TÉCNICO LISBOA

# Real-Time Collaboration on the World Wide Web in a Peer-to-Peer Mode

**Emanuel Sérgio Ruivo Almirante**

Thesis to obtain the Master of Science Degree in

**Telecommunications and Informatics Engineering**

Supervisors: Prof. Rui António dos Santos Cruz

**January 2017**

# Acknowledgments

Firstly, I want to start by giving my greatest appreciation to my supervisor, Prof. Rui Santos Cruz, for all the patience he had and all the help he gave me.

I also want to thank my friends André Pires, André Bernardino, Bruno Barros, Carlos Ribeiro and Gonçalo Borges for all the support and friendship since the beginning of my journey in Instituto Superior Técnico. To my friends Andreia Tibério, Diana Pereira, Francisco Chaves, and Sandra Lopes I want to extend a special thanks for helping and motivating me for all these years. I also thank to the other people that helped me along the way and that left a mark in my life. For that I am deeply grateful for it helped me become what I am today.

Lastly, I wish to thank profoundly to my parents, Manuela Ruivo and Sérgio Almirante, for the support during the good and the bad moments of my life. Without them none of this would have been possible.

Thank you all.

# Abstract

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo, sed imperdiet lectus est ornare odio. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin consectetuer velit in dui. Phasellus wisi purus, interdum vitae, rutrum accumsan, viverra in, velit. Sed enim risus, congue non, tristique in, commodo eu, metus. Aenean tortor mi, imperdiet id, gravida eu, posuere eu, felis. Mauris sollicitudin, turpis in hendrerit sodales, lectus ipsum pellentesque ligula, sit amet scelerisque urna nibh ut arcu. Aliquam in lacus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nulla placerat aliquam wisi. Mauris viverra odio. Quisque fermentum pulvinar odio. Proin posuere est vitae ligula. Etiam euismod. Cras a eros.

# Keywords

Peer-to-Peer (P2P), World Wide Web (WWW), Web Real-Time Communication (WebRTC), Peer-to-Peer Streaming Protocol (PPSP), Real-Time Communications (RTC), Real-Time Collaboration

# Resumo

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Aliquam aliquet, est a ullamcorper condimentum, tellus nulla fringilla elit, a iaculis nulla turpis sed wisi. Fusce volutpat. Etiam sodales ante id nunc. Proin ornare dignissim lacus. Nunc porttitor nunc a sem. Sed sollicitudin velit eu magna. Aliquam erat volutpat. Vivamus ornare est non wisi. Proin vel quam. Vivamus egestas. Nunc tempor diam vehicula mauris. Nullam sapien eros, facilisis vel, eleifend non, auctor dapibus, pede.

# Palavras Chave

Peer-to-Peer, Rede Mundial de Computadores, Comunicações em Tempo Real na Web, Protocol de Streaming Peer-to-Peer, Comunicações em Tempo Real, Colaboração em Tempo Real

# Contents


**1 Introduction** **1**

1.1 Background . . . 3

1.2 Proposed Solution . . . 4

1.3 Thesis Contribution . . . 5

1.4 Outline . . . 5

**2 Fundamentals** **7**

2.1 P2P Networks and Real-Time Collaboration . . . 9

2.2 WebRTC and Real-Time Communication in Web-browsers (RTCWeb) . . . 10

2.2.1 WebRTC . . . 11

2.2.2 RTCWeb . . . 13

2.3 Signaling in WebRTC . . . 15

2.3.1 WebSocket . . . 16

2.3.2 Session Initiation Protocol (SIP) . . . 17

2.3.3 Extensible Messaging and Presence Protocol (XMPP) . . . 18

2.3.4 Signaling-On-the-Fly . . . 18

2.4 Peer-to-Peer Streaming Protocol . . . 19

2.4.1 The Peer Protocol . . . 19

2.4.2 The Tracker Protocol . . . 20

2.5 Dynamic Adaptive Streaming over HTTP . . . 21

**3 Related Work** **25**

3.1 P2P Media Streaming with HyperText Markup Language 5 (HTML5) and WebRTC . . . 27

3.2 Integration of WebRTC and SIP . . . 29

3.3 Webtorrent . . . 30

3.4 WebRTC Technology Overview and Signaling Solution Design and Implementation . . . 31

3.5 Low Delay Moving Picture Experts Group-Dynamic Adaptive Streaming over HTTP (MPEG-DASH) Streaming over the WebRTC Data Channel . . . 34

**4 Architecture** **39**
4.1 Requirements . . . 41
4.2 Architecture Design . . . 42
4.3 Components . . . 46
4.3.1 Peer Components . . . 46
4.3.2 Peer Coordinator, Web Server and Media Server . . . 48
4.3.3 Signaling Server . . . 51
**5 Implementation** **53**
5.1 Implementation Options . . . 55
5.2 Architecture . . . 56
5.2.1 Peer Components . . . 56
5.2.2 Peer Director, Web Server and Media Server . . . 57
5.2.3 Tracker and Signaling Server . . . 57
**6 Evaluation** **59**
6.1 first . . . 61
6.2 second . . . 61
**7 Conclusion** **63**
7.1 Conclusions . . . 65
7.2 System Limitations and Future Work . . . 66
**A Code of Project** **71**
**B A Large Table** **73**

# List of Figures


2.1 Types of networks . . . . . 10
2.2 WebRTC and RTCWeb architecture . . . . . 11
2.3 WebRTC API with signaling . . . . . 12
2.4 Web browser RTC models of deployment . . . . . 14
2.5 MPEG-DASH System Overview . . . . . 23

3.1 Overall architecture of BitTorrent . . . . . 27
3.2 Overall architecture of WebRTC with SIP . . . . . 29
3.3 Hybrid Webtorrent system . . . . . 31
3.4 Overall architecture of WebRTC . . . . . 32
3.5 Message Sequence Diagram to establish a session . . . . . 33
3.6 Experiment setup . . . . . 35
3.7 MPEG-DASH topology . . . . . 35
3.8 WebRTC topology . . . . . 36
3.9 WebRTC datachannel establishment flows . . . . . 37

4.1 System's architecture for the proposed solution . . . . . 43
4.2 System's participants interaction . . . . . 45
4.3 Peer collaborates with content interaction . . . . . 46
4.4 Peer modules . . . . . 47
4.5 Peer Coordinator, Web Server and Media Server modules . . . . . 49
4.6 Signaling module . . . . . 51

# List of Tables

4.1 Table of association of Internet Protocols (IPs) and names . . . . . 49
4.2 Table of association streams and their sources . . . . . 50

# List of Algorithms

# Listings

# Acronyms

| | |
|---|---|
| **AAC** | Advanced Audio Coding |
| **API** | Application Programming Interface |
| **APIs** | Application Programming Interfaces |
| **BOSH** | Bidirectional-streams Over Synchronous HTTP |
| **CDN** | Content Delivery Network |
| **CDN** | Content Distribution Network |
| **CERN** | Conseil Européen pour la Recherche Nucléaire |
| **CPU** | Central Processing Unit |
| **CSMA** | Carrier Sense Multiple Access |
| **DASH** | Dynamic Adaptive Streaming over HTTP |
| **DCE** | Direct Code Execution |
| **DTLS** | Datagram Transport Layer Security |
| **DHT** | Distributed Hash Tables |
| **HTML** | HyperText Markup Language |
| **HTML5** | HyperText Markup Language 5 |
| **HLS** | HTTP Live Streaming |
| **HTTP** | Hypertext Transfer Protocol |
| **ICE** | Interactive Connectivity Establishment |
| **IETF** | Internet Engineering Task Force |

| | |
|---|---|
| **IP** | Internet Protocol |
| **IPv4** | Internet Protocol version 4 |
| **IPv6** | Internet Protocol version 6 |
| **IQ** | Info/Query |
| **LEDBAT** | Low Extra Delay Background Transport |
| **MEGACO** | Media Gateway Control Protocol |
| **MMUSIC** | Multiparty Multimedia Session Control |
| **MPD** | Media Presentation Description |
| **MPEG-DASH** | Moving Picture Experts Group-Dynamic Adaptive Streaming over HTTP |
| **uTP** | Micro Transport Protocol |
| **MPEG** | Moving Picture Expert Group |
| **NAT** | Network Address Translation |
| **P2P** | Peer-to-Peer |
| **PPSP** | Peer-to-Peer Streaming Protocol |
| **PPSPP** | Peer-to-Peer Streaming Peer Protocol |
| **PPSTP** | Peer-to-Peer Streaming Tracker Protocol |
| **PTSN** | Public Switched Telephone Network |
| **QoE** | Quality Of Experience |
| **QoS** | Quality Of Service |
| **RTC** | Real-Time Communications |
| **RTCWeb** | Real-Time Communication in Web-browsers |
| **RTP** | Real time Transport Protocol |
| **RTSP** | Real Time Streaming Protocol |
| **SCTP** | Stream Control Transmission Protocol |
| **SDP** | Session Description Protocol |

| | |
|---|---|
| **SMS** | Short Message Service |
| **SIP** | Session Initiation Protocol |
| **SRTP** | Secure Real time Transport Protocol |
| **STUN** | Session Traversal Utilities for Network Address Translation |
| **TCP** | Transport Control Protocol |
| **TLS** | Transport Layer Security |
| **TURN** | Traversal Using Relays around Network Address Translation |
| **UDP** | User Datagram Protocol |
| **URL** | Uniform Resource Locator |
| **VoD** | Video On Demand |
| **VoIP** | Voice over Internet Protocol |
| **VPN** | Virtual Private Network |
| **WebRTC** | Web Real-Time Communication |
| **WG** | Working Group |
| **WWW** | World Wide Web |
| **W3C** | World Wide Web Consortium |
| **XML** | Extensible Markup Language |
| **XMPP** | Extensible Messaging and Presence Protocol |

# 1

# Introduction

## Contents


1.1 Background . . . . . 3
1.2 Proposed Solution . . . . . 4
1.3 Thesis Contribution . . . . . 5
1.4 Outline . . . . . 5

## 1.1 Background

The Internet was created with the intention of having a network of computers that could communicate, share and collaborate with each other, no matter where they were physically. Its focus, in the beginning, was to perform simple tasks such as, creating and sharing documents or sending email messages [1]. The success of this simple infrastructure paved the way for one of the biggest changes in human history.

This change would reach its most important milestone with the proposal of the WWW, originated at the *Conseil Européen pour la Recherche Nucléaire (CERN)*, by Tim Berners-Lee [2]. The Internet was revolutionized, and evolved from a simple infrastructure, where only the creation and sharing of simple documents and sending email messages was possible, to something that, additionally, would allow access to websites, send text messages, transfer binary files (programs/applications, images, audio, video), and much more.

The evolution of the Internet and of the WWW has not stopped since, and nowadays it is possible to pursuit a multitude of activities, such as online shopping, audio and video calls, play video-games online, live-streaming and on-demand multimedia, even the virtualization of services (e.g. software, application platforms or infrastructures) is possible as well. Almost every service we can think of is now accessible online, which shows how important the WWW has become to society.

As aforementioned, one of the most important aspects of the Internet is the possibility to collaborate with anyone in the network. With the introduction of the WWW and the expansion of the Web, millions of people from around the World can engage in real-time collaboration with each other (between two or more users) using technology present in applications like Google Docs, Overleaf or Slack. Real-time collaboration changed how professionals from various fields of work would work with each other, allowing for a more efficient collaboration by getting an answer immediately. However, these applications use the classic client-server system which means that there is a single point of failure and the server becomes a bottleneck, causing problems of scalability.

A possible solution to mitigate this problem is by using P2P. Theoretically, a P2P system works better as more users are online. This would mean that, the Quality Of Service (QoS) would stabilize no matter how many people used the applications, because a user would act simultaneously as a client and as a server, reducing or even eliminating the workload from the main server. Although P2P systems are used by millions of users, they have some disadvantages, like, for example, needing extra software or plugins to be installed, which may cause compatibility problems between different devices.

But recently a new technology named WebRTC has emerged. This technology allows web browsers to connect directly in a P2P fashion, providing real-time device-to-device communication without the need to install any software, besides the web browser, or plugins. WebRTC has already been adopted by the majority of web browsers, like Google Chrome, Mozilla Firefox, and Opera, immediately delivering WebRTC to a huge user base, and making it attractive to developers.

Considering that video streaming is one of the most bandwidth consuming activities, and one were it is necessary to spend more money in servers, bandwidth, Content Distribution Network (CDN), among others, to provide a good enough service for all the users, it would benefit greatly from the implementation of these technologies. Due to this fact, live or on demand video streaming would greatly benefit from this technologies, and would be a good way to test if a system based on said technologies would still provide a good enough service, while reducing costs.

Using both the P2P to provide real-time collaboration, and the WebRTC to simplify the use of P2P for everyone and enable it to be used in any device, at any time, this system would present itself as a self-scalable solution by decreasing the high costs with bandwidth, because each user would also act as a server. Additionally, the combined use of these technologies provides the necessary tools to develop applications that are easy to integrate with the ones already being used, making it simple to transition to a P2P system.

## 1.2 Proposed Solution

The primary objective of this project is to take advantage of the advantages of WebRTC, making real-time P2P collaboration available using only a simple web browser. This will turn the users into content providers too, causing them to also spread the content they are accessing.

Just referring to a system that provides a real-time P2P collaboration is very vague and can lead to some confusion. In this solution it will be considered a real-time P2P collaboration using video streams from peers. This is, every peer can transmit video, as long as it is authorized.

In the solution proposed there will exist a Peer Coordinator, that manages the registrations of peers, their authorizations to collaborate with video and manages the network of peers, like a tracker does; a normal Web Server, which the peers will access, and provides the web page and the web application; and a Media Server, that will act like a super-peer, i.e., it will be the first to have video available for the peers and will always be online, so that there is always at least one source in the P2P network. These three functionalities of the system will all be done in a single node, composed of three modules.

There will also exist a Signaling Server that will be responsible for helping the peers to connect with each other using WebRTC. Although this Signaling Server is described and represented in the architecture, for the purposes of simplification it will be used a public signaling server. This Signaling Server will be another node, although it may be included in the same node of the Peer Coordinator, the Web Server and the Media Server.

Finally, the user will interact with the system through a web application, which will provide an interface through which the users watch the video and also contribute with their own video stream.

## 1.3 Thesis Contribution

This project proposes an architecture and a solution of a system that allows users to collaborate with video streams in a P2P fashion, meaning that they can also act like a server while browsing the web pages that use this system.

Being a collaborative oriented system, it is necessary to have a simple way for users to collaborate with each other. That is provided by a web application, which is only compatible with web browsers that support WebRTC.

Summarizing, the contributions the thesis are the following:

- Present an architecture and a solution for the system described above;
- Develop a functional prototype;
- Evaluate the implemented prototype.

## 1.4 Outline

This document is structured as follows:

- **Chapter 2** describes the main technologies that are used in the system;
- **Chapter 3** describes some of the work already done in this field;
- **Chapter 4** describes the requirements and the architecture of the system;
- **Chapter 5** describes the technologies and how the solution was implemented;
- **Chapter 6** describes the tests performed to evaluate the solution and the results obtained;
- **Chapter 7** draws conclusions about the work developed in this thesis and the system limitations and future work.

# 2

# Fundamentals

## Contents


2.1 P2P Networks and Real-Time Collaboration . . . . . 9
2.2 WebRTC and RTCWeb . . . . . 10
2.3 Signaling in WebRTC . . . . . 15
2.4 Peer-to-Peer Streaming Protocol . . . . . 19
2.5 Dynamic Adaptive Streaming over HTTP . . . . . 21

This chapter provides some insight about the technologies that will be used in the project. It starts by giving an overview of the basics behind a P2P network and real-time collaboration works using these networks. It is also explained how the WebRTC and the RTCWeb technologies can be used to provide P2P capabilities to a browser, as well as some of the signaling technologies compatible with them. Finally, the concept behind MPEG-DASH, a new technology for the streaming of multimedia content, is explained.

## 2.1 P2P Networks and Real-Time Collaboration

P2P is an alternative model to the traditional client-server paradigm. It consists in a group of nodes that cooperate through the Internet in a way to reach a common goal, e.g., exchange files. But P2P can also be used to establish live multimedia communication, e.g., videoconferencing, or share resources based on P2P concepts [3].

The P2P network functions on top of the physical network of the Internet, more specifically it is overlayed on the IP networks. The P2P overlay network makes a direct connection between two users that know each other. This is, no matter how many physical nodes exist between them, it will be like they are directly connected with no routers or servers or other nodes between them. It is an abstraction of the physical network, to simplify [4].

Networks based on P2P distinguish themselves by making each peer a client and a server simultaneously. In this model the peers send requests and answer to incoming requests from other peers, instead of only sending requests and waiting for an answer [5].

The implementation of a P2P network can be done with one or more central entities or without them. The first is commonly known by hybrid P2P and the second by pure P2P [6].

A hybrid P2P network, has a central server, better known as a “tracker”. This tracker has a list of the peers on the network and what content each of them is sharing. When a peer wants to download something, it sends a request to the tracker, and the tracker helps the peer to search for other peers having the desired content.

In contrast, a pure P2P network does not have any central server (tracker), it only has peers. Hence, each peer maintains a list of their neighboring pairs, knows the content that each one has, and it connects to them without help of any tracker.

Concerning real-time collaboration in a P2P system, the peers connect directly to each other, obviously, and each peer transmits to the other peers the content they want to send, like text messages, audio, video, binary files, and others. This will only depend on the web application and what type of collaboration it will allow to happen between the peers. Some applications will be of videoconferencing only, others will permit a whiteboard collaboration, others will group all of these in one.

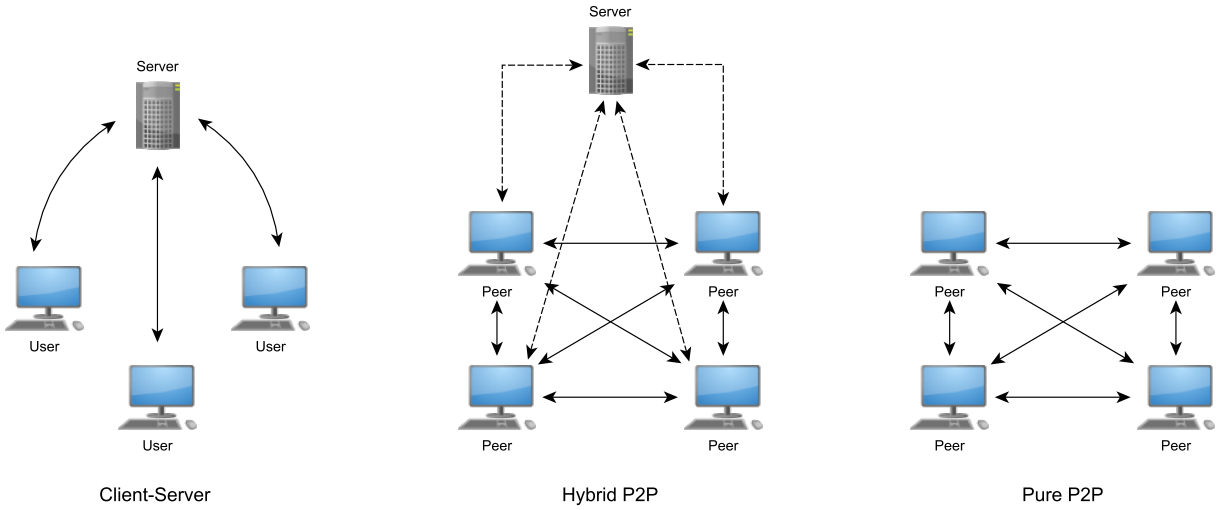


**Figure 2.1:** Types of networks, adapted from [6]

In the event of an application allowing the transmission of live video streaming there will be an original content provider or source peer, that will stream the live video to other peers. But the original peer does not transmit the content to all peers. Instead, the original peer only transmits to a limited number of peers, for example, five, and then those peers also transmit to a limited number of peers, and so on. In this situation, there might be some delay, but nothing that will relevantly affect the user's experience [7].

Nowadays, the most famous P2P network is the BitTorrent, dedicated to the distribution of files over the Internet. It accounts for 35% of all Internet traffic and more than 50% of all P2P traffic. BitTorrent is the proof that P2P networks can be an alternative to the traditional server-client networks [8].

RSC: reviewed

## 2.2 WebRTC and RTCWeb

WebRTC [9] and RTCWeb [10] are two different technologies that complement each other, in order to achieve a common goal: to support direct browser-to-browser communication.

WebRTC is an Application Programming Interface (API) that provides programmers with the tools to create applications that communicate directly browser-to-browser, creating a P2P connection which can be used for a multitude of things, like videoconferencing, stream video (live or not), exchange files, among others [11].

RTCWeb is a protocol that encompasses various existing protocols, to make possible that the WebRTC applications communicate between themselves with audio, video, and data in a P2P mode.

Together, WebRTC and RTCWeb provide an environment where JavaScript embedded in any web page that is viewed in a compatible web browser and authorized by its user is able to set up communi-

cation, in a P2P fashion, using audio, video and data [10].

In Figure 2.2 is illustrated the overall architecture of WebRTC and RTCWeb. While WebRTC is the API that provides a connection to the web browser RTC function, RTCWeb defines the protocols used.

RSC: reviewed

### 2.2.1 WebRTC

Basically, WebRTC is an API that puts real-time communications capabilities into web browsers. These capabilities will then be available to developers through the combination of HTML5 tags and JavaScript Application Programming Interfaces (APIs).

Because the WebRTC API includes HTML5 and JavaScript it is possible to have web applications with functionalities similar to Skype but without the need to install anything else besides a simple web browser [11].

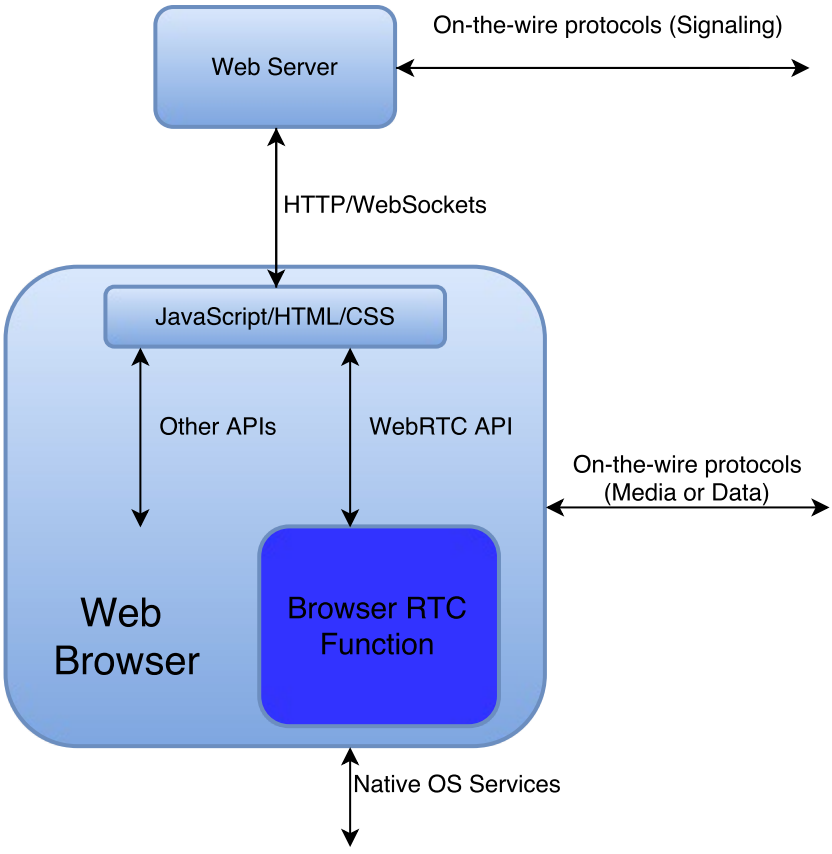


**Figure 2.2:** WebRTC and RTCWeb architecture, adapted from [11]

The components to implement real-time communications capabilities of the WebRTC API are accessed with JavaScript APIs. The most important API, and the ones necessary to establish a live video

streaming, are:

- MediaStream - allows the web browser to access the camera and microphone;
- PeerConnection - establishes peer-to-peer communications, allowing two peers to communicate directly;
- DataChannel - sets up a bidirectional channel through which data can be exchanged between two peers.

These three API will allow two or more users to communicate browser-to-browser, in a real-time and P2P fashion [12].

To start a WebRTC session it is necessary to complete four main steps:

- Web browser client obtains local media;
- Sets up connection between the web browser and the other peer through signaling;
- Attach the media and data channels to the connection;
- Exchange the session description from each other.

After the above steps performed, which can be observed in Figure 2.3, the media stream will start being exchanged through the real-time P2P media channel. All of these steps are implemented by the WebRTC API.

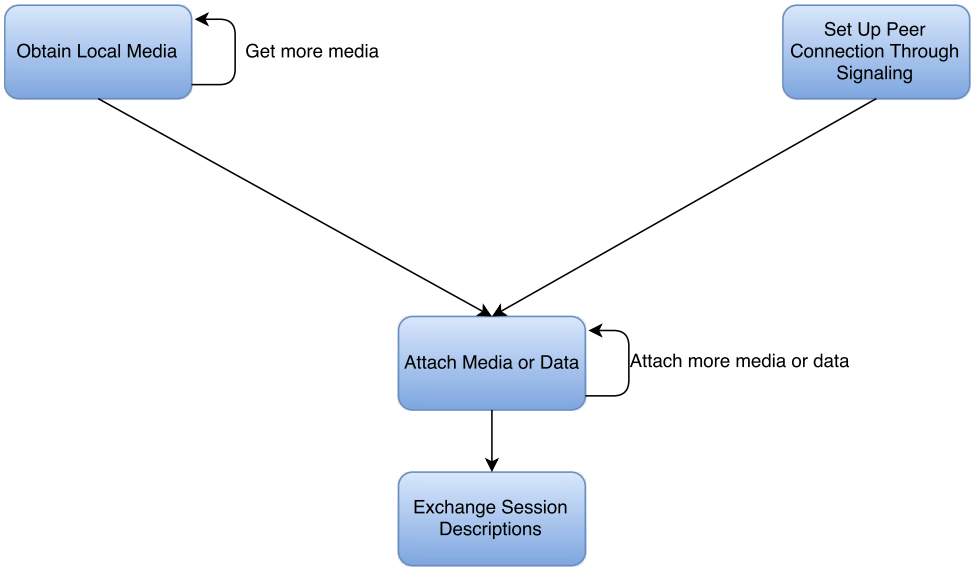


**Figure 2.3:** WebRTC API with signaling, adapted from [13]

The signaling part in Figure 2.3 is also an important component of the system. WebRTC needs a signaling mechanism to coordinate communications and to send control messages. Because signaling

methods and protocols are not specified by the WebRTC API, developers can choose the messaging protocol they prefer, as long as it is a two-way communication channel.

Signaling serves to exchange three types of in information [13]:

- Session control messages - initialize or close communication and report errors;
- Network configuration - let the outside world know the computer's IP address and port;
- Media capabilities - decide the codecs and resolutions that can be handled by the web browser and the web browser it is trying to communicate with.

The signaling, i.e., the exchange of information, must be successful in order for the P2P connection to begin.

The WebRTC API is currently being standardized by the World Wide Web Consortium (W3C) and it is already supported by the main web browsers (Chrome, Firefox, Opera) and platforms (Android, iOS) [9].
RSC: reviewed

### 2.2.2 RTCWeb

RTCWeb is a protocol specification, standardized by the Internet Engineering Task Force (IETF), that defines a method of exchanging data over the Internet [10].

This protocol specifies a set of protocols to allow an implementation to communicate with another implementation using audio, video and other data. The participants will communicate in a P2P fashion, meaning they will be directly connected, without intermediaries.

The models of deployment for which the protocol is best suited are the *triangle* and the *trapezoid* [11].

As it can be observed in Figure 2.4, the media path goes directly between the web browsers, so it is critical to be in accordance with the RTCWeb protocol. The signaling path is not as critical because it goes through servers that can modify or translate the message as necessary. It is noteworthy that the signaling path in the trapezoid model is not of exclusive use of the servers, and the web browsers can communicate through it, in the beginning of a session, to establish the Media Path, for example.

In the triangle model, the signaling runs over Hypertext Transfer Protocol (HTTP) or WebSocket. In the trapezoid model the servers need to agree on the signaling mechanism. SIP or XMPP are two examples of protocols that can be used [11].

Between the web browser and the server a standards-base or proprietary protocol can be used. For example, if both servers use SIP as signaling mechanism then SIP over WebSocket can be used to communicate between the application running in the web browser and the web server [10].

As defined in [14], it is necessary that both User Datagram Protocol (UDP) and Transport Control Protocol (TCP) with the ability to use Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) are available to the implementations.

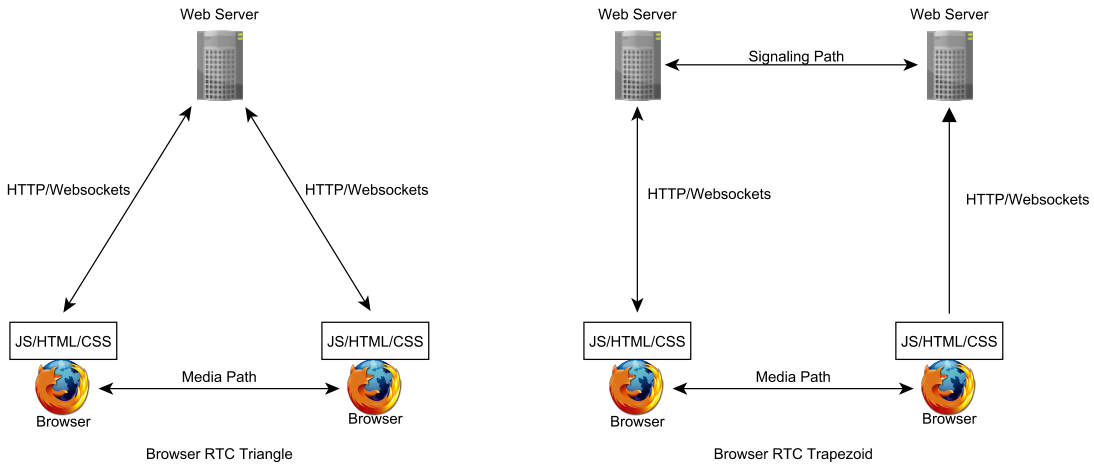


**Figure 2.4:** Web browser RTC models of deployment, adapted from [11]

Also necessary is the support of a full Interactive Connectivity Establishment (ICE) implementation in order for both participants to be able to communicate when they are behind a Network Address Translation (NAT). For this, Session Traversal Utilities for Network Address Translation (STUN) or Traversal Using Relays around Network Address Translation (TURN) servers will be used, with the preference being TURN servers. If the participants are behind a firewall that blocks all UDP traffic, the mode of TURN that uses TCP between the client and the server must be supported, and the mode of TURN that uses Transport Layer Security (TLS) over TCP between the client and the server must also be supported.

The transport of media is typically done by using secure Real time Transport Protocol (RTP). Key exchange must be done using Datagram Transport Layer Security (DTLS)-Secure Real time Transport Protocol (SRTP). For data transport over the data channel, Stream Control Transmission Protocol (SCTP) over DTLS over ICE must be supported. The negotiation of this transport is done in Session Description Protocol (SDP) multiplexing of DTLS and RTP over the same port pair must also be supported [14].

One of the most fundamental aspects is the connection management. In this aspect of the RTCWeb protocol it is important to have interoperability and freedom to innovate. It is also essential that the methods, mechanisms and requirements needed to set up, negotiate and tear down connections to follow these principles:

- The media negotiations will be capable of representing the same SDP offer/answer semantics that are used in [15], in such a way that it is possible to build a signaling gateway between SIP and the media negotiation;
- It will be possible to gateway between legacy SIP devices that support ICE and appropriate RTP/SDP mechanisms, codecs and security mechanisms without using a media gateway. A signaling gateway to convert between the signaling on the web side to the SIP signaling may be needed;

- When a new codec is specified, and the SDP for the new codec is specified in the Multiparty Multimedia Session Control (MMUSIC) Working Group (WG), no other standardization should be required for it to be possible to use that in the web browsers. Adding new codecs which might have new SDP parameters should not change the APIs between the web browser and JavaScript application. As soon as the web browsers support the new codecs, old applications written before the codecs were specified should automatically be able to use the new codecs where appropriate with no changes to the JavaScript applications.

The aspects of the RTCWeb protocol described are the most important. There are other secondary aspects, like the audio and video codecs or the security of the communications, that should be considered, but are not as important as these ones mentioned in [10].

To conclude, the RTCWeb protocol is always used in collaboration with the WebRTC API to be implemented, with the WebRTC API being merely a facilitator to deploy the requirements of the protocol, and it may function in any type of device, as long as the requirements are fulfilled.

In this project this collaboration will be explored.

RSC: reviewed

## 2.3 Signaling in WebRTC

Even though WebRTC has the objective of connecting users directly without any servers serving as intermediary, to establish a connection between two or more users it is still necessary to have servers so that users can exchange metadata to coordinate communication. This is called signaling.

Signaling in WebRTC has the job of passing messages back and forth between clients, that exchange information like:

- Session control messages, to initialize or close communication and report errors;
- Network configuration, to let the outside world know the computer's IP address and port;
- Media capabilities, to decide the codecs and resolutions that can be handled by the web browser and the web browser it is trying to communicate with.

WebRTC does not implement any type of signaling, leaving this aspect to the will of the developers. This happens because different applications may need or prefer to use different signaling protocols, like SIP [16], XMPP [17], WebSocket [18], or even a custom one.

### 2.3.1 WebSocket

Web applications that required bidirectional communication between a client and a server were required to abuse the HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls. This form of bidirectional communication brought a variety of problems, including:

- The wire protocol has a high overhead, with each client-to-server message having an HTTP header;
- The server is forced to use a number of different underlying TCP connections for each client: one for sending information to the client and a new one for each incoming messages;
- The client side script is forced to maintain a mapping from the outgoing connections to the incoming connection to track replies.

The WebSocket protocol [18] offers a simpler solution by using a single TCP connection for bidirectional traffic. This protocol, combined with the WebSocket API [19], can provide an alternative to HTTP polling for bidirectional communication from a web page to a remote server.

The protocols operation consists of an opening handshake between the client and the server. If the handshake is successful then the data transfer can begin. The data transfer occurs through a bidirectional channel, and each side can send data at will, without any dependence on the other. In the end of the communication a closing handshake is sent and the connection is closed, with all the data transfer being stopped.

This protocol is designed to substitute the existing bidirectional communications that use HTTP as a transport layer. These type of communications were implemented as trade-offs between efficiency and reliability because HTTP was not meant to be used for bidirectional communications. The WebSocket protocol works over HTTP ports 80 and 443 and supports HTTP proxies and intermediaries.

The design of the WebSocket protocol does not limit it to HTTP, meaning that future implementations may use a simpler handshake over a dedicated protocol. Being this flexible is an important part of the protocol because the traffic patterns of interactive messaging do not closely match standard HTTP traffic which can induce unusual loads on some components.

Summing up, the WebSocket protocol provides a mechanism for web applications that need bidirectional communication with servers that do not support the opening various HTTP connections. It can be used for games, stock tickers, multiuser applications with simultaneous editing, user interfaces exposing server-side services in real-time, among others.

### 2.3.2 SIP

SIP [16] is a protocol used for the establishment of communication sessions between two or more users. SIP creates, modifies, and terminates those communication sessions. These sessions can be Internet telephone calls, multimedia distribution, and multimedia conferences. This protocol is text-based, modeled on the request/response model used in the HTTP.

There are five characteristics of establishing and terminating multimedia communications that SIP supports:

- User location - determination of the end system to be used for communication;
- User availability - determination of the willingness of the called party to engage in communications;
- User capabilities - determination of the media and media parameters to be used;
- Session setup - ”ringing”, establishment of session parameters at both called and calling party;
- Session management - including transfer and termination of sessions, modifying session parameters, and invoking services.

SIP can also be used with other IETF protocols in order to have a more complete multimedia architecture and provide a better experience to the users.

Usually, in these systems, SIP is used together with the RTP for transport of real-time data and to provide QoS feedback, the Real Time Streaming Protocol (RTSP) for controlling the delivery of streaming media, the Media Gateway Control Protocol (MEGACO) for controlling gateways to the Public Switched Telephone Network (PTSN), and the SDP for describing multimedia sessions.

Although SIP should be used together with other protocols, to provide a complete service to the users, its basic features and operation are not dependent of any other protocol.

Concerning the use of SIP as a signaling protocol for WebRTC, it is SIP over WebSocket [20]. WebSocket works as a transport protocol carrying the SIP messages, instead of the proprietary messages.

This is necessary because the main transport protocols of SIP, i.e., UDP, TCP, TLS, and SCTP, are not directly accessible from the web browser. This means that there is no possibility to have a two-way real-time communication between clients and servers in web-based applications without transporting the SIP messages over WebSocket.

Also, SIP over WebSocket enables web communications with SIP networks, mobile and fixed phones, by giving web browsers the same capabilities as a mobile phone, like audio, video and Short Message Service (SMS).

### 2.3.3 XMPP

XMPP [17] is a communications protocol that allows real-time exchange of data between two or more network entities. It is based on Extensible Markup Language (XML), which makes it attractive for applications that need structured messages and rich hypermedia.

It started by being used only for instant messaging, presence information and contact list maintenance by the Jabber community [21]. But due to its extensibility, the protocol has also been used for Voice over Internet Protocol (VoIP), file transfer, video, and others, by the addition of extensions [22, 23]

The bidirectional communication of XMPP is done over HTTP, using the Bidirectional-streams Over Synchronous HTTP (BOSH) protocol, that defines how arbitrary XML elements can be transported efficiently and reliably over HTTP in both directions between a client and server [24]. This communication can also be attained by WebSocket [25].

Each node on a XMPP network uses XMPP content namespaces. For client-to-server communication it is used *jabber:client*. For server-to-server communication it is used *jabber:server*. It is used *jabber* instead of, for example, *xmpp*, because XMPP was created by the Jabber community, as aforementioned.

Users using XMPP are identified by an ID, that is formed using a combination of the username and the server name to form a *username@server* type of ID. This allows to easily exchange messages with users in different servers.

The users profiles are represented using the protocol Data Forms [26] and is registered and updated through Info/Query (IQ) requests to the server. The server will answer with and IQ response that has information in order to retrieve the registration fields needed to complete the users profile.

Concerning the connection between the user and the XMPP server, this can be performed using the same web server the user was already connected to. This will put a lot of stress on the web server, making it responsible for handling a lot of connections, and greatly impact the system performance. When using web applications, users can connect directly to the XMPP servers using *Strophe.js*, a JavaScript library [27].

### 2.3.4 Signaling-On-the-Fly

A very interesting concept to solve the problem of signaling is Signaling-On-the-Fly, also known as SigOfly [28]. It uses the ID of each peer and JavaScript scripts, with the objective of achieving interoperability between any WebRTC service provider domains and providing inter-domain communications.

With SigOfly, WebRTC applications use JavaScript scripts to implement signaling protocol stacks. The signaling protocol stack can be selected, loaded and instantiated during runtime. This characteristic is the one that enables signaling protocols to be selected by each WebRTC conversation, to ensure

full signaling interoperability among peers. Basically, the peers download all the code necessary to communicate with each other.

The biggest advantage of SigOfly is the fact that users are not restricted to only one type of signaling implementation. It also allows for multiparty conversations with more than one user coming from different domains, through a Mesh Topology with a Hosting peer or a Multipoint Control Unit based Topology with a Hosting peer.

## 2.4 Peer-to-Peer Streaming Protocol

As the P2P streaming traffic continues to experience a substantial growth, more providers are transitioning to P2P systems. Some CDN providers have started to use P2P systems and technologies to distribute their streaming content [29, 30].

Most of those systems use proprietary P2P protocols, which means that new developers need to develop their own protocols, and the lock-in effects lead to substantial integration difficulties with other players, e.g., CDN. This means that there is a need for an open and standard streaming signaling protocol.

To suppress that necessity, the PPSP was developed. It standardizes the signaling operations in every P2P streaming systems to solve the aforementioned problems. This protocol includes the peer (Peer-to-Peer Streaming Peer Protocol (PPSPP)) and the tracker (Peer-to-Peer Streaming Tracker Protocol (PPSTP)) protocols, which will be described next [29].

### 2.4.1 The Peer Protocol

PPSPP is designed to transmit, in a streaming fashion, the same content to a group of users (peers). The protocol is based on the P2P model where a user (peer) consuming content also acts like a server and disseminates that content to other users (peers), to create a system where everyone can provide upload bandwidth. PPSPP can stream prerecorded content and live audio and video content [31].

The protocol is generic, so it can run directly on top of UDP, TCP or other protocols. Currently, PPSPP runs on top of UDP using Low Extra Delay Background Transport (LEDBAT) for congestion control. LEDBAT makes possible for PPSPP to deliver the content without disrupting other tasks that the user (peer) might be doing, and that use the users network connection [32].

To prevent the interruption of the streams by malicious peers, the design of PPSPP has a short time-till-playback for the end user. To achieve this, the content in PPSPP is identified by a single cryptographic hash that is the root hash in a Merkle hash tree calculated recursively from the content. This tree, that can be used for prerecorded or live content, allows every peer to detect when another peer tries to distribute fake content [33].

One of the big problems with P2P systems is the fact that some peers only download content and never upload to other peers. PPSPP has flexible and extensible mechanisms to prevent this and to promote user cooperation. It also offers the possibility to have different schemes for chunk addressing and content integrity protection besides the defaults, so that each case has a scheme fit for it, and the possibility to work with different peer discovery systems, like centralized trackers or fast Distributed Hash Tables (DHT).

Relatively to peers, by default, PPSPP only maintains a small amount of state per peer. It also assumes that a peer gets a list of peers from a centralized tracker beforehand, and only after having that list the peer is able to connect to other peers, request content, and discover other peers disseminating the same content.

The overall operation of the PPSPP can be summarized in three actions:

- Join a swarm - to join a swarm, the peer starts by register with the tracker, following the specification of the PPSTP. Then the tracker will provide the peer with a list of peers in the swarm. After that, the peer will begin to exchange messages with other peers, using the PPSPP, so that it can connect to them. The peers also exchange information about the content and availability of it;

- Exchange chunks - a peer sends messages requesting chunks of a certain content. Peers that have those chunks will answer this request. The first peer receives the chunks, checks the integrity of them, and when it finishes to receive them, it sends a message to the other peers telling them that the reception of the chunk was successful. A peer is not obliged to answer every request and can decide when it accepts requests for the chunks it has. After the complete download of the content, a peer becomes a seeder and only sends messages to peers that still do not have the complete content, to only send messages to peers that still need the content;

- Leave a swarm - when a peer wants to leave a swarm, it sends a specific message to all its peers and also sends a message to the tracker, following the specification in the PPSTP, to unregister from it. The other peers remove the peer that left from their list. In case a peer crashes unexpectedly, the other peers will remove it from the list as soon as they detect that it no longer sends messages.

### 2.4.2 The Tracker Protocol

Peers are usually a user device that participates in sharing media content, receiving and transmitting content. They are organized in one or more swarms. In each swarm there are only peers that are streaming the same content at any given time. The role of the tracker is to coordinate the peers in a

swarm. Trackers maintain a list of peers that are storing chunks for a specific content, answer queries from peers and collect information about the state and the activity of peers.

To be able to do this, is necessary to have a form of communication between peers and trackers. Nowadays, this communication is performed using a variety of different protocols, turning difficult to achieve interoperability between every device.

It was with this in mind that there was the development of an open protocol, PPSTP, that standardizes all the communication process between peers and trackers and allows interoperability. It allows the peers to send meta information to the trackers, report streaming status, and obtain lists with the peers in a swarm.

A typical PPSTP session, if a peer wants to receive content, consists in:

- A peer connects to a tracker, registers in that tracker, and joins a swarm;
- That peer receives a list of other peers in the swarm from the tracker;
- Peer starts communicating directly with the other peers without the intervention of the tracker.

If a peer wants to share content, the session consists in:

- A peer connects and registers on a tracker;
- The peer sends information about the swarms it belongs to to the tracker;
- Peer waits for other peers to connect with them.

While a peer is connected to a swarm it needs to periodically report its status to the tracker. This allows the tracker to know that the peer is still active and to update the information regarding that tracker. If a peer is disconnected in any way, it can always communicate with the tracker and rejoin the corresponding swarms, to continue the previous activity [31].

RSC: reviewed

## 2.5 Dynamic Adaptive Streaming over HTTP

Delivery of video content over the Internet has been done using the RTP [34]. This protocol defines packet formats for both audio and video content, along with stream session management that allows delivery of low overhead packets. However, RTP is optimized for managed IP networks and, in today's Internet, CDN have replaced these type of networks. To make matters worse many of the CDN do not support RTP streaming, making it useless. Other problems with RTP consist of the fact that RTP packets are often not allowed through firewalls and that it requires the server to manage separate streaming sessions for each client, which makes large scale deployment very resource intensive.

The evolution of the Internet was accompanied by a big increase in the bandwidth available and also with a huge growth of the WWW. Considering these two factors, the necessity of delivering audio or video content in small packets has became almost irrelevant. Now, multimedia content can be delivered in large segments, in a very efficient way, using HTTP.

Using HTTP streaming has several benefits:

- The Internet infrastructure has evolved to efficiently support HTTP;
- HTTP is firewall friendly, since almost all firewalls are configured to support its outgoing connections;
- HTTP server technology is a commodity and therefore supporting HTTP streaming for millions of users is very cost effective;
- In HTTP streaming the client manages the streaming without the need of maintaining a session state on the server, meaning that provisioning a large number of streaming clients does not impose any additional cost on server resources beyond standard web usage of HTTP and can be managed by a CDN using standard HTTP optimization techniques.

Considering all the advantages of HTTP it was created the MPEG-DASH [35, 36]. MPEG-DASH is a standard for HTTP streaming of multimedia content. It was developed by the Moving Picture Expert Group (MPEG) due to the fact that the already existing platforms that use HTTP streaming, e.g., Apple's HTTP Live Streaming (HLS) [37], Microsoft's Smooth Streaming [38], and Adobe's HTTP Dynamic Streaming [39], required the device to support the corresponding proprietary protocol.

With MPEG-DASH audio and video are sent as a series of small files, called Segments. Usually, a Segment contains 2 to 10 seconds of media. The clients receive an index file, called the Media Presentation Description (MPD), that provides them with Uniform Resource Locators (URLs) for them to access the Segments. The MPD also allows the client to control the media delivery by requesting the Segments using HTTP and put them together before decoding and play-out.

The media is encoded at several bit rates to enable the client to adapt the download speed to the bit rate channel available to him. This leads to a significant reduction in buffer underruns and re-buffering. Because media is delivered through HTTP, the already existing HTTP infrastructure can be reused, without the necessity of deploying new MPEG-DASH compatible servers. HTTP caches and proxies in CDN can also be reused, for an efficient content delivery. Problems with firewalls and NAT are greatly reduced when compared with RTP streaming.

In the Figure 2.5 can be observed the MPEG-DASH system and how, in a basic way, that system works.

In this example, the Segments are from audio instead of video, but it would be identical for Segments of video. The audio is encoded at two bit rates: 64kpbs (rep1-64.mp4) and 24kpbs (rep2-24.mp4). These

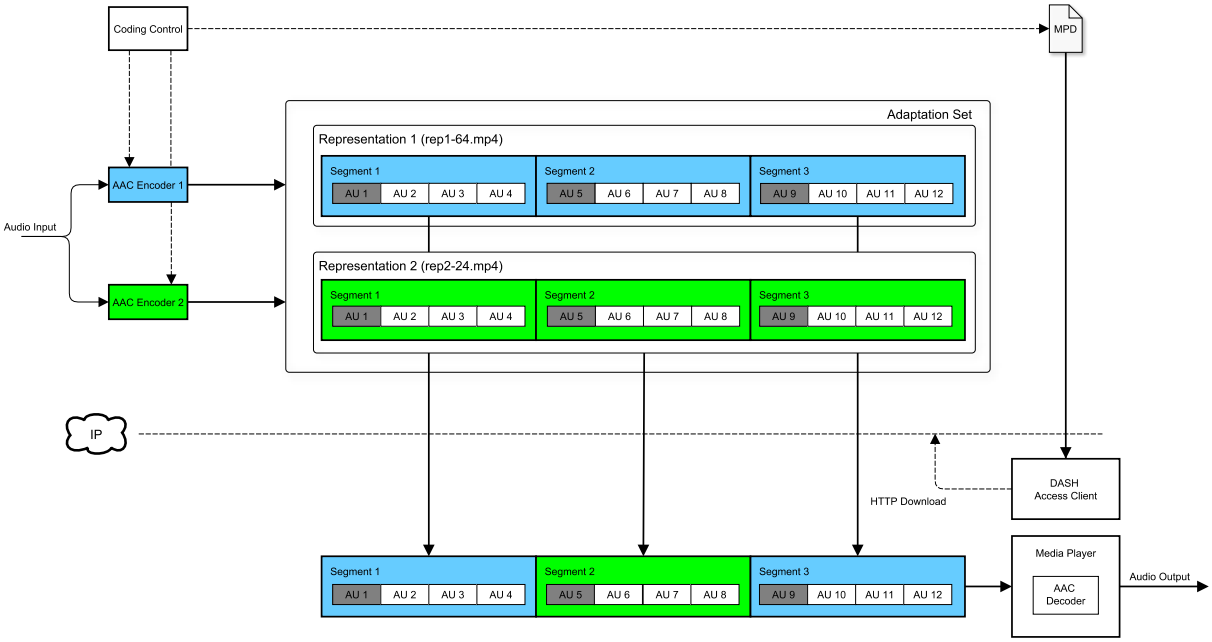


**Figure 2.5:** MPEG-DASH System Overview, adapted from [36]

encodings are called Representations. Representation 1 is for when the network is working normally, and Representation 2 is for congestion periods. All Representations which are encodings of the same content are grouped into an Adaptation Set. The Representations in one Adaptation Set can be used by the client to dynamically adapt the bit rate during stream.

Representations are divided into Segments, each with a duration of 2 to 10 seconds, allowing for dynamically switching by the client. Each Segment contains Access Units, which are Advanced Audio Coding (AAC) encoded audio frames. The number of Access Units is much higher than the one in Figure 2.5, reaching approximately 100 Access Units for a Segment of 2 seconds of duration.

On the encoder side, content is generated in a process comprised of two stages. First, the media is encoded into Segments and stored, in this example, in the corresponding MP4 files. Second, the MPD is generated with the description of how the Segments are encoded and how they can be accessed through HTTP downloads. This information is encoded in XML and stored in the MPD file.

On the client side, to process the content, the first thing to do is for the Dynamic Adaptive Streaming over HTTP (DASH) Access Client is to download the MPD, and then analyses it and decides which Segments are downloaded first. After the download of a Segment, it is passed to the Media Player API to be decoded and to start the play-out. The download of the first Segment also allows an estimation of the available bit rate on the channel. If necessary, the HTTP Access Client will download Segments of a Representation with less quality, to adapt to the channel conditions. It is relevant to note that the AAC Decoder inside the Media Player will always keep the same instance running, even when there is a switch in the Representation from which the Segments are downloaded.

# 3

# Related Work

## Contents


3.1 P2P Media Streaming with HTML5 and WebRTC . . . . . 27
3.2 Integration of WebRTC and SIP . . . . . 29
3.3 Webtorrent . . . . . 30
3.4 WebRTC Technology Overview and Signaling Solution Design and Implementation 31
3.5 Low Delay MPEG-DASH Streaming over the WebRTC Data Channel . . . . . 34

This chapter presents a few projects that were developed using some of the technologies referred in Chapter 2, and that are relevant for the project developed in this thesis. Most of the projects described in this chapter were implemented in a controlled environment, a laboratory environment, so to say, but they can be easily adapted to real world situations. For this, they are considered relevant and an important learning tool for this thesis.

## 3.1 P2P Media Streaming with HTML5 and WebRTC

The authors in [40] describe a system of Video On Demand (VoD), for services similar to YouTube, but using a P2P system, in which the users will also serve as peers and will propagate the video to other peers, not depending on just a few centralized servers from the service providers.

To achieve that, they will communicate directly through the web browser, without the necessity of having any plugins. HTML5 WebRTC standard is responsible for this browser-to-browser connection and communication. This standard provides real-time communication and also makes possible UDP communication to web browsers, instead of only TCP communication, which is the norm with HTTP.

The architecture implemented in this project is very similar to the architecture of BitTorrent, as it can be observed in Figure 3.1. It includes a Tracker and Peers, just like BitTorrent, but also adds a new element, a Seeder, not present in the BitTorrent architecture.

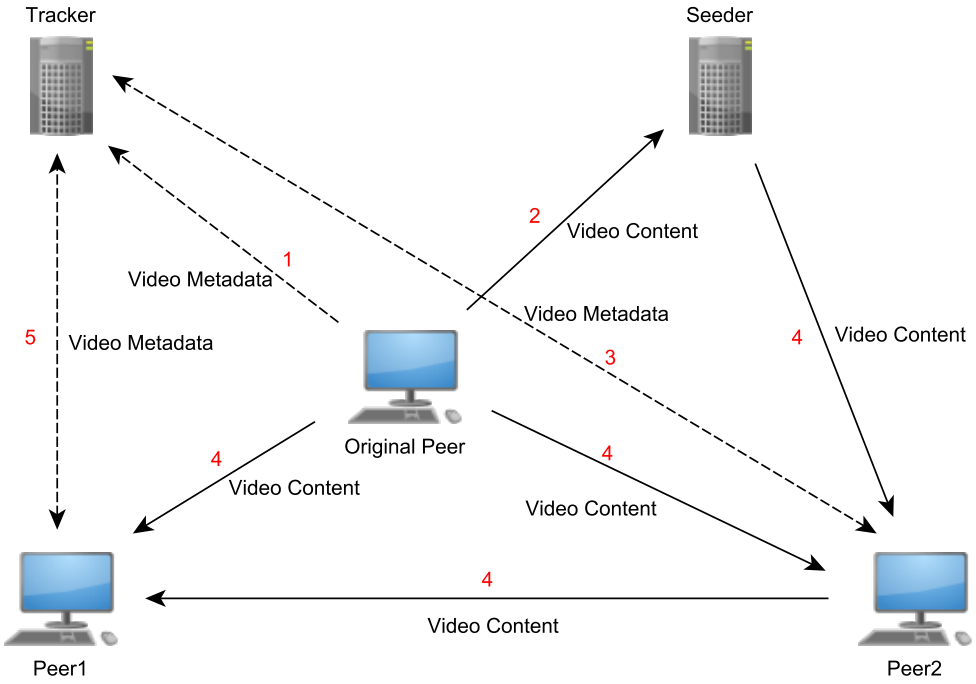


**Figure 3.1:** Overall architecture of BitTorrent [40]

Figure 3.1 also shows how the system works. (1) When a peer selects, automatically or manually, a

video to share, this video is moved to a location accessible to the web application. After, it is created the meta-data of that video, along with hashes of the pieces of the video, that will have cryptographic MD5 hash values. When this is over, the video is moved to a permanent location, using the “IndexedDB” API. Finally, the torrent file is uploaded to the Tracker.

(2) Besides that, the video itself is uploaded to the Seeder. The Seeder is a server that stores a copy of all videos and will also act like a peer. This way, even if the original uploader disconnects from the network, it is always possible to retrieve a copy from the Seeder.

(3) The next peer trying to download the video, needs to download the torrent file to his web browser and a list of peers that have the video, from the Tracker. (4) Then, it connects to those peers and the web application, running in the web browser, will begin to download pieces of the video from different peers.

In the beginning there will only be the original uploader and the Seeder, which means that they will have a bigger load than the other peers. But as more peers enter the network and start downloading and seeding the video, the load on them will begin to decrease.

(5) Due to the constant changes that the network may suffer, it is necessary to periodically access the Tracker to request the newest peer information.

During the implementation of the project, it was not possible to do experimental tests, since there was not any support for the Data Channel API on any of the conventional web browsers.

Alternatively, the authors performed tests on what could be measured: the performance of MD5 on JavaScript and how much load the transmission of the video would generate on the WebRTC based solution.

Regarding the MD5 hashes, there was a concern that the calculation of them would use too much Central Processing Unit (CPU) consumption and would affect the speed of upload the content. There was an additional concern with the mobile devices, because this could affect the overall consumption of the system and deplete the battery quicker.

Possible solutions for making the MD5 hashes calculations less expensive from an energy point of view would be to provide an open API as standard for the most used hash functions by web pages, improve JavaScript in web browsers or improve the JavaScript implementations of the hash functions.

About the load generated, there were also concerns, because of the video decoding and encoding processes. But the authors reached the conclusion that those processes are feasible, both on desktop and mobile devices.

All in all, the tests showed that it would be possible to have a VoD service, although, at the time, some limitations existed in the web browsers regarding WebRTC and the fact that the streaming applications would need to be optimized for the resources that mobile devices offer, which are far less than other devices.

RSC: reviewed

## 3.2 Integration of WebRTC and SIP

The authors in project [41] show how to surpass the difficulties in making interoperable the SIP and WebRTC integration, to achieve RTC sessions between browser-to-browser, browser-to-SIP communicator, or web browser to legacy phone, for example

WebRTC enables browser-to-browser RTC connection, but it requires a signaling protocol. In this case, SIP will be the one used, because it solves the issue of interoperability with the already built SIP systems. But the choice of protocol is decided by the programmers of the JavaScript WebRTC application and it is not required to use only SIP, evidently.

The integration of WebRTC and SIP carries some problems, especially at the media and signaling plane. For the signaling plane, the authors provided the SIP signaling functionalities through SIP WebSocket API. For the media plane, it was used a media gateway, namely the *mediaproxy-ng* and *webrtc2sip*.

For the implementation of this solution, it will be necessary a WebRTC client, a SIP signaling server and a media gateway. The architecture of the solution is shown in Figure 3.2.

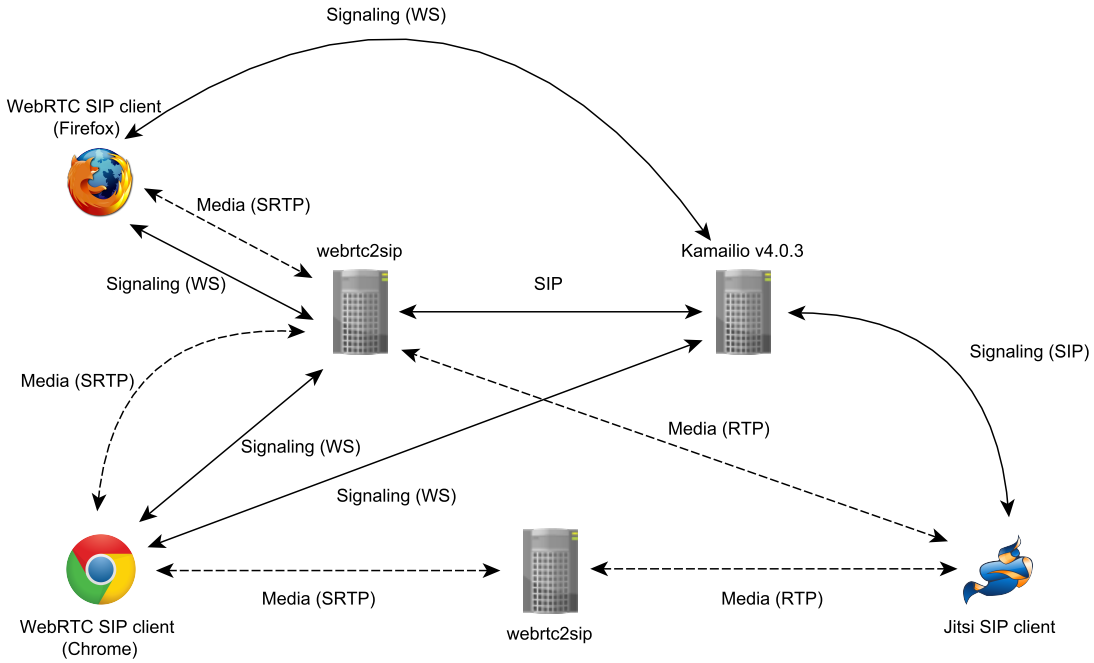


**Figure 3.2:** Overall architecture of WebRTC with SIP [41]

The WebRTC client signaling functionalities are implemented using WebSocket SIP API, so that WebRTC-to-WebRTC or WebRTC-to-SIP sessions can be established. The WebRTC client uses a

WebRTC SIP application created by the authors and it has, as expected, WebSocket and SIP characteristics for signaling.

The signaling server used is a Kamailio SIP proxy server, that supports the WebSocket protocol, because a WebRTC client uses the WebSocket protocol as signaling mechanism, so it is necessary to have a SIP entity that has a WebSocket and SIP protocol integrated.

It is also necessary to have a media gateway, to work as a translating mechanism. In this case, it will be used a *webrtc2sip* gateway. This is due to the fact that WebRTC demands support of protocols that are not present in simple SIP clients and because different web browsers do not support the same key exchange mechanisms, so it is necessary an intermediary (*webrtc2sip*) to translate the different protocols supported by Firefox (supports DTLS-SRTP) , Chrome version 33.0 (supports SRTP-SDES) and a normal SIP client (supports ZRTP-SRTP).

In the experimental tests, the authors noticed that sessions between peers using the same web browser would work without any problems but sessions between peers using different web browsers needed to access the media gateway for translation, as explained before. Although for the case of peers using different web browsers it is necessary to have an intermediary gateway, and they are not directly connected to each other, it can be considered that the integration of WebRTC and SIP was successful and it was possible to establish communication between two peers.

This implementation shows that is possible to integrate the WebRTC protocol with other technologies relevant to the build of a communication system compatible with all users, using only free and open source components and technologies.

RSC: reviewed

## 3.3 Webtorrent

Webtorrent is the first “real world” project, based on the already existent BitTorrent protocol, that really takes advantage of the capabilities of the WebRTC API. It is a streaming torrent client that operates in `node.js`, and in the web browser [42].

Regarding the `node.js` , the Webtorrent behaves as a simple torrent client, using TCP and UDP to communicate with other torrent clients, whether they use BitTorrent or Webtorrent.

In the web browser it does not need plugins, extensions, or to be installed, due to being completely written in JavaScript, and does not support UDP/TCP peers in web browser.

Still regarding the web browser for communication between peers, Webtorrent uses WebRTC data channels as the transport protocol, instead of the TCP/Micro Transport Protocol (uTP) protocols, like BiTorrent.

Due to the protocol changes necessary to do in order to make Webtorrent work over WebRTC, a web

browser Webtorrent client can only communicate with other clients that support Webtorrent/WebRTC.

A solution to connect both BitTorrent peers and Webtorrent peers is to have intermediary servers, called hybrid servers, that translate BitTorrent into Webtorrent and vice-versa. This solution can be seen in Figure 3.3.

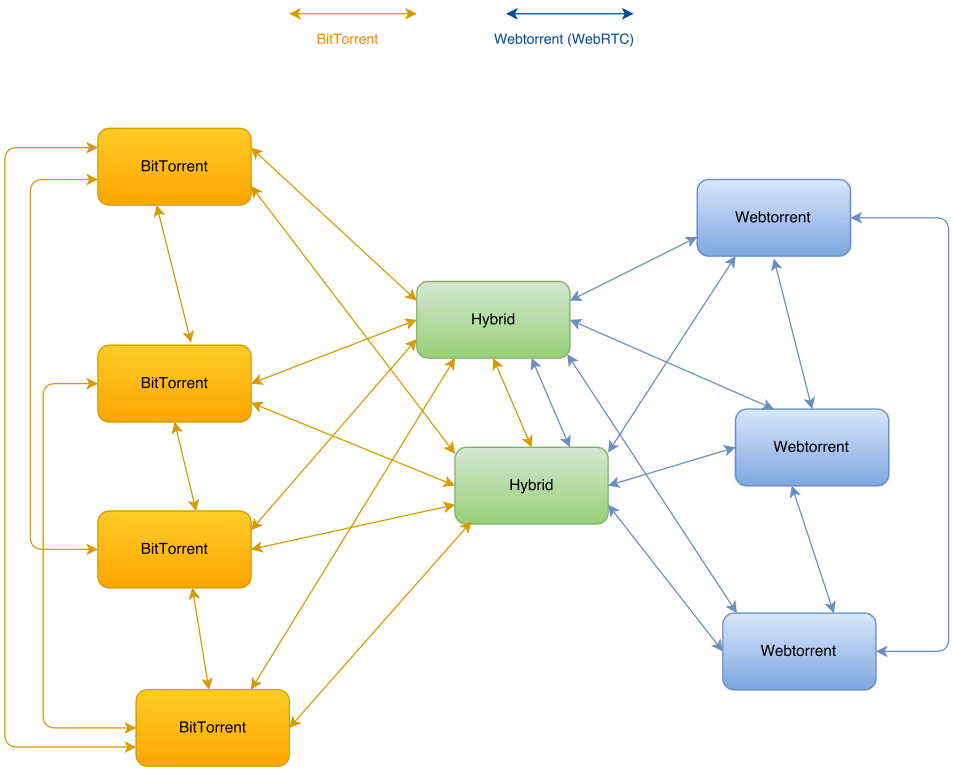


**Figure 3.3:** Hybrid Webtorrent system, adapted from [42]

By simplifying the process for the users, making unnecessary for them to understand `.torrent` files, magnet links, NAT, etc, needing only a web browser to work, it can attract a new wave of users willing to participate in this P2P system.

RSC: reviewed

## 3.4 WebRTC Technology Overview and Signaling Solution Design and Implementation

Sredojev et al. [12] implemented a videoconferencing and chat application that was able to connect peers through the web browsers without any external plugins, using WebRTC.

This work was made considering communications inside an enterprise and its branches, were the communications were carried out through a Virtual Private Network (VPN). Even so, the techniques

applied in this project can be used in more scenarios than just inside enterprises.

The overall architecture of the project is very simple, as can be observed in Figure 3.4. It it only necessary a signaling server, two peers and a way for the peers to communicate (signal) with the server.

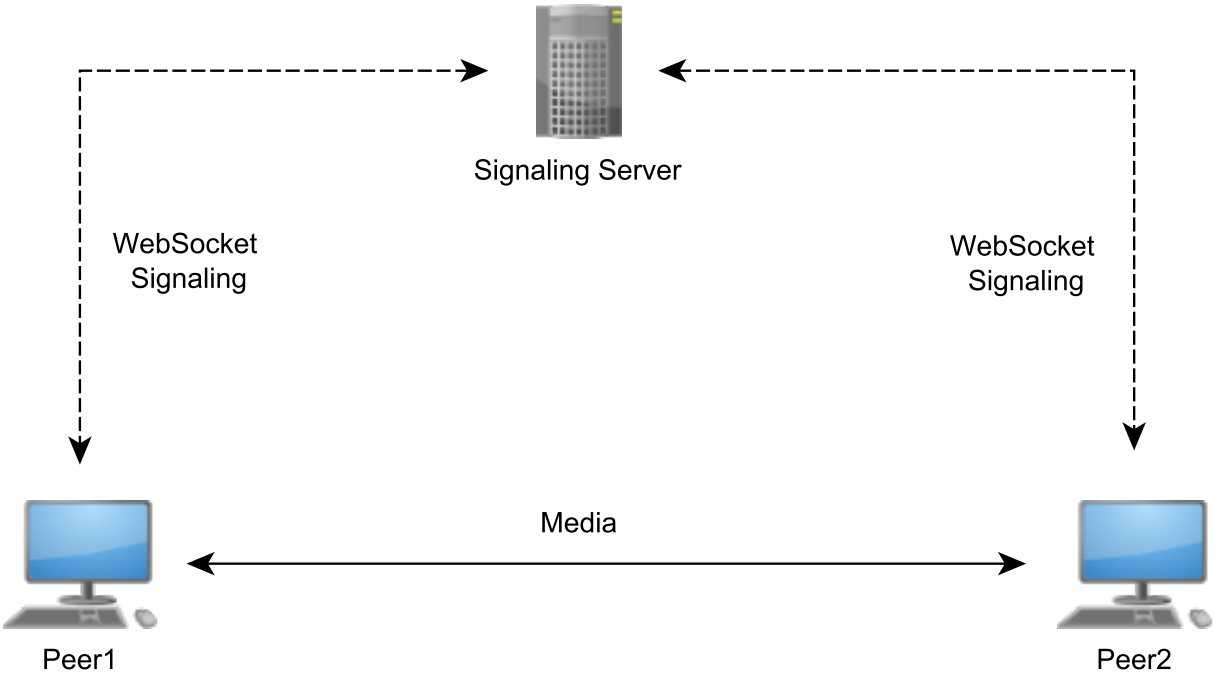


**Figure 3.4:** Overall architecture of WebRTC [12]

The Signaling Server is a WebSocket server, written in Node. The decision of implementing the server in Node was due to the fact that this is an open source, cross-platform runtime environment for the server-side and networking applications. Node optimizes the throughput and scalability of an application, something essential for when the server needs to handle more than two peers at the same time. The server also handles the control messages: “initialize”, “initiator”, “got user media” and “peerChannel”; media information messages: “offer” and “answer”; network information messages: “candidate”; and manages the peers.

As expected, the application for the users (peers) was implemented using the WebRTC API. This API is used, mainly, for a P2P connection between the web browsers of the users (peers). Before the connection is established, exchanges local and remote descriptions of the audio and video media information. For this there will be used the *setLocalDescription()* method and the *setRemoteDescription()* method, that are in the API.

For communicating (signaling) with the Signaling Server, it is used the WebSocket protocol, which allows to establish a session between the peer and the server. The WebSocket protocol maintains the session open until it is closed.

The signaling solution of this project used four types of control messages, as mentioned before:

- initialize;
- initiator;
- got user media;
- peerChannel.

Depicted in Figure 3.5 is a diagram of a session establishment between two peers.

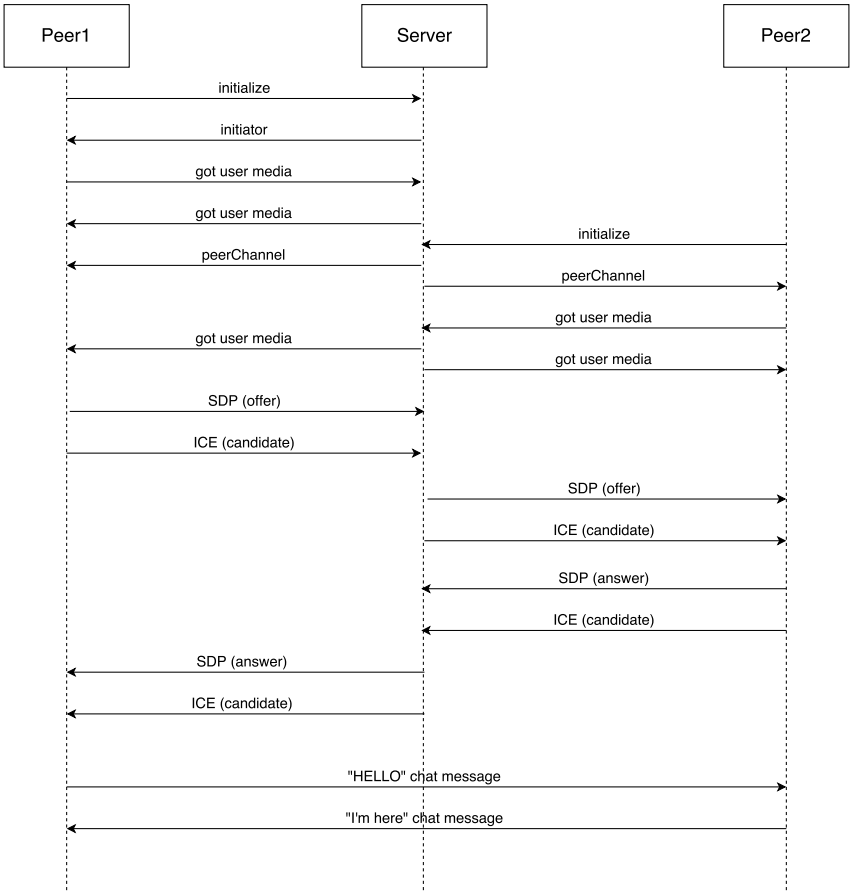


**Figure 3.5:** Message Sequence Diagram to establish a session, as seen in [12]

The `initialize` message is the one sent by any peer that wants to signal the server, to register himself, as it can be seen, in Figure 3.5. In this case, Peer1 is the one to initiate the session, as confirmed by the `initiator` message sent by the Server. After knowing that it is the initiator of the session, Peer1 will configure the user media following the reception of the `got user media` message. Then Peer1 waits for Peer2. When Peer2 wants to connect, it sends a `initialize` message to the Server and becomes aware that it is not the session initiator. The Server then sends `peerChannel` and `got user media` for both Peer1 and Peer2.

After this exchange of messages, Peer1 and Peer2 can start to establish a P2P session. They first need to inform each other of the media type, format, codecs, and all the other properties that will be used in the session. This information will be in the ”offer” and ”answer” messages that are transmitted using the SDP. Peer1 sends the ”offer” to the Server, this one sends it to the Peer2. Peer2 sends the ”answer” to the Server that sends it to Peer1.

They also need to know in which will be the protocol used, and in which IP address and port the session will be established. For this ICE “candidates” messages will be sent. Again, the Server will intermediate this exchange of messages: Peer1 sends its “candidate” to the Server, this one sends it to the Peer2. Peer2 sends its “candidate” to the Server, this one sends it to the Peer1.

When the exchange of SDP and ICE messages is over, the connection is established and the peers can communicate without using the Server as an intermediary. Peers use their web browsers to communicate with each other and nothing else. There is no need to install any plugins.

In the experimental tests, the application worked perfectly and can be concluded that the implementation of the project was a success. The authors also concluded that it is feasible to send other types of media and files with WebRTC and to encrypt them too, which is something very important, not as much as in a VPN, but more on public networks with a high risk of having their signals intercepted.

This project demonstrates that WebRTC is a powerful technology that can improve the overall communications, changing the traditional way media and files are exchanged.

RSC: reviewed

## 3.5 Low Delay MPEG-DASH Streaming over the WebRTC Data Channel

In the project [43] it was explored a new low delay solution for the streaming of MPEG-DASH using the WebRTC data channel. The results were then compared with a normal MPEG-DASH streaming system, to reach a valid conclusion.

For the experiments, the authors used a NS-3 simulated network, and deployed two Linux containers using a Linux LXC library and Chrome, to run the simulated network with an existing DASH.js player and WebRTC applications. The network link bandwidth is limited to 1 Mpbs. The DASH.js player and WebRTC were deployed as user level applications, that ran on top of two different Linux containers, connected by tap devices using an NS-3 simulated Carrier Sense Multiple Access (CSMA) network, like aforementioned. This setup is represented in Figure 3.6.

Using LXC in the [43] project allows to take full advantage of virtualized network topology with various network conditions and there is no need to rewrite existing applications, only deploying them. LXC is also more flexible compared to NS-3’s Direct Code Execution (DCE) technique. Each LXC node has two

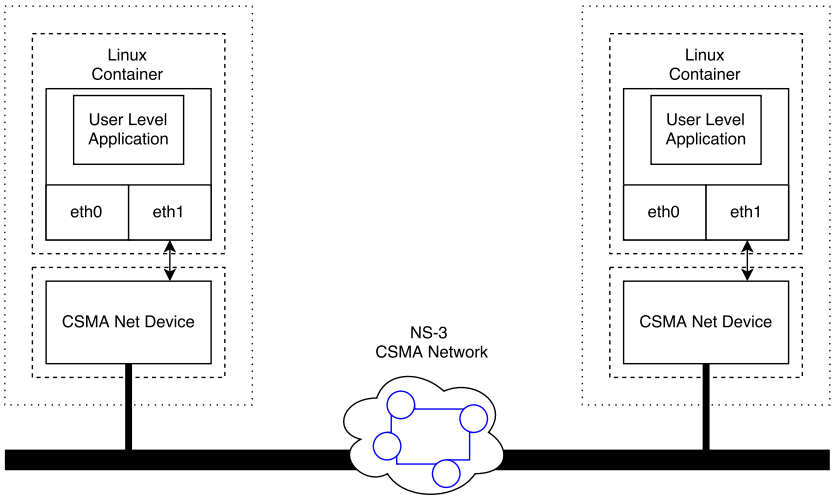


**Figure 3.6:** Experiment setup adapted from [43]

network interfaces. These interfaces are used for data transfers and WebRTC signaling, respectively. This allows to use either a local signaling server as well as a remote one such as Firebase.

In this system, where it is used MPEG-DASH, streaming bitrate is controlled by the client. The client, based on the available network bandwidth, will request the next proper bitrate. The media server sends the requested segment to the client, after receiving the requests.

With the bitrate being controlled by the client, extended playback delay and decrease jitter time can be avoid, in order to improve user experience by downloading the adaptive bitrate segment. The media server itself has no knowledge of network conditions, because its main function is to provide the various bitrate segments.

The authors started by simulating a normal MPEG-DASH streaming system. The topology of this system can be seen in Figure 3.7. It was deployed a DASH.js v2.1 player on one LXC container and a DASH segments' Apache Server on the other end of the network topology.

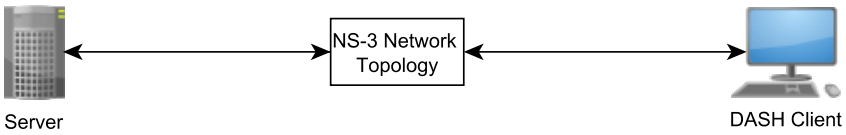


**Figure 3.7:** MPEG-DASH topology adapted from [43]

Between the media server and the DASH client, the simulated NS-3 network is a CSMA network. It has 1 ms of delay between each node and there was no background traffic while the tests were being conducted.

The results of the tests showed a maximum playback rate of around 0.8 Mbps, that the playback bitrate kept changing, showing an ON/OFF pattern, and that the initial requests start from a low bitrate segment and then rise to the maximum bitrate segments. After a period of constantly requesting high

bitrate segments and downloading them, the network bandwidth was all used, so, the DASH client started to request low bitrate segments, in order to not cause any playback delay and buffer.

A big problem with this system is that the typical DASH client, like DASH.js, uses HTTP as the data transport layer protocol. This runs on top of TCP that has a slow start. As soon as the TCP window collapses, the playback rate on the client side drops drastically. Despite this drawback, the DASH client can adaptively keep changing the segment bitrate and keep a high user experience.

To try and mitigate the issues with the normal MPEG-DASH streaming implementation, the authors used the *datachannel* function of the WebRTC that uses UDP instead of TCP for binary transfers. The topology of that solution can be seen in Figure 3.8. It consists of a WebSocket server, for signaling, and two peers that will use the *datachannel* function of the WebRTC to transfer binary data between them.

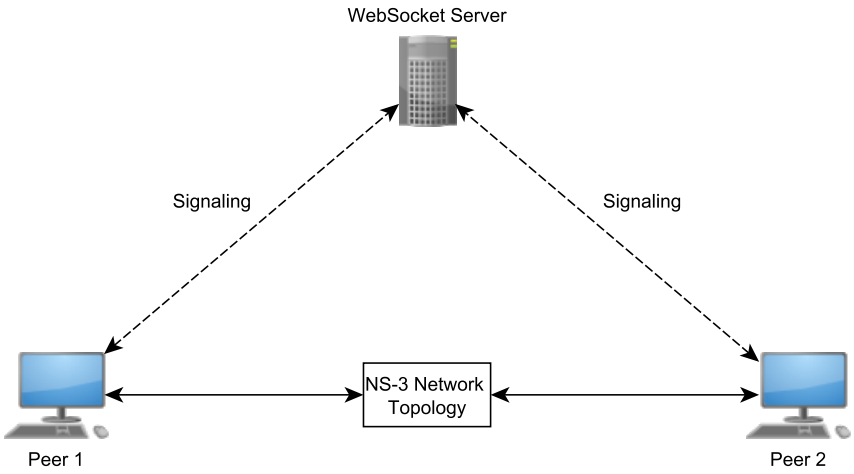


**Figure 3.8:** WebRTC topology adapted from [43]

The authors of [43], when using WebRTC data channel transfer media binary data, implemented a push based and self-pacing transport control, different from the DASH.js client's pull based control. Chunks of 16 kB were used and the "bufferedamountlow" event for sender side flow control. In the simulation, the receiver kept the maximum throughput of around 0.9 Mbps and the the time difference of the initial receiving, between the sender and the receiver, is of 170 ms.

Figure 3.9 represents the workflow of the signaling process between the peers (clients) and the WebSocket server. The objective of this process is to establish the WebRTC datachannel.

In this workflow, Peer-1 is the one to initiate the data channel connection. It first connects to the server, shown by the message `connect`, and then it sends a request to the WebSocket server to start the creation process (`Create a datachannel`).

After, the server approves the data channel and sends it back to the initiator (Peer-1), and also sends a message of confirmation, `Emit datachannel`. It keeps waiting for any connection from other peers, after it sends the message of confirmation.

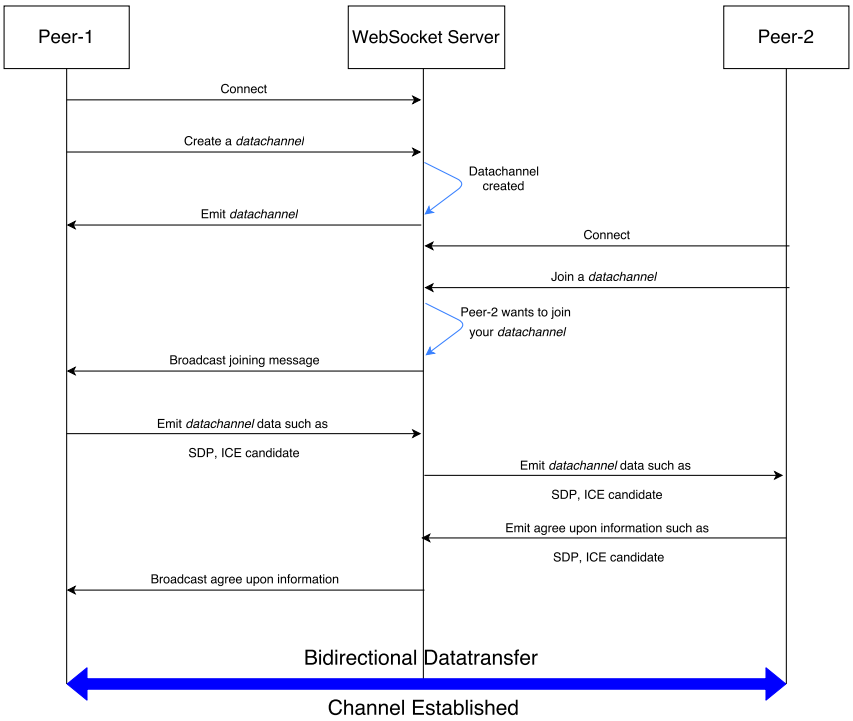


**Figure 3.9:** WebRTC datachannel establishment flows as seen in [43]

Peer-2 then connects to the WebSocket server (`connect`) and sends a `Join a datachannel` to the server in order to join the previously created data channel by Peer-1.

Posteriorly to receiving the joining request (`Join a datachannel`), the WebSocket server broadcasts that message to all the other peers using the data channel. In this case, there is only one other peer, Peer-1.

Peer-1, after receiving the message that there is another peer that wants to join the data channel, sends, to the WebSocket server, its information, through SDP, about the media type, format, codecs, and the other properties that will be used in the session. It also sends, through ICE, the protocol to be used, and in which IP address and port the session will be established. The WebSocket server transmits this information to the Peer-2.

Peer-2 will receive the information and, if it agrees with the parameters, send an agreeing message (`Emit agree upon information such as SDP, ICE candidate`) to the WebSocket server that will deliver it to Peer-1.

Once both peers agree about the parameters to communicate with each other, a bidirectional WebRTC data channel is created, and peers will communicate directly without any server as an intermediary.

Compared to the traditional DASH.js client, the solution using a WebRTC based DASH client side playback performance has a much lower initial streaming start delay and a very high link capacity utiliza-

tion. This results in a expressive constant high bitrate data segment transfer without showing an ON/OFF pattern like DASH. This validated that using WebRTC transport layer protocol can be an advantage for the DASH data transfer.

This implementation of MPEG-DASH shows that it is compatible with the WebRTC data channel and it is more efficient than a a typical DASH.js client over the HTTP link.

# 4

# Architecture

## Contents


4.1 Requirements . . . . . 41

4.2 Architecture Design . . . . . 42

4.3 Components . . . . . 46

This chapter describes the architecture of the proposed solution. It starts by listing the requirements, and after it presents the design of the system. Lastly, each module present in the architecture is briefly described, for a complete understanding of the system.

## 4.1 Requirements

This solution is intended to be applied to a multitude of distinct scenarios and systems, but always with the objective of implementing real-time collaboration in a P2P fashion focused on video streaming, using only a simple web browser. Because of that, there are some requirements that need to be fulfilled independently of the scenario or system:

- **Allow for P2P collaboration -** the main objective of the project developed in this thesis is to allow for peers to collaborate with each other, with video streaming, in a P2P fashion using WebRTC;
- **Peers can collaborate -** only peers in the same web page, watching the same video stream, can collaborate with each other;
- **Only authorized peers can contribute -** in order to avoid unnecessary content or spam, only authorized users can stream video to the P2P system. Although any peer can access the stream and watch, not everyone will be able to provide an original stream for the P2P system;
- **Self-scalable -** the solution must be able to maintain Quality Of Experience (QoE) and QoS no matter how many peers are using it;
- **Signaling Server coordinates the connections -** the Signaling Server will be responsible for connecting the peers between them, exchanging peers metadata to coordinate communication between them and to cope with NATs and firewalls;
- **Peers update their state frequently -** peers need to communicate to the Peer Coordinator the streams they are peers of, i.e., the streams they can also upload to other peers. This update is not requested, it is automatically done by the peer, from time to time. In case a peer stops communicating for a certain period it will be considered offline;
- **Stream of video -** system needs to provide to the peers the ability do stream video and watch the streamed video they want;
- **Reduce the server load -** the Media Server present in the system is considered a super-peer. The first peers connect to this super-peer to receive a video stream, but, after the P2P network starts to compose, this server will stop to have peers connected to it, reducing its load;

- **Ability to interact with the system -** the peer should be able to interact with the system in order to fully take advantage of all the functions that it permits.

The core of this solution is the software that will be implemented, especially the web application as it is responsible for combining all technologies in charge for permitting to achieve real-time collaboration in a P2P fashion focused on video streaming, and to let the user to take advantage of them in an easy way. Relatively to the hardware, nothing out of the ordinary is used, mainly consisting in servers, and devices with access to the Internet.

## 4.2 Architecture Design

Based on the related works studied and on the research work done, it was concluded that the best system architecture for the solution is the one illustrated in Figure 4.1. This architecture, consisting of a Peer Coordinator, Web Server and Media Server, a Signaling Server, and an arbitrary number of peers, is designed to be easily integrable with any web stack in the Internet.

To be easy to illustrate and explain, the system is shown only with three peers, but, like aforementioned, it is designed to be able to service as many peers as necessary. These three peers represent a swarm. The Peer Coordinator in this solution, that works like a tracker, does not communicate with other Peer Coordinators. This means, in practice, that the peers need to directly access the Peer Coordinator responsible for this swarm to be able to join this P2P network and access the video streams. The web application is not represented in the architecture, but it will be implemented only in the peers.

It should be noted that although Peer Coordinator, Web Server and Media Server are represented in the same physical node, they are in fact different modules, each with a different function. Also, the Media Server module will act as a peer, a super-peer, that will distribute the media to the first peers that request the video stream.

The peer node consists of two very important modules: the Content Loader and the Connection API. The Content Loader handles everything related with the video stream and decides from which peer or peers the stream should be requested from. The Connection API handles the connections between the peers.

In Figure 4.2 can be observed the interactions between the various participants of the system. This flow only represents the interaction of two peers, but the interaction for more peers is linear and can be perceived from the flow. The flow presupposes that the system is started at the beginning of it and that there are no network errors during it. The interaction is described in the following list:

- The Media Server announces to the Peer Coordinator that it is the source of the stream.
- Peer 1 accesses the webpage and registers itself in the Peer Coordinator.

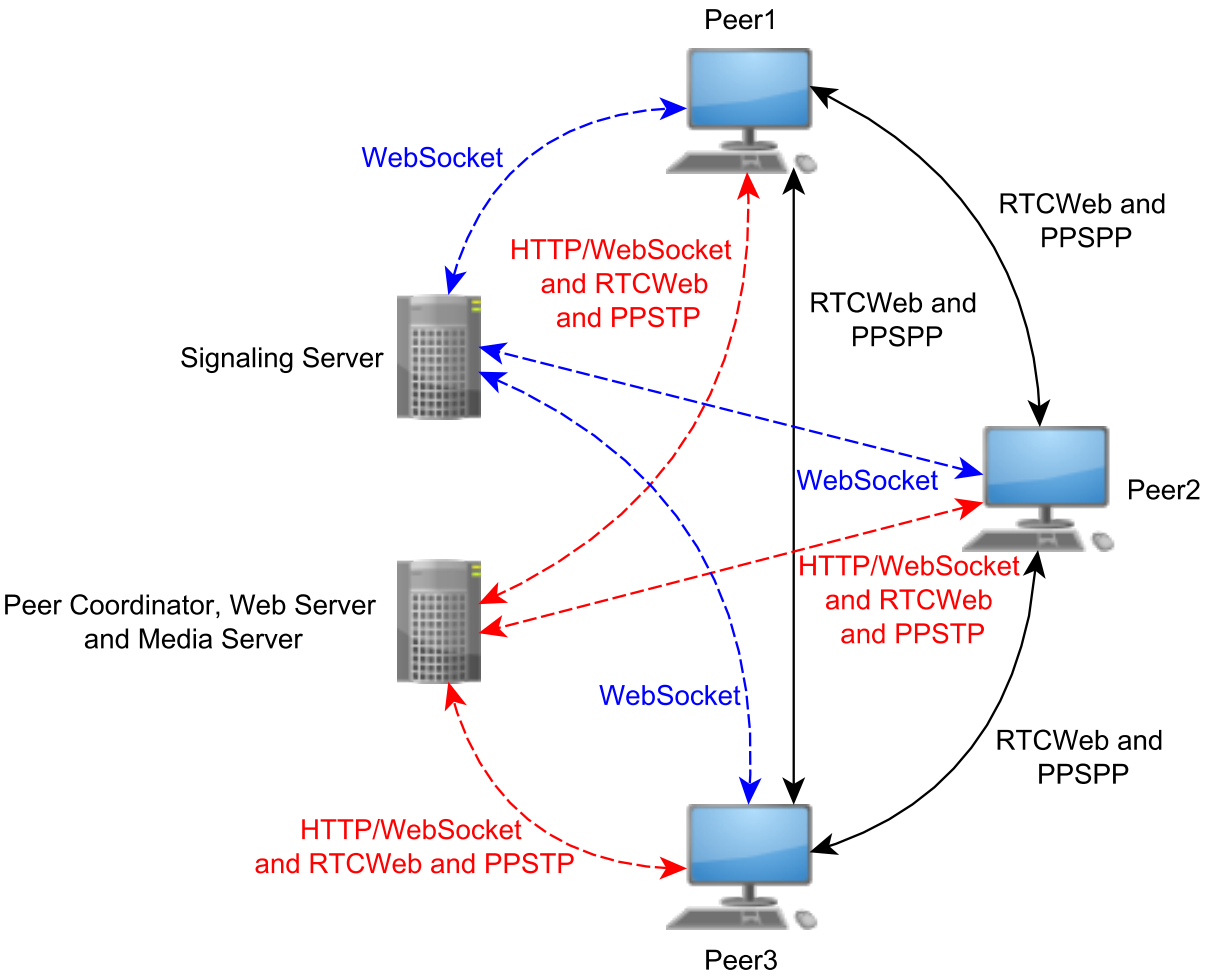


**Figure 4.1:** System's architecture for the proposed solution

- Peer 1 receives the webpage and a confirmation of its registration.
- Peer 1 then requests the stream it wants to the Content Loader that indicates to the Connection API that Peer 1 wants the stream.
- The Connection API requests the list of peers that are streaming that specific stream to the Peer Coordinator.
- The Peer Coordinator sends the list to the Connection API, that sends it to the Content Loader.
- The Content Loader decides from which peer, or peers, Peer 1 should connect to receive the stream from.
- The Connection API will then signal the Signaling Server, to gather ICE candidates and, after gathering them, will offer a SDP to the Peer Coordinator.

- The Peer Coordinator will transmit to the Media Server the SDP that was offered by the Peer 1.
- Then, the Media Server will also gather its ICE candidates and answer the SDP of the Peer 1, sending that answer to the Peer Coordinator, that will send it to the Peer 1.
- A bidirectional WebRTC data channel is established between Peer 1 and the super-peer, the Media Server.
- Finally, the Connection API will send the stream that is receiving to the Content Loader, that displays it to the peer, and announces to the Peer Coordinator that it is now a source for the stream.
- Peer 2 interaction is identical to the one of Peer 1, with the exception that Peer 2 establishes a WebRTC connection with Peer 1 and not the Media Server.

This flow excludes secondary interactions of the elements present in the system, such as the peers informing automatically the tracker of their state, the automatic actualization of the MPD, or the Tracker sending up to date peer lists to the Signaling Server. Although excluded, they should be taken into consideration when implementing the system.

The second interaction, represented in Figure 4.3 involves a peer wanting to collaborate with a video that only he has. It is assumed that the interaction begins after there are already WebRTC data channel open and no network errors occur during the interaction. The following list describes the interaction:

- Peer 1 informs its Content Loader that wants to stream video. Then the Content Loader, in turn, informs the Connection API.
- The Connection API asks authorization to the Peer Coordinator, to stream video.
- After receiving authorization it informs the Content Loader, which will then send the stream to the Connection API.
- The Connection API sends the video stream to the Media Server, because it is a super-peer.
- Next, the Media Server updates the MPD on the Web Server, in order for peers to be able to receive the new video stream, and announces itself to the Tracker as source of the new stream.
- Finally, the Connection API of Peer 1 also informs the Peer Coordinator that it is a source of the new video stream.

Once again, this flow also excludes secondary interactions of the elements present in the system, namely the peers informing automatically the tracker of their state or the constant and automatic actualization of the MPD, just to name some.

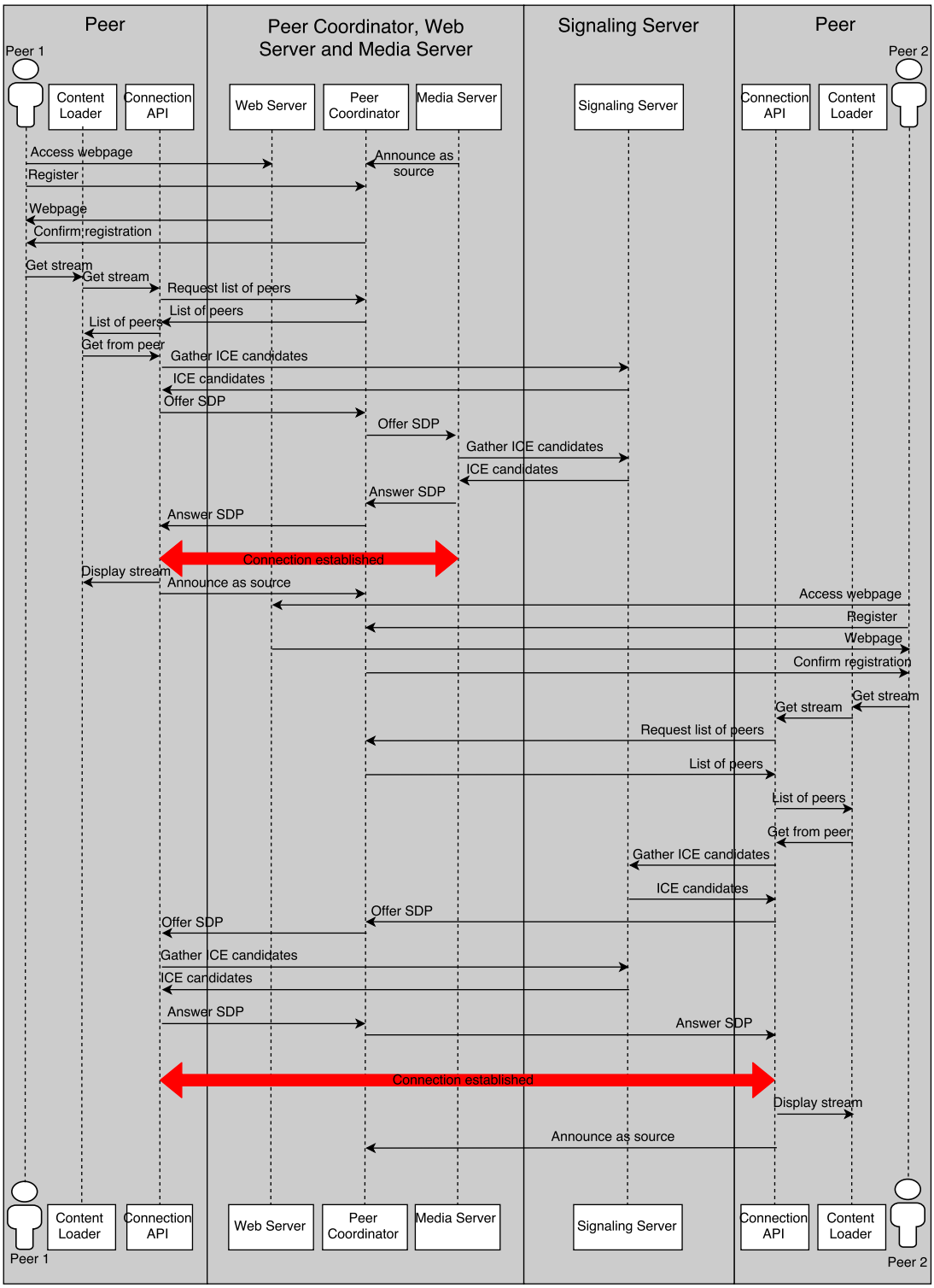


**Figure 4.2:** System's participants interaction

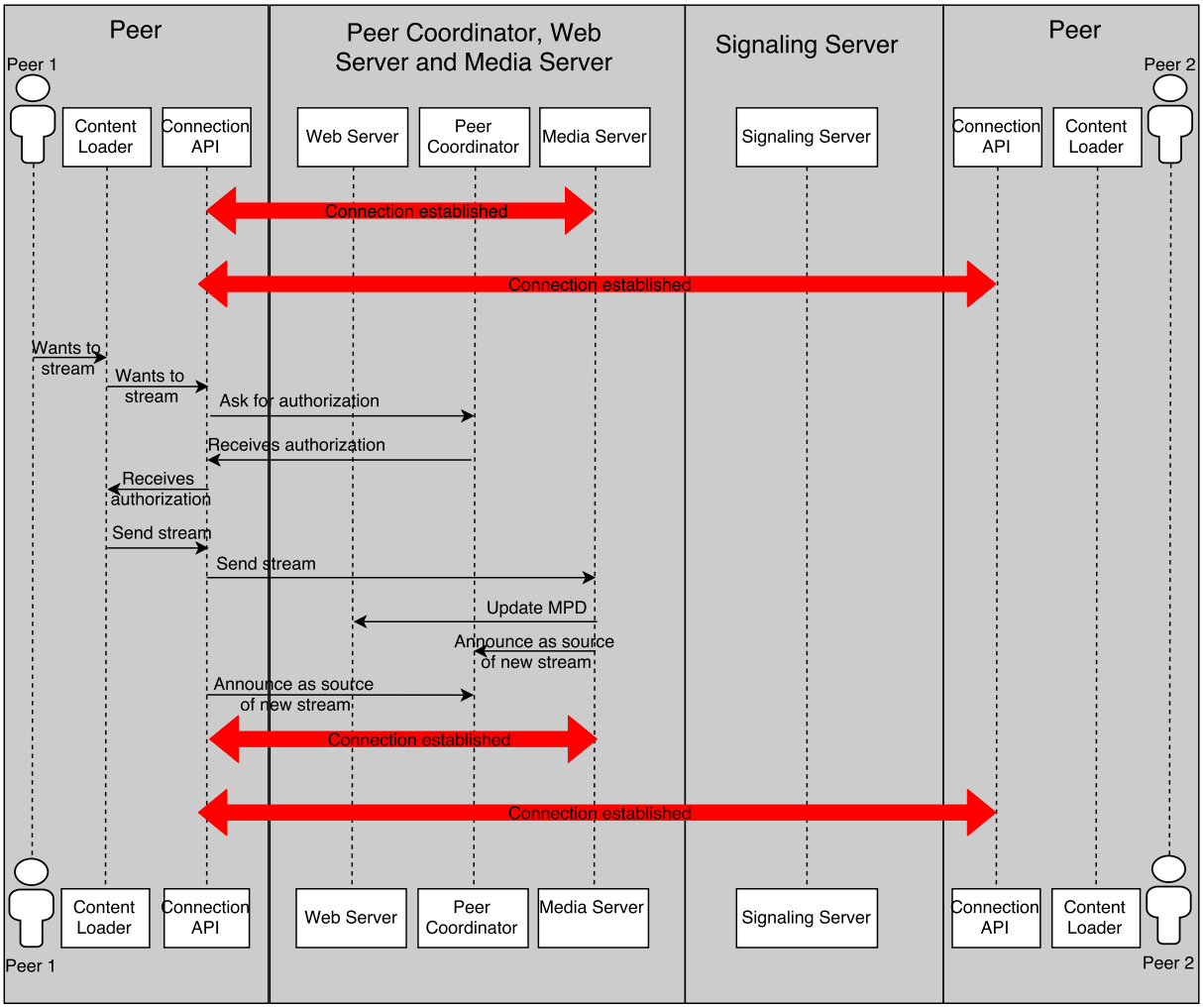


**Figure 4.3:** Peer collaborates with content interaction

Another aspect important to consider when implementing the system is how a user will be registered. In this solution, the Peer Coordinator attributes an ID to a peer that registers. This ID is used to give the user anonymity and will also be used in the list that the Peer Coordinator compiles of the peers that have a stream.

## 4.3 Components

### 4.3.1 Peer Components

Peers are users that not only download a video stream but also upload it to other peers that want that same video stream. In this solution, only authorized peers can collaborate with their own video stream. When a peer collaborates with a video stream, this stream will overlay the original video stream and it

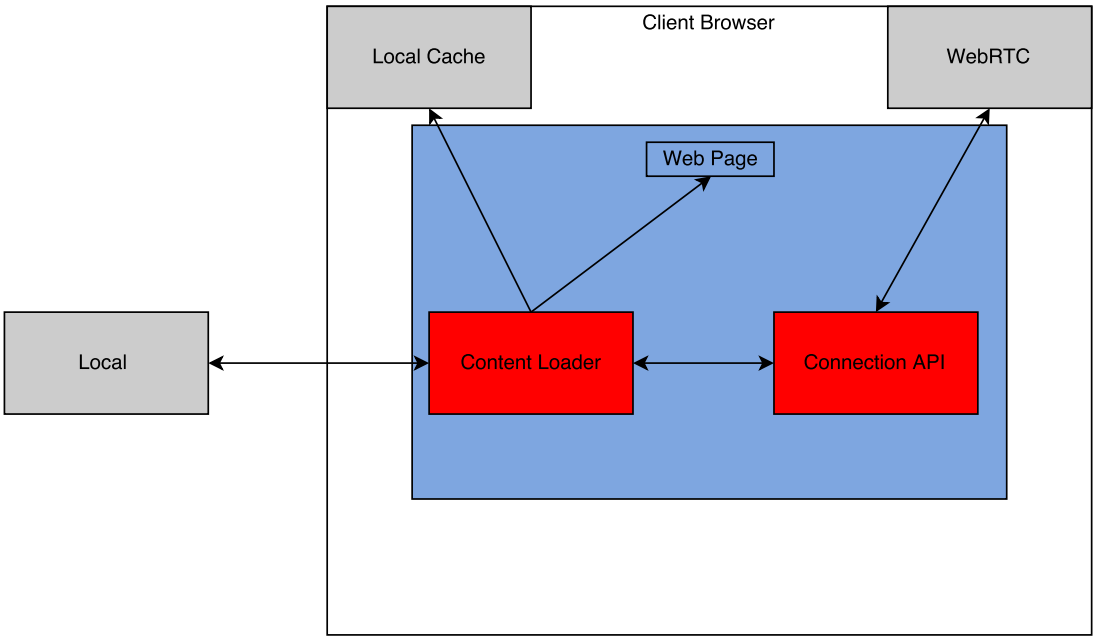


**Figure 4.4:** Peer modules

will be shown, in real-time, to the other peers in the swarm.

Two modules compose the peer: the Connection API and the Content Loader. The Connection API simplifies the WebRTC functionalities regarding the bidirectional data channel. The Content Loader takes decisions regarding the video and cooperates with the Connection API when it is required.

These two modules will be executable on the browser and are described in the next sections.

### Connection API

For this solution to be fully implemented, it is necessary to have a module that communicates with the Peer Coordinator and the Signaling Server, and that transforms the creation of the WebRTC data channel between two peers in an easy to use function.

To connect to another peer the Connection API will start by selecting a STUN server to use. It is noteworthy to mention that the IP address resolution should be done by the Signaling Server, but in the implementation of this project a Google's public STUN server will be used.

In case a STUN server can not resolve the address of a peer, there will be left only two options: use a TURN server or transmit an error message to the peer. Using a TURN server implies to have some costs and also will not provide a direct connection between the two peers. So, the second option will be used, the transmission of an error message to the peer, in case the IP address resolution fails.

After the STUN server resolves the IP address, the next step is to send the SDP to the Peer Coordinator which will send it to the intended peer. SDP is a set of rules that defines how multimedia sessions can be set up to allow all end points to effectively participate in the session. When the other

peer answers positively to the SDP it received the two peers can connect directly through a WebRTC data channel.

Through this newly created data channel the peers can send and receive video from each other.

#### Content Loader

The Content Loader will determine if a resource is already loaded, were it is going to be loaded from, and coordinates with the Connection API to open and manage the peer connections. In this solution the Content Loader loads images of the website and video streams from one or more peers.

In the beginning, if the peer already has the web page in cache, the Content Loader prevents it from re-download all objects of the web page. The Content Loader will also interact with the local device, in case a peer wants to contribute with an original stream.

After a connection with other peer is made, there will be a continued stream of data arriving. The module will reassemble it and, after the video is reassembled, it is displayed to the peer. The correctness and authenticity of the video will be assured by the fact that only authorized peers, hence trustworthy, will be able to collaborate with video streams. After starting to load the stream, the Content Loader notifies the Tracker that this peer is now a source of that stream.

There are no mechanisms to avoid failure when a peer can not connect to other peers. But other solutions can implement them, like, for example, having a normal media server streaming the video and peers connect to them instead of using P2P.

When a peer closes the browser window, the listener will warn the Peer Coordinator to remove the peer from the list of sources. In the event of the browser or the computer crashing or the connection having problems the peer is removed from said list if it does not respond to any messages for more than a predefined time. In this solution, that predefined time will be 3 seconds.

### 4.3.2 Peer Coordinator, Web Server and Media Server

This server will be composed by three modules with distinct functions and functionalities: the Peer Coordinator, the Web Server, and the Media Server. These modules, which are represented in a single node in this solution, can be divided into three different nodes, in case it is necessary.

The main objective of this server and its modules is to always provide the most up to date video and to control the network of peers.

#### Peer Coordinator

In a simple way, the Peer Coordinator will be the responsible for registering the peers and to, or not, to give and revoke the authorization of peers to collaborate with a video stream, and to assume the role of a tracker.

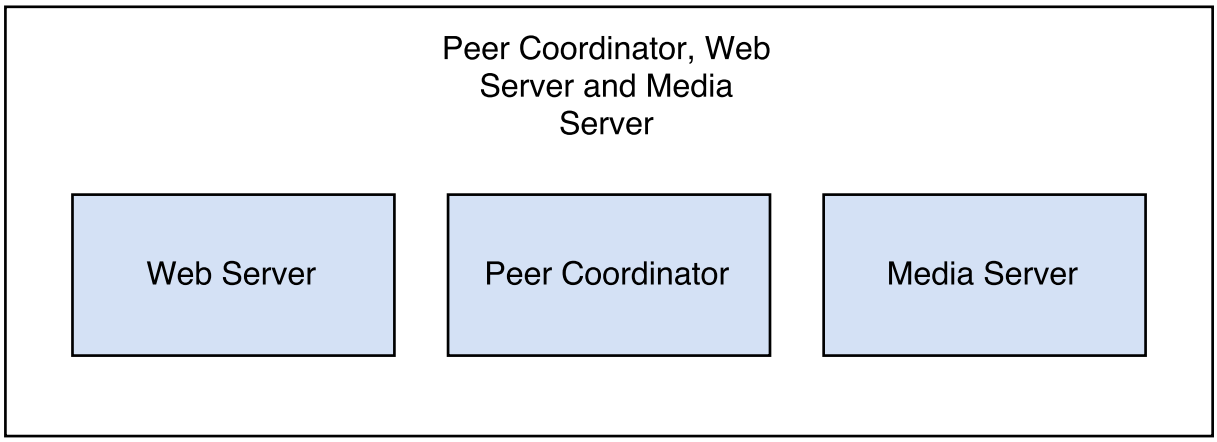


**Figure 4.5:** Peer Coordinator, Web Server and Media Server modules

The registration of a peer is processed when it reaches the web page. It sends a request to the Peer Coordinator and the Peer Coordinator will give it an ID. It will also give it, or not, authorization to collaborate with a video stream. This authorization is to prevent abuses from the peers, e.g., spam, publicity without consent, among others, and it is only granted when the peer wants to collaborate with an original video, i.e., a video that is still not being streamed.

A peer ID only lasts for the time the peer is on the web page. After leaving the web page, the peer needs to register again and is given a new ID. Also, the ID is a way to give the peer some anonymity during the process of establishing a WebRTC data channel.

The Peer Director also assumes the role of a tracker, keeping a list of peers that have a certain stream and giving that list to the peers that request it. This list needs to be updated frequently, in order to keep a good QoS for all the peers in the network, meaning that the peers need to communicate their state automatically from time to time.

This list of peers is in reality a table that associates a stream to one or more peers. To have this list of peers it is necessary to create first a table that associates the IP and ID of a peer to a name, avoiding the distribution of information like the IPs between peers. This association can be seen in 4.1.

**Table 4.1:** Table of association of IPs and names

| Name | Information |
|---|---|
| Super-Peer | [146.66.222.144, 31] |
| Peer1 | [172.55.35.75, 79] |
| Peer2 | [117.241.171.107, 8] |
| Peer3 | [169.228.99.225, 66] |

After each peer has a name associated to it, it is associated to the streams it is source of. If a peer stops being the source of a stream, it is removed and stopped being associated with said stream. This association can be observed in 4.2.

In the eventuality of a peer crashing, e.g., due to an error of the web browser or problems in the

**Table 4.2:** Table of association streams and their sources

| Stream | Peer |
|---|---|
| stream1 | [Super-Peer], [Peer2], [Peer3] |
| stream2 | [Super-Peer], [Peer1] |
| stream3 | [Super-Peer], [Peer1] |
| stream4 | [Super-Peer], [Peer1], [Peer2] |

connection, it is removed of both tables show above.

Although in the solution only the IP and the ID are used in the list, more information can be used to improve the quality of the P2P network and to deliver the streams in a more efficient way. Information like the geographical position or the peers bandwidth can be used to improve the QoS.

The Peer Coordinator also works as an intermediary for passing the SDP from a peer to another peer. This is due to the fact that, when a peer receives the list of peers with the video, it does not receive the IP address of each peer, but only the name of the peer, as seen in 4.1.

#### Web Server

The Web Server will host the web page of this service, like mentioned before. It is a simple server, with no special characteristics, that distributes HyperText Markup Language (HTML) and it directs the peers to the Peer Coordinator, for them to be registered.

It also provides a graphical interface to the peers in which the video is displayed, and in which the peers can fully interact with the system. Only peers that are accessing the web page will be able to collaborate with each other.

#### Media Server

This last module is essential to the solution. It is considered a super-peer, i.e., the peers use P2P to connect to it and it is always online, providing a lasting source of the stream. As soon as this super-peer is initiated, announces to the Peer Coordinator that it is a source of the stream. After that it waits for peers to connect to it.

Due to the fact that there are no peers, in the beginning, a certain number of peers will connect directly to it. When this number is reached, the other peers will only connect to other peers and not to the Media Server. In this solution only the first peer connects directly to it.

During the process of establishing a connection with other peer, the Media Server also gathers ICE candidates and sends its information through SDP. In the end of the process, a WebRTC data channel is established and the exchange of data, in this case the video streaming, begins.

In the eventuality of a peer collaborating with a video stream, the Media Server is the first to receive it, with the objective of helping streaming it and updating the MPD on the Web Server.

### 4.3.3 Signaling Server

The Signaling Server will provide the peers with ICE candidates in order to be possible for them to establish connections between them. ICE is used to cope with NAT and firewalls, so that, no matter where a peer is, it can always connect to another peers.

Peers receive, in the ICE candidate, information about the IP address and the port to be used for the exchange of data, among other important information. This information is then transmitted, through the SDP, to other peers interested in establishing a connection between each other.

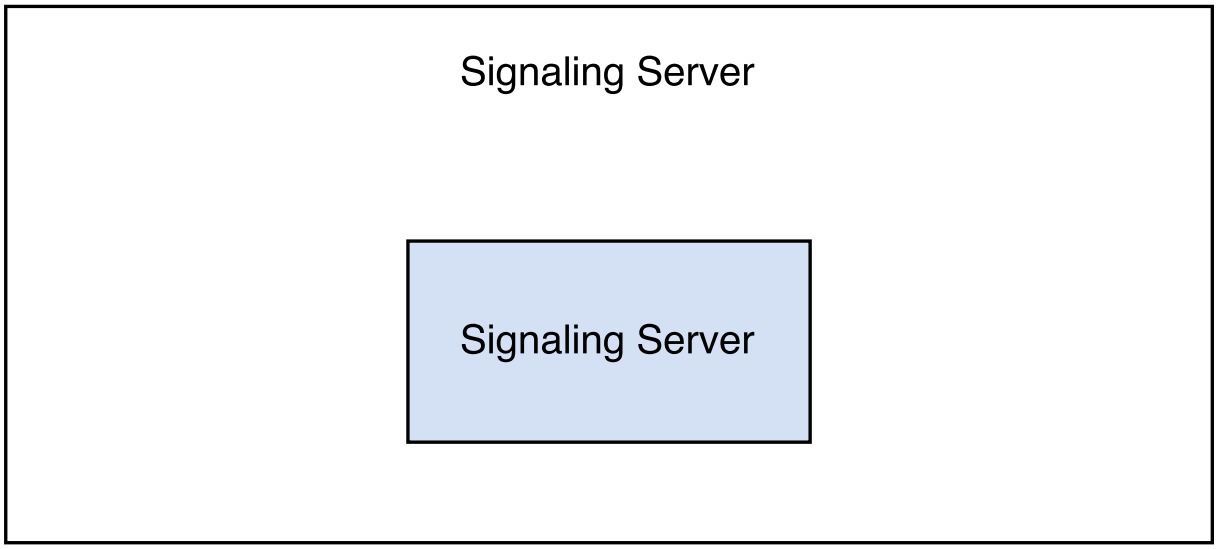


**Figure 4.6:** Signaling module

Although this Signaling Server is represented in the architecture, in the project it will be used, for simplification purposes, a public signaling server, provided by Google.

# 5

# Implementation

## Contents

5.1 Implementation Options . . . . . 55
5.2 Architecture . . . . . 56

This chapter gives an explanation of how the solution presented in this thesis was implemented. It begins by explaining the decisions taken relatively to the programming language and the development environment, before detailing how each component of the architecture was implemented.

## 5.1 Implementation Options

To implement the prototype of the solution presented in this thesis it is necessary to have a network of peers and to have all the components described in 4. Everything could be built from scratch, but that would take too much time and would lead to an effort too great. For this reason it was decided to use tools that would abstract and simplify this process.

This tools, due to the fact of being open-source, can be modified to suit the needs of the prototype, making them a good option for implementing it. One of the more important tools is the PeerJS. PeerJS is an open-source WebRTC API that wraps the implementation of WebRTC on the browser in an easy-to-use and configurable P2P connection API. PeerJS also has a connection broker, of optional use, that helps connecting the peers to each other.

REVER

As the objective of this implementation is to verify the feasibility of a real-time collaboration system in a P2P mode using video stream, the use of PeerJS to abstract the WebRTC core functions is acceptable and will save a lot of programming work. Also, the PeerJS Server allows the extension of its functionalities, making adaptable for the requirements necessary for this solution.

Although the Media Server present in this solution works as a peer, it is still a module of a normal server, so the traffic to it should be kept to a minimum. In this prototype the priority is to have P2P video stream, so, when there is a peer that can stream the video, this will be chosen, meaning that only the first peer in the P2P network will connect to the Media Server. It can be allowed to have more peers connecting to the Media Server, if found necessary or to improve the QoS.

When there is more than one peer with the video, it will be chosen randomly, the peer or peers to transmit it to the peer requesting the stream. The Media Server is excluded from this choice. Also, the first stream of video will originate in the Media Server.

The streams are identified by a torrent file that is constantly being updated and that is accessed when the user enters the website. The download of the torrent is automatically performed and all subsequent updates are also automatically. If a new video stream starts to be streamed the torrent for this stream is also downloaded automatically. As soon as the torrent is downloaded and all the processes necessary are completed, the video stream is displayed to the peer in the same webpage.

In this prototype it is assumed that all peers are good peers, i.e., they don't send virus, spam, malware, or any other type of data that can put in risk the other peers. For this reason, no verification

mechanism of the live video stream data integrity is executed.

**PeerJS**

REVER

PeerJS wraps the implementation of WebRTC on the browser in an easy-to-use and configurable P2P connection API. Nothing else but an ID is necessary for a peer to create a P2P data or media stream connection to another peer. This ID, which is a string, can be chosen by the peer itself or assigned by server, that generates a random ID.

It is necessary a server to act as a connection broker to actually connect the peers to each other. This server is provided by PeerJS, with the PeerJS Server, an open source implementation of this connection broker, written in Node.js. The PeerJS Server is only needed for connecting the peers to each other. After that, any communication happens directly between the peers.

Since PeerJS and the PeerJS Server were developed in JavaScript, the language used for the implementation of the project will be JavaScript, in order to facilitate any modifications necessary.

**Node.js**

REVER

Node.js is an open-source, cross-platform runtime environment for developing server-side applications and to build scalable network applications. It is a JavaScript runtime built on Chrome's V8 JavaScript engine, making code execution very fast.

It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, making it perfect for data-intensive real-time applications that run across distributed devices. Node.js package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Even though Node.js is not a JavaScript framework developers can write new modules in that programming language, because many of the basic modules of Node.js are written in it.

**Express.js**

REVER

Express.js is a web application framework for Node.js that provides a robust set of features for web and mobile applications, making it easy for developers to build web applications.

## 5.2 Architecture

### 5.2.1 Peer Components

**Connection API**

**Content Loader**

### 5.2.2 Peer Director, Web Server and Media Server

#### Peer Director

#### Web Server

#### Media Server

### 5.2.3 Tracker and Signaling Server

#### Tracker

#### Signaling Server

# 6

# Evaluation

## Contents

6.1 first . . . . . 61
6.2 second . . . . . 61

Pellentesque vel dui sed orci faucibus iaculis. Suspendisse dictum magna id purus tincidunt rutrum. Nulla congue. Vivamus sit amet lorem posuere dui vulputate ornare. Phasellus mattis sollicitudin ligula. Duis dignissim felis et urna. Integer adipiscing congue metus.

## 6.1 first

bla

## 6.2 second

bla

# 7

# Conclusion

## Contents

7.1 Conclusions . . . . . 65
7.2 System Limitations and Future Work . . . . . 66

Pellentesque vel dui sed orci faucibus iaculis. Suspendisse dictum magna id purus tincidunt rutrum. Nulla congue. Vivamus sit amet lorem posuere dui vulputate ornare. Phasellus mattis sollicitudin ligula. Duis dignissim felis et urna. Integer adipiscing congue metus.

## 7.1 Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis. Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus. Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dictum elit, vel lacinia enim metus eu nunc.

Proin at eros non eros adipiscing mollis. Donec semper turpis sed diam. Sed consequat ligula nec tortor. Integer eget sem. Ut vitae enim eu est vehicula gravida. Morbi ipsum ipsum, porta nec, tempor id, auctor vitae, purus. Pellentesque neque. Nulla luctus erat vitae libero. Integer nec enim. Phasellus aliquam enim et tortor. Quisque aliquet, quam elementum condimentum feugiat, tellus odio consectetuer wisi, vel nonummy sem neque in elit. Curabitur eleifend wisi iaculis ipsum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In non velit non ligula laoreet ultrices. Praesent ultricies facilisis nisl. Vivamus luctus elit sit amet mi. Phasellus pellentesque, erat eget elementum volutpat, dolor nisl porta neque, vitae sodales ipsum nibh in ligula. Maecenas mattis pulvinar diam. Curabitur sed leo.

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo, sed imperdiet lectus est ornare odio. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin consectetuer velit in dui. Phasellus wisi purus, interdum vitae, rutrum accumsan, viverra in, velit. Sed enim risus, congue

non, tristique in, commodo eu, metus. Aenean tortor mi, imperdiet id, gravida eu, posuere eu, felis. Mauris sollicitudin, turpis in hendrerit sodales, lectus ipsum pellentesque ligula, sit amet scelerisque urna nibh ut arcu. Aliquam in lacus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nulla placerat aliquam wisi. Mauris viverra odio. Quisque fermentum pulvinar odio. Proin posuere est vitae ligula. Etiam euismod. Cras a eros.

Nunc auctor bibendum eros. Maecenas porta accumsan mauris. Etiam enim enim, elementum sed, bibendum quis, rhoncus non, metus. Fusce neque dolor, adipiscing sed, consectetuer et, lacinia sit amet, quam.

## 7.2 System Limitations and Future Work

Aliquam aliquet, est a ullamcorper condimentum, tellus nulla fringilla elit, a iaculis nulla turpis sed wisi. Fusce volutpat. Etiam sodales ante id nunc. Proin ornare dignissim lacus. Nunc porttitor nunc a sem. Sed sollicitudin velit eu magna. Aliquam erat volutpat. Vivamus ornare est non wisi. Proin vel quam. Vivamus egestas. Nunc tempor diam vehicula mauris. Nullam sapien eros, facilisis vel, eleifend non, auctor dapibus, pede.

# Bibliography


[1] W. Howe, “A Brief History of the Internet.” [Online]. Available: http://www.walthowe.com/navnet/history.html

[2] I. H. of Fame, “Internet Hall of Fame Innovator - Tim Berners-Lee.” [Online]. Available: http://internethalloffame.org/inductees/tim-berners-lee

[3] R. Schollmeier, “A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications,” in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, 2001, pp. 101–102.

[4] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, “A Survey and Comparison of Peer-to-Peer Overlay Network Schemes,” *IEEE Communications Surveys and Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[5] The Government of the Hong Kong Special Administrative Region, “Peer-to-Peer Network,” p. 14, 2008. [Online]. Available: http://www.infosec.gov.hk/english/technical/articles{_}peer.html

[6] T. Aspelund, “Peer-to-Peer Networks: Sharing Between Peers,” in *Network and System Administration Research Surveys vol. 1*, F. E. Sandnes, K. Begnum, and M. Burgess, Eds. Akershus University College of Applied Sciences, 2004, pp. 27–35. [Online]. Available: http://www.cs.hioa.no/{~}frodes/rm/

[7] A. F. I. Ibrahimy, F. Anwar, M. I. Ibrahimy, and M. R. Islam, “Two Dimensional Array Dased Overlay Network for Reducing Delay of Peer-to-Peer Live Video Streaming,” *Proceedings - 5th International Conference on Computer and Communication Engineering: Emerging Technologies via Comp-Unication Convergence, ICCCE 2014*, vol. 5, pp. 185–188, 2015.

[8] N. Lake, “BitTorrent Technology - How and Why It Works,” p. 5, 2005. [Online]. Available: http://www4.ncsu.edu/{~}kksivara/sfwr4c03/projects/4c03projects/NELake-Project.pdf

[9] Google, “WebRTC,” 2012. [Online]. Available: https://webrtc.org/

[10] H. Alvestrand, “Overview: Real Time Protocols for Browser-based Applications,” Working Draft, IETF Secretariat, Internet-Draft draft-ietf-rtcweb-overview-14, 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-rtcweb-overview-14.txt

[11] A. B. Johnston and D. C. Burnett, “WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web,” in *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*, 3rd ed. Digital Codex LLC, 2014, vol. 1, pp. 1–15.

[12] B. Sredojev, D. Samardzija, and D. Posarac, “WebRTC Technology Overview and Signaling Solution Design and Implementation,” in *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*, 2015, pp. 1006–1009.

[13] X. Chen, “Unified Communication and WebRTC,” Ph.D. dissertation, Norwegian University of Science and Technology, 2014.

[14] H. Alvestrand, “Transports for WebRTC,” Working Draft, IETF Secretariat, Internet-Draft draft-ietf-rtcweb-transports-10, 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-rtcweb-transports-10.txt

[15] J. Rosenberg and H. Schulzrinne, “An Offer/Answer Model with Session Description Protocol (SDP),” Internet Requests for Comments, RFC Editor, RFC 3264, 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3264.txt

[16] Dynamicsoft, J. Rosenberg, H. Schulzrinne, ICIR, C. U., G. Camarillo, Ericsson, A. Johnston, WorldCom, J. Peterson, Neustar, R. Sparks, Dynamicsoft, M. Handley, AT&T, and E. Schooler, “SIP: Session Initiation Protocol,” *Internet Engineering Task Force*, pp. 1–269, 2002.

[17] P. Saint-Andre and Cisco, “Extensible Messaging and Presence Protocol (XMPP): Core,” *Internet Engineering Task Force (IETF)*, pp. 1–211, 2011.

[18] I. Fette, I. Google, A. Melnikov, and I. Ltd., “The WebSocket Protocol,” *Internet Engineering Task Force*, pp. 1–71, 2011.

[19] I. Hickson, “The WebSocket API.” [Online]. Available: https://www.w3.org/TR/2012/CR-websockets-20120920/

[20] I. B. Castillo, J. M. Villegas, Versatica, V. Pascual, and Quobis, “The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP),” pp. 1–25, 2014.

[21] J. Miller, “Jabber.” [Online]. Available: http://www.jabber.org/

[22] T. Muldowney, M. Miller, R. Eatmon, and P. Saint-Andre, “SI File Transfer,” 2004. [Online]. Available: http://xmpp.org/extensions/xep-0096.html

[23] P. Saint-andre, “XEP-0045 : Multi-User Chat,” 2016.

[24] I. Paterson, P. Saint-Andre, L. Stout, and W. Tilanus, “XEP-0206: XMPP over BOSH,” *XMPP Standards Foundation*, 2014. [Online]. Available: http://xmpp.org/extensions/xep-0206.pdf

[25] E. L. Stout, J. Moffitt, Mozilla, E. Cestari, C. Industries, and &yet, “An XMPP Sub-protocol for WebSocket,” pp. 1–17, 2014.

[26] R. Eatmon, J. Hildebrand, J. Miller, T. Muldowney, and P. Saint-Andre, “XEP-0004: Data Forms,” 2007. [Online]. Available: http://xmpp.org/extensions/xep-0004.html

[27] J. Moffitt, “Strophe.js.” [Online]. Available: http://strophe.im/strophejs/

[28] P. Chainho, K. Haensge, S. Druesedow, and M. Maruschke, “Signalling-On-the-fly: SigOfly: WebRTC Interoperability Tested in Contradictive Deployment Scenarios,” *2015 18th International Conference on Intelligence in Next Generation Networks, ICIN 2015*, pp. 1–8, 2015.

[29] Y. Zhang and N. Zong, “Problem Statement and Requirements of the Peer-to-Peer Streaming Protocol (PPSP),” Internet Requests for Comments, RFC Editor, RFC 6972, 2013.

[30] Peer5, “Peer5.” [Online]. Available: https://www.peer5.com

[31] A. Bakker, R. Petrocco, and V. Grishchenko, “Peer-to-Peer Streaming Peer Protocol (PPSPP),” Internet Requests for Comments, RFC Editor, RFC 7574, 2015.

[32] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, “Low Extra Delay Background Transport (LEDBAT),” Internet Requests for Comments, RFC Editor, RFC 6817, 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6817.txt

[33] A. Bakker, “Merkle Hash Torrent Extension,” 2009. [Online]. Available: http://bittorrent.org/beps/bep{_}0030.html

[34] V. J. H. Schulzrinne, S. Casner, R. Frederick, “RTP : A Transport Protocol for Real-Time Applications,” *The Internet Society, RFC 3550*, pp. 1–89, 2003.

[35] I. Sodagar, “The MPEG-dash standard for multimedia streaming over the internet,” *IEEE Multimedia*, vol. 18, no. 4, pp. 62–67, 2011.

[36] I. E. C. Jtc, R. Call, V. Coding, T. Approved, and T. Cfp, “AAC Implementation Guidelines for DASH,” *Test*, pp. 9–12, 2011. [Online]. Available: http://mpeg.chiariglione.org/sites/default/files/files/standards/docs/w15072.zip

[37] R. Pantos, W. May, and A. Inc., “HTTP Live Streaming,” pp. 1–24, 2011.

[38] A. Zambelli, “IIS Smooth Streaming Technical Overview,” *Microsoft Corporation*, no. March, 2009.

[39] D. Streaming, “HTTP Dynamic Streaming,” *Access*. [Online]. Available: http://www.adobe.com/products/hds-dynamic-streaming.html

[40] J. K. Nurminen, A. J. R. Meyn, E. Jalonen, Y. Raivio, and R. G. Marrero, “P2P Media Streaming with HTML5 and WebRTC,” in *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, 2013, pp. 63–64.

[41] P. Segeč, P. Palúch, J. Papán, and M. Kubina, “The Integration of WebRTC and SIP: Way of Enhancing Real-Time, Interactive Multimedia Communication,” in *Emerging eLearning Technologies and Applications (ICETA), 2014 IEEE 12th International Conference on*, 2014, pp. 437–442.

[42] F. Aboukhadijeh, “Webtorrent,” 2014. [Online]. Available: https://github.com/feross/webtorrent

[43] S. Zhao, Z. Li, and D. Medhi, “Low Delay MPEG-DASH Streaming over the WebRTC Data Channel,” *2016 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pp. 1–6, 2016.

# Code of Project

# B

# A Large Table