Schahin Rajab, sr2@kth.se

6 October 2015

Comparing different network topologies for WebRTC conferencing

Table of contents

Abstract

This study compares a mesh and star (bridged) architecture for conference calls using the WebRTC framework. Measurements showed that a mesh network worked well for a small group chat of up to 4 people, but for a larger conference, a bridge that mixes the audio streams is highly recommended

1 Introduction to WebRTC

Web Real-Time Communication (WebRTC) allows web browsers (which supports HTML5) to communicate directly with each other, i.e., peer-to-peer. This means that browsers can transfer media in real time directly between each other. To set up a connection between the browsers, an optional signalling protocol such as Session Initiation Protocol (SIP) is used. However, other than the signalling, everything else is communicated in a peer-to-peer fashion. This means that the user does not need to instal any plugins, they simply set up a connection and start stream their video, audio, data or other directly with each other. Since the communication does not go through any server, unlike the typical browser-server model, but goes directly between clients, the latency is very low.

WebRTC is still in development and was released as an open-source project in 2011 by Google[1]. Since then, the code has been going through a lot of work to standardise what protocols should be used, what codecs should be selected, and so on. WebRTC only works in browsers with HTML5 support, which of October 2015 includes Microsoft Edge[2], Firefox[3], Chrome[4], and Opera[5]. On 22nd of September 2015, Mozilla released a new update of Firefox with a new feature called “Firefox Hello” integrated in their browser[6]. Firefox Hello is an instant messaging client based upon WebRTC. Another messaging client that makes use of WebRTC is Facebook’s own “Messenger”[7].

It take only a lines of JavaScript code to write a chat application using the WebRTC framework. The first step is for the the local browser to get access to the the user’s videocamera and microphone. This is performed using a function called *getUserMedia*. The getUserMedia API takes three parameters, one of which is constraints. This argument restricts what media source will be used. One can use the RTCDataChannel API to send data between peers. The second step is to use the RTCPeerConnection API to establish a connection between the web browsers and start transferring media.

2 Main components of WebRTC

The WebRTC API is based on three main components: MediaStream, RTCDataChannel, and RTCPeerConnection.

2.1 MediaStream API

MediaStream represents, as the name reveals, a stream of media. That could be video, audio or both. So a media source such as a microphone provides audio streams. For the browser to obtain a MediaStream, we simply request the getUserMedia() method. When this method is executed, the user will be asked for permission to allow access to the media source. If the user accepts the request, the media source will be attached to an element in the HTML5 file and displayed on the local browser.

With getUserMedia we can input a constraint as a parameter to specify what type of media we want to return from the user, the resolution we want to display, the frame rate and so on. The other two arguments this function is required to take in is successCallback, which is invoked when the user gives his or her permission to the browser getting access to the media source, and errorCallback, which is called when the browser is denied access to the end users media sources..

2.2 RTCDataChannel

The DataChannel API allow web browsers to send arbitrary data with each other. Since the data is being exchanged in a peer-to-peer fashion, the latency is very low. Using a method called createDataChannel(), we can configure the reliability of the data. So for example, if we set {reliable: true} as an argument, the data will be transferred using the Stream Control Transmission Protocol (SCTP). SCTP is similar to the Transmission Control Protocol (TCP) but is considered to be more reliable. This is good to use if we want to ensure that no packet gets lost by retransmitting these packages like for example when we send a file, we are going to want the whole file to be transmitted with no packets being lost.

The data will on the other hand be sent using User Datagram Protocol (UDP) packets if we set reliable to false and also configures the amount of retransmissions and whether the packets are meant to be sent in an ordered way or not. The consequence will then be that we sacrifice the guarantee that every message gets across and in what order they will get there in compensation for speed (as we will limit the number of retransmissions using maxRetransmits or set a time during which retransmission are allowed using maxRetransmits. These two properties cannot be set simultaneously[8, Chapter 3, p.53].). This is ideal during gaming when we are more concerned about sending data in real-time than we are about reducing packet loss.

2.3 RTCPeerConnection

We use RTCPeerConnection to send over the stream that was obtained from getUserMedia. How this works is that we add the stream from the local browser to the local peer connection and send it to the remote peer. When the remote peer receives the local RTCPeerConnection, the stream will be added to their remote peer connection and then displayed on their HTML5 element. RTCPeerConnection handles a lot of different aspects to create a connection between browsers.

RTCPeerConnection creates the Session Description Protocol (SDP) which handles the transport information and what codecs are to be used during the media transfer.

RTCPeerConnection also cancels out echo and reduces noise, manages bandwidth, manages routing through NATs and firewalls, encrypts the data, etc. All of this is handled automatically under the hood by RTCPeerConnection so the web developer will not have to think about any of that.

3 Signalling

In order for the browsers to start exchanging media in a peer-to-peer fashion, they first have to find each other. This is made possible is by signalling which connects the browsers to a server and allows the peer to communicate with the rest of the peers connected to this server. A signalling protocol is used to establish and terminate the connection between peers. This is actually not specified by WebRTC, hence WebRTC app developers can choose any messaging protocol they wish to use; SIP, XMPP, Jingle, or any other protocol to find remote peers.

Signalling makes use of the SDP for gathering the network addresses and port numbers that can be used for the media exchange. Once each browser has sent its own session description object and also received the session description from the other peer's browser the media exchange can begin between the clients.

When one of the clients wants to disconnect from the communication, the client sends a 'BYE' message to the server who then notifies the other client about it and terminates the communication for the disconnecting peer. The client who received the "BYE" message responds back with an "ACK" message to the server.

4 Inter-network connectivity

The Network Address Translator (NAT) is a device made for assigning public addresses to devices inside a private local network. This is to alleviate the depletion of IPv4 addresses and also adds security since the NAT can hide the user's real IP address. This makes it more complicated to route a peer-to-peer communication between web browsers. The devices inside a private network holds private addresses, and it is not possible to make a connection with someone outside the private network without a public address. The solution to this is to use something called Interactive Connectivity Establishment (ICE)[9] and its duty is to gather communication paths between the peers.

4.1 The STUN protocol

STUN stands for Session Traversal Utilities for NAT, and when a client wants to know their public IP address they ask the STUN server. The STUN server responds with the public IP address and now does the WebRTC application know its public address and if the remote peer has also obtained its public IP address, the clients can send media to each other through their remote peer's NAT. If STUN does not know the public IP address for the end user who

requested it, the client will relay the traffic with another protocol known as TURN (Traversal Using Relays around NAT).

4.2 The TURN protocol

If it was not possible for STUN to provide the host with a public IP address when requested, then TURN will address this problem by relaying the traffic through the cloud. And this will always work since it is out in the public internet and therefore anybody can contact it. The reason this is not the first option to use is because this method consumes a lot of bandwidth and STUN is a cheaper option.

4.3 The ICE protocol

ICE finds communication paths between peers. ICE first requests the end user's public IP address from the host's operating system, but if the host is behind a NAT then that method will fail. So the second strategy will be to employ a STUN server and ask for the address. If that also fails then the last remaining method ICE will use is a TURN server to relay the communication. When ICE finds an address, it adds this to the RTCPeerConnection object. ICE also checks the connectivity between the peers.

5 RTP

WebRTC uses RTP traffic. RTP stands for Real-time Transport Protocol and it is a protocol that transmits real time video/audio data over IP. RTP is UDP based since we do not want to increase the delay by retransmitting the data packets with TCP. The audio or video data that is being transmitted is divided into chunks, covered with a RTP header that runs on top of UDP.

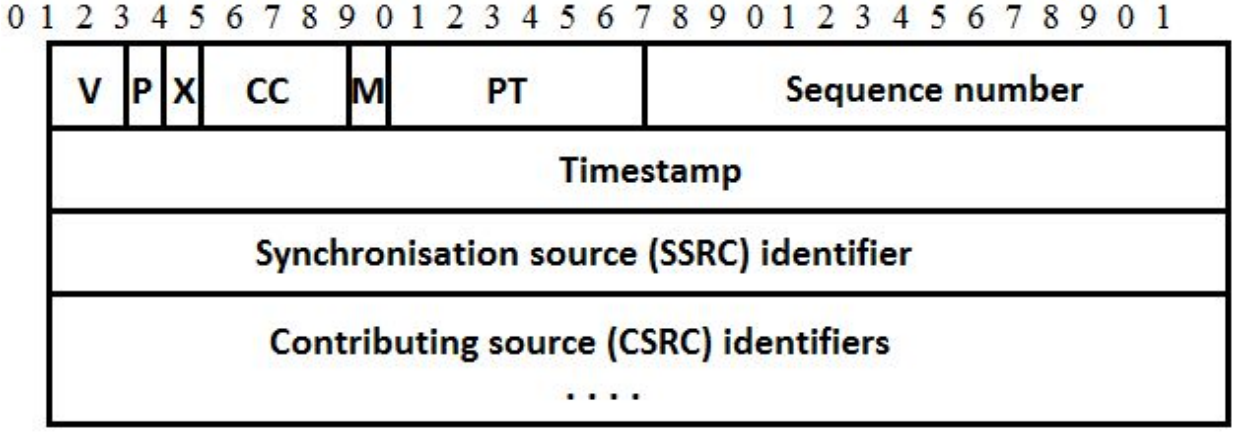

Figure 1, Illustration of a RTP header with al its fields and how many bits each field takes up

The image above represents the header structure of RTP.
“V” stands for the version of RTP.
“P” stands for padding and it is configured with a boolean value; true or false.
”X” is for extension, RTP is an expandable protocol and this is set using true or false.
“CC” Contributing Source Identifiers Count, the number of sources for the data.
“M” Marker, used by specific applications.

“PT” Payload type, indicates what type of data that is being carried, what codec was used.
“Sequence number” the initial RTP data packet that is transmitted has a sequence number that starts from a random number (for security purposes) and increments by one for every packet being sent. This is so the receiver can keep track of which packets was sent first, to avoid replay and to detect packet loss.
“Time stamp” reflects when the packet content was sampled. The timing is related to the sampling rate
“SSRC” Synchronisation Source, identifies the source of the real time stream
“CSRC” Contributing Source, in case the media sources are mixed, CSRC identifies these different sources. It can only list up to 15 sources.

The interesting fields here are the sequence number and the time stamp because this adds reliability to UDP in the same style as TCP without increasing the delay for sending packages that TCP has.

As mentioned before, the main purpose of having a sequence number is to detect packet losses. The sequence number is a 16 bit value and the initial RTP packet has a randomly generated value that gets incremented by 1 for each packet being sent. Since the sequence number only goes up to 65535 because of the bit field size, it will restart from 0 after going past that.

The timestamp helps with playing the media in the correct order. It can synchronise video images with the audio. The timestamp is a 32 bit value and the value for the initial packet is randomly generated. The increment depends on the media.

RTP is made up of two components, the first one is RTP that transmits media data in real-time and the second one is RTCP (Real-time Transport Control Protocol) and it keeps statistics of the quality of the call by controlling the transmitted media data.

5.1 RTCP

The RTP Control Protocol provides feedback about the quality of the transmitted data. To maintain a quality of service (QoS), RTCP sends statistics to each participant in a RTP session of things such as packet loss, packet delay, jitter and so on. With this information, applications can adapt the sending rates and buffer sizes according to the current QoS.

There are different types of RTCP packets that does different things. The interesting ones are the sender report (SR), receiver report (RR), source description (SDES) and BYE. SR and RR sends back reports regarding packet loss, round-trip delay and jitter. SDES sends information about the sending user such as user name, e-mail address, phone-number etc. A BYE message is sent from the end user that is leaving a call.

5.2 SRTP

SRTP stands for Secure RTP and provides security to RTP and RTCP traffic. SRTP utilises AES (Advanced Encryption Standard) to encrypt the data. For providing RTP with message authentication it makes use of the HMAC-SHA1 algorithm[10]. And for protection against

replay attacks, the receiver compares the sequence number of the received RTP packet and verifies that it has not been played before.

6 Network topologies

WebRTC is not only about communicating in peer-to-peer, it can also be used to make a multiparty call, aka a conference call. When dealing with a multiparty call we must choose an architecture for our application. This is an important decision because how the users are arranged will play an important role in how well the conference will scale. The two most common network topologies are the mesh network and the star network.

6.1 Mesh

The most common topology is called a mesh where every peer in the network sends their data to the rest of the peers in the same network. A full mesh network, every peer establishes a connection with every other peer in the network, hence there are n*(n-1) number of connections where n is the number of peers. For example, a full mesh network with 4 users has 12 connections.

No servers are needed for this to work, which makes this an inexpensive option to use. The downside with this is that as the number of participants increases, a lot more bandwidth and CPU processing will be needed. As a result, this architecture is unsuitable for a large network.

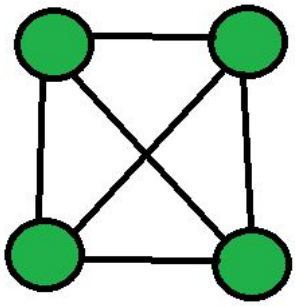

Figure 2. Representation of a fully connected mesh network

6.2 Star

With a star network architecture, we choose the most capable device as the hub of the network. The central hub receives all of the data for the conference call and sends a suitably combined stream back to each peer. Unlike a mesh network, this makes it easier for a new user to join or leave the network.. The star can mix the individual streams and then distribute the mixed stream appropriately. Being able to transmit a single stream reduces the bandwidth and CPU usage for each of the peers in the network, at the cost of a lot of processing by the hub (often called a conference bridge).

The drawbacks to this topology are that the network depends on the hub, if it goes down, the whole network will go down. The number of peers that can join the network and the amount of data that can be shared depends solely on the capacity of the hub.

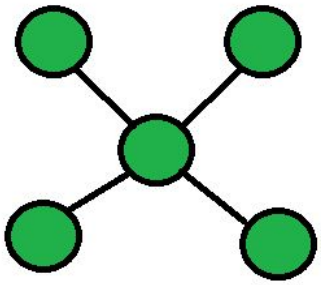

Figure 3. A representation of a star network

6.3 MCU

Multipoint Control Unit (MCU) and it is similar to the star topology with the exception that instead of using one of the user's device as the central hub, we will be using a server that is optimised for processing real time data instead. So this will make the processing easier, while just as in the star case reducing the network load for all of the peers in the network. However, the cost for the MCU makes the network more expensive.

7 Comparison between a mesh network and a star network

This study compares how well a mesh network scales with an increasing number of participants in a conference call and compared the results with how well it scales in a star network.

7.1 Method description

Connected two computers (I used a laptop and a desktop computer) to form a mesh network. I found that https://appear.in/ has a mesh achiteture built to it. appear.in is a group chat application that makes use of the WebRTC API to allow a multiconference call between browsers without having to instal anything or having to register an account[11]. I simply insert a chat room name and it creates a chat room with my chosen name and connects my web browser to this room. Anyone who knows my room name can connect. When a user tries to connect to the chat room, the browser will first ask the user for permission to gain access to the webcamera and microphone. The user has to accept this request in order to join the chat room.

As I was only interested in sending audio streams over the network, I only allowed the browser access to the audio source of both computers and not the video source. To simulate a conversation in the chat room I had a mp3-player placed next to the desktop computer and set it to play John F. Kennedy's speech "We choose to go to the Moon".

I measured the CPU usage using the Windows Task Manager on the laptop before and after the connection was established. I also used a tool called DU Meter[12] to measure the bit rate by adding together the download rate with the upload rate. To increase the number of peers, I simply opened new browser tabs on the desktop computer and connected these tabs to the chat

room. And after each new connection to the room was made, I measured how that affected the CPU usage and the bit rate on the laptop computer.

Appear.in states that their application can only handle up to 8 participants in the same room simulaneosly before the room becomes full, so my idea was to increment the number of participants until I reach that cap, or until the local CPU reaches its maximum load.

I could not conduct measurements on the desktop computer because my laptop was not capable of handling so many open tabs. Otherwise I would have done so as well to compare how each computer performed. The laptop was equipped with AMD A6-3420M 1.5 GHz processor and 4 GB of RAM.

After doing measurements on appear.in, I switched to an application that used the star architecture. https://meet.jit.si is such an application. Jitsi Meet is a WebRTC bridge that receives audio from every participant in the chat room and then relays this to every user in the network. Jitsi Meet does not have any limitation on the number of users allowed in one chat room[13].

Jitsi Meet actually has a tool that generates traffic called “Jitsi-Hammer”. What Jitsi-Hammer does is that it can send RTP traffic from fake users in a Jitsi Meet conference call[14]. This sounded like the perfect tool for my experiment, but after attempting for days trying to make the program start without any success I had to leave it behind me and returned to my previous method again of using a secondary computer to create new participants in the voice conference call.

I performed the same measurement on Jitsi Meet as I did with in appear.in, I used the desktop computer to create new connections to the group chat and I measured the CPU usage and the network activity on my laptop.

7.2 Results

The mesh network consumed a lot more CPU, and network activity compared to the star network. In the mesh network test, I did not try connecting an 8th participant to the room since the CPU usage was already at its maximum capacity and the latency was also very high. However, the star network had no problem allowing more users to join the room.

The two graphs below represents the measurements I did on the CPU usage and the bit rate for the two topologies. The first graph is the discrete graph of the CPU usage where the star network was pretty much on the same level (18%-20%9 independent of the number of users. And the second discrete graph is of the bit rate. The bit rate is a sum of the upload rate and the download rate. The bit rate for the mesh network got higher than the star network after a 4th user joined the chat room, and it only continued to grow at an increasing rate after that-

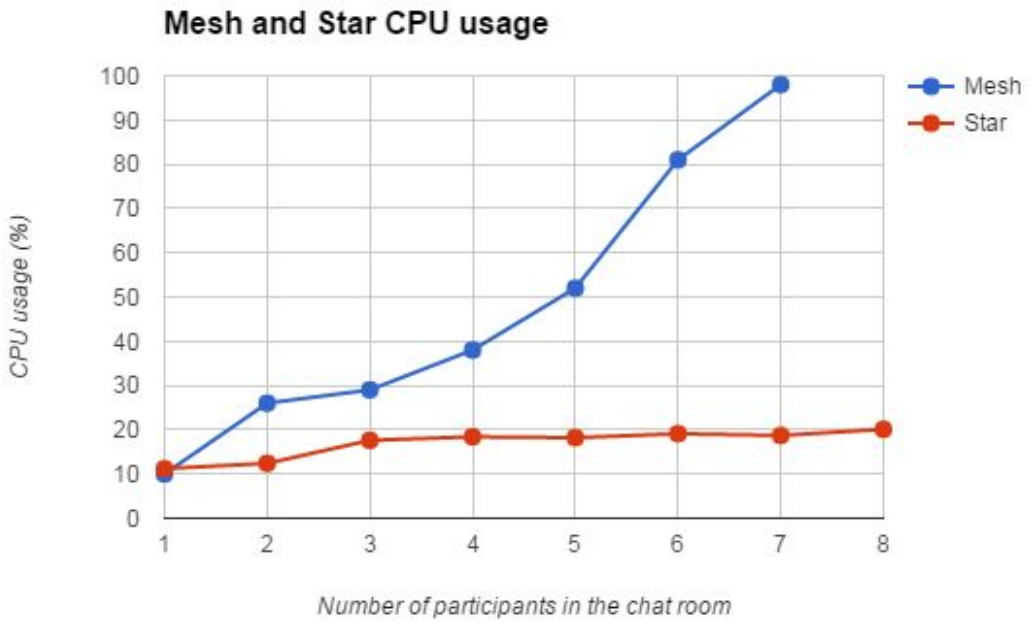

Figure 4, Discrete graph over the CPU usage measured in percentages for the two networks

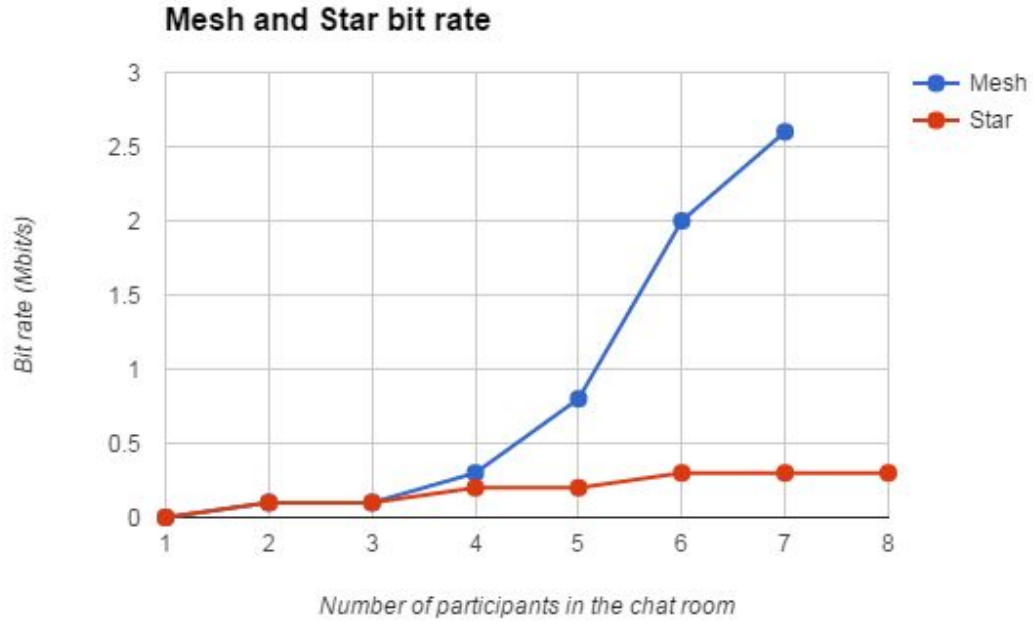

Figure 5, Discrete graph over the bit rate measured in Mbit/s for the two networks

7.3 Analysis

The star architecture was much more scalable compared to the mesh network. The mesh network used a lot more CPU power and the laptop did not manage to handle more than 7 users in the group chat. The laptop handled a larger number of users much better when it was using the star topology.

The mesh conferencing call worked fine for 2-4 users in the group chat, but the latency grew quickly very high after that. Because of the mesh architecture, sending audio to every user in the room consumes a lot of bandwidth since the sender has to distribute the audio stream to each user. Therefore, the more users in a mesh network, the more bandwidth will be required to send data over the network. And also, since the browsers will have to do all of the encoding and decoding, the CPU load becomes a lot higher for each user as well. This explains the mesh network's high CPU usage and bit rate in this study.

Meanwhile, for the star network I used an application with a conference bridge where there is a central hub in the network receiving audio streams from every sender and then delivers a stream each to every user. Hence, the low CPU load and bit rate no matter how many users joined the group chat in this study.

.

A more sophisticated method (if I had 8 computers at my disposal) would be to connect to a group chat instead of using multiple tabs on the same computer. Had I had more computers with different specifications I could have measured how each computer performed and then compared the results. Also as I mentioned before, if I could have gotten Jitsi-Hammer to work, this would have been a better way of generating RTP traffic for my test.

8 Conclusions and future work

My work focused purely on the CPU usage and network activity for a low-end laptop during a voice conference call. I have shown from my test results that a mesh network works fine for a smaller group chat of around 4 people, but for a larger conferencing, a MCU with a server that mixes the audio streams is highly recommended.

For future work one could compare the packet loss rate for each network architecture. And as I have previously explained, generating RTP traffic from fake users would get a more accurate view on how these network topologies actually scales. Also, I have limited myself to audio conferencing call, it would be interesting to see how much CPU usage and bit rate would be affected for a video conference call and/or sending text messages in a group chat.

WebRTC is still in development stage, a lot of standardisations of the protocols remains to be made by the IETF (Internet Engineering Task Force) and the APIs by W3C (World Wide Web Consortium). Maybe upcoming changes will improve the performance and the WebRTC experience overall for end users.

References

[1] Harald Alvestrand, 'Google release of WebRTC source code', 31-05-11 [Online]. Available: http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html. [Accessed 05-10-15]

[2] Microsoft Edge Team,'ORTC API is now available in Microsoft Edge', 18-09-15 [Online]. Available: https://blogs.windows.com/msedgedev/2015/09/18/ortc-api-is-now-available-in-microsoft-edge/. [Accessed 05-10-15]

[3] Team Firefox, 'Firefox notes - desktop', 25-06-13 [Online]. Available: http://website-archive.mozilla.org/www.mozilla.org/firefox_releasenotes/en-US/firefox/22.0/releasenotes/. [Accessed 05-10-15]

[4] Google Chrome Team, 'Chrome - WebRTC', [Online]. Available: http://www.webrtc.org/web-apis/chrome. [Accessed 05-10-15]

[5] Magnus Peter Langeland, 'Download Opera 18 for desktop', 19-11-13 [Online]. Available: http://blogs.opera.com/news/2013/11/opera-18/. [Accessed 05-10-15]

[6] Steve Dent, 'Firefox's latest browser has built-in instant messaging', 23-09-12 [Online]. Available: http://www.engadget.com/2015/09/23/firefox-41-browser-instant-messaging/. [Accessed 05-10-15]

[7] Tsahi Levent-Levi, 'Forget Whatsapp - Facebook Messages goes WebRTC - big time', 13-04-15 [Online]. Available: https://bloggeek.me/facebook-messages-webrtc/. [Accessed 05-10-15]

[8] Salvatore Loreto and Simon Romano, Real-Time Communication with WebRTC: [peer-to-Peer in the Browser], 1. ed. Sebastopol, Calif.: O'Reilly & Associates, 2014, ISBN: 978-1-4493-7187-6.

[9] Jonathan Rosenberg, 'Interactive Connectivity Establishment (ICE): A methodology for Network Address Translator (NAT) traversal for offer/answer protocols', April 2010 [Online]. Available: https://tools.ietf.org/html/rfc5245. [Accessed 05-10-15]

[10] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. 'HMAC: Keyed-Hashing for Message Authentication', Internet Request for Comments, vol. RFC 2104 (Informational), Feb. 1997 [Online]. Available: https://tools.ietf.org/html/rfc2104. [Accessed 05-10-15]

[11] 'Appear.in – one click video conversations'. https://appear.in/. [Accessed 05-10-15]

[12] Hageltech, 'Download DU Meter', 03-08-15 [Online]. Available: http://www.hageltech.com/dumeter/download. [Accessed 06-10-15]

[13] 'Jitsi Meet | Jitsi'. https://jitsi.org/Projects/JitsiMeet. [Accessed 05-10-15]

[14] 'Jitsi/jitsi-Hammer'. https://github.com/jitsi/jitsi-hammer. [Accessed 05-10-15]