Iciar Puigpelat Barrado

Real-Time Communication applied to Orchestration Graphs

Bachelor Thesis

Computer-Human Interaction in Learning and Instruction
Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland

Supervision

First Supervisor
Haklev, Stian
Second Supervisor
Fornes de Juan, Jorge

January 2018

Abstract

Computer Supported Collaborative Learning (CSCL) activities have become an arising field on the educational sector, aiming to enrich student's cognitive state through rich verbal interactions which help students to internalize their knowledge. At the same time, education is shifting to on-line classes, which allow students to choose from a variety of courses without any geographical restriction. Combining on-line interactions and collaborative learning has become one of the main challenges for CSCL softwares.
FROG is an educational software that allows teachers to build easy and understandable models of pedagogical scenarios, called Orchestration Graphs. These models allow the extrapolation from few students up to big scale. FROG focuses on developing activities which engage collaborative learning in real-time, but the usage of this technologies is limited to face-to-face interactions.

Therefore, in this project, we have focused on the development of a video-conferencing system which allows students working on collaborative learning activities to achieve richer interactions. To build the web-based video-conferencing system we have used the WebRTC technology, that allows two browsers to exchange data in real-time, in our case, we have focused on audio and video streaming.
The development of the project has marked a before and after of the usage of this software, FROG, for on-line classes. Allowing up to 6 users to be interconnected through video and audio while they collaborate through different activities. In the following thesis, we will discuss the decisions taken in order to achieve this result.

Keywords: WebRTC; Real-Time; Collaborative Learning; Orchestration Graphs.

Resum

L'aprenentatge col·laboratiu amb recolzament informàtic (CSCL en anglès) ha esdevingut un camp emergent en el sector educatiu, el qual busca enriquir l'estat cognitiu de l'estudiant a través de riques interaccions verbals que ajuden l'estudiant a interioritzar els seus coneixements. Al mateix temps, l'educació s'està traslladant a classes en línia, que permeten als estudiants triar entre una gran varietat de cursos sense cap restricció geogràfica. La combinació d'interaccions en línia i aprenentatge col·laboratiu s'ha convertit en un dels principals reptes dels programes CSCL.
FROG és un programari educatiu que permet als professors construir models senzills i comprensibles d'escenaris pedagògics, anomenats Gràfics d'Orquestració. Aquests models permeten l'extrapolació de classes amb pocs alumnes fins a escenaris a gran escala. FROG es centra en el desenvolupament d'activitats que involucren l'aprenentatge col·laboratiu en temps real, però l'ús d'aquestes tecnologies es limita a les interaccions cara a cara.

Per això, en aquest projecte ens hem centrat en el desenvolupament d'un sistema de videoconferència que permeti als estudiants treballar en activitats d'aprenentatge col·laboratiu per aconseguir interaccions més satisfactòries. Per construir el sistema de videoconferència basat en web hem utilitzat la tecnologia WebRTC, que permet l'intercanvi de dades en temps real entre dos navegadors, en el nostre cas, ens hem centrat en la transmissió d'àudio i vídeo.
El desenvolupament del projecte ha marcat un abans i un després de l'ús d'aquest programari, FROG, per a classes en línia. Permetren que fins a 6 usuaris s'interconnectin a través de vídeo i àudio mentre col·laboren ens de diferents activitats. En la següent tesi es discutiran les decisions preses per aconseguir aquest resultat.

Paraules clau: WebRTC; Temps Real; Aprenentatge Col·laboratiu; Grafiques d'Orquestració.

Resumen

El aprendizaje colaborativo con apoyo informático (CSCL en inglés) se ha convertido en un campo emergente en el sector educativo, el cual busca enriquecer el estado cognitivo del estudiante a través de ricas interacciones verbales que ayudan al estudiante a interiorizar sus conocimientos. Al mismo tiempo, la educación se está trasladando a clases en línea, que permiten a los estudiantes elegir entre una gran variedad de cursos sin restricción geográfica. La combinación de interacciones en línea y el aprendizaje colaborativo se ha convertido en uno de los principales retos de los programas CSCL.
FROG es un software educativo que permite a los profesores construir modelos sencillos y comprensibles de escenarios pedagógicos, llamados Gráficos de Orquestación. Estos modelos permiten la extrapolación de clases con pocos alumnos hasta escenarios a gran escala. FROG se centra en el desarrollo de actividades que involucran el aprendizaje colaborativo en tiempo real, pero el uso de estas tecnologías se limita a las interacciones cara a cara.

Por ello, en este proyecto nos hemos centrado en el desarrollo de un sistema de videoconferencia que permite a los estudiantes trabajar en actividades de aprendizaje colaborativo para conseguir interacciones más satisfactorias. Para construir el sistema de videoconferencia basado en web hemos utilizado la tecnología WebRTC, que permite el intercambio de datos en tiempo real entre dos navegadores, en nuestro caso, nos hemos centrado en la transmisión de audio y vídeo.
El desarrollo del proyecto ha marcado un antes y un después del uso de este software, FROG, para clases en línea. Permitiendo que hasta 6 usuarios se interconecten a través de vídeo y audio mientras colaboran en diferentes actividades. En la siguiente tesis se discutirán las decisiones tomadas para conseguir este resultado.

Palabras clave: WebRTC; Tiempo Real; Aprendizaje colaborativo; Gráficas de Orquestación.

Acknowledgements

Today, I remind myself: "One chapter ends, yet another begins". After long but rewarding years, my bachelor chapter is arriving at its last pages. This chapter has signified growth, not only regarding knowledge state but also on a personal level. Although my years of learning have just begun, I will always remember the impact this period of my life has left on me.

I would like to thank all people who have made it possible. From the teachers who taught me to fall in love with programming, to the EDpuzzle company who showed me combining two of my passions, coding and education, could be more than just a hobby. A special thanks is directed to my family and loved ones who have supported me in times of defeat, and they are a pillar of who I am today.

Finally, I would like to thank my supervisor, Stian, for his valuable guidance and patience. In addition to all the EPFL stuff and members, for creating such a welcoming community and allowing me to live a never-forgetting experience in Switzerland.

Thank you so much for all the support,

Iciar Puigpelat Barrado
Morges, January 2018.

Contents

Chapter 1

Introduction

This chapter provides an overview of the evolution of video conferencing, followed by its growing relationship with the educational sector. After a brief introduction of the background technology and the research motivations, the goals, and approach of the thesis will be described. Concluded by an outline of the discussion of the thesis.

1.1 Background

The continuous evolution of technology has changed all the aspects of our everyday life, from how we communicate to how we teach and learn. Communication has shifted in the last centuries; from post letters and telephones, to instant messages and video conferences (VC). Currently, there are thousands of applications that allow us to communicate with people across the globe instantly. Some of this applications focus on video conferencing, in order to allow two or more users to transmit video and audio streams in real-time. The necessity of simulating face-to-face communication has exploded as companies are expanding internationally. But at the same time, it is crucial for those companies to keep communication close and straightforward as possible. Therefore it is necessary to assure this connection happens in real-time.
Real-Time Communication for the web has evolved with the introduction of WebRTC, primarly focused on video conference. WebRTC, which stands for Web Real-Time Communication, is a fairly new technology which provides the browser with the capacity to establish a peer-to-peer (P2P) communication with another browser in order to exchange audio, video and data files in real-time without the need of installing an external software or plugin. This technology has arisen to the point it has become a common technology being used in our everyday lives.

Another developing sector in the past decade is education, which has focused on absorbing the benefits of the new emerging technologies. Many modern software are expanding the way people learn and teach, from including self-paced lessons to interactive learning programs.
Creating and developing software which generates an adjustable and self-made lesson using technology has been one of the main focus of the Computer-Human Interaction in Learning Instruction (CHILI) team for a few years now. The Fabricating and Running Orchestration Graphs (FROG) software is one of the results of all those efforts. FROG is an open-source software based on a modelling language for pedagogical scenarios called Orchestration Graphs. This modelling language based on geometrical graphs helps to scale up to 20.000 students, rich learning activities, often designed for small classes [1]. FROG characterizes itself as a flexible software capable of fabricating orchestration graphs as well as running the pedagogical exercises modelled on the graph in the classroom or at home using a browser application. Teachers create a model of a lesson, which can be structured in classroom, group or individual activities. The majority of the activities developed on the FROG environment focus on collaborative learning, an educational approach which encourages to learn by working together with other learners.

Combining those two fields; learning and communication, to promote collaboration has become ubiquitous in the educational sector. It has mainly been focused on real-time communication to engage participation and teamwork.

1.2 Project motivations

Currently, different projects and ideas are being developed on FROG, which have flourished by the needs of users as well as new ideas thought by the developers. One of those ideas is introducing a real-time voice or audio communication into group and classroom activities, in order to enhance the collaboration between students that are working collaboratively. This idea will be the main focus of this project.

Prior to this thesis, students working on collaborative activities had no way to communicate to each other other than verbal face-to-face communication or via an existing FROG chat activity. This limited the ability to use FROG's collaborative activities on classes that were not physically present or with a large number of students, due to a slow interaction between users via chat messages. In order to communicate with each other in real-time students had to use a third party software or an external plugin. Therefore, introducing video conferencing to FROG would ease the collaboration between students.

1.3 Statement of purpose

The main research question for this thesis is:

How can video conferencing be included and developed in the FROG software?

This principal question can be itemized into different sub-questions:

Q1 What is WebRTC?

Q2 Is WebRTC the best technology for VC in web browsers?

Q3 How does WebRTC establish the P2P connection?

Q4 Which P2P network topology should be used?

Q5 How will the signalling process be implemented?

Q6 Is the usage of a STUN and TURN needed?

Q7 How will the activity code be structured?

1.4 Requirements and specifications

The project objective is the creation of an activity for the FROG software which will support video conferences between multiple users. We decided to use a leading edge technology: WebRTC. This technology allows the creation of real-time connections between browsers without the need of an external plugin, to exchange information. For this project, we will focus only on video and audio streaming.
This activity needs to integrate with FROG in a user-friendly interface as well as in the most efficient, light-weight and fast way possible. Other requirements for the WebRTC in FROG project are the seamless functionality in the standard browsers as well as the capability of being used in restricted network environments.

1.5 Approach

The scope, needs and future uses of WebRTC in the FROG program were not entirely defined, therefore it was crucial to structure and define the project to prevent a significant amount of time lost. The beginning was characterized by a substantial amount of research on WebRTC technologies, especially on its limits and issues. The conducted research on the topic and the amount of time dedicated to organize and design the structure has played a significant role and has benefited the development of the implementation. The work plan was divided into five tasks which are briefly described below. For further knowledge of the project organization, refer to Figure 1.1 and previous work provided to the university, such as Project Design Review and Critical Design Review.

1. *Project organization and first approach:* Conducting extensive research of the WebRTC technology its limits and possible issues, as well as, developing a prototype of peer-to-peer WebRTC communication between two devices.

2. *WebRTC multi-peer video-call:* Include video and audio to the prototype version as well as upgrading it to be able to bear more than two peers. Developing it first in a mesh configuration and later in a server-client configuration.

3. *TURN and STUN servers:* The use of FROG in educational and company environments may cause potential issues with WebRTC. Therefore it is necessary to investigate and develop a TURN and STUN server in case the Internet connection does not allow IP detection or media transmission via P2P.

4. *Integration with FROG:* Aggregate the extended prototype version to the running FROG environment and adjust it to the possible necessities.

5. *Documenting:* Be able to capture all the information gained and as well as the issues and future achievements to be accomplished in the WebRTC implementation for FROG.

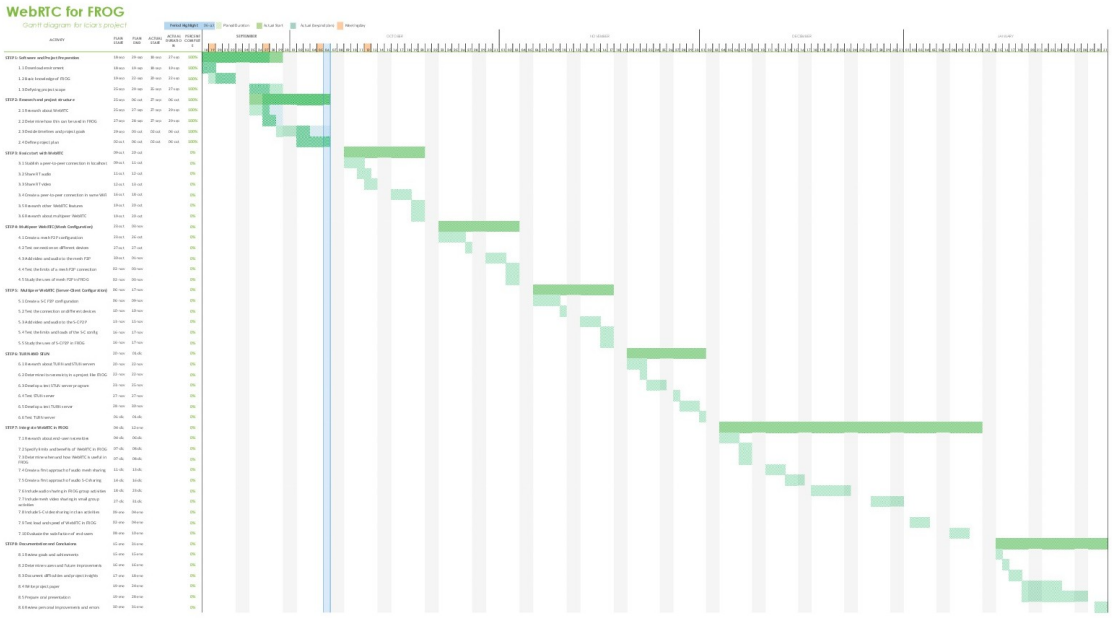


Figure 1.1: Original Gantt Diagram for project 'WebRTC for FROG'

1.6 Outline

The thesis is structured in chapters. An in-depth analysis of the WebRTC technology is given in chapter 2, describing the technical background needed to understand the project development. After the introduction to the technology used, in chapter 3, related work is studied regarding the uses of WebRTC for video conferencing especially in educational environments. The description of the project development, such as issues encountered completed by the solutions provided, and technical decisions are commented in chapter 4. A general overview of the final result and a discussion of the product is provided in chapter 5. And this thesis concludes with chapter 6 summarizing the conclusions, future work and recommendations.

Chapter 2

Web Real-Time Communication

This chapter provides a basic overview of the WebRTC technology. In section 2.1, a short introduction to the principals is given, followed by an explanation of signalling method needed to establish the connection, in section 2.2. Section 2.3 answers to the question why STUN and TURN servers are required. The different real-time topologies for multi-party conferences are explained in section 2.4. Lastly, the chapter concludes in section 2.5 with a short explanation of browser compatibility, open-source libraries and the Application Programming Interface (API) the browsers provide.

2.1 Introduction

WebRTC can be defined as an arising technology which provides Real-Time Communication (RTC) functionalities to browsers. It uses a conjunction of protocols that allow the exchange of information without the use of any external plugins. WebRTC which stands for Web-based Real-Time Communication provides web applications with the capability of exchanging audio and video streaming as well as data transfer in real time. Different to most browser communications based on a server-client (S/C) configuration, which the information flows through a shared server, this technology connects the browsers directly with each-other following a peer-to-peer (P2P) configuration.

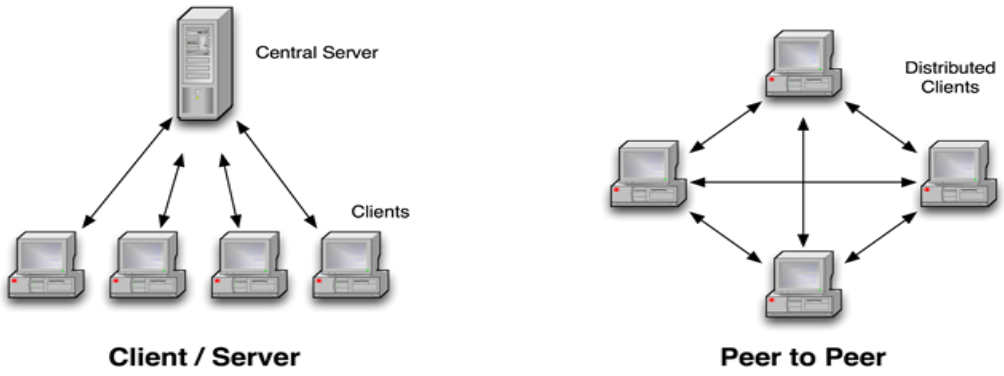


Figure 2.1: Difference between server-client communication and peer-to-peer communication

Google introduced this technology in mid-2011 with an open-source project which was later included in Google Chrome late 2012. As it is a relatively new technology, it is yet being standardized by World Wide Web Consortium (W3C) [2] in collaboration with the Internet Engineering Task Force (IETF) [3, 4]. W3C created a JavaScript API to ease the development of these WebRTC applications, and IETF focuses on defining the WebRTC protocols.

Prior to the introduction of this technology, video-conference browser applications used external or internal plugins, such as Adobe Flash, in order to exchange the user's web-camera stream [5]. The main flaw of this approach is the lack of standardization and the need of the installation of a specific plugin for the web-application. Also, it consumed a considerable amount of battery and created some security flaws [6].

In this document, we will focus on the transfer of video and audio streams using the JavaScript API provided by W3C. The underlying WebRTC protocol will be mentioned, but not in depth due to the small role it plays in the development of the project.

2.2 Signaling Process

As mentioned in the previous section, WebRTC establishes a P2P connection for data transmission but to create this link a signalling server is required. The central server transfers the necessary information for the establishment of the connection between peers. Therefore it is a shared external point. This signalling service is not defined in the WebRTC specs, to prevent the loss of flexibility in the usage of WebRTC for each application. But is usually treat as a bridge where peers exchange control messages (ready, offer, answer, disconnect), network data (IPs, interfaces, ports), media information (codec specifications, etc.) and error messages.

The principal objects needed for the establishment of a WebRTC peer connection are: RTCPeerConnection object, the Session Description Protocol (SDP) and the Interactive Connectivity Establishment (ICE) candidates:

- The RTCPeerConnection object is the component that creates the communication of data between the peers. Figure 2.2 describes a WebRTC architecture diagram on which we can see the RTCPeerConnection object encapsulates a significant complex structure that allows real-time communication and manages things as: packet loss, codecs, echo cancellation, bandwidth adaptivity, jitter buffering, gain control, noise reduction and suppression [7].

- The SDP is used to describe the streaming media parameters of the connection [8], specifically session creation, invitation and answer. SDP does not support media itself but is used to negotiate media constraints (audio and video), the codec and resolutions used, the decoding, network information and other parameters.

- The ICE framework is in charge of finding the networks interfaces and ports needed for the Peer Connection establishment [9, 10]. It is used to bypass NATs and firewalls, in order to connect peers using the best path available. The process for choosing the route will be explained in section 2.3.

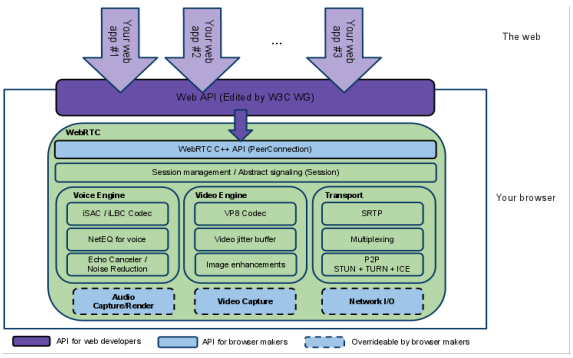


Figure 2.2: WebRTC architecture. Image from `webrtc.org/architecture`

The signalling process consists of the exchange of those SDPs between peers. Each user creates an RTCPeerConnection object which contains its own SDP, and then they exchanged the information with the other peer. The SDP information can be divided into two main sections: the network information and the media information. The obtention and exchange of these segments are carried in parallel. For an easier manipulation of the RTC peer connections, the SDP logic is encapsulated in the JavaScript Session Establishment Protocol (JSEP) [11].
The network information will be provided by the ICE framework, which will find the best candidate possible for the establishment. With the usage of the Trickle ICE, a procedure that allows the exchange of the network information as it becomes available, the signalling process has become more efficient. Without Trickle ICE it would be necessary to collect the data from all candidates before deciding which one will be used. Since the candidate information is requested to external services, this would bottleneck the establishment of the connection.

The Figure 2.3 represents how the media information is exchanged between two peers, using a signalling server. It represents the transfer of the SDP information, such as media information, resolution and codec, between Alice and Bob. Below a brief description of the information contained in the figure is given, The explanation describes the process followed to deliver the required information to the other peer, such as methods used and objects created:

1. Alice calls the method *getUserMedia()*, passing a parameter of which media she wants to obtain: video and/or audio. This method returns a MediaStream object.
2. She creates a RTCPeerConnection object to which she adds her MediaStream.
3. Calls the method *createOffer()* from the RTCPeerConnection object.
4. Once the offer is created, Alice calls *setLocalDescription()*, which will add Alice's SDP information into the localDescription variable of the RTCPeerConnection object.
5. Alice sends her offer to Bob using the signalling server.

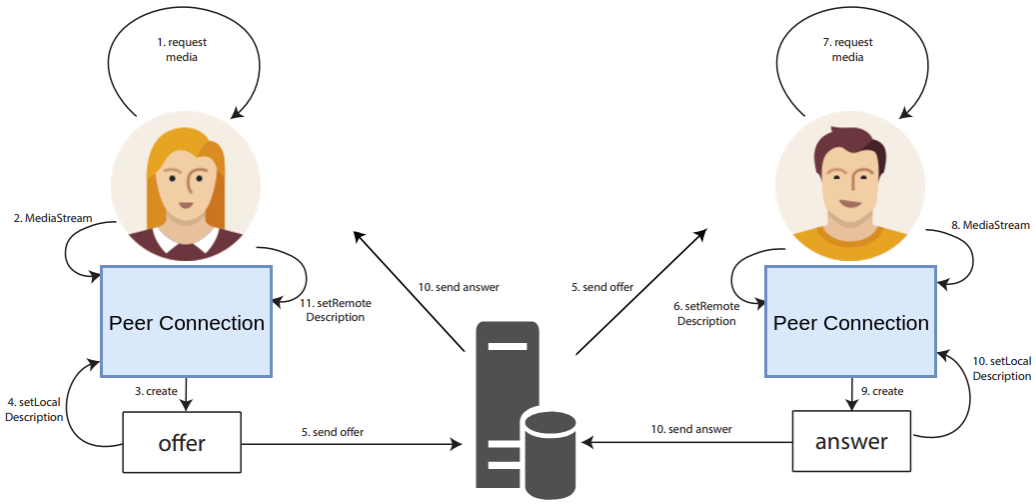


Figure 2.3: SDP exchange process between peers

6. Bob receives the offer and gets his MediaStream object.
7. He creates a RTCPeerConnection object. He adds his MediaStream and Alice's SDP information, by calling *setRemoteDescription()* method.
8. Bob calls the method *createAnswer()* from the RTCPeerConnection object.
9. Once the answer is created, Bob sets his local SDP and sends the answer to Alice.
10. When Alice receives the answer, sets Bob SDP as the remote Description of her RTCPeerConnection.

Once the media information exchange process has finished the connection is established. Although, it is required to obtain the network information using the ICE framework in parallel to this SDP transfer, in order to create a real-time communication.

2.3 STUN and TURN servers

Section 2.2 explained that it is necessary to exchange information; media specifications and network interfaces, in order to establish a P2P connection. The ICE Framework is responsible of learning the network interfaces and ports available of each user, and deciding the best route to establish the connection. If every peer had a unique address, they could just exchange it and communicate freely.

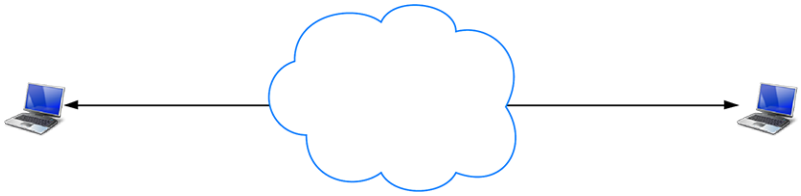


Figure 2.4: Peer to peer connection without NATs and firewalls [7]

But most users are behind Network Translation Layers (NAT), protection software that prevent connection through specific ports or protocols, and many are behind proxies and corporate firewalls with strict security rules. Therefore a direct contact is not usually possible.

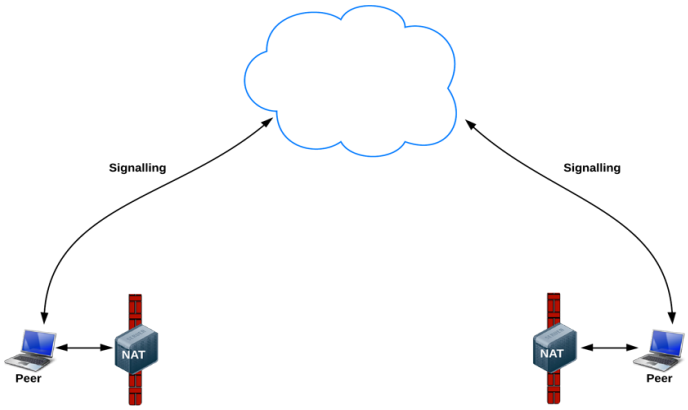


Figure 2.5: Peer to peer connection in the real world [7]

WebRTC uses the ICE framework to bypass the restrictions and find the best way to create a connection between peers. There are three stages in order to determine the most efficient way to

establish the connection; the messages exchanged in these stages are called ICE Candidates:
First, the ICE tries to contact the other peer directly via UDP using the host address obtained from the device OS and the network card. Since in most cases users are behind NATs, this candidate fails.
Secondly, the ICE framework will obtain a public address and ports available using a Session Traversal Utilities for NAT (STUN) server [12] and then try connecting to the peer using the TCP protocol: first with HTTP and then HTTPS. This method will usually fail if the user is behind a NAT transversal and a corporate firewall.
Third, the traffic will be routed to an intermediate server, the Traversal Using Relays around NAT (TURN) [13] server, which will act as a bridge between peers.

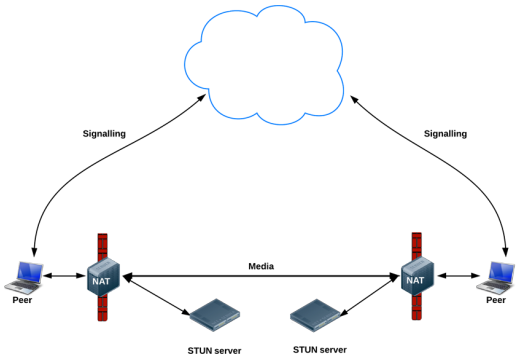


Figure 2.6: STUN servers [7]

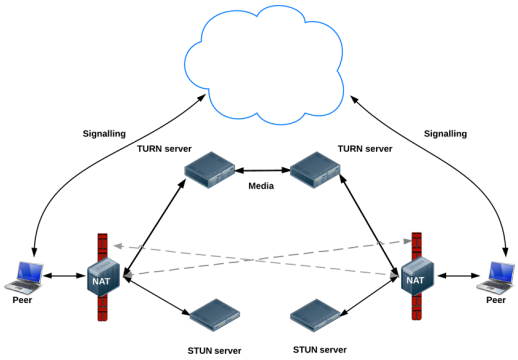


Figure 2.7: TURN servers [7]

In conclusion, a STUN server is needed to learn an external address of the peer and a TURN server to relay traffic if the peer-to-peer connection fails. All the candidates are gathered and exchanged through the shared server at the same time. Then, the ICE framework decides the best route possible to establish the P2P connection, following the order explained previously. The ICE framework is also responsible for maintaining the link alive and switching if a better route becomes available.
The application specifies the STUN and TURN using URLs, which are passed to the RTCPeerConnection object. There are public STUN server credentials which can be used, but TURN servers are expensive to maintain since they route big amounts of data. Up to a 86% of the WebRTC connections are successful with the usage of a simple STUN server [14], but it is vital to assure the well-functioning of WebRTC even on strict networks, such as educational environments.

2.4 Network topologies

In previous sections, we have focused on one-to-one connections. Sometimes video conferences comprise multiple users, from 3 peers to one peer's video being streamed to hundreds of users. Therefore it is important to assure that WebRTC can support these requirements. To do so there are different network topologies used to arrange the users at multi-party's conferences. We can define different architectures [15] depending on the conditions of our application, these revolve around two key points:

- **Server/Client vs peer-to-peer:** Server/Client (S/C) is defined as a centralized structure where all information flows through a server, therefore the server establishes a peer to peer connection with each user and it is the responsible for delivering all the other peer's information to each client. In the other hand, the peer-to-peer (P2P) configuration consists on the creation of a P2P connection between each client. The main differences are the number of resources needed and the capability of monitoring and extracting data for each approach. S/C demands the usage of a powerful server but allows more control over the data flow.

Opposite to P2P connections which do not require a powerful server, thus it is only going to be used for signalling but prevents the gain of any statistics of the data flow.

- **Mixing vs routing:** Mixing and routing structures are used on S/C architectures depending on the way data is shared to each peer, but both approaches allow the server to monitor the shared data. The mixing structure receives each peer's data flow through the server, who combines it and delivers it to each peer. On the contrary, the routing structure only re-directs the information received from one peer to the other, in order to do so it establishes as many connections with each peer as clients are connected. Again, the main differences are the amount of resources needed and the ability to modify the data for each specific client.

Combining these key-points, we obtain three different and principal structures; P2P with routing, S/C with routing and S/C with mixing, which will be described below.

Mesh structure: P2P with routing

The simplest and more popular architecture among WebRTC applications [16]. It creates multiple one-to-one peer connections between all users.

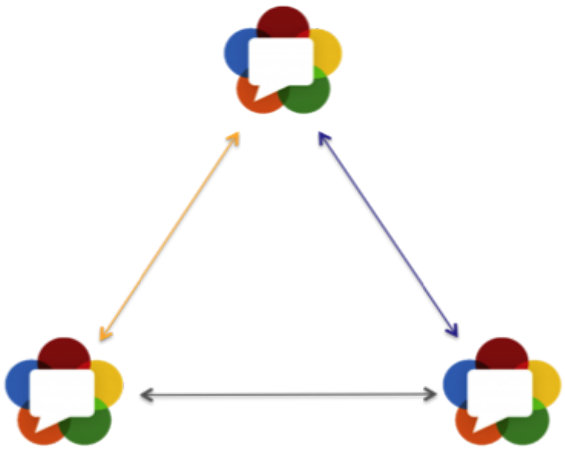


Figure 2.8: Mesh topology diagram [16]

The main advantages of this approach are the lack of need of a powerful server since it is only going to be used as a signalling bridge, and the lowest delay possible with an independent rate adaptation for each peer. The throwbacks are the amount of up-link bandwidth needed to send the same media to all the different clients as well as CPU power to encode the same media multiple times. Therefore this approach it's considered sustainable only for a few users. The exact number depends on the capabilities of the computer and network used, but it is estimated to work reliably with 5 or 6 users [17].

Mixer / MCU (Multipoint Control Unit): S/C with mixing

The MCU approach creates only one peer-connection between each user and the server. The central point is in charge of decoding each stream integrating it together and then delivering it to each participant.
The benefits of the mixer approach are the reduced amount of up-link bandwidth and CPU needed compared to the previous solution, as well as the liberation of the logic in the client side. Also, it allows the creation of different outputs for each client with a full bit rate adaption because the

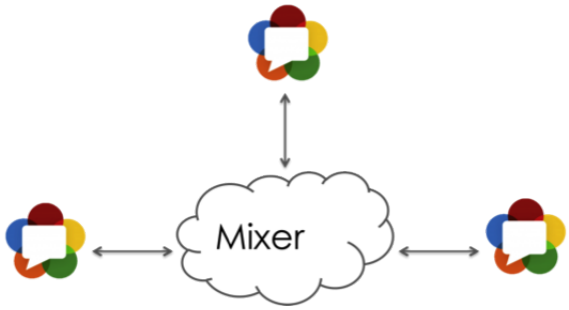


Figure 2.9: MCU topology diagram [16]

mixer can generate qualities for each receiver. Although, all those benefits cost a certain delay and loss quality due to the mixing process as well as the decoding and re-coding of each stream. An MCU network topology requires having a robust central point to process many streams. This approach is considered useful for a multi-party conference with many clients when a powerful server is available, and it is required to handle the data flow.

Router / SFU (Selective Forward Unit)

The SFU solution is becoming one of the more popular architectures for those applications who need to evaluate and keep control of the data flow, also want to reduce the amount of CPU and up-link bandwidth for the client, but do not want to use a powerful central point. This architecture arranges as many connections with one peer as clients are connected but limits the up-link connection to only one. This SFU server inspects and forwards the packages without having to encode and decode the media, which is expensive.

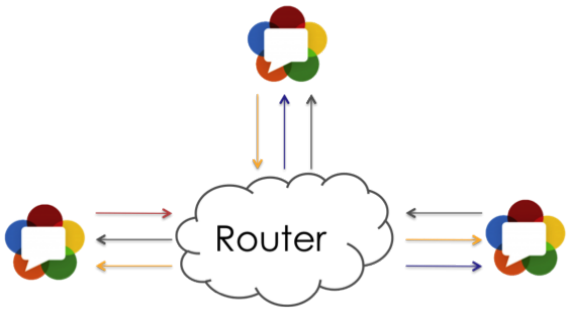


Figure 2.10: SFU topology diagram [16]

The main assets are the reduction of delays and the improvement of the stream quality compared to the MCU approach. And the reduction of the CPU used on the client side in comparison to the mesh solution. The drawback is the need of a central point powerful enough to create many connections.

In conclusion, we determine that P2P structures are useful for a reduced number of users since its delay is minimum, but it is not a scalable topology. For video conferences with many peers, a S/C structure is recommended, due to the reduced amount of CPU and up-link bandwidth required. Depending on the application's requirements and the availability of a powerful media server it is recommended to use an MCU approach or an SFU architecture.

2.5 Browser Compatibility

The WebRTC technology is still expanding; although it has come a long way since its beginning, partially thanks to W3C and IETF's work, there is still a long way to go. At the moment this thesis is being written, most common browsers support the RTCPeerConnection object, refer to Figure 2.11, although there are small differences between them, regarding codec support and the API used. In order to bypass the small differences, there is an open source library, Adapter.js, which unifies the API methods and necessities.
W3C has standardized different RTCPeerConnection methods, which are the principal components used in the WebRTC connection and can be accessed via a JavaScript API. The most important methods are:

- *getUserMedia:* Method from the MediaDevices object which returns a MediaStream object, composed of a VideoTrack and AudioTrack object. It is used to win access to the audio or/and video input of the user, which can later be displayed and shared with other peers.
- *addStream:* RTCPeerConnection object method which adds the previously acquired MediaStream to the connection object.
- *createOffer and createAnswer:* Methods from the RTCPeerConnection object, that are used to create offer and answer messages in order to establish peer connections.
- *setLocalDescription and setRemoteDescription:* RTCPeerConnection object methods which handle the SDP messages from the peers.
- *addIceCandidate:* Method from the peer connection object, in charge of adding candidates to the peer connection object, which will be processed by the ICE framework.
- Handlers: Many handler methods used to trigger functionalities as come available.

In chapter 4 it will be commented the role those methods play in the development of the project as well as issues arisen.

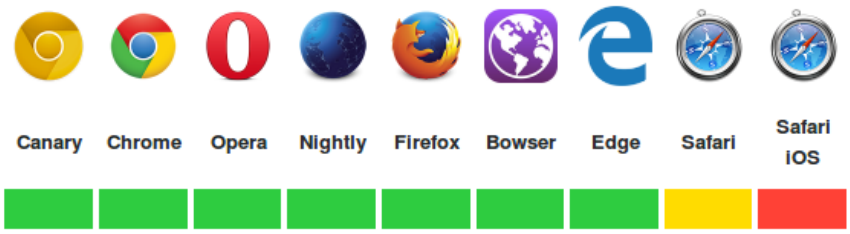


Figure 2.11: Browser support overview [18]

Video conferencing focuses on exchanging video and audio streams between users, in section 2.2 it was mentioned that in the SDP messages peers transferred between each other, it was essential to establish the codec used. A codec is used for compressing the file size, for it's easier and faster transmission. The encoder takes the raw data and compress is it, which will later be received by the decoder who will decompress it. In this process, the quality of the stream is lost, but the duration and bandwidth used for the transmission are reduced. It is important to keep the right balance between the quality and the delay introduced to assure the peer a Quality of Service (QoS) while guaranteeing the codec selected is supported by both peers' browsers. The major video codecs are VP8, its successor VP9, and H.264, which are focused on reducing packet loss and cleaning up noisy images. Regarding audio codec, the most common one is the Opus audio codec, supported by all browsers. Opus is an open-source highly versatile audio codec which supports low and high bit-rates and different sampling rates and was standardized by IETF [19]. Although there are other audio codecs used in WebRTC applications such as G.711, iSAC, iLBC.

Chapter 3

Related Work

This chapter provides a review of the literature regarding the WebRTC technology. It will be divided into three stages; section 3.1 will explain the different existing video conferencing applications, followed by a study, in section 3.2, of the previous WebRTC research papers focused on the educational environment. This chapter will conclude with section 3.3 which will contain a comparison between prior mentioned applications and this thesis' approach.

3.1 Existing Applications for Real-Time Communication

There are various existing collaboration application, especially video conferencing (VC) systems, in this following section we will overview a few, explaining their technological approach:

- *Appear.in:* Free of charge VC software up to 4 users, with no registration required. Based on WebRTC mesh topology for the free version.
- *Google Hangouts:* Google's VC system was prior developed using an external plugin, but it has completely shifted to use the WebRTC technology.
- *FaceTime:* VC application for iOS devices, also based on WebRTC.
- *Slack:* Powerful chat platform, used in many companies and teams, which included it's own WebRTC-based VC system in 2016.
- *Skype:* Well-known Microsoft software for VC, uses its own VoIP protocol, Skype Protocol, also based on peer-to-peer connections. Although it is possible to use Skype on a browser, it is mostly used in its own software application.
- *WebEx:* A platform from Cisco, which is mainly used in webinars or training applications. It requires the installation of external software to be used [20].
- *GoToMeeting:* Citrix's platform for collaboration focused on middle size companies. Also, demands the use of an additional plugin for its usage [20].
- *BigBlueButton:* Focused on education, provides remote students the ability to learn online. It was started in 2007 by Richard Alam for the Carlton University, and it is written in Flash for the exchange of video stream and uses WebRTC for audio stream [21]
- *Visimeet:* Desktop software application for VC, used in colleges such as Illinois State University for its online classes [22].
- *Adobe Connect:* Web-based VC platform used for eLearning and webinars. Developed in Adobe Flash to enable web-camera stream sharing [23].

3.2 Real-Time collaboration in education

The previously mentioned applications are used in various sectors, from the company environment to the education world. Although few VC applications are only focused on education, there is a big movement towards real-time collaboration software. In the following section state of the art research in WebRTC focused on education will be discussed.

A study conducted by Ilinois State University [24] was focused on developing a WebRTC video-conferencing system for e-Learning, using a mesh topology approach. A similar study was conducted at the University of Madrid [25] but using an MCU approach for the video-conferencing system. These studies set a starting point for WebRTC in educational environments, but it cannot be implemented on a real scale e-Learning software.
Another study, conducted in the University of Senegal [26], was focused on the implementation of a WebRTC plugin for the e-learning platform, Moodle. This plugin supported virtual classrooms being conducted in real-time through the integration of video, audio, chat and audio recording. This study revealed the increase in the student's engagement through virtual classes. Although the paper is focused on the implementation of real-time online classes, there are a few activities that promote student collaboration in real-time.
Other related products [27, 28, 29, 30, 31] are focused on the development and implementation of real-time application software based on WebRTC for the interaction between teacher and students.

3.3 Proposed improvements

While comparing previous existing applications for real-time communication in the educational world, I realized that many of state-of-the-art research papers are focused on the implementation of real-time communication between teacher and students, and not on engaging collaboration between students.
FROG software characterizes itself for being a platform which contains many activities focused on real-time student collaboration [32]. Therefore introducing verbal communication into an active collaboration will engage even more the interaction between students. It is important to acknowledge that using one of the multiple platforms mentioned in the first section would also allow users to communicate freely. But this communication would have to be pro-active or proposed by the teacher since it would require the use of an external website or plugin. Therefore including WebRTC for VC in the same FROG platform would allow an easier and faster approach, as well as, it would reduce the necessity of creating a free or paid account for using one of those softwares.

In conclusion, the proposed improvements for the "WebRTC in FROG" project compared to related work or software are; lack of external plugin or website for video conferencing while using the FROG platform, no requirement to create a free or paid account, engage student-to-student collaboration instead of teacher-to-student real-time communication.

Chapter 4

Architecture of the Project

In this following chapter, we will comment the main sections of the final product, together with the development and implementation information such as issues and solutions. Beforehand, we will provide an overview of the final products' structure, in section 4.1. Afterwards, we will discuss the three key parts; in section 4.2 the creation of the peer connections followed, in section 4.3, by information of the TURN and STUN module and lastly an overview of the video and audio capture module in section 4.4.

4.1 Components of the product

The resulting product provides a video conferencing system, using WebRTC, for the FROG software. This module is written in React, due to the usage of this library through the entire FROG project. Although the first prototype of the thesis, which consisted of a simple WebRTC video-conference application, was written in pure JavaScript. In order to adjust to the new library, it was compelling to learn the use of React Component native functions as well as the usage of state and life-cycle events. In the final product, we have used the following life-cycle methods:

- *Constructor:* Gets called only once when the activity it is first created. It sets state and creates some global variables.
- *Component Did Mount:* Once the component has loaded it gets users media, changes state and then notifies it's ready to join the call.
- *Component Should Update:* If there is a change in the props or the state this method gets called. It is used for handling new messages from other users.
- *Component Will Unmount:* Before un-mounting the component, we close all open connections and set it back to the initial state.

When upgrading from the prototype version to the FROG module, we studied the main libraries used for WebRTC connections in order to simplify the methods used. Several libraries were investigated and ranked based on accessibility (open source or not), signalling transport method and protocol, support of multiple connections, community range, and documentation provided. The results made us consider the following libraries: Janus, Jitsi, Kurento, and Licode. Although, in the end, no library was used in order to give more flexibility to the project and to get a deeper understanding of the WebRTC functionality.

The final WebRTC module can be divided into three key aspects, based on the parts of a WebRTC connection which are explained in chapter 2. In the following sections, we will comment those three parts; the P2P connection structure, the network information and the media description. Refer to Figure 4.1 for an overview structure of the code flow which will be later explained.

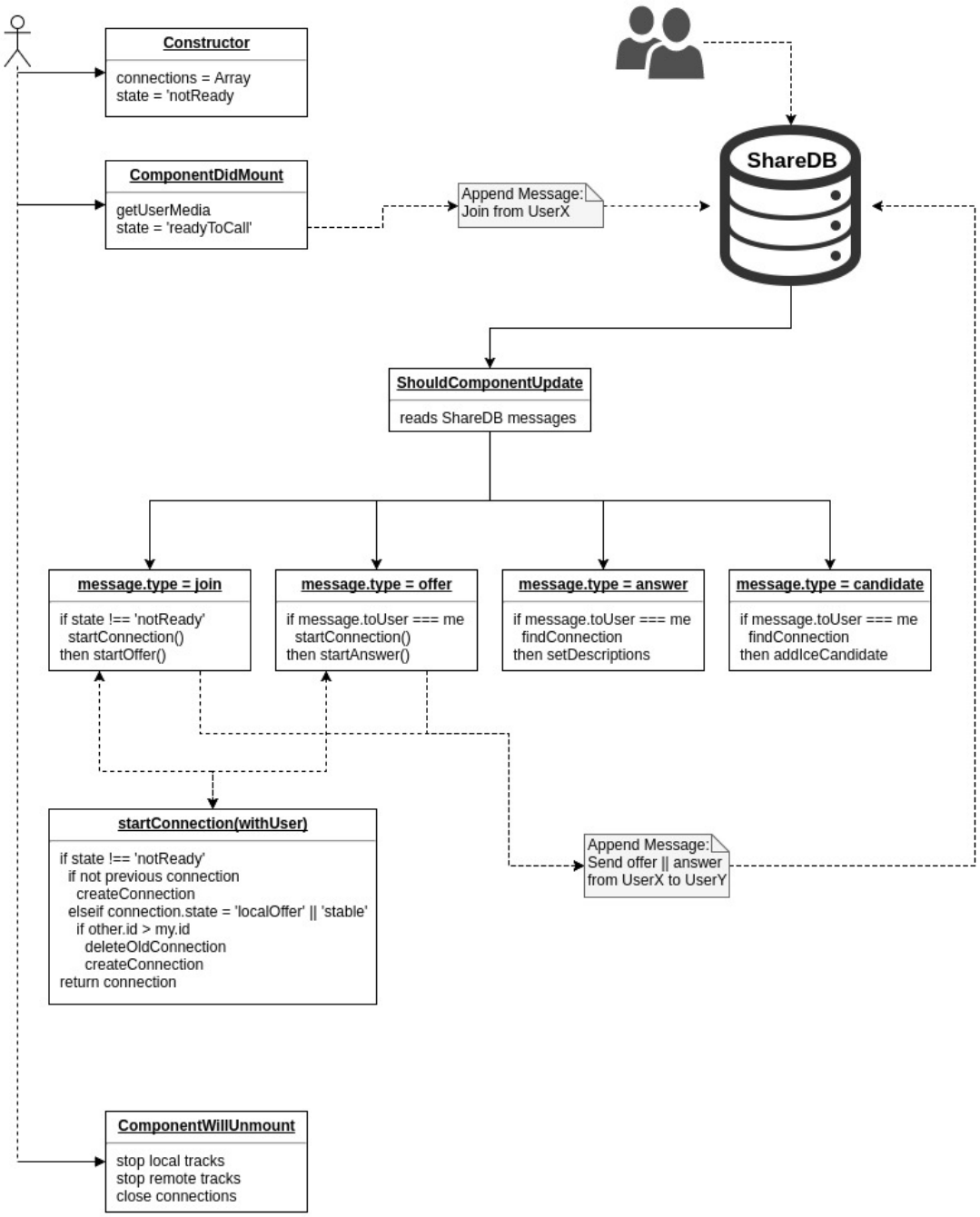


Figure 4.1: Schematic code-flow of the relevant methods in the WebRTC activity

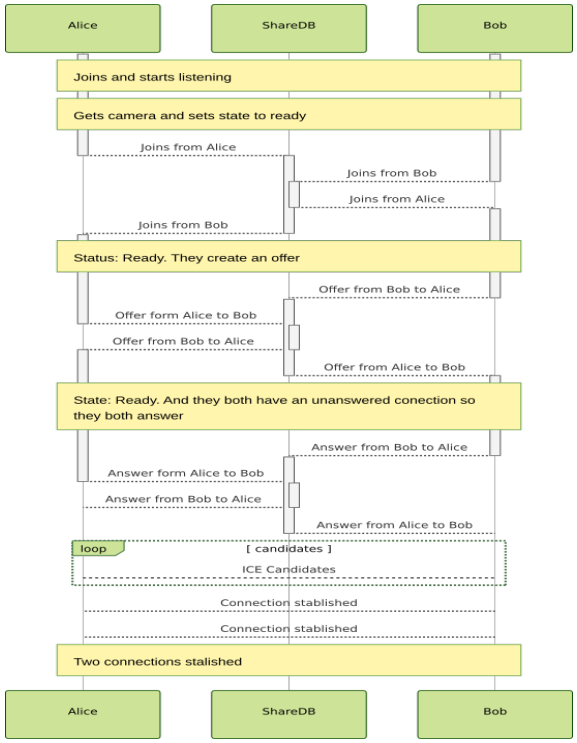
4.2 Peer Connection

In this section we will comment the main aspects regarding the establishment of the peer connection; from the signalling server used, to the network topology and ending with the establishment procedure. It is assumed that readers have a deep understanding of the WebRTC technology, otherwise please refer to chapter 2 for an explanation.

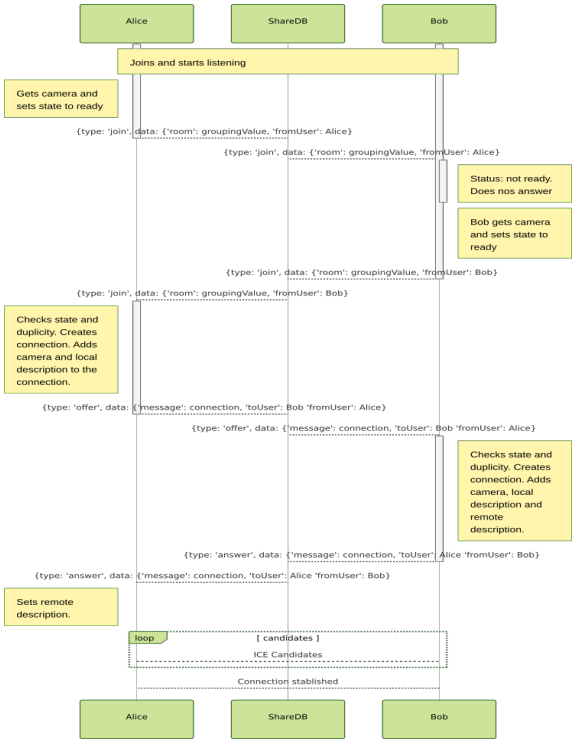
The signalling process for the establishment of the peer connection in the WebRTC activity is done through a shared database, instead of an external signalling server which redirects the messages between clients and handles different rooms/groups, as commented previously. In the prototype version, there was a simple server code written using the Socket.io library which created a WebSocket connection with each client, whom also used the Socket.io library. During the integration, it was discovered that the importation of the library Socket.io was difficulted due to an already existing wrapper over the WebSocket protocol.
Therefore it was necessary to exchange all the code to support regular WebSocket communications instead of the Socket.io wrapper in both, the client and the server code. During the development of this new stage some synchronicity errors appeared and also the configuration of the rooms was difficult to implement. For that reason, the central server approach was shifted, and instead of using the external server with WebSockets we decided to use the already existing shared database between users.
FROG has many activities that require real-time collaboration, and this real-time is accomplished by the usage of a shared database, ShareDB, between the users of the same group. The ShareDB is used in the final product as a "wall of messages" for the clients to exchange information of each other. Since this ShareDB manages the synchronicity of the messages as well as groups, we decreased the complexity of the activity. It is also positive the reduction of an external server to process the messages.
The ShareDB information is distributed among users with a prop called data. Each time this data is modified the shouldComponentUpdate method is called, therefore every time a user "publishes" a new message this method is triggered, and the other users can act depending on the contents of the message.

Another thing to consider with the creation of the WebRTC activity was the topology for multi-party conferences. After analyzing the main architectures for real-time network topologies, commented on section 2.4, and considering the requisites needed for WebRTC in FROG. We concluded that a mesh scope would adapt best to the necessities. A full-mesh architecture consists on the creation of multiple one-to-one peer connections between all users. The principal usage of the technology is to establish a video-conference with a small-medium group of people working in the same activity, therefore it will usually not bypass 6 users per room. The mesh configuration would allow the interconnection with less delay and quality loss of the peers without the need of an external server to process the peer connections.

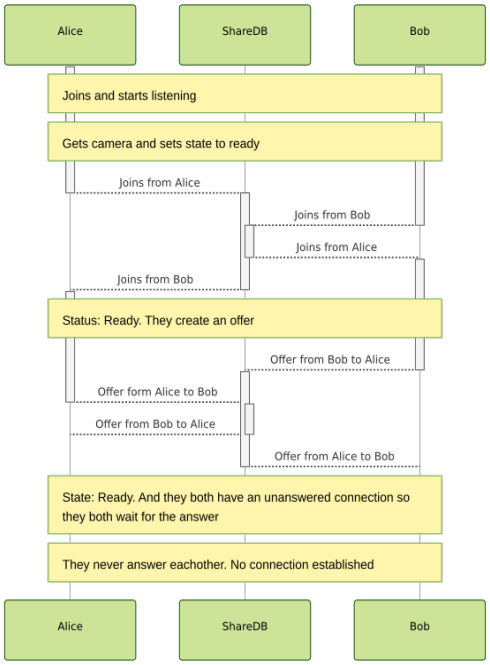
Lastly, it was crucial to establish a bulletproof signalling process. One of the main challenges of the project had to do with the synchronicity, specifically in the signalling phase. It was necessary to work either with users joining at different times or all together at the same time. The first case happens if the WebRTC is the first activity, or if a user joins in late to FROG. And the second case scenario happens when all users are already connected and a new WebRTC activity is called. Therefore it was key to manage the changing of activities seamlessly, and the key to this lies in handling the closing correctly as well as having the correct signalling system.
The signaling procedure used on the prototype was simple. It was based on a notification, offer, answer process without taking into account that Alice and Bob might join at the same time. In Figure 4.2 you can see the errors this approach may cause:



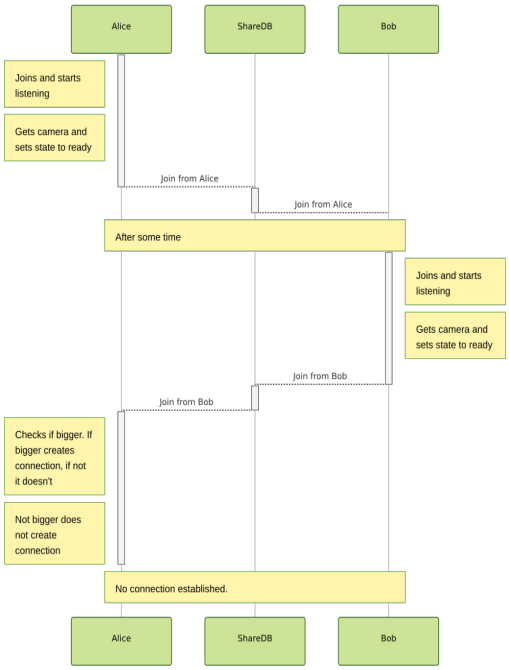
(a) Two Connections



(b) Not ready yet



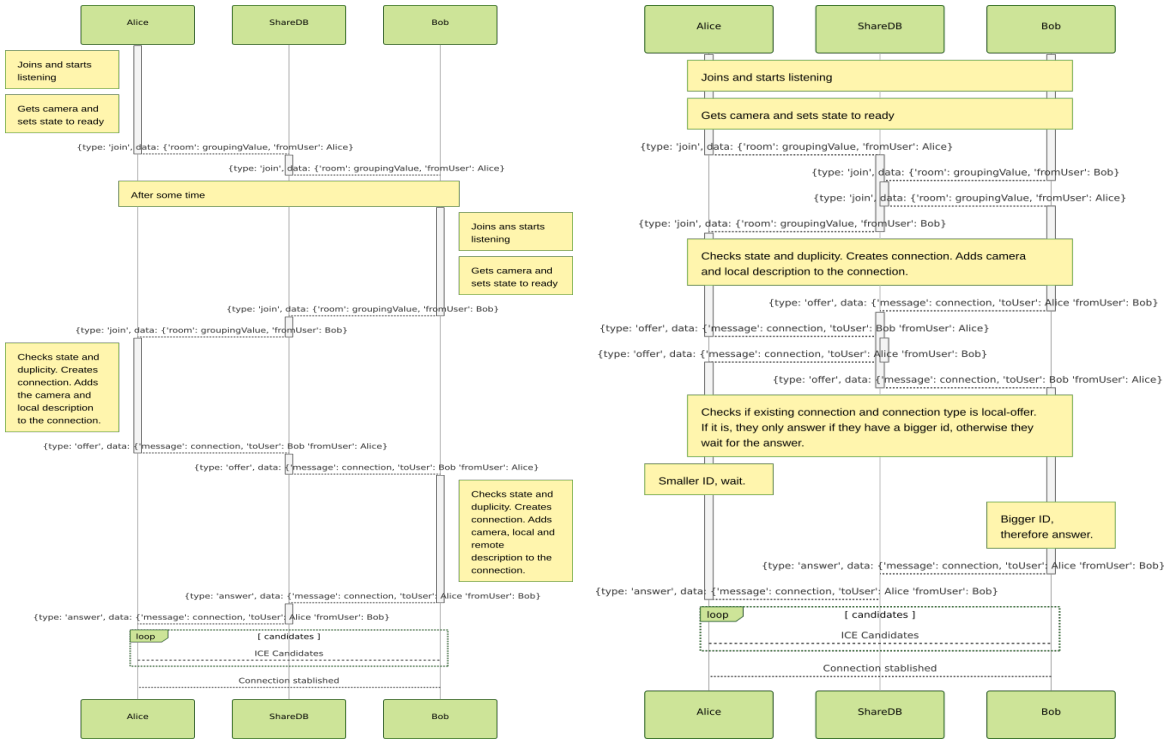
(c) Wait for each other



(d) Only if bigger

Figure 4.2: Issues with the synchronicity of the signaling system

- **Fig. 4.2(a):** If two users join at the same time without any control method two connections will be established between those users. This is the result of importing the signalling system from the prototype, where there were no synchronicity issues, to the FROG environment.
- **Fig. 4.2(b):** Another case to take into account was the users joining at the same time but setting state ready in different times. It is necessary to discard those messages read while the Component is still creating, therefore the state is 'notReady'.
- **Fig. 4.2(c):** In order to solve the synchronicity issues one approach was to check if they had already created a peer connection (an offer), and do not answer if they do. But this leads to the creation of no connections since they both wait for the other.
- **Fig. 4.2(d):** A future approach was setting a decider parameter in which only one would start the connection if is bigger than the other. But if this parameter is set right at the begging point causes issues when users join at different times.



(a) Join at different times

(b) Join at same time

Figure 4.3: Definitive product signaling process flow

The final solution is a combination all the previous approaches. In Figure 4.3 we can see both use cases for users joining at different times (Figure 4.3(a)) and for users joining at the same time (Figure 4.3(b)). The key things to assure that the connection is correctly done without duplicates are the following:

1. Only process messages from other users if your own state is different than 'notReady'.
2. When receiving a 'join' notification check if you already have an existing connection. Probably not, then create one. Next, set your local description and send an offer.
3. When receiving an 'offer' notification check if you already have an existing connection. There are two cases:

(a) *Does not exist:* Usually happens if the users did not join at the same time. Then create a peer connection, set local description and copy the description from the offer to the remote description. Lastly, create an answer.

(b) *It exists:* Usually happens when users have joined at similar times. The existing connection may have different signalling states (have-local-offer, stable, complete...). If signalling states are 'have-local-offer' or 'stable' only the smaller user will answer, the bigger user will wait for an answer.

4. When receiving an 'answer' notification set the remote description from the one provided in the message.

4.3 TURN and STUN servers

FROG is mostly used in educational environments and usually the networks that surround those sectors have strong and restrictive firewalls. Therefore it is crucial to include TURN and STUN servers to assure the well functionalism of the WebRTC activity in FROG.
After studying multiple existing source codes for STUN and TURN servers, we chose an open-source code named CoTurn [33] which can be easily deployed and its used in multiple applications. The code was downloaded and ran on a server and the resulting URL was added to the source code, it was crucial to modifying the file named *'iceServers.js'* located on *'FROG/ac/ac-webrtc/src/iceServers.js'* to include our server's IP. In order to deploy the CoTurn server we executed the following command:

```
turnserver -a -v -n --no-dtls --no-tls -u test:test -r
"someRealm" --syslog -L YOURSERVERIP -f --log-file=stdout
```

Lastly, to test the existing TURN and STUN server it was necessary to include some modifications to the code since the EPFL environment did not block the WebRTC connections. To do so it was necessary to discard all the ICE Candidates that did not include the parameter relay, therefore it was necessary to discard all the direct connections and STUN information of the ICE Candidates.

4.4 Video and Audio Capture

It is important to define beforehand which type of data will flow through your WebRTC connection. In our case, since it is focused on video-conferencing the peer connection will transfer video and audio streams. To acquire the web camera and microphone inputs from the client it is necessary to use the MediaStream API of the browser, refer to section 2.5 for more information. Although W3C and IETF have been working on an standardization, the capture of these inputs has not been yet regularized through the variety of browsers, therefore each browser uses its own methods and has its own restrictions. It is required to point out that most browsers only allow the video and audio capture if the connection is secure (https) or it is running on localhost. In order to unify the different WebRTC methods, it was necessary to use an external library, Adapter.js, which unifies all the different browsers necessities. Adapter.js [34] is a JavaScript shim, created and maintained by Google, and later on with the help of Mozilla as well as the WebRTC community, that overcomes browser differences and spec changes [35].

The improvement of the prototype version to support more than two users, rooms and remote hang-ups was challenging but without any main interference along the way. Although, the introduction of the multiple users created a problem with the audio quality, due to an enormous amount of echo noise. After multiple days of research that lead to a change in the comparison method of the audio to the Opus codec, refer to section 2.5 for more information. Although the usage of this codec has improved the echo cancellation, it is still encouraged to use headphones and external microphones when using WebRTC on FROG.

Chapter 5

Results

This chapter will provide a brief overview of the final product, summarizing previous chapters into a short description. It will begin with an explanation of the overall application, followed by an explanation of the technology used and will conclude with a summarization of the most important structures.

The module developed is part of the FROG web application. As explained previously FROG is a web-based platform for the creation and implementation of Orchestration Graphs, a logical and structured way of designing pedagogical activities. FROG is focused on the creation of collaborative group activities in order to engage student interaction. The module developed in this thesis creates video conferences for groups of students who are working together in the same FROG activity, in order to ease and engage a more effective collaboration, without the need of an external service.

To create this video conference module for a web application, we used the WebRTC technology which is an arising technology that connects browsers in real-time through a peer-to-peer connection. This P2P connection allows the real-time exchange of media streams, such as video and audio obtained from the client. To set this connection, it is important to follow a signalling procedure, where peers exchange the information needed to create this connection. The exchange of information is done through a shared database and consists of the transfer of the network information, obtained through the ICE candidates, and the media information, via SDP messages. The signaling process has considered the possible synchronization errors that may occur, users joining at the same time or at a different time, and works consequently. The module connects peers forming a full-mesh topology for faster interconnection and to obtain the lower delay possible.

The creation of the WebRTC activity for FROG has achieved mostly all of its requirements, in 5.1 we can see a simplification of those. It is important to remark that due to the usage of the mesh configuration the video-conference it is limited to a reduced number of users, recommended 5 or 6 maximum, although this adapts to the project specs.

The WebRTC activity for FROG can be used together with existing activities in order to engage student's communication and collaborative learning. In image 5.2 we can see a caption of the WebRTC module working alongside with the collaborative activity for peer-coding.

WebRTC connection	✓
Multi-party (up to 5 users)	✓
Multi-party (more to 5 users)	✗
Video streaming	✓
Audio streaming	✓
Video + audio streaming	✓
Strict enviroments	✓
Loss of connection handeling	✓
Echo cancelation	✓
Good audio quality	✓
Low audio delay	✓
Good video quality	✓
Low video delay	✓
Between different networks	✓
Between different locations	✓
Between different countries	✓
Not many delay	✓
Browser compability	✗

Figure 5.1: Requirements and Accomplishments

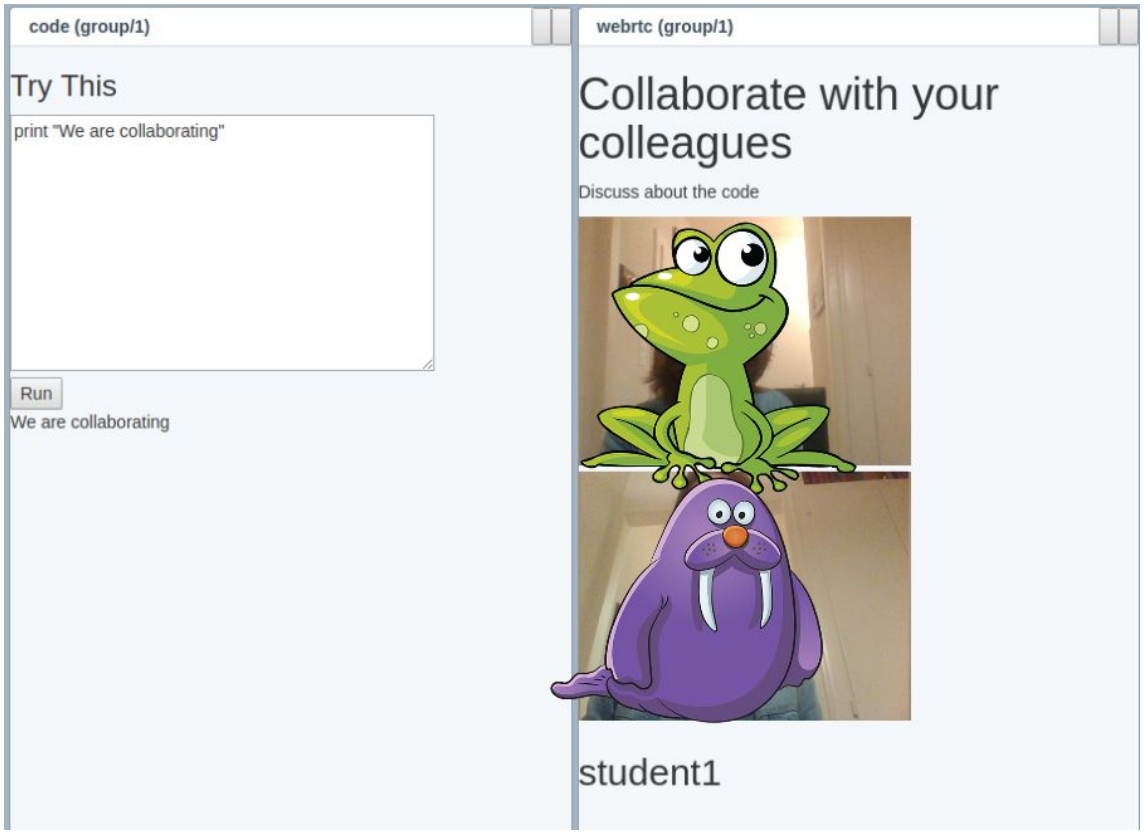


Figure 5.2: Picture of the WebRTC activity alongside with a collaborative learning activity

Chapter 6

Conclusions and future work

To conclude we will provide a discussion of the final product all together with future related work that could be accomplished and a few recommendations for future developers of WebRTC or the FROG software.

To create a video-conference module for the web software FROG, we have developed a real-time communication using WebRTC technology. This technology has proven to be the most effective, fast and easier way to create a real-time conference between browsers, overcoming his predecessor, the plugin based video-conferencing systems. This is corroborated thanks to several reasons. The WebRTC VC systems do not require the installation of non-standardized internal or external plugins or softwares, allowing a faster interconnection between users. Also, they create P2P connections between its browsers, with no need of an external shared server to enable the communication, which allows a quicker interconnection between the users due to the reduced amount of intermediate infrastructure. The lack of a Media Server also reduces the connection delay, compared to the use of the flash player plugin, as well as it improves the audio latency and quality.
The architecture used to connect multiple users for the VC system is a full mesh network topology which introduces the lowest latency possible and assures the highest audio and video quality. Also, it requires no external server, therefore it is the cheapest infrastructure available. Since the WebRTC for FROG activity is designed for group activities, and these activities usually comprehend up to 6 users, this architecture suits perfectly the requirements of the project.

There is much future work to be accomplished in the WebRTC activity for FROG. Some expansions should focus on the improvement of its UX and UI, since currently all video inputs are displayed in stack. Another aspect that could be developed would the change of network topology to SFU in order to allow multi-party conferences with many people such as an entire class. Or to an MCU topology where the video would be processed in order to extract data from its participants and could be shown to the teacher.

For people who want to engage in this project, either WebRTC or FROG, I would recommend they get documented beforehand and they plan the requirements and goals before developing. It is important to think, determine and investigate all case scenarios, especially with synchronization before deciding on one method.

Appendix A

Budget

The following appendix will contain overview information of the budget glossary for the thesis. This budget has been established following the standards of the leaving cost in Switzerland, such as salary and hardware cost. The overall time has been measured through the work accomplished during 21 weeks of work.
We conclude that the development of this thesis has a total budget of 14427€.

Description	**Unit Price**	**# Units**	**Total [€]**
Personnel	20.5 €/h	504 h	10332
Hardware	1499 €/u	1 u	1499
Server	0.024 €/h	3528 h	84.67
Printed resources	1.4 €/u	5 u	7
Total			11922.67
Tax (21%)			+2503.76
Total			**14426.43**

Table A.1: Budget overview

Appendix B

File Architecture

The source code of the WebRTC activity can be found in the '*ac/ac-webrtc*' folder of the FROG software. The files, contained in the '*src*' folder, are divided for an easy manipulation:

- *index.js:* Combines all the functionalities and exports the activity. Refer when modification of the 'meta' data or 'mergeFunction' wants to be made.
- *config.js:* The structure of the configuration needed for the activity to run. The teacher will be asked to set the following information title, subtitle, and sdpConstrains.
- *iceServer.js:* JSON file containing the STUN and TURN servers used for the ICE candidates. To be modified each time application is used, with own IP address for the STUN and TURN server
- *codec.js:* Methods to modify the SDP message in order to prefer OPUS codec.
- *Video.jsx:* JSX elements for Local and Remote Videos.
- *ActivityRunner.jsx:* Main file. Contains all rendering logic as well as the peerConnection establishment.

Appendix C

Commands

In order to run the project please follow this steps. If you do not have the FROG environment installed, make sure you have the latest version of node, npm and meteor installed and then run:

```
git clone https://github.com/chili-epfl/FROG.git
git checkout webrtc
```

Once you are set in the correct branch it is necessary to install some libraries, some issues may arise if using Windows. To do so, from the FROG folder run the installation files with the command:

```
./initial_setup.sh
```

If the installation was not successful please contact the authors of FROG. Otherwise type:

```
./run_and_watch_all.sh
*open a new tab and go to the same FROG directory*
cd frog
meteor
```

Then you can open a web browser and type the following address "`localhost:3000?login=teacher`". In order to test the WebRTC module by yourself type the following address afterwards "`localhost:3000/preview/ac-webrtc/0?windows=2`".

It is important to remember that in strict networks it may be needed to deploy a STUN and a TURN server. In order to do so install the the coturn server and then follow this command to run it:

```
turnserver -a -v -n --no-dtls --no-tls -u test:test -r
"someRealm" --syslog -L YOURSERVERIP -f --log-file=stdout
```

Do not forget to modify the '*iceServers.js*' file located on '*FROG/ac/ac-webrtc/src/iceServers.js*' to include your server's IP.

List of Figures

Bibliography

[1] P. Dillenbourg. *Orchestration graphs : modeling scalable education.* Number 226087 in EPFL-Book. Lausanne : EPFL Press, 2015.

[2] A. Bergkvist, D. C. Burnett, C. Jennings, A. Narayanan, B. Aboba, T. Brandstetter. Webrtc 1.0: Real-time communication between browsers. *Working draft, W3C*, 91, November 2017. [Online] Available at `http://w3.org/TR/webrtc/`.

[3] H. Alvestrand. Overview: Real time protocols for browsers-based applications. Internet-Draft draft-ietf-rtcweb-overview-19, Internet Engineering Task Force, November 2017.

[4] C. Jenngins, T. Hardie, and M. Westerlund. Real-time communications for the web. *Communications Magazine, IEEE*, 51:20–26, April 2013. [Online] Available at `http://ieeexplore.ieee.org/document/6495756/`.

[5] P. Rodriguez, J. C. Arriba, I. Trajkovska, and J. Salvachua. Advanced videoconferencing services based on webrtc. In *Proceeding of IADIS multi conference on computer science and information systems*, pages 1–8, 2013.

[6] R. Dillet. Safari puts yet another nail in flash's coffin, June 2016. [Online].

[7] S. Dutton. Getting started with WebRTC. [Online] Available at `https://html5rocks.com/en/tutorials/webrtc/basics/`, July 2012.

[8] D. C. Perkins, M. J. Handley, and V. Jacobson. Sdp: Session description protocol. RFC 4566, RFC Editor, July 2006. [Online] Available at `https://rfc-editor.org/rfc/rfc4566.txt`.

[9] M. Westerlund and C. Perkins. IANA Registry for Interactive Connectivity Establishment (ICE) options. RFC 6336, Internet Engineering Task Force, July 2011.

[10] S. Loreto and S. Romano. *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser.* " O'Reilly Media, Inc.", 2014.

[11] J. Uberti, C. Jennings, and E. Rescorla. Javascript session establishment protocol. Internet-Draft draft-ietf-rtcweb-jsep-24, Internet Engineering Task Force, October 2017. [Online] Available at `https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-24`.

[12] M. Petit-Huguenin and G. Salgueiro. Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN). RFC 7350, RFC Editor, October 2015.

[13] P. Matthews, J. Rosenberg, and R. Mahy. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766, RFC Editor, April 2010.

[14] J. Uberti. WebRTC: Plugin free real-time communication. [Online] Available at `"http://io13webrtc.appspot.com/"`.

[15] M. Westerlund and S. Wenger. RTP Topologies. RFC 7667, Internet Engineering Task Force (IETF), November 2015.

[16] V. Pascual. WebRTC beyond one to one. [Online] Available at "https://webrtchacks.com/webrtc-beyond-one-one", February 2014.

[17] B. A. Jansen. Performance analysis of WebRTC-based video concerencing. Master's thesis, Delft University, Netherlands, August 2016.

[18] P. Hancke. Is WebRTC ready yet? [Online] Available at: http://iswebrtcreadyyet.com/, 2017.

[19] J. M. Valin and K. Vos. Updates to the Opus Audio Codec. RFC 8251, Internet Engineering Task Force (IETF), October 2017.

[20] M. Heller. Review: WebEx vs. GoToMeeting vs. MyTrueCloud. [Online] Available at: https://infoworld.com/article/3010043/collaboration/review-webex-and-gotomeeting-meet-their-match.html, December 2015.

[21] BigBlueButton team. History of bigbluebutton. [Online] Available at: https://bigbluebutton.org/about/, 2012.

[22] IOCOM. Professors at Illinois State University use Visimeet. [Online] Available at: http://janet.iocom.co.uk/docs/ISU_case_study.pdf, 2013.

[23] Adobe Connect. Adobe Web Conferencing Software. [Online] Available at: http://www.adobe.com/products/adobeconnect.html.

[24] K. Bissereth, B. Lim, and A. Shesh. An interactive video conferencing module for e-learning using webrtc. In *International Conferences*, pages 1–4, 2014.

[25] P. Rodríguez Pérez, J. Cerviño Arriba, I. Trajkovska, and J. Salvachúa Rodríguez. Advanced Videoconferencing based on WebRTC. 2012.

[26] S. Ouya, K. Sylla, P. M. D. Faye, M. Y. Sow, and C. Lishou. Impact of integrating webrtc in universities' e-learning platforms. In *2015 5th World Congress on Information and Communication Technologies (WICT)*, pages 13–17, December 2015.

[27] H. Oh, S. Ahn, J. Choi, and J. Yang. Webrtc based remote collaborative online learning platform. In *Proceedings of the 1st Workshop on All-Web Real-Time Systems*, page 9. ACM, 2015.

[28] T. Balan, A. Stanciu, F. Sandu, and S. Surariu. Webrtc based e-learning platform. In *The International Scientific Conference eLearning and Software for Education*, volume 2, pages 48–55, 2017.

[29] Y. Liao, Z. Wang, and Y. Luo. The design and implementation of a webrtc based online video teaching system. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 137–140, October 2016.

[30] K. Jain, A. Himmatramka, A. Bhandary, A. D'Silva, and D. Barge. Synchronized development using webrtc real-time collaboration in webrtc. *International Journal of Engineering Science*, 6(4), 2016.

[31] B. P. Fazakas, O. C. Iuonas, C. Porumb and B. Iancu. Collaborative learning tools for formal and informal engineering education. In *2017 16th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pages 1–6, September 2017.

[32] S. Håklev, L. Faucon, T. Hadzilacos, and P. Dillenbourg. Frog: rapid prototyping of collaborative learning scenarios. In *EC-TEL*, number EPFL-CONF-230014, 2017.

[33] CoTurn repository. [Online] Available at: "https://github.com/coturn/coturn".

[34] Adapter.js repository. [Online] Available at: "https://github.com/webrtc/adapter".

[35] T. Levent-Levi. What is WebRTC adapter.js and why do we need it? [Online] Available at: "https://bloggeek.me/webrtc-adapter-js/", January 2018.

Nomenclature

Acronyms and Abbreviations

API	Application Programming Interface
CHILI	Computer-Human Interaction in Learning Instruction
CSCL	Computer Supported Collaborative Learning
FROG	Fabricating and Running Orchestration Graphs
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IP	Internet Protocol
JSEP	JavaScript Session Establishment Protocol
MCU	Multi-point Control Unit
NAT	Network Address Translation
P2P	Peer-to-Peer
QoS	Quality of Service
RTC	Real-Time Communication
S/C	Server-to-Client
SDP	Session Description Protocol
SFU	Selective Forwarding Unit
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TURN	Traversal Using Relays around NAT
UDP	User Datagram Protocol
VC	Video Conference
VoIP	Voice Over Internet Protocol
W3C	World Wide Web Consortium
WebRTC	Web Real-Time Communication

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Title of work:

Real-Time Communication applied to Orchestration Graphs

Thesis type and date:

Bachelor Thesis, January 2018

Supervision:

First Supervisor
Haklev, Stian
Second Supervisor
Fornes de Juan, Jorge

Student:

Name: Iciar Puigpelat Barrado
E-mail: iciar.puigpelat@gmail.com

Submission:

A Degree Thesis Submitted to the Faculty of the Escola Tecnica d'Enginyeria de Telecomunicacio de Barcelona (ETSETB) Universitat Politecnica de Catalunya (UPC) by Iciar Puigpelat Barrado.

In partial fulfillment of the requirements for the degree in Telematics Systems.