components</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Example of shuffling between node 2 and 9</td>
<td>23</td>
</tr>
<tr>
<td>4.3</td>
<td>A two-dimensional spring system</td>
<td>25</td>
</tr>
<tr>
<td>4.4</td>
<td>Steps for Close Peer Exploration</td>
<td>26</td>
</tr>
<tr>
<td>4.5</td>
<td>Illustration of similarity for close peers. The blue peer shares two neighbours with the red peer and since they both have each other as neighbours their similarity is 50% (3 out of 6).</td>
<td>26</td>
</tr>
<tr>
<td>4.6</td>
<td>Illustration of one peer’s responsibility calculation, responsibility factor 2 will be chosen as it is the highest possible factor. Blue boxes marks responsible peers.</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Figures represents the implemented simulation scenarios.</td>
<td>30</td>
</tr>
<tr>
<td>5.2</td>
<td>Descriptions of how segment requests are processed in a regular vs. peer-to-peer assisted player.</td>
<td>33</td>
</tr>
<tr>
<td>5.3</td>
<td>Out of sync scenario peers ends up 1 segment duration off.</td>
<td>34</td>
</tr>
<tr>
<td>5.4</td>
<td>Time components of a segment request in peer assisted player</td>
<td>34</td>
</tr>
<tr>
<td>5.5</td>
<td>Establishment of WebRTC-connections</td>
<td>36</td>
</tr>
<tr>
<td>6.1</td>
<td>Convergence in Vivaldi</td>
<td>38</td>
</tr>
<tr>
<td>6.2</td>
<td>Visualization of Vivaldi. The color of the nodes is determined by the simulated ISP which adds more latency between ISPs. Coordinates are two-dimensional with a height vector.</td>
<td>38</td>
</tr>
<tr>
<td>6.3</td>
<td>This figure shows how close peer similarity changes by the close peer set size (network with 1000 peers).</td>
<td>39</td>
</tr>
<tr>
<td>6.4</td>
<td>This figure shows how close peer similarity is affected by joins of new peers. At cycle 300, 100 new peers joined the network consisting of 1000 peers.</td>
<td>40</td>
</tr>
<tr>
<td>6.5</td>
<td>Accuracy of responsibility coordination in a network with 1000 peers. Upload capacity is randomly distributed between 0.5-5</td>
<td>41</td>
</tr>
<tr>
<td>6.6</td>
<td>Segment request statistics in a network with 1000 peers. Upload capacity is randomly distributed between 0.5-5 and 0.5-10</td>
<td>42</td>
</tr>
</tbody>
</table>
6.7  Global statistics of test with 11 peers and video in sync. . . . 43
6.8  Global statistics of test with 11 peers and video out of sync . . 43

7.1  Theoretical maximal offloading . . . . . . . . . . . . . . . . 47
7.2  The peer assisted live streaming player built upon dash.js . . 49

A.1  Intersections of circles for a peer with only four neighbours . 57
A.2  Intersection of two circles . . . . . . . . . . . . . . . . . . . . 58
# List of Acronyms and Abbreviations

This thesis requires the reader to be familiar with terms and concepts of computer science. In this section acronyms and abbreviations are described shortly before using them in the thesis.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVC</td>
<td>Adaptive/Advanced Video Coding</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Distribution Network</td>
</tr>
<tr>
<td>DASH</td>
<td>Dynamic Adaptive Streaming over HTTP</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash Table</td>
</tr>
<tr>
<td>DRM</td>
<td>Digital Rights Management</td>
</tr>
<tr>
<td>HDS</td>
<td>HTTP Dynamic Streaming</td>
</tr>
<tr>
<td>HLS</td>
<td>HTTP Live Streaming</td>
</tr>
<tr>
<td>HTTP</td>
<td>The Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICE</td>
<td>Interactive Connectivity Establishment</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>MSE</td>
<td>Media Source Extensions</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>PEX</td>
<td>Peer Exchange</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>STUN</td>
<td>Session Traversal Utilities for NAT</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>SVC</td>
<td>Scalable Video Coding</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>WebRTC</td>
<td>Web Real-Time Communication</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Streaming media is one of the most popular and traffic consuming internet services. It represents nearly half of the aggregated internet traffic of the world during 2013 [1]. The trend do also indicate that traffic will increase due to higher video and audio quality but also due to an increasing number of media consumers. Naturally, a content provider has to pay for all network traffic consumed by their servers and due to the large traffic consumption in especially video streaming it is an area in need of improved distribution methods in order to save costs. This thesis proposes a solution for peer assisted distribution of live video content that helps in reducing such costs.

1.1 Background

Distributing a video stream over the internet is normally done by having a set of servers providing a total capacity that is at least the total capacity required by the viewers (clients). Naturally, the load of a live streaming service as such depend on the content popularity.

Having a live streaming provider to configure their servers to always accommodate for high user load is expensive regarding to both server and bandwidth costs. Instead a common solution is to let the servers scale with the user load. This scaling must be done in advance to a predicted increase of viewers and unused capacity will be a waste of money.

A possible solution to minimize the need for prediction is to enable peer-to-peer sharing of the live stream between the viewers. The service would still be dependent of servers distributing the video stream to a subset of viewers. Those viewers will serve others via the peer-to-peer network. If there is a peak in the load of a live stream it can be served by the peer-to-peer network, utilizing viewers’ upload capacity. This will offload traffic
Chapter 1. Introduction

from the origin servers, and thus allow for fewer servers and lower bandwidth usage.

Services that make use of peer-to-peer techniques are mainly implemented as desktop applications or plugins. This thesis presents a solution for enabling peer assisted live streaming in browsers without the use of plugins.

1.2 Problem

To find a solution like the one described, many questions need to be answered. How should the peer-to-peer overlay be formed? Should it be a structured or unstructured overlay? Included in these questions are the issues of peer information exchange, group membership and handling of churn.

Since live streaming is considered, how should quality of service be guaranteed? When should peers fall back to fetch video segments from the server instead of other peers? How do peers choose which other peers to cooperate with? This does also affect the question of how can the underlying physical network be used efficiently. Should peers coordinate the retrieval of different packets by means of an elected leader or should they act independently?

Regarding the video segments, how long should they be to allow for the most efficient cooperation? Should conventional encoding be used for different bitrates or is it possible to use layered video encoding to enable for cooperation between peers viewing the video at different bitrates?

These questions need to be answered in order to build a solution which thereafter can be evaluated. The evaluation includes questions about reduction of used resources and quality of the service. Can server and network resources be saved? Could popular video streams be scaled at a quicker rate with peer assistance?

1.3 Purpose

This thesis presents answers to the questions in the previous section and an evaluation of the proposed solution. Although the context here is browser-native peer connections with WebRTC, the answers and results could, with consideration, be extended to other contexts.
1.4 Goals

The goal of the project is to come up with a solution that could benefit content providers that offer live video streaming. It is desirable that the solution should be made as compatible as possible with current main technologies. A precondition is to support peer assisted live video streaming without having the end-user to install any browser-extensions. In other words, it should be browser-native to lower the threshold for an end-user to participate. Thus both content providers and end-users could benefit from it.

1.5 Methodology and methods

When conducting research the two most commonly used methodologies are quantitative and qualitative. Research that is of qualitative nature is based on collecting data that are not measurable, consisting of people’s opinions and experiences about some phenomenon or product. The collected data are then analyzed in order to gain understanding or to create hypotheses related to the research question. The qualitative research is commonly used within social science studies. [2]

Research that addresses data of a measurable kind is called quantitative, its focus is to prove or verify hypotheses and/or functionalities of a computer system or algorithm (in the case of computer science). The quantitative research draws conclusions from the data collected through experiments and testing. As this research project addresses development and performance evaluation of a live streaming solution it comes naturally to use the quantitative research method due to the fact that the evaluation of the developed system’s performance yields quantifiable data. [2]

The research methods applied are mainly applied research but also to some extent experimental. Applied research is used as the main problem is to solve the practical problems related to develop peer-to-peer assistance for a video streaming service. The experimental method is used to study how different configurations of the system and algorithms affect the performance of the end product.

Due to the fact that the feasibility and performance of the system is to be proven and measured the deductive research approach is chosen. The deductive approach is suited to verify the validity of a proposed hypothesis and to express the relationships of dependent variables in the system.

A more detailed explanation is presented in chapter 3 of how these theories are applied practically in the thesis project including other aspects such as
research strategies, data collection, and data analysis.

1.6 Delimitations

The presented streaming system is designed for use with adaptive bitrates but it is only evaluated with a fixed bitrate. The choice of using a fixed bitrate is to simplify the testing by requiring less participants. The choice is also followed by the reason that the bitrate adaption algorithm needs to be re-designed when peer assistance is used since the segment requests are done with a longer time frame than without peer assistance. Bitrate adaption is player specific and outside the scope of this thesis project.

1.7 Outline

The thesis is structured with the following chapters:

Chapter 2
Provides an extended background about video streaming technologies, WebRTC, and peer-to-peer concepts. Additionally related work that inspired this thesis are presented.

Chapter 3
Describes how the presented research methods are applied in this thesis.

Chapter 4
Presents the requirements and design choices done in the implementation of the streaming system.

Chapter 5
Presents how the browser extension and its corresponding simulations were implemented.

Chapter 6
Presents results from simulations and the evaluation of the browser extension.

Chapter 7
Analyses the results collected through the simulations and testing.

Chapter 8
Concludes the work done in the thesis project and presents future work.
Chapter 2

Extended background

This chapter presents a wider background to related technologies and previous work. First, concepts within video streaming and peer-to-peer will be explained and lastly related solutions to peer-to-peer video distribution is presented.

2.1 Video streaming

Streaming refers to the process of simultaneously delivering media, from a provider to a consumer, and presenting it for the consumer. A typical example is that a consumer can start viewing a video before all of it has been transmitted and that transmission will continue while viewing. The media could be of various types but do most commonly refer to video or audio. In this report, the term video may often refer to the combination of both video (image) and audio.

2.1.1 Difference between live and on demand

There are basically two different applications for streaming video, live or on demand. On demand video means that the provider has the media available on a file, ready to be transmitted when the consumer requests it. Live video could further be divided into two different applications, real-time video and relaxed live video. Real-time video refers to applications such as video conferences where a lag higher than a few hundred milliseconds causes negative impact on the user experience [3]. Relaxed live video refers to applications without interactivity such as Internet TV where a lag of, for example, 30 seconds is acceptable. In fact, similar lags are common for
Chapter 2. Extended background

2.1.2 Protocols

There are numerous ways to achieve streaming video. Some protocols are designed for real-time video, e.g. RTP, and therefore build upon the User Datagram Protocol (UDP) which only has a best-effort transmission model [4]. The lack of guarantee for delivery is however preferable over resending packets which anyway will be outdated at arrival.

Common to all real-time protocols is that they inherently need to be push based, i.e. the media is sent to the other part directly after being captured. For example, a video frame is sent as soon as it is available. On demand video could also use this approach but the possible loss of packets raises a dilemma, should a client player implementation proceed and thus lower the video quality or should the client player wait for the packets to be resent and thus pause the video? The solution is to buffer an upcoming part of the video and use this to hide retransmission time of lost packets. Reliable transmission protocols with guarantees of delivery are thus used in solutions that use buffers.

To achieve a buffer in a push based system, the player would have a reasonable lag behind the true stream to allow for retransmissions. However, the common approach to this is to have a pull based system, i.e. the client requests parts of the video as they are needed including upcoming parts to allow for a buffer. Relaxed live video could also use this approach, the server provides parts within an interval that the client is able to prefetch. The main current technologies for pull based streaming do all use HTTP as transport protocol, these will be briefly described in the next section.

2.1.3 HTTP based protocols

For push based streaming, the servers do naturally need to have streaming capabilities. These kind of servers are not as ubiquitous as regular HTTP servers. A main advantage of of having HTTP as transport protocol is that general purpose content distribution networks (CDNs) can be utilized. These are in general more cost effective than solutions specific to push based streaming [5]. Examples of protocols which are HTTP based are HTTP Live Streaming (HLS), Microsoft’s Smooth Streaming, Adobe’s HDS and MPEG-DASH.

According to conversations held with employees at Millicom Digital Ventures that have been working with various Internet TV services.
2.1. Video streaming

2.1.3.1 Adaptive Video Coding

Adaptive Video Coding (AVC) is a technique for letting the client dynamically change video quality while playing. The original video file is encoded into different representations where quality, resolution and codec can vary. Each representation is also divided into several smaller files called segments. Information about the representations and segments are placed in a manifest file which is used by the client to choose representation and start playing. While playing, the client can monitor the bandwidth conditions and if necessary, change representation by downloading upcoming segments from another representation. This allows for seamless adaptation to changing bandwidth conditions on the client side.

2.1.3.2 Proprietary streaming protocols

HTTP Live Streaming (HLS) is Apple’s protocol for AVC [6]. It is widely adopted but also compete with Microsoft’s Smooth Streaming [7] and Adobe’s HTTP Dynamic Streaming [8].

2.1.3.3 MPEG-DASH

MPEG-DASH is the result of an attempt to start standardize HTTP streaming formats across all kinds of devices. It is an open standard, as opposed to the proprietary standards mentioned in the previous section, and standardized by ISO.

DASH manifests are in XML format and supports a templated layout from which the segments to play can be calculated. This is in contrast to having to specify every segment explicitly in a timeline. In the manifest, different video stream options are presented: video and audio bitrates and also language options. The standard is also agnostic about DRM (Digital Rights Management) and supports common encryption (CENC). The choices of DRM is specified in the manifest as well as codecs used for the audio and video segments.[9]

Through the manifest the stream can be adjusted to have specific playback options. Some of the options include recommended buffer length and time shifting capabilities. In a live scenario the buffering is achieved by a sliding window, where the playback is behind the live edge. The amount of offset behind the live edge limits the maximum buffer capacity.[9]
2.1.3.4 SVC

Scalable Video Coding (SVC) is an approach that achieves adaptive video quality without the need for switching between representations. Instead the video is divided into different layers - a base layer and multiple enhancement layers. The enhancement layers can provide either better image quality, resolution or frame rate. Segments are still used and video players need to fetch as many layers as required for each segment. Using SVC in encoding of a stream only needs one encoder and takes up less storage capacity compared to AVC. [10]

SVC is an interesting approach for peer assisted video since some layers are the same even if clients are at different bitrates and thus clients could still cooperate. However, the main problem is the lack of support in web browsers and it can thus not be used in this project.

2.2 WebRTC

Video conference applications based on video and audio are due to its nature typical real-time communication (RTC) applications. They depend on very low latencies to be functional and are thus best implemented with peer-to-peer connections. Services offering these applications in browsers have — until recently — had no other choice but to implement this with browser extensions that need to be installed by end-users. This is an unwanted obstacle for these service providers. For example, installing and updating browser extensions can be problematic for less skilled users. Another drawback from a service provider’s point of view is the need to develop and maintain a proprietary browser extension for their service. WebRTC - which integrates real-time communication into browsers - is a solution to these problems.

The history behind WebRTC is that in 2011 the Web Real-Time Communications Working Group was created with the goal to solve the issues around RTC in a standardized way, supporting browsers on different kinds of devices without the need of browser extensions. Recently WebRTC has defined open standards for browser-native real-time video, audio and data communication. [11]

WebRTC defines an API that has to be implemented for supported browsers. At the moment, the only browsers that supports WebRTC are Google Chrome, Firefox, and Opera [12, 13].

The WebRTC API has three main components used for real-time communication:
2.2. WebRTC

**MediaStream** The MediaStream API offers functionality to capture synchronized audio and video from devices and store it in a MediaStream object. Each MediaStream object can accommodate multiple tracks of audio and video, which are sent to a connected peer by plugging in the MediaStream object as input to a RTCPeerConnection. [14, 15]

**RTCPeerConnection** The RTCPeerConnection is used to set up the actual connection between peers and to send the media contained in the MediaStream object. Connections to peers behind Network Address Translation (NAT) and firewalls can be established by RTCPeerConnection with the use traversal protocols. Additionally with WebRTC it is possible to encrypt [13] and manage bandwidth over the RTCPeerConnection. [15, 16]

**RTCDataChannel** To send arbitrary data, an RTCDataChannel has to be created. It uses an already established RTCPeerConnection. RTCDataChannel uses the same API as web-sockets and provides low latency traffic which can be reliable or unreliable. As encryption protocol for arbitrary data it uses DTLS between peers. [15, 16]

### 2.2.1 NAT and firewall traversal in WebRTC

In real time peer-to-peer communication, Network Address Translation (NAT) and firewalls are big obstacles. A connecting peer may be behind a NAT and thus it has no knowledge of its external IP-address and port (external address of NAT). A technique called *hole punching* solves the problem of connecting peers behind NAT, which in WebRTC is implemented through the Interactive Connectivity Establishment protocol (ICE) [17]. The ICE-protocol consists of three mechanisms to detect a set of candidate addresses. These are detected through local interfaces, STUN-, and TURN-protocol [17]. [18]

The first method is intended to work if the peer has an public address or if the peers are located behind the same NAT or firewall. It will scan local interfaces and all addresses of the network interfaces are added to the candidate set (host candidate). [17]

As a second method it uses the STUN protocol with an external STUN-server which the client behind a NAT will issue a request to. The STUN-server will investigate the request packet and extract the source address of the packet, which points to either the address of the NAT or directly to the client (if not behind a NAT). The source address is sent as a response to inform the client of its external IP-address and added as a candidate. [17, 18]
The third method is with the TURN-protocol, instead of detecting the address it provides the client with an address to a relay-server (relay address). All traffic to the client will be sent to this relay and then forwarded to the client. [17, 18]

When connecting to a client, the candidate addresses will be evaluated until one that works is found. The first address to test is the client’s host candidate which only will work if both clients are behind the same NAT or not behind NAT’s. If no connection can be established through host candidate the server reflexive address is tested. The last attempt is done through the relay address which always will work. [17, 18]

2.2.2 Signaling

Even though WebRTC enables real-time communication between peers in browsers it still needs to rely on servers for certain tasks to establish a connection. This is commonly referred to as signaling. Two clients that wish to set up, for example, a video chat connection are not able to just connect to each other without any prior knowledge. Both clients has to find out how to locate each other, which is done by exchanging their candidate addresses obtained by the ICE-protocol. Moreover the clients exchange session control messages to open and close connection, metadata (codecs and bandwidth information) and session keys used to secure the data transfer. The signaling protocol is not standardized in WebRTC and has to be implemented by the service provider to fit each service. The motivation to have a custom signaling protocol is that existing services are already using established protocols such as SIP or Jingle which can easily be integrated in WebRTC. [19]

2.2.3 Security in WebRTC

To mitigate potential threats, encryption is used for all WebRTC components and signaling phases. The WebRTC application do also execute in a sandboxed environment within the browser. To gain access to camera and microphone, browsers typically require the user to explicitly allow it. [20]

2.3 Peer-to-peer networks

In peer-to-peer systems the connected clients are referred to as peers. All peers that participate in sharing a certain file or video stream are grouped into a swarm. A peer that wants to participate in a swarm must know at least one existing peer in that swarm to be able to join. When fully connected
to the swarm a peer holds a subset of the peers called neighbours that it communicates with to exchange data. This arrangement of peers is called an overlay and it can be either structured or unstructured. The difference lays in how neighbours are chosen. The selection can be strict and based on some rule to form a structured overlay such as a tree or DHT ring. It could also have random elements to form an unstructured overlay such as a mesh. Knowledge about peers available in the swarm is obtained from a peer discovery service often called a tracker, which can be implemented in a centralized or decentralized way [21]. How the different types of trackers work are described in the following sections.

2.3.1 Centralized

In the centralized approach the tracker is implemented as a server holding a global view of all peers in a swarm. When a new peer is about to join a swarm it contacts the tracker and registers itself as a member of the swarm. The tracker will reply with a set of peers which will be used as neighbours to the new peer. In a fully centralized solution the tracker is the only source of updating neighbours to maintain the overlay. This results in scalability issues when the size of the swarm grows bigger. A centralized approach is fast in terms of updating neighbours and optimizations of how peers are selected can easily be applied. Conversely it is a single point of failure and if the tracker crashes, peers in the swarm will not get the possibility to update their neighbours until the tracker is restarted. A tracker failure could thus lead to partitions in a structured overlay in the presence of peers failing or leaving the swarm. In BitTorrent the conventional approach is to have a centralized tracker. [21]

2.3.2 Decentralized

Decentralized peer discovery is an approach to solve the issues of scalability and availability of centralized trackers. In this approach peers can cooperate to provide the peer discovery service by maintaining a Distributed Hash Table (DHT) that contains information about the peers in the swarm. Updating a neighbour set is done through lookups in the DHT which is both scalable and failure resistant, but to maintain the DHT structure creates a lot of communication overhead.

Peer discovery can additionally be done through gossiping, where all peers in the swarm continuously exchange their neighbour sets and cooperatively maintain the structured or unstructured overlay. Using gossiping will create a highly scalable and failure resilient peer discovery service. In BitTorrent
Chapter 2. Extended Background

a protocol called PEX is implemented in several clients [refs to clients using PEX] to do peer discovery by gossiping, it has been shown that PEX reduces the download time of torrents due to effective peer discovery [21].

Even if the techniques described are called decentralized they still require a centralized bootstrap server which helps new peers to find its way into the swarm. A connecting client will receive one or more addresses to peers in the swarm to be used as an initial set. After bootstrapping, the peer can continue to exchange neighbours without the use of the bootstrap server through a decentralized peer discovery protocol.

2.3.3 Overlays

An overlay is what connects the peers in a peer-to-peer network. Depending of their type, overlays have specific characteristics. When building a peer-to-peer network it is crucial to select the overlay which is the most appropriate in respect to the service. Construction of an overlay can either be structured or unstructured depending of how neighbours are maintained. The two following sections will present overlays and their properties in video streaming systems.

2.3.3.1 Structured overlays

A typical example of a structured overlay is a single-tree overlay. This is commonly used together with push based dissemination of video segments between peers with its root located at the streaming server. Each peer will join the tree at a specific location (level) according to the directives from the tracker service. An internal peer has one parent peer which it receives segments from, additionally it has one or more child peers that it serves by pushing video segments to. The outer most peers without children is called leaf peers and these will not contribute at all in the data dissemination since video segments are only disseminated in an descending way through the tree. The single-tree overlay is easy to manage and offers a short and predictable latency. The ease of management and its predictive latency comes with some drawbacks: it is sensitive to node failures and upload capacity of the leaf nodes will be unused. In the case of node failure, the subtree rooted under the failing node will have a disruption of the stream until the tree is restructured. [22, 23]

One approach to utilize the upload capacity of leaf nodes is to to split the stream into sub streams, creating one tree for each sub stream. This approach is a push based overlay called multiple-tree and each peer has to join all of its trees, the difference to single-tree is that peers join on different places in each tree. A peer that is a leaf in one tree can be placed as an
2.4 Peer selection

This thesis project focuses on peer assisted distribution of content while making efficient use of the network. It is therefore important to have an overlay which gives precedence to nearby peers when choosing neighbours. The closeness of a peer can be measured in distance of communication delay or by measures of ISP and IP subnets. By having peers close in communication distance the service can be optimized to allow minimal delay in peer-to-peer communication. Costs of inter ISP traffic can be reduced by preferring peers within the same ISP. Typically, the communication delay is also lower within an ISP than between ISP’s.

internal peer in the other trees, allowing its upload capacity to be utilized. Multi-tree overlays are more resilient to node failures than single-trees and offers good load balancing. A failure will still disrupt the dissemination of a sub stream but with smaller impact than using single-tree. [22, 23]

2.3.3.2 Unstructured overlays

Unstructured overlays are typically mesh based. These address the problem with single point of failures in tree based overlays. The mesh based overlay allows peers to initiate peer connections without hierarchy (no parents or child peers), each peer can have multiple neighbours which will improve the resistance to failures. In a mesh the neighbouring peers are a randomly chosen subset, with the intention to create an overlay topology as close to a random graph as possible. Data dissemination is commonly pull based, pushing data between peers will result in overhead when peers receive multiple copies of the same video segment from its neighbours. The overhead created by pushing can be solved by coordinating neighbours to not push duplicate segments, in contrast the coordination introduces some overhead making pull based a strong candidate. To minimize overhead in pull based systems the neighbours will exchange buffer maps describing which video segments are available for request. The pull based mesh overlay is a good solution to use in HTTP based streaming protocols since HTTP is request (pull) based. [22, 23]

The randomness of the mesh and frequent exchange of buffer maps makes latency prediction hard. There can exist several paths from the streaming source to a peer which results in variable latency. The advantages of a mesh based overlay is: it is resilient to failures due to multiple non hierarchical peers, it has good load balancing and it is very simple to implement and maintain the overlay. [22, 23]
A lot of work has been done in the field of using the networks efficiently, ranging from topology aware and ISP friendly protocols [24, 25] to synthetic coordinate systems as Vivaldi [26]. Vivaldi is especially interesting since network coordinates based on latencies are assigned in a fully decentralized manner and with very little overhead. These coordinates are then utilized for predicting the latency to any other node where its coordinate is known.

2.5 Related Work

There are of course interesting services which already make use of peer-to-peer technology. Some of them are proprietary with very limited documentation while other are well explained. The services that influenced this thesis project will be described in this section.

2.5.1 Peer-to-peer assisted CDN

With the popularity of peer-to-peer techniques several services try to take advantage of it to save bandwidth and build scalable services. Hosting media at a CDN is a good solution but it is still costly. SwarmCDN[27] and PeerCDN[28] are two companies that offers an extension to using a traditional CDN to lower the distribution costs. Their solutions make use of the visiting user’s browser by letting it serve static content from the cache to other visitors. This is all done with a small javascript extension and peer connections through WebRTC. If the swarm can not provide the content in time, the original CDN is used as a fallback [27, 28]. The saved bandwidth compared to regular CDN distribution in SwarmCDN is 70% and 90% in PeerCDN [27, 28].

2.5.2 Streaming using peer-to-peer

In VOD and live video there are several benefits of assisting the service with peer-to-peer technology, it reduces the amount of bandwidth needed at the streaming service and adds scalability to the service. StreamRoot[29] and Peer5[30] are two companies which have released proprietary peer-to-peer VOD streaming services. Both execute browser-native with WebRTC to enable peer-to-peer communication and relies of the streaming server as a fallback. Peer5 showed that their technology was able to save 90% of the bandwidth costs when deployed for the VOD service Kaltura, with a startup delay and buffering well under industry average[31]. Streamroot presents that the bandwidth reduction is between 50-90% [29].
An upcoming service for peer-to-peer live video streaming is BitTorrent Live[32] which is a proprietary protocol recently released by the founders of BitTorrent. It builds upon the successful BitTorrent protocol and requires the user to have a client installed in order to watch the live stream. A second service called SopCast[33] also provides a peer-to-peer live streaming service, which also requires a separate client to be installed.

### 2.5.2.1 Clive

In live streaming, the timing of video segments needs to meet a soft real time constraint. When built with peer-to-peer, meeting these constraints can be problematic due to bandwidth limitations of the swarm and streaming server. A paper[34] by Amir H. Payberah et al. presents a cloud-assisted peer-to-peer live streaming system called CLive which tackles the problem about the time constraints, to provide the end user a guaranteed QoS. The CLive paper defines QoS in live streaming as high playback continuity and short playback delay. The two metrics are negatively correlated to each other, since a higher playback continuity may require more buffering and thus the playback delay is increased. [34]

The idea behind CLive is to dynamically assist the peer-to-peer network with servers the authors call helpers to balance the load of the peer-to-peer swarm. These helpers are cloud resources of two types, Amazon EC2 as active helpers and Amazon S3 persistent storage as passive helpers. Active helpers will assist as participants in the mesh based swarm to increase its upload capacity. If a peer can not receive a video segment in time the passive helper is contacted as a fallback to retrieve the segment.

By using distributed aggregation algorithms the size of the swarm and its upload capacity is estimated. The estimations are used as a metric for scaling the number of active helpers. If the swarm is too large for its current upload capacity, active helpers are added. In the proposed solution there are only one passive helper since it is considered to be sufficient. The results of CLive showed that with adaptive number of active helpers the cost was reduced by 45% compared to without the use of active helpers [34].

### 2.5.2.2 SmoothCache

A paper by Roberto Roverso et al. addresses the topic of using HTTP based live video streaming. A system called SmoothCache is proposed that utilizes peer-to-peer data dissemination by enabling peers to act as HTTP caches of video segments requested within the swarm. While prioritizing peer-to-peer traffic, the streaming server is used as a fallback if the swarm fails to deliver
segments in time. A motivation to use HTTP as protocol is that it is firewall friendly in comparison to the real time protocols. [35]

The proposed system is compared to a baseline HTTP streaming system without peer-to-peer support. In the report, several enhancements of SmoothCache are implemented and compared to the baseline system, some of them are:

- Traffic prioritization using DTL[36] where requests of video segments with a close deadline is prioritized over non-urgent segments.

- To maximize the utility of a peer with high upload capacity in the swarm, SmoothCache uses Manifest manipulation. By modifying the manifest, higher bandwidth peers can be served the stream a bit earlier than other peers. This allows data to be available with a higher probability at higher bandwidth peers when requested by low bandwidth peers.

- Each peer is associated with the bitrate it is currently using. This knowledge is used when selecting peers for the neighbour set. The peers are selected with a gaussian distribution centered around the selecting peer’s current bit rate. This will ensure a concentration from the current as well as some from the other bit rates which allows peer connectivity when the player changes bitrate.

- Proactive caching is used to download and cache segments in advance, even before the video player requests them. This is a risk since it could be possible that the player will change bitrate before this element is required. This would mean unnecessary work since the other segment still need to be fetched.

The evaluation of SmoothCache showed that the enhanced peer-to-peer solution saved 78% of the costs and 76% of the viewers had no more delay than 5 seconds compared to the baseline system which has the same latency. [35]
Chapter 3
Method

The main method, methodologies, and their motivations were presented in section 1.5. In the following sections a more detailed description is presented of how the thesis project was conducted in respect to approach and methods.

As mentioned in section 1.4, the goal of the project is to develop a browser-native peer-to-peer assisted live streaming solution with the intent to reduce costs for the service provider.

3.1 Research approach

To conduct this thesis, the project has gone through several phases. The first phase of the project was to explore the area of peer-to-peer live streaming through a literature study. Through this study insights into issues and solutions related to the area were gained. These are presented in chapter 2. Knowledge from the literature study were used to define the problems in section 1.2 and to outline a design for the proposed streaming system. Mathematical proofs and simulations were done to verify the validity of the proposed solution (see chapter 4). From the mathematical proofs and simulations it was possible to improve the proposed solution before implementation.

The applied research focused on coming up with an peer-to-peer assisted live streaming prototype to be used for the experimental research in an real world setting and in simulations. As a base for the implementation an existing MPEG-DASH player was used since implementing a player is outside the scope of this project. The actual work lies in developing a peer-to-peer extension to be used in combination with an existing player.

The experimental phase focused on gathering data, this was done in two methods; through simulations and tests of the prototype. Both methods
focused to verify the behaviour and performance of the proposed solution. The simulation method was used to abstract away all video player operations and networking parameters, such as delay, message loss with the intention to prove the developed algorithms in a delimited setting. Through simulations it is feasible to test the solution in larger peer-to-peer networks without having to deploy a large scale online test on a peer cluster.

Tests of the implemented prototype were conducted on a small scale - but in a non delimited setting in contrast to the simulations. These test data gave a better view of some aspects not present in the simulations.

Additionally the experimental data were used to illustrate and draw conclusions about the systems performance. Through the experiments that lead to greater knowledge about the systems behaviour some improvements were suggested for future implementation.

### 3.2 Data collection

Through simulation and experiments data were gathered about the proposed system. In the project two methods were used to collect the data, live statistics from the prototype player and simulation data of the algorithms.

Simulations were conducted in a simulation framework called PeerSim [37] especially developed for peer-to-peer simulations. It has the functionality to simulate protocols in a cycle driven or event based approach. In the cycle driven approach the simulation runs for a number of cycles and each protocol is triggered to run one time each cycle. When working with event driven simulations the protocols are triggered by emitted events and the simulation ends at a specific time limit (simulation time). Event driven simulations are suited to simulate event based protocols and can be combined with the cyclic method. In the case of this thesis project cycle driven simulation was used since the protocols of the prototyped system executes with fixed intervals. Due to determinism and reproducibility of the simulations it is possible to evaluate and improve the algorithms of the prototype from the simulation data. Though, simulations may overlook problems that only occur for the implemented prototype since simulations are mostly focused towards the algorithms.

To gather data about the implemented prototype on a smaller scale, tests with employees at Millicom Digital Ventures were performed.¹

Global data of the test was collected by a statistics server that received client reports of statistics related to the streaming performance.

¹The thesis was conducted at Millicom Digital Ventures.
3.3 Data analysis

The process of running simulations and then analysing the results was automated by using a Python plotting library called matplotlib and R for statistical analysis. Usage of matplotlib and R varied between different tests and simulations, ranging from plotting graphs to find relationships between parameters of the system; to visualisation of the peer-to-peer topologies in 2D and 3D animations.

To get reliable data for the analysis the simulations where repeated 10 times each to create 10 different data sets of the simulation. This generated a sample size sufficient for calculation of confidence intervals. Intervals of 95% where used in all plots that contains confidence intervals.

3.4 Ethical considerations and sustainable engineering

Throughout the thesis project some choices have been made regarding ethics of the conducted research which are presented in this section.

The implemented peer assistance extension for the video player is licensed as open source to allow for further development by anyone and thus drive the area of peer-to-peer live streaming forward. One of the positive factors is that an open source peer assisted streaming solution enables companies or organizations with a limited budget to distribute a live stream to their viewers. On the other hand, it can also lower the bar for illegal distribution of copyrighted material. Illegal distribution of video was not considered to be a limiting factor that affects the project since the streaming content is a responsibility of the distributor. The streaming content chosen for the tests with the prototype was licensed as Creative Commons Attribution 3.0 which means that copyright was not infringed.

The ethics behind utilising each viewers bandwidth to distribute the stream was discussed. A user will contribute with its resources but will get a service that may cost less in the case of a subscription based streaming service. For users with an internet connection with a limited data plan peer-to-peer traffic can possibly be a negative aspect. A study could be done to find the users opinions about peer-to-peer traffic utilisation to evaluate how it is viewed by end users, but that is outside the scope of this thesis.

The project has environmental aspects since a reduced need for streaming servers would lead to smaller data centers. Smaller or fewer data centers lowers the power consumption and reduces CO2 emissions.
Chapter 4

Design

This chapter describes the design of the components in the streaming system and the underlying requirements from which the design is derived.

4.1 Requirements

As described in the introduction chapter, the goal is to offload servers with peer assistance to enable quicker scaling of popular streams. Special care has to be taken so that the underlying physical network is not overloaded. This is of particular importance for ISPs which also has to bear the costs for the network utilization. To efficiently use the network, cooperation should only be performed with close peers. A way to find close peers in the peer-to-peer network is therefore needed. This implies that we need a member management mechanism to create the peer-to-peer network. New peers should only need an initial set of peers from a bootstrap server to be able to join the network. In addition this, responsibility coordination for the segments is also needed. Communication for this needs to be kept to a minimum to lower the overhead.

4.2 Components

This section will explain the components, their design and the relationships between them. A simple illustration of the components and their relationships can be seen in figure 4.1. The protocols are first briefly described to give an overview picture and are later further detailed in subsections.

Cyclon [38] is a membership management protocol which efficiently creates a graph with low diameter and also is able to handle high node churn. It is thus used to keep the peer-to-peer overlay network intact.
Vivaldi [26] is a protocol for calculating synthetic network coordinates in a fully decentralized manner. It uses random peers from Cyclon to calculate a position where the distance between two points should estimate the communication latency between the corresponding peer.

Close Peer Exploration is a protocol developed for this thesis project. It holds the closest peers that are known around the position calculated by Vivaldi. These peers initially come from Cyclon and later also by chance from Cyclon. However, they do mainly come from exchanges with other peers. The goal is to have the closest possible peers available for cooperation.

On top of this, responsibility for the different segments needs to be assigned among the cooperating peers. This coordination is basically deterministic since it is decided based on a unique ID of a peer and the segment number. Responsibility decision is done optimistically at each peer based on the known upload capacities in their peer sets. The level of offload do however vary due to diverse upload capacities.

### 4.2.1 Membership management: Cyclon

Fundamentally in peer-to-peer systems, peers need to be able to find each other to be able to communicate. This can be done both with a central service and decentrally. In line with the purpose of this thesis, a decentralized solution is used. Cyclon which is an unstructured overlay has been chosen since structure based on the ID of the peers is not desirable. Instead, the structure comes from the positions based on Vivaldi so that the closest neighbours can be found through Close Peer Exploration. Cyclon does instead provide peer sampling and keeps the overlay network connected. The
4.2. COMPONENTS

reason Cyclon has been chosen and not a similar protocol like NewsCast is that Cyclon is more efficient and provide good random graph properties, such as low diameter and low clustering and also is resilient to high amounts of churn [38, 39].

(a) Before shuffling  
(b) Shuffle operations  
(c) After shuffling

![Diagram](image)

Figure 4.2: Example of shuffling between node 2 and 9

Each peer executing the Cyclon protocol holds a cache which is constantly updated through shuffle operations with other peers. The cache in cyclon contains peer descriptors which represents the neighbours of the peer. Choosing a peer to shuffle with can be done in two ways: random peer (basic) or the oldest peer (enhanced). The enhanced solution uses an age parameter for each peer descriptor, which is increased in every shuffle operation for every peer in the cache. The oldest peer is removed from the cache when selected, limiting the lifetime of descriptors and stops dead peers to circulate in the network. In the basic solution it is also removed, still peer descriptors can circulate indefinitely since there is no age limit controlling the removal.

The presented solution uses Cyclon with the enhanced shuffling to keep the network resilient and free from dead descriptors. Using Cyclon instead of NewsCast saves bandwidth, since NewsCast exchange its complete cache compared to a subset ‘L’ in Cyclon. A description of the Cyclon protocol is presented below where P is the node initiating the shuffle operation. [38]

1. Increase by one the age of all neighbours.

2. Select neighbour Q with the highest age among all neighbours, and L-1 other random neighbours.

3. Replace Q’s entry with a new entry of age 0 and with P’s address.

4. Send the updated subset to peer Q.

5. Receive from Q a subset of no more that L of its own entries.
6. Discard entries pointing at P and entries already contained in P’s cache.

7. Update P’s cache to include all remaining entries, by firstly using empty cache slots (if any), and secondly replacing entries among the ones sent to Q.

A scenario with enhanced shuffling can be seen in figure 4.2 where the shuffle is initiated by peer 2 which selects its oldest peer 9. Peer 2 removes its link to peer 9 and pushes its own descriptor among others to node 9 which responds with three random nodes. The effect of the operation is some neighbour changes including a reversed link between peer 2 and 9. In each shuffle interval one shuffle operation is completed at every node in the overlay, as mentioned this creates a overlay resilient to membership changes.

4.2.2 Synthetic network coordinates: Vivaldi

This project aims to offload traffic from content distribution servers while not affecting the load on the physical network to the worse. Therefore inter-ISP traffic should be avoided as well communication over longer geographic distance. For this, IP addresses and lookups of the autonomous systems (AS) can be used but they are neither precise nor directly related to what is important - network distance. This distance, embodied in round trip time, makes for a far better foundation for deciding neighbours. Pairwise measurements of round trip time between all peers in a peer-to-peer network is however not feasible.

Vivaldi is a fully decentralized protocol which assign synthetic network coordinates to peers based on round trip time measurements. A peer do only need to measure its distance to a subset of all peers and can from this find its position. The absolute position does not have any meaning, it is just in relation to other peers’ positions it is useful. The distance between two peers’ positions should estimate the round trip time between those peers. The coordinate system can be of any dimension, e.g. two dimensional or three dimensional. To conclude why Vivaldi is a good fit is that it provides a general solution to find close peers and avoid inter-ISP traffic.

4.2.2.1 Vivaldi in detail

Vivaldi was inspired by spring systems [26], an example is shown in figure 4.3. The position of a peer corresponds to a vertex in the graph and each pairwise latency is represented by a spring between the two peers. The estimated latency between two peers is defined as the distance \( est_{ij} \) between them. However, the real latency \( rtt_{ij} \), which corresponds to the spring’s rest
length, may be different and thus introduce an error $e_{ij} = |est_{ij} - rtt_{ij}|$. The total potential energy in such a spring system is the sum of all pairwise errors squared, $E = \sum_i \sum_j e_{ij}^2$. Each node computes and continuously adjusts its coordinates to minimize this error. Measurements of round trip time are only needed to a subset of nodes. In such a measurement, the coordinates of the measured peer are sent back. Thus can the error and the direction be computed which combined constitutes a force vector. The peers can then adjust its coordinates a small step according to the force vector. These steps should not be too big to avoid oscillation. Vivaldi peers has an uncertainty factor which makes new and uncertain nodes move faster while more stable nodes are more reluctant to changes.

![A two-dimensional spring system](http://en.wikipedia.org/wiki/File:Elastic_network_model.png)

4.2.3 Peer selection: Close peer exploration

On top of Cyclon for membership management and Vivaldi for synthetic network coordinates, a mechanism for finding the closest neighbours in the peer-to-peer overlay is needed. For this, Close Peer Exploration has been developed. A limited set of neighbours are kept and in each cycle a random neighbour is asked for the neighbours it knows that are closer to the requesting peer than itself. Peers from Cyclon can also be merged into this peer set. Although since they are random, they are not likely to be very close. Figure 4.4 shows how one peer asks another peer for closer peers. If the size of the neighbour set is bigger than the limit after merging in the new peers the most furthest will be pruned from the set. By iteratively asking neighbours for even closer neighbours, the protocol will eventually obtain the
closest neighbours possible (assuming that positions will converge to a stable state).

(a) The blue peer asks the red peer for close neighbours.
(b) The red peer replies with the two neighbours that is closer than itself.
(c) The blue peer adds the new peers as neighbours.

Figure 4.4: Steps for Close Peer Exploration

This protocol aims to achieve as high similarity of close peers as possible. This similarity is defined as the amount of overlapping peers in the close peer set among the close peers. Figure 4.5 shows how a similarity of 50% is calculated between two peers. This does also show that an average similarity of 100% is not achievable except in some special cases.

Figure 4.5: Illustration of similarity for close peers. The blue peer shares two neighbours with the red peer and since they both have each other as neighbours their similarity is 50% (3 out of 6).

4.2.4 Responsibility coordination of segments

Offloading is achieved by downloading segments from other peers and naturally some peers need to download it directly from the server before
sharing it with other peers. The problem here is to decide who and how many that should download directly from the server. This is called responsibility coordination, a responsible peer is a peer which should download from the server and serve other peers. This is done for each segment for the reasons that the overlay is dynamic and also to utilize all peers’ upload capacity.

Too much communication decreases the benefit of peer assistance. Therefore, a communication-less protocol is developed for deciding responsibilities. Cyclon, Vivaldi and Close Peer Exploration should provide the closest neighbours to each peer and thus close peers should be able to come to similar decisions. This is an optimistic approach where different decisions will still be practically working with a good probability.

4.2.4.1 The protocol

The protocol works as following. A peer has a set of neighbours which includes itself. For all of these, their ID and upload capacity is known. The bitrate of the media is also known. Let us say that responsibility for segment number $s$ has to be decided. This is done by deciding a responsibility factor $rf$. A responsibility factor of, for example, 16 means that on average, every 16th peers should be responsible.

The decision protocol starts with trying a high responsibility factor, which also is a power of two. A peer will be marked responsible if the modulo operation of the ID and the responsibility factor is equal to the modulo operation of the segment number and the responsibility factor (equation 4.1). This is done for all peers. The upload capacity of all peers marked as responsible will be summarized and compared to the required total upload capacity. The required total upload capacity is equal to the bitrate for the media multiplied with the number of not responsible peers. If the total upload capacity is enough, this will be the final decision. If not, the responsibility factor will be divided by two and the same protocol will be tried again until a working responsibility factor is found. In the worst case, this means that the responsibility factor will be equal to one and thus mean that all peers should download the segment from the server.

\[
\text{responsible}(id,s,rf) = \begin{cases} 
\text{true}, & \text{if } (id \mod rf) = (s \mod rf) \\
\text{false}, & \text{otherwise}.
\end{cases}
\]  

(4.1)

Neighbours will, with maybe a few exceptions, have different sets of neighbour sets and this means that they may come to different decisions. However, peers responsible at one responsibility factor will always be
Table 4.2. Neighbour List: Neighbour IDs, Upload Capacity, Bitrate, and Responsibility Factor.

<table>
<thead>
<tr>
<th>Neighbour IDs</th>
<th>Upload Capacity</th>
<th>Bitrate</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>900</td>
<td>300</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>7</td>
<td>700</td>
<td>300</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>8</td>
<td>400</td>
<td>300</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>16</td>
<td>450</td>
<td>300</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>18</td>
<td>400</td>
<td>300</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>21</td>
<td>350</td>
<td>300</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>28</td>
<td>500</td>
<td>300</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>33</td>
<td>300</td>
<td>300</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sum</th>
<th>4000</th>
<th>2400</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment number</th>
<th>Total Upload Capacity</th>
<th>Required Upload Capacity</th>
<th>Missing Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>2100</td>
<td>1900</td>
<td>150</td>
</tr>
</tbody>
</table>

Figure 4.6: Illustration of one peer’s responsibility calculation, responsibility factor 2 will be chosen as it is the highest possible factor.

Blue boxes marks responsible peers.

responsibility at a lower responsibility factor. Different decisions will thus partially coincide. Figure 4.6 illustrates responsibility coordination from the view of one peer. Let us say that this peer has ID 5 and that it will fetch the segment from any of the responsible peers 8, 16, 18 or 28. Even if all four of them decides for a higher responsibility factor like 4, three would still regard themselves as responsible.

Some neighbours in a set might have a significant higher upload capacity than the others. The concept of super peers was used to utilize these powerful peers, even if they were not otherwise considered responsible. In the proposed solution, a peer was considered to be a super peer if its upload capacity was two times higher than the average of the neighbour set. A peer that has the super peer status is considered responsible over all responsibility factors. This leads to better utilization of upload capacities, resulting in higher possible offloading.
Chapter 5

Implementation

This chapter presents work done in the implementation phase, both for the simulations and the live streaming implementation.

5.1 PeerSim simulations

The algorithms of the proposed system were tested and evaluated by implementing them in a simulation framework called PeerSim. The framework is written in Java that allows for easy simulation implementations and can easily simulate dynamics in the network. A PeerSim simulation is built up by nodes that execute some protocols invoked once every cycle. Several protocols can be assigned to a node which will execute sequentially to represent a complete protocol stack.

Through the thesis project three different simulation cases were developed using different combinations of the components in the protocol stack of the streaming system as shown in figure 5.1.

PeerSim has some control protocols that were used in the simulation implementations, initializers and observers. Initializers are the first protocols to be executed when a new node is spawned. One specific initializer called WireKOut is used that will initialize a node with bootstrapping peers to simulate a bootstrapping server in a real application. This is used in all simulations to ensure initial connectivity. PeerSim uses data collection nodes called Observers to collect data during the simulation. Observers are custom implemented for the specific protocol or task to monitor, which can be scheduled to run at every cycle or on certain times of the simulation. How the observers were implemented are described in the following subsections about the different implemented simulations.

The dynamics of peers in the network was implemented by having the
simulator add new nodes during the simulation. Node failures was also simulated in a controlled way by having the simulation engine crashing nodes in. Both joins and leaves (crashes) is done in an deterministic pattern specified to the simulator.

The underlying peer-to-peer network topology in all simulations is built by the group membership protocol called cyclon. The cyclon protocol maintains a topology close to a random graph that is kept alive even during frequent membership changes (joins and leaves).

![Diagram](https://via.placeholder.com/150)

(a) Vivaldi simulation. (b) Close Peer Exploration simulation.

(c) Responsibility coordination simulation.

Figure 5.1: Figures represents the implemented simulation scenarios.

### 5.1.1 Vivaldi simulation

The goal of the Vivaldi simulation is to evaluate effectiveness in finding the coordinates of the peers and the time for the network to converge to a stable state (stable coordinates). The complete Vivaldi protocol presented in section 4.2.2 was implemented as a protocol in the simulator as shown in figure 5.1a. Vivaldi will execute once every cycle for each peer and choose another peer from the underlying cyclon protocol. With the other peer it will execute the Vivaldi protocol to improve its own coordinate.

A Vivaldi observer was implemented to monitor the network’s properties in every cycle during the simulation. It monitors every peer’s coordinate, their error relative to every other node in the network, and specific Vivaldi details. The average error for the nodes in the network was calculated from the sum of errors $E = \sum_i \sum_j e_{ij}^2$. 
5.2. Implementation of peer assisted live streaming

5.1.2 Close Peer Exploration simulation

Close peer exploration was simulated to evaluate the protocol and to verify that it works as intended in clustering peers which are close to each other in terms of Vivaldi coordinates. Peers in this protocol are the set from which the final neighbours for the responsibility coordination are chosen from. The simulation uses Cyclon and Vivaldi as shown in figure 5.1b, to ensure connectivity and provide coordinates. The protocol is implemented in PeerSim as described in the section 4.2.1. Executing once every cycle in which every peer contacts one other neighbour to improve its current neighbour set.

To collect data from the simulation an observer monitors the final subset of the peers that are selected to be used by the responsibility coordination. These peers are the N closest (in Vivaldi distance) of the neighbour set of the Close Peer protocol. Since the responsibility coordination builds upon similarity of the neighbours the observer calculates the similarity of the N neighbours at each peer and its N neighbours neighbour sets.

5.1.3 Responsibility coordination simulation

A simulation of the responsibility coordination was done to evaluate the accuracy in responsibility calculations and get a metric of offloading. The simulation uses Cyclon, Vivaldi, and Close Peer Exploration as shown in figure 5.1c and the responsibility coordination is implemented as described in the section 4.2.4. This simulation is a model of the algorithms used in the prototype player, in each cycle every peer requests a certain segment. For a segment request every peer will do responsibility calculations out of the peer set provided the by Close Peer Exploration.

The data collection is done by an Observer that monitors how much traffic are offloaded from the streaming server. Out of the segments that are requested directly from the server it monitors the reason of the request. Was the server request a fallback since none of the responsible peers actually were responsible or was the peer responsible for the requested segment.

Upload capacities for peers in the simulation were set as a factor of the media bitrate. The upload capacities used in the simulations were randomly assigned in the span 0.5 to 5 times the media bitrate.

5.2 Implementation of peer assisted live streaming

The proposed system for peer assisted live streaming was implemented to test the complete solution in web browsers using WebRTC. The following sections describes the choices done during the implementation phase.
5.2.1 Prototype player

The development started with an existing Video On Demand (VOD) player called dash.js. It is developed by the DASH industry forum as a reference player for the MPEG-DASH standard. The player by itself is fully capable of streaming VOD and has a limited (still in development) support for Live streaming (hereafter referred to as basic player).

With the basic player the procedure of requesting segments is as shown in figure 5.2a. As seen in the flowchart, when the basic player needs to download the next video segment it issues a request for a new segment. This request is sent as an HTTP-request to the streaming server, the segment is received and delivered to the buffer of the player.

The flowchart in figure 5.2b shows the behaviour of the basic player extended with the peer-to-peer assistance (hereafter referred to as prototype player). The prototype player will try to request a segment from neighbouring peers if enough time is available until playback of the segment. If there is not sufficient time the player will issue an HTTP-request to the streaming server as fallback to ensure delivery of the segment in time. If the time is sufficient the player runs the responsibility calculation on its current neighbours, if the peer itself is responsible it requests the segment from the streaming server. For a peer that is not responsible it will try to request the segment from the responsible peers. To distribute the load on the responsible peers they are randomly chosen with a weight relative to their upload capacity. Peers with larger upload capacity will thus get more requests. If any of the responsible peers can serve the segment request within the time available the segment is delivered to the buffer. In the case that peer requests takes too long time the player will stop the peer assistance and download the segment from the server as fallback.

The presented flow of the player is executed for each segment that the player requests. Since buffering is used segments may have a varying time until playback deadline. The time until playback limits the time available for the peer assistance module, the next section discusses timing and synchronization when requesting segments.

5.2.1.1 Timing of segment requests

When watching a live stream the peers may end up a bit out of sync in retrieval of video segments. Two peers can come out of sync when starting the stream as shown in figure 5.3. Peer 1 starts the stream in the end of the current live edge segment and will thus play from the beginning of that segment. Peer 2 starts shortly after but ends up with the live edge
5.2. Implementation of peer assisted live streaming

(a) Basic player without peer-to-peer.  

(b) Assisted player with peer-to-peer.

Figure 5.2: Descriptions of how segment requests are processed in a regular vs. peer-to-peer assisted player.

at the second segment. The peers will end up with streams that are in theory one segment duration out of sync. In practice parameters that affects peers to come out of sync is the segment duration and how fast the player starts playing the first segment (player delay). These parameters results
in a maximum time difference: $\text{maximumTimeDiff} = \text{segmentDuration} + \text{playerDelay}$

![Figure 5.3](image)

Figure 5.3: Out of sync scenario peers ends up 1 segment duration off.

The download time of a segment is considered to be no longer than the segment duration, if requested from the streaming server. Otherwise it would not be possible to have a continuous playing live stream. A video player adjusts the bitrate of the stream accordingly to the bandwidth of the player to ensure smooth playback. The download time may differ a bit when using peer-to-peer technology due to lower upload capacity of peers than the streaming server. Therefore the download time is allowed to be twice the segment duration ($\text{peerDownloadTime}$) when downloading from peers. When the player starts a stream some segments are fetched directly from the streaming server to ensure that the player has some buffered segments before starting the peer assistance. Ensuring a quicker start of video playback and it increases the time available for the following peer segment requests.

When using peer assistance the player will fetch a segment from the server if the time until playback is less than $\text{peerLimit} = \text{peerDownloadTime} + \text{fallbackTime}$. The $\text{peerLimit}$ stops peers from trying to use the peer-to-peer network when the time until playback is insufficient. When sufficient time is available the player uses the peer assistance module. If the executing peer itself is not responsible it will try to fetch the segment from its responsible peers. The time to request and deliver a segment to the buffer consists of three parts as shown in figure 5.4, Wait time, Peer download time, Fallback Time.

![Figure 5.4](image)

Figure 5.4: Time components of a segment request in peer assisted player

Wait time is used to delay a request before it is issued to a responsible peer to handle out of sync clients. During the wait time the responsible peer has time to fetch the segment from the server before the request comes in. The idea about wait time was derived from the solution of manifest manipulation presented in Smooth Cache [35]. The wait time is variable depending on the time until playback, but is no more than $\text{waitTime} = \text{maximumTimeDiff} +$
segmentDuration. After waiting the player will issue requests to responsible peers, if none of them are completed within the peer download time the player will download the segment from the server. Fallback time is used to allow the player to have sufficient time after a failed peer request to fetch the segment from the server before its playback deadline.

By using time limitations, the player offers peer assistance when possible since continuous playback of the stream is prioritized over utilization of the peer-to-peer network.

5.2.2 Peer connections

The communication between peers in the network was implemented with WebRTC, which offers realtime communication between web browsers. In the thesis project a framework called PeerJS was used to handle connections over WebRTC. PeerJS offers an API to create connections and monitor connections in case of errors or disconnects. It abstracts away the process of setting up a RTCPeerConnection and creation of a RTCDataChannel, providing a simple method to establish connections. To provide a complete method to establish connections PeerJS uses a peer-server that assists in connecting peers. A peer will connect to the peer-server through a web-socket which is persisted as long as the peer is using the streaming service. The peer-server is also responsible of assigning unique ID’s to all connecting nodes, an ID that is used in the responsibility calculations.

As discussed in section 2.2.1 establishing peer connections has some difficulties if the user is behind a NAT or firewall. This problem was solved by using PeerJS which utilizes the ICE protocol to find candidate addresses through a STUN server. After running the ICE protocol a peer has knowledge about the addresses it can be contacted at. The connecting peer (P1) uses the web-socket connection as shown in figure 5.5, where it first issues a request for P2’s candidate set. The peer-server will contact P2 through its web-socket connection to get the candidate set of P2 which is returned to P1. As the last step P1 connects to P2 via the PeerJS API on one of the addresses provided in the candidate set. Each component of the video player that needed peer-to-peer communication utilizes the PeerJS API to setup reliable WebRTC connections.

PeerJS solved the issue of establishing connections, but an earlier problem of the connection process is how a new peer finds its way into the network. The peer discovery was done through a bootstrap-server that keeps track of all connected clients in the peer-to-peer network. When a new client wants to join the network it will query the bootstrap-server for a random subset of the existing peers. These random peers are used as the initial
cyclon neighbours for the newly connected peer. Naturally after running the protocols for a while the application the peer will find its Vivaldi position and closest neighbouring peers.
Chapter 6

Results

This chapter presents results from simulations and tests of the prototype player.

6.1 Vivaldi simulations

The correctness of Vivaldi is proven in the paper in which it was presented. The paper also shows that Vivaldi adapts to network changes [26]. The results in figure 6.1 shows how quickly Vivaldi converges. At cycle 0, 1000 peers are added to the network. The average latency is 1 and therefore is the average error as well 1. Recall that the error corresponds to the difference between estimated latency and real latency. The error is typically down at zero after 150 cycles. At cycle 200, 100 new peers are added to the network. It is shown that these peers find their position in a few cycles in the otherwise stable network.

Vivaldi was also tested with a configuration where nodes were not only assigned a position but also a simulated ISP. There was an extra latency added to communication between different ISPs to better imitate the conditions of the Internet. The goal was to see that nodes within the same ISP grouped together in Vivaldi. The result of this can be seen in figure 6.2. Note that the other results in this chapter use a simpler model without assignments to ISPs and associated latencies.
(a) This figure shows how the average of average error decreases cycle by cycle. The grey area shows a 95% confidence interval.

(b) This figure shows how the median of average error decreases. The grey area shows the 95th percentile for a single execution.

Figure 6.1: Convergence in Vivaldi

(a) Convergence after a few cycles when nodes are still added
(b) All nodes have been added but not yet found their positions
(c) Eventually, all nodes of the same ISP are grouped together

Figure 6.2: Visualization of Vivaldi. The color of the nodes is determined by the simulated ISP which adds more latency between ISPs. Coordinates are two-dimensional with a height vector.
6.2 Close peer simulations

The close peer exploration protocol has been validated by letting a network of peers stabilize over many cycles and then calculating the similarity of close peer sets. Figure 6.3 shows the distribution of similarity among all peers. Each peer can calculate its similarity with its neighbours by checking how many neighbours it have in common with them. An example of this was shown in section 4.2.3. The average similarity is around 60% for all three tested peer set sizes, although slightly higher when using a bigger size. The following results uses a close peer set size of 32.

![Close Peer Similarity](image)

Figure 6.3: This figure shows how close peer similarity changes by the close peer set size (network with 1000 peers).

For testing adaption in changing network conditions, a simulation where new nodes were inserted to an existing stable network was done. This can be seen in figure 6.4. 1000 peers get 300 cycles to stabilize and then 100 new nodes are added. The figure clearly shows that these new nodes quickly gain a high similarity without major impact on the already stable nodes.
Figure 6.4: This figure shows how close peer similarity is affected by joins of new peers. At cycle 300, 100 new peers joined the network consisting of 1000 peers.

6.3 Responsibility coordination simulation

The responsibility coordination was tested by simulating a set of peers which consumed a media stream. In each cycle, a peer calculated a responsibility decision that said which peers it believed to be responsible. If the peer itself was responsible, the segment was said to be fetched from server. If not, requests to the responsible peers was done. Figure 6.5 shows the probability for how many requests that were needed. Most of the times, the first request was successful. When not, the reason was that the peer receiving the request did not regard itself as responsible. In these cases, a request could be sent to another responsible that most likely regarded itself as responsible.

Figure 6.5 also shows a reduction in accuracy when the network is experiencing churn. For high churn 10 peers are removed and 10 new peers are added in each cycle. Thus, 1% of the network is changed in each cycle. As low churn 1 peer is removed and 1 is added per cycle, changing only 0.1% of the network. The lower accuracy is caused both by requests to peers which are dead and the fact that new peers does not make as accurate decisions.

Simulation results of the average offloading achieved with the responsibility coordination algorithm are shown in figure 6.6. The data were collected
6.3. Responsibility coordination simulation

Figure 6.5: Accuracy of responsibility coordination in a network with 1000 peers. Upload capacity is randomly distributed between 0.5-5 in a stable network with 1000 nodes. The percentage of segments requested from the server is the combination of fallback requests and the requests done by responsible peers. The achieved offloading is shown in the bars where segments are fetched from peers. Using the lower upload capacity range, offloading average was 58.9% and the higher range resulted in an average of 75.4%. Upload capacity is specified as a factor of the media bitrate.
Figure 6.6: Segment request statistics in a network with 1000 peers. Upload capacity is randomly distributed between 0.5-5 and 0.5-10

6.4 Peer assisted live streaming player

To get results of the prototype player, a performance test was done on a small scale. The test was conducted with 11 connected peers. The shown global statistics includes what has been reported during the last 10 seconds.

During the test, the observed offloading reached up to 70%. Statistics seen in figure 6.7 shows this. ‘No decision yet’ responses to peer requests means that a peer received a request before it had even made a decision about that segment. This indicates that players are out of sync. When comparing to figure 6.8 it is noticeable that peers were in better sync at this moment.

When playback of the video is out of sync, the offloading was lower as seen in figure 6.8. Here, 25 segments were fetched from peers which regarded themselves as responsible for it. However, 57 segments (‘No peers’) had to be fetched from server since they could not be served through the peer-to-peer network.
6.4. Peer assisted live streaming player

Figure 6.7: Global statistics of test with 11 peers and video in sync.

Figure 6.8: Global statistics of test with 11 peers and video out of sync
Chapter 7

Analysis

In the following chapter, the design, implementation, and results of the simulations and the prototype are analysed.

7.1 Vivaldi

It was shown in the figure 6.1 of the Vivaldi simulation that the algorithm works as intended, it typically converges after 150 cycles to an average error of zero after the 1000 initial nodes are added. This means that until convergence each peer has contacted 150 nodes out of the initial 1000. In the stable network, 100 nodes join and find their positions in only 25 cycles. This means that after 25 messages, a peer can estimate the latency to all other nodes in the network. The distinct convergence of Vivaldi is important since the close peer protocol is highly dependent on the coordinates, new peers joining the stream can in few cycles be able to find their closest peers.

7.2 Close Peer Exploration

When running simulations of the Close Peer similarity, the results (figure 6.3) showed that the average similarity was around 60% or slightly above for all of the close peer set sizes. A peer’s similarity in a perfect network with an infinite number of peers can be mathematically proven to be 58.7%. This is shown in appendix A. There are two reason for why a higher similarity is achieved in the simulations. First, clustering, i.e. peers grouping close together, promotes higher similarity. Second, edge nodes achieves higher similarity since they can only reach inwards to find neighbours.

Choice of the close peer set size has not a large impact on the achieved similarity when comparing sizes of 32 and 64. Moving from size of 16 to the
two larger ones is an improvement since the left tail of the graph is shortened
and thus increases the lowest level of similarity in the network. Increased
similarity with increased peer sets comes at a cost; larger peer sets also
increases the time it takes for peers to achieve high similarity.

In figure 6.4, it can be seen that the initial nodes need 200 cycles to
achieve the average similarity of 60%. Note that the results of the close peer
similarity is dependent on the convergence of Vivaldi that is running at a
lower layer. When joining 100 nodes on the already stable network they
gain high similarity fast. Already after 25 cycles the similarity was increased
from 0% to 50%. A correlation can be seen between this and the convergence
time of Vivaldi that is also 25 cycles (figure 6.1) when joining 100 nodes. A
similarity of 50% in 25 cycles is considered to be good since it is close to the
stable average of 60%. The last 10% similarity is gained in a slower pace,
where it reaches 60% after 75 cycles.

The simulations of the close peer exploration protocol demonstrates its
good properties in clustering peers to gain similarity which is crucial for the
responsibility coordination.

7.3 Responsibility coordination

The accuracy of calculations done by the responsibility was measured in how
many peer requests that are needed to download a segment. Figure 6.5 shows
that the low churn of 0.1% has nearly no effect on the accuracy. Without
churn and with low churn there is a probability of at least 90% to have a hit in
the first request. A network that experiences high churn, here incorporated
as 1% in each cycle, has an accuracy of 95% in the range of 1-3 requests.

Peers that have to do 8-15 requests occurs but it is extremely rarely, the
experiments was run 10 times for 100 cycles with 1000 nodes resulting in 1
million data points. The larger request amounts only happened a few times
in a million segment downloads.

The simulation results from the responsibility coordination also shows
that the average achieved offloading is related to the average upload capacity
in the network. In the simulation case with random upload capacities ranging
from 0.5 to 5 times the video bitrate, the offload was on average 58% as seen
in figure 6.6. With an upload capacity ranging from 0.5 to 10, the offload
was 75%. It is natural that a larger average upload capacity in the network
means that a peer typically can serve more peers.

The average offloading for a certain network can be mathematically
calculated since the presented responsibility algorithm is based on upload
capacities. A peer’s upload capacity is equivalent to how many peers it can
serve. This can be extended to calculate that offload for the full network by using the average upload capacity. With average upload capacity denoted as $C_U$, the relative server load can be written $1/(1 + C_U)$. The denominator $1 + C_U$ comes from the fact that a peer serves other peers with its upload capacity but also itself by fetching the segment from server. The formula $\text{offloading} = 1 - 1/(1 + C_U)$ can then be used to calculate the offloading based on the average upload capacity. This can be seen if figure 7.1.

![Graph showing theoretical maximal offloading](image)

**Figure 7.1: Theoretical maximal offloading**

A network with upload capacities randomly distributed in the range from 0.5 to 5 has an average upload capacity of 2.75. With $C_U = 2.75$ the average offloading is calculated to be 73% according to the formula and the simulated
value was 58.9%. For the simulation with upload capacity in the span from 0.5 to 10, the average upload capacity was 5.25 and thus have a calculated average offload of 84% where the simulated value was 75.4%.

The gap between simulated and calculated offloading values is due to the use of the communication-less responsibility algorithm and coarse grained responsibility factor \((rf)\) values - 1, 2, 4, 8, 16 and so on. If the \(rf\) values instead would be steps of 1, a peer could maximize its usage of the available upload capacity in its neighbour set. For example, assume the calculation would choose an \(rf\) value of 7 where the coarse calculation in this scenario would choose \(rf\) 4. Since every 4th peer then would be responsible the actual offloading becomes less than the maximum possible with every 7th being responsible. As mentioned in section 4.2.4.1, the motivation of coarse \(rf\) values as a power of two is that peers responsible in an higher \(rf\) are also responsible in an lower. This makes the decisions of two peers with different \(rf\) values to intersect at some peers, that makes the decisions compatible with each other and a peer request can with high probability be correctly resolved. The big advantage of the communication-less algorithm is that decisions can be done without delay and avoid communication overhead.

Fallback did not occur with the lower average upload capacity. In the case of higher capacity, it did occur with a probability of 0.1%. The reason for fallback is probably that with higher upload capacities fewer peers become responsible. If there then is discrepancy in decisions, there will be fewer peers to ask for a segment and thus a fallback may be needed. This shows that with higher upload capacities, the close peer set should also be larger.

### 7.4 Peer assisted live streaming player

The browser implementation of the live video player (figure 7.2) worked well with up to 70% offload, as long as the video was more or less in sync between peers. When the video was out of sync between peers, offloading decreased considerably. The synchronization of video between peers is the main obstacle, partly solved with the wait time introduced for segment requests. To improve this, it is needed rewrite several parts of the player in favor for the peer-to-peer extension.

Peers in a stream that is out of sync got a lot of negative responses to segment requests as seen in figure 6.8. The reasons of peers responding negatively to a request were: requests issued before the responsibility calculation is done, peer was not responsible, and that the segment is in progress to be downloaded.

The buffer length used for the prototype was 30 seconds. This was choosen
so that there were well enough of time to allow for peer requests. In fact, a lower buffer may have been used. Although, as mentioned previously, the synchronization problem hindered fine-tuning of this parameter. In fact, there was also a related problem where the buffer was running dry.

The length of segments was chosen to be 1 second. They should neither be too long or too short. Since a peer need to wait at least the length of a segment before sending a request to another peer, they should not be very long. Also, if they are too long, broken responses will have a bigger impact. If the segments instead are too short, they will cause unnecessary overhead because of the increased amount of requests.

![Image of DASH Reference Client](https://example.com/dash.png)

**Figure 7.2:** The peer assisted live streaming player built upon dash.js

The usage of PeerJS simplified the implementation of WebRTC peer connections, where connections were easily set up including NAT-traversal. At the time of this thesis project there are limited support for WebRTC and MSE. Only Google Chrome supports both WebRTC and MSE\(^1\). Only Google Chrome supports both WebRTC and MSE and can thus

---

\(^1\)MSE: Media Source Extensions - allows browser-native adaptive streaming
utilize the peer assistance in the browser implementation. Internet Explorer
and the coming version of Safari supports MSE but not WebRTC which
makes it capable of playing the live stream but without peer assistance.
Chapter 8

Conclusion

This thesis project presented a solution to peer assisted live video streaming. The goal has been to offer a solution to reduce distribution costs without lowering the user experience. Thus a prototype which allows for browser-native playback has been developed and evaluated. This means that a user does not need to install any browser extension to consume the content and participate in assisting other peers. The fact that no browser extension is needed is of importance for content providers that want to keep the bar to their service as low as possible.

It has been shown that the solution can cut a significant part of the distribution costs for live content. Although, the impact of the solution is highly dependent of the users’ upload capacities. A content provider already using HTTP-based streaming could without too much effort replace their current player to this peer assisted player. The only additional architecture needed are the services used for helping peers to connect.

An ethical consideration to reflect over is how the users’ resources are utilized. A content provider may need to ask the user for permission before enabling peer-to-peer communication. For example, consideration should be taken for customers on mobile connections where data traffic may be limited.

The solution incorporates well proven protocols such as Cyclon and Vivaldi with our novel approach to coordinate peers. These protocols makes it possible to find the peers that are closest in the network and thus utilize the network efficiently and avoid inter-ISP traffic.

The responsibility coordination of segments takes an optimistic approach which does not require communication at that level. The key is that neighbours will come to similar decisions because they have similar neighbours. It is shown that this approach works well, even under non optimal conditions. That is at least true for the simulations. Showing the same behaviour for the prototype implementation was not as easy.
The prototype had a problem related to synchronization of players. When this was under control the prototype demonstrated good behaviour related to offloading. However, when out of sync the overall offloading performance decreased drastically. The reason is that our solution assumes that playback at different peers are more or less synchronized. This is obviously not a strange pre-condition since we are considering live video. To solve the problem of synchronization more parts of the reference player has to be rewritten. This is a proposal for future work since we believe this is the main obstacle to achieve better performance.

Another consideration for future work is the Close Peer Exploration protocol. This protocol is a simple approach to find close neighbours. Although, the protocol could be tuned to lessen the communication when a stable state has been reached.

Peer to peer connections through WebRTC are per se not a security issue. However, the integrity of the content needs to be controlled against malicious peers. This is currently not implemented and this is a suggestion to look further into.

We hope that this work is going to be beneficial for parties who wish to broadcast live content. As a consequence of that, the player will be licensed as open source to allow for further development and contributions.
Bibliography


[27] Swarm CDN, “Swarm reduce bandwidth costs today. add swarm :: Add swarm to your current host or CDN and save money.” [Online]. Available: http://swarmcdn.com/


Appendix A

Close peer similarity proof

This section will present a mathematical proof for why the average similarity, in an ideal two-dimensional network, between a peer’s close peer set (closest neighbours) and its neighbours’ close peer sets are 58.7%.

Figure A.1: Intersections of circles for a peer with only four neighbours

An ideal network means that peers are distributed evenly over a two-dimensional space so that the closest neighbours to a peer are within a circle of radius $R$ and that all peers within this circle are seen as neighbours. A neighbour would also have a set a neighbours which are within a circle of radius $R$. It is the intersection, relative to the area of a circle with radius $R$, of these circles that defines the similarity. This is exemplified in figure A.2.

It is clear that the similarity depends only on how far away a neighbour is. Let us denote this distance as $r$. When $r$ approaches zero, similarity approaches 100%. The distance $r$ can be at maximum $R$, otherwise the two peers would not be neighbours. Then, to find the average similarity between a peer and its neighbours, the similarity of a peer at any point (in the circle)
Figure A.2: Intersection of two circles

can be integrated over the full circle and divided by the area of the circle. See equation A.1.

\[ \text{sim}_{\text{avg}} = \left( \int_0^R \int_0^{2\pi} \text{sim}(r) \, d\phi \, dr \right) / (\pi \times R^2) \quad (A.1) \]

Since the similarity is a relative measurement, the actual radius of the circle does not matter. Instead, we let \( R = 1 \). Equation A.2 shows the intersection area of two circles [40, 41].

\[ \text{intersection} = R^2 \times (q - \sin(q)) \quad \text{where} \quad q = 2 \times \cos^{-1}(r/2R) \quad (A.2) \]

By dividing this intersection area with the area of the circle we get the similarity as seen in equation A.3.

\[ \text{sim}(r) = R^2 \times (q - \sin(q)) / (\pi \times R^2) \quad \text{where} \quad q = 2 \times \cos^{-1}(r/2R) \quad (A.3) \]

This can be further simplified (\( R = 1 \)) as seen in equation A.4.

\[ \text{sim}(r) = (q - \sin(q)) / \pi \quad \text{where} \quad q = 2 \times \cos^{-1}(r/2) \quad (A.4) \]

Equation A.1 and A.4 combined gives us equation A.5.

\[ \text{sim}_{\text{avg}} = \left( \int_0^1 \int_0^{2\pi} (q - \sin(q)) \, d\phi \, dr \right) / \pi^2 \quad \text{where} \quad q = 2 \times \cos^{-1}(r/2) \quad (A.5) \]

The equation A.5 can be evaluated to 58.7% which is what we aimed to prove.