e-ISSN:2582-7219

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH IN SCIENCE, ENGINEERING AND TECHNOLOGY

Volume 5, Issue 5, May 2022

International Journal Of Multidisciplinary Research In Science, Engineering and Technology (IJMRSET)

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.54|

| Volume 5, Issue 5, May 2022 |

| DOI:10.15680/IJMRSET.2022.0505032 |

Web Real-time Communication Scalability using Multipoint Conferencing Unit

Kshitij Srivastava[1], Deepak Arora[2], Puneet Sharma[3]

Department of Computer Science & Engineering, Amity University, Uttar Pradesh, Lucknow Campus, India

ABSTRACT: Web-based communication has become a significant part of our daily life. Over the last decade, the number of Real-time Communication (RTC) products has skyrocketed, thanks in part to cheaper high-speed internet and more capable devices, but also to an open-source, free to use a platform known as Web Real-time Communication (WebRTC). WebRTC allows web browsers to interact in real-time using JavaScript APIs without any need for extra plugins. However, as the number of users or participants grows, so does the need for resources and bandwidth. The primary goal of this study is to examine the performance of multi-party video conferencing using the Multipoint Conferencing Unit (MCU) and how well WebRTC applications can be scaled utilizing the MCU technique.

KEYWORDS: WebRTC, MCU, P2P, Videoconferencing.

I.INTRODUCTION

Real-time communications, sometimes known as RTC, refer to any form of telecommunications in which all users are able to share information instantaneously or with very little lag time or transmission delay. In this particular setting, the terms live and real-time are interchangeable. There is never an indirect route in RTC; rather, there is always a direct link between the source and the destination. The data is sent from the source to the target without being stored at any point along the route, despite the fact that the network may comprise numerous intermediary nodes. Asynchronous or time-shifting communication, such as email and voicemail, on the other hand, always require data storage between the source and the destination. When something like this occurs, it is normal for there to be a lag time between the information being sent and when it is received.

Real-time communications underwent a further revolution in the 20th century thanks to advancements in high-speed internet, mobile telephony, and smart devices. These advancements made it possible to engage in instant messaging (IM), video calling, video conferencing, and other activities.

Another significant step forward in the evolution of real-time communications technology was taken in 2011 when Google made available the open source WebRTC project. WebRTC is able to provide real-time, peer-to-peer audio and video communication via conventional web browsers by utilising Application Programming Interfaces (APIs) written in JavaScript. This eliminates the need for specialised plugins or stand-alone programmes.

There has always been an increasing need for real-time multi-party video conferencing, and WebRTC (Web Real-Time Communications) has sparked a lot of interest because the API is supported by a variety of modern browsers. However, because WebRTC was developed for browser-to-browser communication, creating a multi-party conferencing model is difficult and costly. There will be N*(N-1)/2 links to support N conference participants on a pure Mesh network. The bandwidth/device capabilities requirements will grow in lockstep with the number of conference attendees.

As a result, a central media server is almost always necessary. A WebRTC media server is essentially a multimedia middleware via which media traffic is routed from source(s) to destination(s). Media servers may handle incoming media streams and provide a variety of results, including Group Communications, Mixing, Transcoding, and Recording.

The MCU (Multipoint Conferencing Unit) architecture for media servers was devised to reduce bandwidth usage. It's a sophisticated concept that creates a composite video and audio stream from all of the peers' video/audio and feeds it back to everyone.

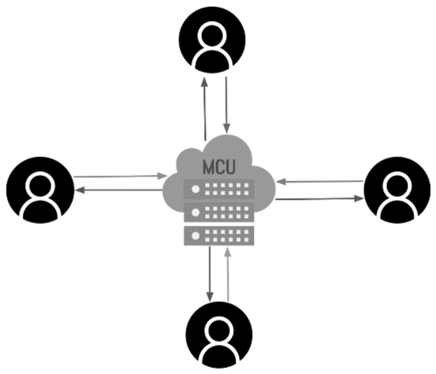


Figure 1: Mixing of streams on MCU and emitting of a single composite stream for each client.

Regardless of how many people are on the call, the MCU ensures that each person only receives one set of video and audio streams as shown in Figure 1. As a result, the computers of the participants don't have to do quite so much work. MCU servers are generally fairly expensive and require a lot of bandwidth because they are often built on fixed and pre-configured hardware. MCU process can vary on each implementation but can be simplified into these five steps as shown in Figure 2. The MCU device is responsible for receiving these media streams from every participant, decoding them, creating a layout that includes all of them, and then encoding them so that they may be eventually transmitted to all of the peers.

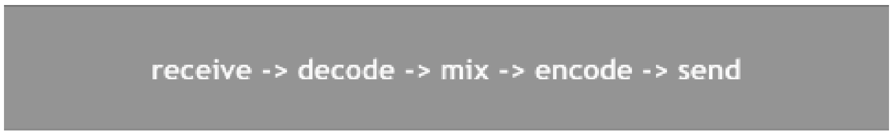


Figure 2: Simplification of MCU Process.

II.LITERATURE REVIEW

Simon Holm and Alexander Lööf published a paper on “The Design and Architecture of a WebRTC application” [1], a study that looked at current design or architectural patterns for WebRTC apps and how they might be implemented using JavaScript. The authors have reviewed the Full mesh-Peer to Peer, MCU, and SFU design patterns, as well as the use cases and limits of these current models. In this study, a model was suggested that combined the Full mesh model with the SFU model.

"Design and assessment of browser-to-browser video conferencing in WebRTC" by NaktalMoaid Edan, Ali Al-Sherbaz, and Scott Turner [2] explain WebRTC and its clients and server. The authors built and implemented WebRTC video conferencing between browsers utilising Chrome and wired/wireless LAN/WAN networks. In this research paper, the authors have also evaluated CPU performance, bandwidth use, and QoE. Also, a WebSocket-based signalling channel between browsers using Node.js has been designed and tested.

A research paper on “P2P Live Video Streaming in WebRTC” has been created by Florian Rhinow, Pablo Porto Veloso, Carlos Puyelo, Stephen Barrett, and Eamonn O Nuallain[3]. The goal of this study is to see if it's possible to utilise WebRTC to introduce live video streaming protocols into online applications. The authors also touched on the constraints and potential future issues that may arise when implementing complicated and scaled P2P systems.

“A P2P-MCU Approach to Multi-Party Video Conference with WebRTC” by Kwok-Fai Ng, Man-Yan Ching, Yang Liu, Tao Cai, Li, and Wu Chou [4], a research project aimed at addressing the problem of high MCU server costs due to pre-specified hardware configuration. The authors of this study developed a P2P-MCU strategy for multi-party video conferencing that effectively supports both conventional smartphones and PCs by integrating the MCU module into the browser to mix and transcode video and audio streams in real-time. The authors are confident that CPU and bandwidth

utilization will be reduced by 64% and 35%, respectively. This model features less than a 500ms delay that is both steady and tolerable.
“Kurento: The Swiss Army Knife of WebRTC Media Servers” is a paper written by BoniGarca, Luis López-Fernández, Micael Gallego, and Francisco Gortázar [5] and published in IEEE Communications Standards Magazine (Volume: 1, Issue: 2, 2017) [5] that explains how Kurento works and the benefits of using it. The writers of this paper conducted comparative research on the Kurento media server, which includes various rich media features to help with application development. Kurento is built on a modular design, and its API is available in Java and JavaScript programming languages, as part of a three-tiered web development approach. Kurento can support SFU, MCU or both.
A paper on “Models for Multimedia Conference between Browsers based on WebRTC” by WajdiElleuch [6] provides two distinct conference models that have been tailored to facilitate WebRTC communication across browsers for both local and large-scale conferences. This work's suggested abstract protocols implement conference formation and browser joining/leaving and maybe mapped using SIP/DSP primitives.
“Performance Analysis of WebRTC-based Video Conferencing”, a thesis by B.A. Jansen [7] in which the author has performed an analysis of different scenarios for a video conferencing system utilizing WebRTC. The author has put up a testbed to analyse WebRTC performance in various situations and network impacts using the newest browsers. Testing latency, packet loss, and bandwidth. Cross-traffic, multi-party, WebRTC, mobile browsers, and video codecs are also studied. The performance study compares WebRTC performance across browsers and mobile devices. Newly implemented support for VP9 and H.264 video codecs are assessed and require improvement.

III.METHODOLOGY

The events of 2020 would have been harder to organize without WebRTC innovators from the past decade. WebRTC helped us deal with being stranded at home. According to Credence Research Inc.'s "Web Real-Time Communication Industry 2016–2023" estimate, the Web-RTC market will reach over $19 billion in 2023, with annual growth rates of 40% or more over the next five years. A similar trend has been seen with other video conferencing services such as Google Meet and Microsoft Teams. Videoconferencing is also projected to continue beyond the epidemic, as Gartner predicts that just 25% of business meetings will be done in person by 2024.
A server that accepts media streams as input and distributes them to the end-users as output is known as a media server. The first and most important step is to exchange the appropriate credentials between the media server and the client. Because every router and anti-virus generally has some form of a firewall to guard against suspicious or undesired connections, the second stage is to build some kind of mechanism to penetrate through this wall of security for the successful establishment of a connection between a peer and the media server, this can be achieved by using a STUN or TURN server (optional for systems with low security). In the third step, the signaling server is used to identify the media server. Signaling is critical because it allows the end-user to determine which media server to connect to. After successfully identifying the media server, the fourth stage is to connect with the media server, which has all of the essential functionality to send the stream to all of the appropriate participants. When the media server receives more than one stream, it begins processing and distributing video codecs to each user.

IV.IMPLEMENTATION

In this research study, the authors have used an MCU media server to enable browser-to-browser interaction while ensuring scalability. As previously stated, the MCU is a centralized model in which user streams are processed, encoded, and mixed before being sent back to each peer as a single composite stream. Sending only one stream decreases CPU utilization and bandwidth for each peer, making the MCU architectural pattern more scalable.
Authors are working on an MCU solution that can adapt, redirect, and translate media streams. This must be a safe path, as MCU not only acts at the transport layer but also at layers underneath that layer as well. Rather than depending on the W3C's continued development of the JavaScript API specification, this leaves flexibility up to the individual application. Here, the authors have proposed an MCU that complies with the standards and is capable of connecting the WebRTC ecosystem as an enabler that delivers the out of the box services.

A. Mechanism

The MCU is sophisticated software that continually connects to WebRTC and can reroute and transcode media streams. MCUs can be installed anywhere on the network and do not represent conference participants; they provide cutting-edge communications services. Even though WebRTC peers build an overlay network by creating two or more connections, an MCU makes sense and is necessary for handling videoconference issues.

In the simplest case, a MCU is responsible for directing media flows among participants in communication by only allowing them to acquire data from other participants, operating as a matrix. There may be a slew of operations supplying the most up-to-date services for video conference-based communication, given that all data must pass via that Multiple Communication Unit. MCU operates as a centralized bridge, allowing multiple devices to connect in real-time.

Media Encoding and Decoding (Transcoding): There is always several communication units can mix and transcode media streams and flows to support heterogeneous access networks and devices. It is recommended that streams be transcoded into a variety of formats and bitrates so that the communication may be modified to varied screen sizes, network circumstances, and other considerations without the MCU having to manage it all by itself. Users operating within a gateway environment, which requires media streams to be localised, might also benefit from this feature.

Media Composition: The Multiple Communication Unit reduces CPU overhead and controls needed for multi-person videoconferencing by creating a single audio and video stream.

Recording of Media: When a communication is transmitted, the MCU is sure to get a copy of all of the streams that are currently active. As was said before, it is able to produce a composite stream by mixing the two streams in question. In the case that an audio recording of the session is required, the flow of the session will be saved by MCU for use in future reproductions [8]. Because these features provide us control over all of the data and information that is accessible during a single session, we are able to deliver the sophisticated services that are routinely requested in multi-conference communications and collaborative applications.

B. Multi-Communication Unit Architecture Utilizing WebRTC

In this part, we will cover MCUs that are able to connect with existing WebRTC apps. To get things started, the MCU is made up of four essential parts, which are stream processing, control, transportation, and API.

API: This section discusses API. Clients can use API functions. Implementations and functions must be reviewed to identify future input streams, processing stages, and output streams. This level comprises WebRTC's abstract signaling, which initiates media exchanges with participants.

Transport: This module handles transport-related functions. The manner in which data is transmitted over the network is the factor that has to be considered with WebRTC compatibility in the highest regard. It brings up concerns with ICE, in addition, to STUN, TURN, RTP, and SRTP.

Processing of streams: This module handles MCU's media processing. It mixes, processes, decodes, and encodes. In addition to this, it is able to analyse numerous streams simultaneously and generate various outputs, all of which will be transmitted through the transport. As was mentioned earlier, the structure of this module is responsible for the separation of the process into mechanisms for processing video and audio respectively. Each of them is equipped with a single or many decoders, mixers, and encoders. In the end, all that a video/audio mixer is utilised for is to produce streams that are comprised of data for both audio and video.

Control of Stream Processing: This section concludes the overall discussion on functions, often known as API-specific functions. It has full control over both the transport layers and the stream processing layers. It establishes connections between the streams that are received through transport and the stream processors that are correctly organised.

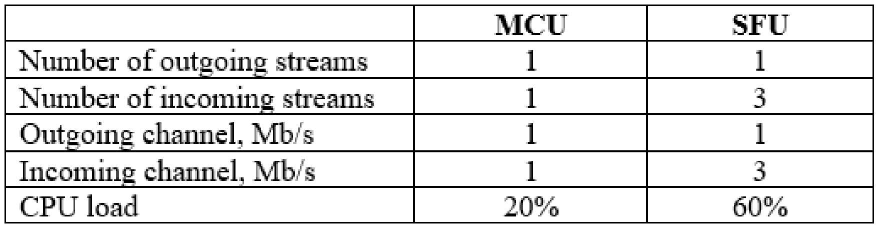
C. WebRTC-MCU Implementation

In this research study, the authors have used an MCU media server to enable browser-to-browser interaction while ensuring scalability. As previously stated, the MCU is a centralized model in which user streams are processed, encoded, and mixed

before being sent back to each peer as a single composite stream. Sending only one stream decreases CPU utilization and bandwidth for each peer, making the MCU architectural pattern more scalable.
After considerable trial and error, the writers succeeded in creating a working prototype of the Multiple Communication Unit application. The transportation component receives the utmost attention during the first stage since it is the one that is responsible for ensuring that all of the requirements are met. Most crucial is making it compliant with WebRTC and adhering to it. We are using well-known open-source C libraries for the SRTP and ICE, such as libsrtp and libnice. Because WebRTC is so new, dealing with it has its own challenges. As standards and applications improve, the environment is in constant motion. There's also no way to strictly follow the requirements. At this point, MCU must improve communication and react to continuing change in order to analyse novel variants or a well-known standard. Currently, our prototype supports the retransmission of a WebRTC stream to other communication participants.

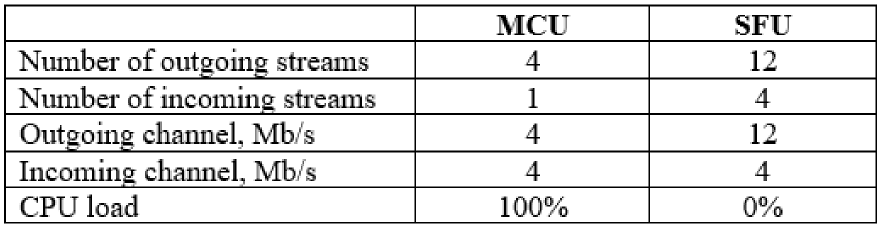
V.RESULTS AND DISCUSSION

Here, the authors have created a comparison table which compares MCU and SFU architecture performance considering the 4-people video conference. The observations in Figure 3 and 4 clearly show that CPU load is lesser (20%) for MCU on the client-side so this architecture is suitable for a large number of participants (>40). While on the server-side CPU load when using MCU architecture is a lot higher (almost 100%) due to the extensive task of mixing and encoding incoming streams from each user.
If the processor load is more than 80% and/or the CPU load reaches 100% on a regular basis, the server is overloaded and lacks the processing capability to complete work tasks, which will inevitably result in stream deterioration.



	MCU	SFU
Number of outgoing streams	1	1
Number of incoming streams	1	3
Outgoing channel, Mb/s	1	1
Incoming channel, Mb/s	1	3
CPU load	20%	60%

Figure 3: Comparison between MCU and SFU on the client-side



	MCU	SFU
Number of outgoing streams	4	12
Number of incoming streams	1	4
Outgoing channel, Mb/s	4	12
Incoming channel, Mb/s	4	4
CPU load	100%	0%

Figure 4: Comparison between MCU and SFU on the server-side

In Figure 5 authors have shown the energy consumption of MCU and SFU topologies on the client-side where the energy consumption of MCU architecture is 0.6 units and that of SFU architecture is 0.8 units. This shows MCU works better on the client-side and the client uses the least amount of CPU since it only needs to decode one stream coming from the media server. A jitter buffer is required because of the increased delay produced by media server decoding.

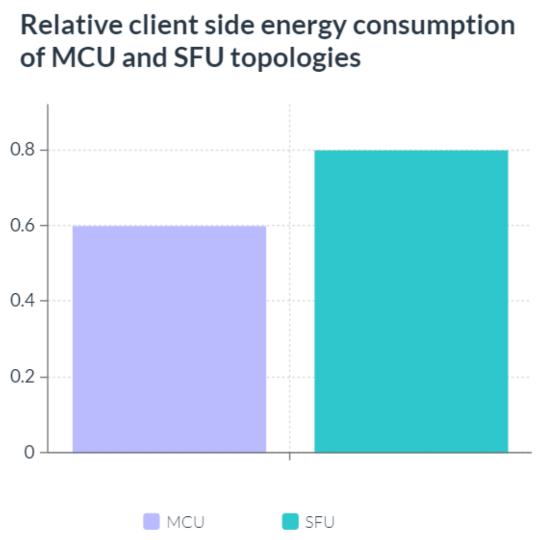


Figure 5: Client-side energy consumption of MCU and SFU topologies

On the server-side, Figure6 illustrates the gradual rise in CPU usage. The rapid variations react to instant changes in the videos that the clients give. At second 200, whenever the 10th client connects, the CPU reaches 100%. There is no way for the session to expand and accept a new participant in this non-distributed scenario. If we pushed another client to join, the CPU would become saturated, resulting in packet loss and lowering the overall communication quality.

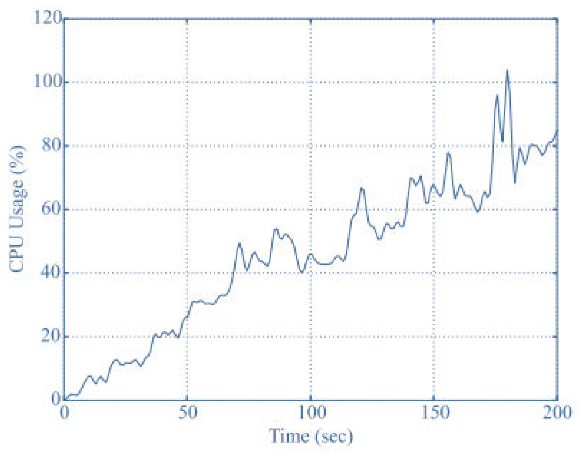
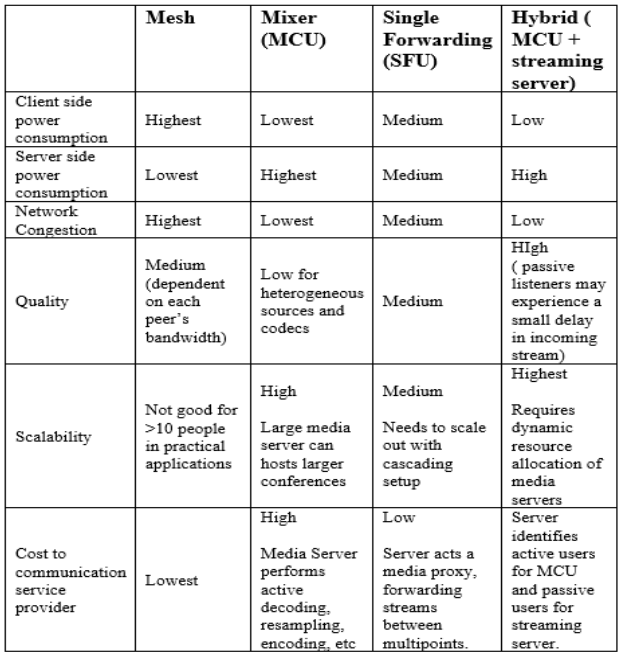


Figure 6: Gradual rise in CPU usage over time in MCU topology.

The authors have also made a comparison table which clearly compares the different topologies on the basis of several important factors. Here is the table:



	Mesh	Mixer (MCU)	Single Forwarding (SFU)	Hybrid (MCU + streaming server)
Client side power consumption	Highest	Lowest	Medium	Low
Server side power consumption	Lowest	Highest	Medium	High
Network Congestion	Highest	Lowest	Medium	Low
Quality	Medium (dependent on each peer's bandwidth)	Low for heterogeneous sources and codecs	Medium	HIgh (passive listeners may experience a small delay in incoming stream)
Scalability	Not good for >10 people in practical applications	High Large media server can hosts larger conferences	Medium Needs to scale out with cascading setup	Highest Requires dynamic resource allocation of media servers
Cost to communication service provider	Lowest	High Media Server performs active decoding, resampling, encoding, etc	Low Server acts a media proxy, forwarding streams between multipoints.	Server identifies active users for MCU and passive users for streaming server.

Figure 7: Comparison between different WebRTC topologies.

VI.CONCLUSION AND FUTURE WORK

The performance and scalability of MCU architecture for WebRTC applications have been thoroughly examined in this research. This is a centralised and integrated video conferencing architecture based on the Multiple Communication Unit of the WebRTC ecosystem. Although the majority of this research focuses on the scalability of WebRTC utilising MCU architecture and the challenges that MCU may face when it comes to designing streaming platforms where there is low latency, P2P, SFU, and MCU each have their own set of advantages and disadvantages. Simple video chat programmes that support two to four participants can work effectively with P2P. The scalability of WebRTC is addressed by MCU and SFU. With a modest number of concurrent participants, SFUs function effectively in multiparty videoconferencing applications. An MCU is essential for large conference systems and other circumstances where many user streams must be shown on a single WebRTC client.
Mixing streams demands a significant amount of processing power, which boosts the server's maintenance expenses. This design is appropriate for meetings with a large number of people (above 40). Because of server-side processing, MCU is also a good alternative for users who need to watch video feeds on low-powered devices like phones.
Despite having a first MCU implementation, the Authors intend to expand the functionality of this MCU implementation by developing new features such as video and audio stream processing for supporting various devices (mobile phones, tablets, and desktops), gateways for communicating with WebRTC clients such as SIP, XMPP [9], or even H.323 [10] clients, session recording, and streaming. In future, the authors are also intending to find out the various ways to minimize the resource utilization by MCU on the server-side to a certain level on the basis of this analysis.

REFERENCES

[1] Simon Holm and Alexander Lööf, "The Design and Architecture of a WebRTC Application", Independent thesis Basic level (Bachelor's Degree), Malmö University, 2019
[2] NaktalMoaid Edan, Ali Al-Sherbaz, and Scott Turner, "Design and assessment of browser-to-browser video conferencing in WebRTC", School of Science and Technology, The University of Northampton, 2017

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.54|

| Volume 5, Issue 5, May 2022 |

| DOI:10.15680/IJMRSET.2022.0505032 |

[3] Florian Rhinow, Pablo Porto Veloso, Carlos Puyelo, Stephen Barrett, and Eamonn O Nuallain, “P2P live video streaming in WebRTC”, School of Computer Science and Statistics, Trinity College Dublin, 2014

[4] Kwok-Fai Ng, Man-Yan Ching, Yang Liu, Tao Cai, Li, and Wu Chou, “A P2P-MCU Approach to Multi-Party Video Conference with WebRTC”, International Journal of Future Computer and Communication, Vol. 3, No. 5, October 2014

[5] BoniGarca, Luis López-Fernández, Micael Gallego, and Francisco Gortázar, “Kurento: The Swiss Army Knife of WebRTC Media Servers”, IEEE Communications Standards Magazine, Vol. 1, Issue: 2, 2017

[6] WajdiElleuch, “Models for Multimedia Conference between Browsers based on WebRTC”, Dep. Computer Science and Applied Mathematics National Engineering School of Sfax University of Sfax, Tunisia, IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), October, 2013

[7] B.A. Jansen, “Performance Analysis of WebRTC-based Video Conferencing”, PVM 2016-086, Master of Science in Electrical Engineering, Delft University of Technology, August 8, 2016

[8] Johnston, Alan, John Yoakum, and Kundan Singh. "Taking on WebRTC in an enterprise." IEEE Communications Magazine 51.4 (2013): 48-54

[9] P. Saint-Andre, 2011. RFC: 6120 Extensible Messaging and Presence Protocol (XMPP): Core. IETF RFC.

[10] ITU-T, 2009. Rec. H.323, Packet-based multimedia communication systems. H SERIES: AUDIOVISUAL AND MULTIMEDIA SYSTEMS.