Tree topology networks in WebRTC

An investigation into the feasibility of supernodes in WebRTC video conferencing

JOHAN GRÖNBERG

ERIC MEADOWS-JÖNSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, May 2014

Tree topology networks in WebRTC
An investigation into the feasibility of supernodes in WebRTC video conferencing

JOHAN GRÖNBERG
ERIC MEADOWS-JÖNSSON

Examiner: LAURA KOVACS

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden May 2014

Abstract

Video conferencing is important in today's society, but most popular solutions require a user to install a program, plugin or similar. With the release of Web Real-Time Communications - WebRTC it has become easy to create video conferencing solutions that work with WebRTC supported browsers, such as Chrome, Firefox, or Opera without any addons. However due to the peer to peer nature of WebRTC it becomes difficult to scale as the technical requirements on all clients grow with each participant.

This report aims to examine how to use supernodes in WebRTC based video conferencing to shift CPU and networking load within a network and how this affects service quality. It then examines the theory behind WebRTC and supernodes to be able to implement solutions that use these concepts within a video conferencing solution. This project starts from an existing WebRTC project and implements a statistics gathering algorithm as well as two implementations using supernodes.

By utilizing the results of the statistics gathering the original service is compared with results from the supernode service to evaluate the impacts on bandwidth and video resolution. We show that networks using supernodes do redistribute bandwidth and can achieve higher resolution quality in conferences given proper supernode selection. Finally, we have identified needs for further research into optimal supernode selection to achieve ping optimization, TURN usage minimization and other efficiencies.

Acknowledgements

We would like to thank our friends for always giving helpful advice and acting as a sounding board whenever we needed it. Our families for their support and understanding while writing this thesis. We would like to thank Svein Willasen for proposing our thesis topic as well as being a great guide and tester whenever we needed it, also all the developers at appear.in for answer our many questions. Many thanks are also given to Thomas Grönberg and Marianne Söderberg for giving feedback on the report. We are also grateful to our examiner Laura Kovacs for all the help and insight that she has provided us. Finally we wish to thank our opponent Jakob Kallin.

Contents

Glossary

DHT Distributed hash table, A distributed system that provides mappings between values similar to a normal hash map. The responsibility for maintaining the different mappings is however distributed amongst the clients..

ICE Interactive Connectivity Establishment, A protocol used for NAT traversal for protocols that use the offer/answer model..

MCU Multipoint control unit, a device that bridges two or more clients together in a video conference..

NAT Network address translation, is the translation of a global IPv4 address to another address which can be used on a local network..

STUN Session Traversal Utilities for NAT, STUN is a protocol that is used by other protocols in the NAT traversal process..

TURN Traversal Using Relays around NAT, an Extension of the STUN protocol to allow for Relays..

W3C World Wide Web Consortium, A consortium that produces and publishes international standard and works for the growth of the Internet..

WEBRTC Web real time communications, A specification aimed at bringing telecom functionality to the web. This standard contains APIs as well as other standards and protocols..

1 Introduction

Most people today are in contact with some form of video conferencing at some point in their life. People have started traveling more and over a wider area than before. With the use of video conferencing people who are traveling, or people who live in different parts of the world and want to keep in touch are able to do so face to face. With the increased usage and also improvements in technology we can today use video conferencing software in our tablets, cell phones, and even our cars, an achievement that would not have been possible even just a few years ago. Video conferencing is also important for businesses, it is frequently used to conduct remote interviews, to talk to colleagues that are travelling and to conduct meetings with remote offices.

The applications used for private and professional video conferencing differ quite widely, mainly because of different usage patterns and demands from the markets. The private market wants something that is easy to use and requires no setup, has good quality and just works. The professional market on the other hand is not as concerned with setup and usage, but is more concerned with reliability and security. This is so because there might be catastrophic results if something were to fail, and even worse if the conference or the computer system were to be compromised.

Today video conferencing continues to grow. Recently Web Real-Time Communications - WebRTC was released which made it possible to have real time communications in any browser that supports WebRTC, without the need for any plugins or third party software. WebRTC makes video conferencing available to anyone that has a modern web browser and makes it easier for people to use. Also, the release of this technology makes it easier for independent developers and companies to develop custom conferencing solutions.

1.1 Problem Statement and Purpose

There are many problems for smaller companies trying to make WebRTC based products marketed at consumers. While WebRTC is inherently peer to peer there is still some server infrastructure needed. The servers needed are for signaling, as the WebRTC specification only states that signaling is preformed by unspecified means [1]. The signaling sets up the call between all participants and also handles events, such as a participant leaving the call. There will also be clients that cannot connect by themselves and will need a server to relay all information to and from them. This is a major cost for service owners as there will be a lot information to relay.

Since WebRTC is also inherently peer to peer, the straight forward implementation of the product uses a complete graph topology in which every client is connected to all other clients, as can be seen in Figure 1 below. Nodes in the graph in Figure 1 denote clients and edges denote connections between clients. In a complete graph topology each client has to send a copy of their stream to every other client in the conference. This places a very high demand on all clients for conferences with more than a few participants. The demand is in both processing power and bandwidth. Since clients do not have unlimited bandwidth and processing power this will cause a decline in the users experience, or even stop the service from working when certain thresholds are reached.

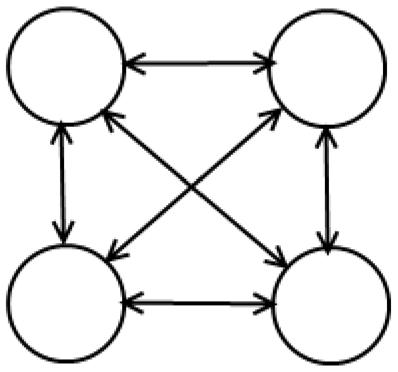
Figure 1: A complete graph; all four peers are connected to each other.

The increased performance requirements on the frontend and hardware and bandwidth requirements on the backend leads to that a video conference service becomes hard to scale. It is hard to scale for the company in regards to server costs. It is also hard to scale the user since users with lower bandwidth and perhaps even lower end computers might be excluded from conferences due to performance issues.

The purpose of this report is to explore and evaluate solutions to front end scalability without negatively impacting backend scalability. This is done in order to allow WebRTC services to be used by more people on more devices while still trying to keep as much of the peer to peer nature as possible, so as to not have to create a larger infrastructure than is needed to support the service.

1.2 Methodology

This project aims at evaluating the impact of changing the network topology used by traditional WebRTC services from the complete graph topology in Figure 1 to a tree topology graph. As can be seen in Figure 2 a tree is formed by all nodes in the graph being connected to each other without having any cycles in the graph.

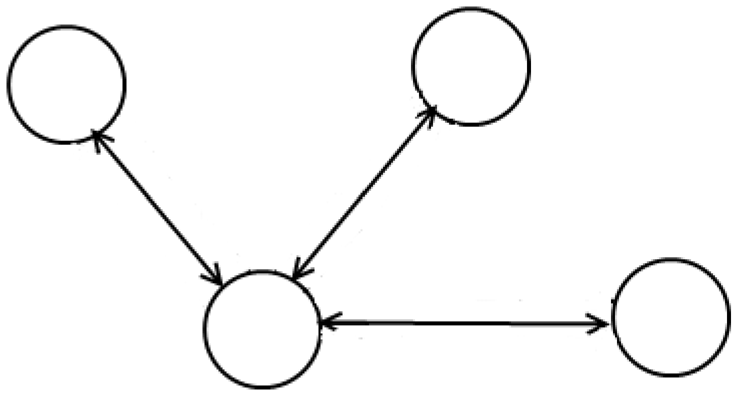

Figure 2: A tree topology graph constructed from Figure 1.

By changing the topology of the graph, as in Figure 2, we think that it would be possible to change and improve on the scalability for both the backend and the user.

Starting from an already existing product, namely https://appear.in, two different prototypes will be developed and then evaluated against the standard implementation. First a very naive proof of concept implementation will be implemented. Other implementations will be based upon this but use different algorithms to construct the tree. All options will then be evaluated and conclusions drawn about what algorithms are good, and which problem areas they address.

The developed prototypes will be compared to the original implementation in these respects:

- Bandwidth usage
- Video resolution
- Perceived latency
- Service usability

Some of these points are very subjective while others are purely data points that can be extracted and viewed. Data will be gathered directly from WebRTC in the browser without affecting the user experience and there will also be a verbal follow up with all testers about the subjective data points.

1.3 Related works

In the summer of 2013, there was a bachelor project at the Delft University of Technology in the Netherlands using WebRTC to implement a browser based game [2]. This project used supernodes to build the network that was used to communicate between nodes. It was aimed at gaining an understanding of WebRTC and using it to implement a game rather than to explore the effect and the impacts of supernodes on a WebRTC network. Furthermore this was a pure data network while our report describes a network used specifically for video conferences.

Another related approach to our work is a software solution called Licode [3] that can be used to communicate between multiple clients and create video conferences similar to the ones in this project. In the same way as in this project, the usage of licode takes some work away from clients since the clients only have to upload one stream no matter the number of participants. However, as opposed to using supernodes, licode uses a multipoint control unit(MCU). The MCU solution requires a dedicated server to relay streams and adds to the backend load. Our project however is focused on distributing that effort amongst clients in the conference themselves such as to alleviate load on nodes as well as not adding any load on the backend.

1.4 Report structure

This report consists of ten main chapters. In Chapter 2 the technical background for the project is explained, mostly focusing on the technology used, specifically WebRTC, but also the basic graph concepts used in this report. In Chapter 3 the problem description and purpose presented in Section 1.1 is developed giving a more thorough explanation of the actual problem. Both a high level and a detailed description is provided. The purpose of this project is also explained in greater detail, as to why this has been done, and why performing this comparison is of interest. Further theory needed to conduct this project is explained in Chapter 4. In Chapter 5 the method applied in the project is explained. There are a few different aspects that are covered, the development, the technology used in this report, as well as how the testing and gathering of data is undertaken. The different implementations that have been developed over the course of this project are described in Chapter 6 while Chapter 7 provides the results gathered as well as analysis of them. Chapter 8 discusses our results, Chapter 9 discusses social and ethical aspects of this thesis, and Chapter 10 concludes this report.

2 Background

This chapter overviews the technical background of the thesis, including peer-to-peer communications, network topologies, session protocols, network connectivities and WebRTC.

2.1 Peer-to-peer

Peer-to-peer is a decentralized form of network communication. There are many types of Internet communication where a client connects to a central server. HTTP, which is used when accessing a website, is one example of that; a client, the browser, connects to a server, the website. With peer-to-peer communication there is no centralized server. All clients connect directly to each other without going through a server.

2.2 Network topology

In graph theory, there are many different graph topologies. There are however two basic topologies that have a very relevant connection to this report.

2.2.1 Complete Graph topology

A complete graph is a very simple graph topology in which every two nodes are connected to each-other [5]. This topology can occur in both directed and undirected graphs. In complete directed graphs every two nodes are connected via two edges, one for each direction, whereas in undirected graphs there is only one edge that connects the nodes in both directions. There is one distinct complete graph for every number of nodes. In Figure 1 the complete graph for four nodes is represented. This is also referred to as K^4.

2.2.2 Tree topology

A tree graph is a graph where all nodes in the graph are connected, meaning that it is possible to get from any node, to any other node in the graph, and in which there are also no cycles. A cycle is when it is possible to go from one node and back to that node without using the same edge twice. An example of a such a tree is shown in Figure 2.

2.2.3 Supernodes

In an application that uses a graph representation, and thus uses nodes, a supernode is a node that is given a special role within the network. As described in [6] there are many different uses for supernodes in applications today. How much extra responsibility or power that is given to the supernode also varies greatly. In peer to peer applications such as Skype [7], Kazaa[8], supernodes are used to store information about the network and other nodes in order to aid in searching through a large network. Supernodes can also be used to forward data from one node to another. This is very close to the usage of supernodes in this project, except that instead of only forwarding information, the node also uses the information itself in order to display video.

2.3 Session description protocol

In establishing a session between two parties a great deal of information is needed from both parties to establish the content and terms of the session. The session description protocol (SDP) provides a standard way of representing and presenting this information. There is however no transport or negotiation protocol included in SDP, and as such the transport and negotiation of a session using SDPs has to be done separately [9].

2.4 Web Real-Time Communications - WebRTC

WebRTC, which stands for Web Real-Time Communications, is a collection of multiple standards that aim at providing real time communications on the web. These standards contains protocols, specifications and APIs [10]. WebRTC is very new and still under development, therefore it is not implemented in all browsers. It is however implemented in working condition in Firefox, Chrome, and Opera. While this technology is not supported by all major browsers, from the data in Table 1 it can be seen that WebRTC is available to approximately 52.8% of web users through these three browsers alone. The WebRTC standards are being maintained by W3C, and are widely available online. A part of the WebRTC standards is dedicated to provide a Javascript API that gives easy access to a computers media, as well as peer connection right in the browser. WebRTC does not only aim at making real time communications functionality available in the browser, but also the other way around, to make web functionality more available for telecommunications.

Browser	Market share(Percent)
Chrome	37.2
Internet Explorer	18.3
Firefox	18.1
Safari	16.6
Opera	2.9

Table 1: Browser market shares in March 2014, gathered from over 74000 websites [11]

2.5 Network connectivity

This Section explains how to establish connections to other peers with WebRTC and the protocols required to traverse NATs.

2.5.1 Network address translation (NAT)

Network Address Translation (NAT) is the process of modifying the address of an IP packet while in transit. With IPv4 there are often multiple machines sharing a single IP address on the Internet. A router handles address rewriting of packets to send them to the correct machine on the local network. The router maps an internal IP and port pair to an external port when the first outgoing packet arrives. It is hard to connect to a peer behind a NAT because the mapping is set up for outgoing packets, and hence the router does not know which internal peer an incoming packet should be sent to unless a mapping has been set up.

2.5.2 Establishing a connection

To establish a connection around NAT, WebRTC uses a series of protocols. The first protocol is called STUN [12] and is a protocol that is designed to aid in the act of NAT traversal. A computer system can contact a server running a STUN software and the STUN server will return an IP address and a port number. This address and port number can then be used by others to try and connect to this computer as described in Section 2.5.1.

This will however not always work as some NATs cannot be connected in this manner. Therefore the second protocol used by WebRTC is called TURN[13] and is by name an extension to the STUN protocol. TURN allows the IP and port candidates to not point to the calling computer, but instead to a TURN server that will relay all information between two clients in a connection. A TURN server is easy to run as there are available open source implementations based on the standard, there is however a significant cost associated with the bandwidth consumed by the server.

STUN and TURN are however not complete NAT traversal solutions but are instead aimed to be used by other protocols to achieve NAT traversal. Therefore WebRTC uses a third protocol called ICE [14] to establish an connection.

The ICE agent in WebRTC gathers connection candidates consisting of an IP address, a port and a transport protocol such as TCP or UDP. These are exchanged via the signaling channel as well as SDPs in an offer/answer fashion. Once offers, answers, and candidates have been exchanged, the ICE agent in both clients sort the candidates and goes through the list and performs a connection check until a connection can be established.

2.5.3 SSRC

Within an SDP session description there is a value called SSRC that contains an ID which is the ID of the stream source. This ID helps track and correlate a single stream in multiple locations [15].

3 Problem Description and Statement

The following sections describe the problem that is addressed by this thesis as well as explain why this problem needs to be addressed. Further, the limitations placed on the report are also discussed and presented.

3.1 Problem Description

Video conferencing is important today and it only continues to grow. With the release of WebRTC it became possible for new actors to simply and cheaply develop applications that make video conferencing in browsers possible. There are different solutions that consumers can easily use to set up video conferences, for example Facetime, Google Hangouts, and Skype. However, all of these have downsides. Google Hangouts requires a plugin to be installed, Facetime can only be used with Apple devices and only supports two people, and Skype requires the users to install a program on their system.

Therefore it should be possible to create a service that is easier to use, and requires less setup than either of the above options using WebRTC. However, as explained in Section 1.1 WebRTC is inherently peer to peer and constructs a complete graph. This means that this type of conference is not very scalable, as each node will need to send and receive video from all others. A video stream of good quality usually uses over one megabit per second of data [10]. If a conversation is to maintain good quality with many members it will require a large amount of bandwidth from each participant. Having to send and receive this many streams will also place other strains on the clients, such as increased CPU workload from encoding and decoding.

While Internet connections are always getting faster it can be noted that the average Internet connection speed was only 3.6 megabits per second as of 2013 [16]. To reach as wide of an audience as possible, it is then of importance to keep bandwidth requirements as low as possible. There are many ways of doing this, for example Google hangouts directs all traffic over their servers where it is mixed into one video to maintain quality [10]. There are similar solutions already for WebRTC, such as Licode, described in Section 1.3, which is a server that handles all connections.

These solutions however put strain on the backend infrastructure of a service. With the amount of bandwidth used for each stream, video conferencing will require a backend with a great deal of bandwidth and a large amount of processing power for any large service. These requirements makes it hard to have a scalable service, as well as for new services to enter the market as they will not have the server resources to use. It does not matter if the backend is on physical on virtual devices because this still places a strain on the company to pay for the required bandwidth and processing power.

In order to find a solution that works well from the consumer end, which allows multiple participants without too much strain on them, and scalable from the service owners standpoint without the need for a big backend, this project wishes to explore the prospect of changing the network topology from complete graph to tree topology by using supernodes. This should allow for all connections to remain peer to peer, as well as re-distribute the workload within the graph by moving parts of the load from nodes to supernodes while not adding any significant server load.

3.2 Purpose

As described above, WebRTC video conferences in their common implementation have scalability issues both for the peers and the service. The peers need a lot of available computing power and bandwidth to be in a conference with many participants. The service needs a big backend to make the service as smooth as possible for the peers.

The purpose of this project is to explore the feasibility of using supernodes with a WebRTC based video conferencing system. Addressing the scalability issue is important from both the peer and the service owners point of view. For the peer this might not seem significant if they are used to conferences with only two or three participants. However,peers might still need to have a larger conversation sometime, and then they should still be able to, even if their connection or computer is not at the highest standards. From the service owners viewpoint it is however about minimizing the backend power, and therefore reducing the economic cost of launching a competitive service using WebRTC.

This report will answer related questions that arise in the usage and implementation of supernodes in a WebRTC based video conference, as well as suggest a way in which to use supernodes for such services. The report will address the following questions:

- Are tree topology networks a feasible and usable alternative to fully connected topologies?
- What is required for supernodes to work in a common WebRTC system?
- Do supernodes impact on how many peers can participate in a conversation?
- Do supernodes increase the perceived quality of a conversation?
- What is the relation between upload and download speed in conventional user systems?
- What is the impact of node drop outs?

The purpose of answering the above questions is mainly to gain an understanding of how supernodes can be used to change the network topology of a WebRTC video conferencing system. The results and conclusions of this thesis can also serve as a baseline for others who are evaluating this technology, who wish to implement it by themselves or aim to do further investigations into the usage of supernodes within WebRTC based video conferences.

3.3 Ethical and social considerations

By reducing the bandwidth and performance requirements on clients, more clients should be able to use a video-conferencing service in scenarios where it has not been possible before. Further, this might also make is easier for multiple services to co-exist. By placing extra load on supernode clients there are however several ethical concerns that arise. For example, is it justifiable to use extra resources from supernode clients? Also, if a solution is too liberal with whom can be used as a supernode there security of information might also be a concern.

3.4 Limitations

This project is limited to the functionality that is currently exposed to developers through Javascript APIs in browsers.

WebRTC is a new standard, it is not yet fully implemented in browsers and has a largely undocumented implementation. There are numerous things that do not correspond between the implementations and the specification, making it time consuming to develop with WebRTC. This fact coupled with the time frame for the project limited the report to studying and implementing only two solutions and performing a comparison against a reference solution. Other possible solutions are left to further investigations.

4 Theory

This chapter overviews the technical theory used when implementing the solutions. It includes the supernode selection schemes and more in-depth explanations of WebRTC and related APIs.

4.1 Network

Within the context of a network are clients correspond to nodes. This section supernode selection schemes and the differences between nodes that can be used together with the supernode selection schemes.

4.1.1 Nodes

If all the nodes in a peer to peer network had been exactly the same or very similar, a network using supernodes would not be a good solution to the stated problem of this report, as nodes would not be able to help each other. Since in a fully connected topology all peers would process the same amount of data, if one node is overloaded, all nodes should be overloaded since they are all the same.

However, in the introduction to his doctoral thesis, J. Sacha [17] talks about nodes in peer to peer networks. For these types of networks, all the nodes are different from each other in many different ways. The node differences that are most important for our purposes are the processing power and the bandwidth, as the supernodes will have to forward a lot of packets which will take processing power and bandwidth. As mentioned in Section 3.1 the average connection speed is 3.6 megabits per second, which would not support many video streams at the highest quality. The network would still work, but in lower quality, as WebRTC scales the quality of streams by available bandwidth. Bandwidth and processing power usually does not vary by small percentages between nodes, but by one, or even several orders of magnitude. This fact suggests that there should be a good candidates for supernodes in our network.

Another important property of nodes is if they are behind a NAT which cannot be traversed and will need to be relayed to be able to be connected. Otherwise, connections can be established directly.

4.1.2 Supernode selection schemes

To select a supernode from a set of nodes many different selection schemes can be used. The schemes can be grouped into four separately distinct groups [17]. The simplest type of supernode selection scheme is when the selection is not a collaborative effort or a distributed decision, but rather performed by a central server, a higher level application, hard coded or even performed manually. This very simple type of selection is amongst others used by Skype.

The second type is a type of distributed decision where the set of nodes is broken down into smaller subsets depending on varying conditions such as location. These subgroups then elect one or multiple supernodes within their own group.

This makes supernode selection easier since it can be split into many supernode selection problems on all the new smaller sets. These new smaller problems can then be solved individually. Systems with this solutions usually differ on how they construct these smaller sets. This approach is used in an algorithm called PopCorn which was used in the bachelor thesis described in Section 1.3.

Distributed hash tables (DHT) are also sometimes used to select supernodes. This is done somewhat similarly as the previous method, since nodes are broken down into groups based on proximity in DHT space and they internally choose a supernode for that group. The work of J. Sacha [17] notes that one disadvantage of this method is however that running a DHT protocol may introduce significant overhead on the peers. Therefore this is not viewed as an appropriate solution for this project.

The last method described in the thesis of J. Sacha [17] is the so-called Adaptive systems method. In this method, super nodes are elected by a set of rules or values which are important to that specific implementation. This could for instance be that each node has to connect to two supernodes or that a supernode has to have a certain percentage of up time.

4.2 Media capture

The Media Capture and Streams [18] standard maintained by W3C defines several APIs to request media streams from a device. In this project only a small part of this specification is however used. The main usage is the *getUserMedia()* function which, if successful, returns a *mediaStream* object which is then used with WebRTC as described in Section 4.3.

Another part of the standard that is important to this project is the concept of *mediaTrack*s. A *mediaStream* object usually consist of several *mediaTrack*s objects, both audio and video which are the individual audio and video components of the stream. These tracks also have a disabled and enabled state that is reachable and changeable from the API. This makes it possible to control what can be seen and heard, and also to make sure of what kind of media is actually present. Video feeds from *getUserMedia* can also scale and re-size themselves to better fit the conditions in which they are being used.

4.3 WebRTC

This Section explains which APIs are of the most importance for this project, what they do, and how they are used in this project. There are two main areas of the WebRTC API that are of importance, *RTCPeerConnection* for connection handling and *getStats()* for getting statistics.

The WebRTC standard also extends upon the *mediaStream*s discussed in Section 4.2 so that they can be used over networks, and not only locally. The main extension that is of importance for this project is that each *mediaStream* object is now assigned an ID upon creation.

The fact that WebRTC uses media streams does mean that the stream can be automatically scaled due to a lack of bandwidth or processing power. This however only mask a problem instead of solving it. If streams start being downscaled, a lower quality in the video-conference is obtained. This however should be avoided, since it decreases the quality of the service.

4.3.1 Peer connections

There are two main sections to the WebRTC API, one is peer to peer connection while the other is data connections. In this project the peer to peer connections API is used. The sending of streams could also be done with a data connection, but a peer to peer connection turned out to be the best choice for this project.

The peer to peer API is exposed through the *RTCPeerConnection* object. Peer connections use the session description protocol in an offer and answer model to establish a session between two peers. Since no connection is started, the answer and offer has to be sent between the clients in a separate signaling channel.

The main reason peer connections are used for video conferences is the ease with which *mediaStream* objects can be added and removed from the connections. Addition and removal of streams are done with one separate API call each. If successful, it will also trigger an event on the other client informing it about the new stream. Further, as the session has now changed, it will also require a new SDP offer and answer sequence. As many streams as possible can be added to a single peer connection object.

4.3.2 Statistics

The WebRTC standard specifies a statistics model that can be used to gather statistics. This statistic model exposes a JavaScript function called *getStats()* to developers. This method is to be called with a selector which the developer wishes to gather statistics about. When the method is called upon a peer-Connection object with an appropriate selector, a *RTCStatsReport* is returned which is a map between IDs and *RTCStats* objects. In Chrome, statistics are shown in Chrome://webrtc-internals to help debugging WebRTC applications as well as seeing current connections.

5 Method

This chapter describes the research and development methods. Additionally it describes the techonologies used and how testing was performed.

5.1 Research

This project revolves around the WebRTC standard and how to use it. Therefore a significant amount of time in the beginning of the project was spent reading and understanding the specification to understand how WebRTC works and which parts are of most importance to this project.

Since this project is based on the existing service https://appear.in, a significant amount of time has also been spent to understand the codebase of appear.in. This is important for several reasons. The project members needed to gain insight into the structure of the service to understand what needs to be changed or expanded upon and also how this could best be accomplished. Since the project is performed against an existing service, it is also important to understand the service in such a way that the result of the project conforms to the standards and style used within that service.

5.2 Development

While gaining an understanding of WebRTC, small solutions and examples using WebRTC were implemented to test different functionalities needed for this project. This was done to explore how the browser implementations of WebRTC conforms with the standard as specified by W3C as well as to test small scale solutions.

All Web browsers have API differences between them, which pose several important problems to solve. Firefox for example does not support re-negotiations within peer connections as described in Section 4.3.1. This means that it is not possible to add or remove streams from peer connections. For simplicity, the solutions described in this report only work under Chrome; however extending these solutions to other browsers, including, Firefox is an interesting task for future work. It might also be fixed automatically be changes in browser implementations.

In the beginning of the project, the choice was made to keep the code base of this project up to date with the code base of the appear.in service by regularly merging the two. This decision was however later reversed as it was found to be too time consuming to keep up to date against an actively developed product when conducting several experiments that change an underlying nature of the service. We are of the opinion that if this project was to be done again, it would either construct a small service from scratch to test and evaluate, or simply work from a snapshot of the original service with no further interaction with it.

In the sequel, different implementations using supernodes will be described and evaluated against a reference implementation. Results will be collected to draw conclusions about the usage of supernodes, benefits of algorithms as well as suggesting future work.

5.3 Technology used

There are many outside technologies and third party packages used to minimize the amount of unnecessary work that has to be done during the course of the present project.

The main technology used by this project is socket.io for the signaling channel described in chapter 4.3.1, and AngularJS to easily be able to change the front end while using data binding between the front and backend.

5.4 Testing

In the end of this project, the developed implementations were compared against the original appear.in implementation that uses a fully connected graph topology. In Section 3.2 certain points on which the implementations are to be evaluated are stated. These points are both data that can be gathered from the service itself, such as amount of bandwidth used, but also very subjective points, such as perceived quality.

To test this and gather the data that is needed, at first a general survey about video-conferences was sent out. To measure the perceived quality, a verbal follow up was performed with testers to gather information about the different implementations. In order to measure other aspects of the different implementations a statistics gathering algorithm was implemented, as described in Chapter 6. By combining these sources of information the project compared different aspects.

Data is gathered by tests that are preformed by the implemented solutions being used by users as they would use the original service. As such it is believed that the test data is representative of normal service usage. However, as this is a test of supernodes and not supernode selection, a suitable supernode is always used, either by entering the conference first or by opting-in first. All tests were preformed by the same computers on the same networks. Further, in order to avoid any differences in implementations in the *getStats()* API between browsers, all tests were in Google Chrome.

6 Implementation

In order to evaluate the impact of supernodes on a WebRTC network three different tests were implemented and preformed. All the tests are based on the same code from the appear.in service.This is to rule out any outside influence on the values gathered by the implementations described in this chapter. The implementations were all ran on a virtual cloud server with both STUN and TURN servers used in configurations. This chapter provides details of each implementation in a separate section. Starting with the reference implementation and then tree-topology implementations. Lastly, it also described architechural changes needed for the algorithms to work.

6.1 Reference implementation

The first implementation in this report is a reference implementation to get data to compare against the other implementations. The reference implementation is based on the appear.in project as it was in the start of this project. A new algorithm for data gathering, as shown in Algorithm 1, was implemented to gather the data needed to draw a baseline. Algorithm 1 goes through every stream that is being uploaded and records the resolution and the reason for any limitations. Further, it also accumulates the amount of bandwidth all streams use together and records this number. While Algorithm 1 is relatively simple, it has some

```
bandwidthAccumulator = 0;
response = null;
foreach peerConnection pc do
    pc.getStats( function(response){this.response = response});
    foreach report in response do
        if report is not an SSRC report then
            return;
        end
        if The report is about data being received then
            return;
        end
        resolution = {report.stat("frameHeightSent") ,
        report.stat("frameWidthSent")};
        limitedBy = get limitation from report;
        bytesSent = report.stat("bytesSent");
        bandwidthAccumulator += average bytes/s sent by this
        connection since last check ;
        send {resolution, limitedBy} to analytics engine;
    end
end
send bandwidthAccumulator to analytics engine;
```

Algorithm 1: Basic algorithm for gathering data from peerConnections.

peculiarities. We are only interested in data about streams and not general WebRTC data, which is why we are only interested in the SSRC reports. As of writing this report, we discovered that the the information about downloaded data was not accurate when adding several streams to one *RTCPeerConnection* object. It was found that in fact it seems like the data about older streams is sometimes not counted after the addition of new streams. Due to this inconsistency, only the upload data is collected. Also, download bandwidth was not a vital statistic for this project as it, unlike upload bandwidth, does not change between implementations.

6.2 Selection scheme

Section 4.1.2 discussed several different selection schemes for supernodes . In this project, we decided to use the supernode selection scheme with a centralized decision. In this case the decision is performed by the signaling server since this server already has connections with all nodes and can easily choose between them and distribute the result of said choice.

The scheme was chosen in this report because in a video conference set up time is important and with a central authority making the decision with information it already possesses without the need to collaborate with others the decision can be made fast. If the decisions are made by the central server it is also easier for it to alter the network topology as needed when nodes connect and disconnect. Another important aspect of the topology is that it should only be changed when needed. Unlike a network that is only used for data that might be changed to fit current conditions better, any change in the network will cause re-connections, re-negotiations of streams and therefore interrupt the service for users.

6.3 Naive tree-construction

In [10], the author proposes using the first node that enters a conference as supernode. Expanding on this idea, the naive implementation assumes equal weight of all nodes in the network and therefore sees all nodes as potential supernodes. Thus when the first node enters a conference it is immediately chosen as supernode, because the server sees it as a good candidate.

As the service will need to handle conferences of all sizes there is a need for multiple supernodes. Otherwise the single supernode will get overloaded since the amount of work placed on it would increase quadratically for each new node connected to it. Therefore a mechanism to create a new supernode on demand is needed. There are a few different algorithms that run both on the server and on the client. Since all nodes are considered equal the algorithm can assign a node to any supernode it wishes, which in the naive implementation implementation is the first supernode with available spots for new nodes. Additionally a new node can be made supernode if no others are available, and then simply connect it to an existing supernode to get all the information in the conference. This is illustrated in Algorithm 2. If a supernode has too many children, it might experience a load that is greater than it is able to handle. Since appear.in has a limit of eight nodes in the network at the moment, the maximum number of children was chosen as three. There will never be more than two supernodes in the network. Setting the number at four would have left one supernode underutilized. When nodes disconnect there are several scenarios to handle for the

```
let n be the number of nodes in the network;
n++;
let k be the number of supernodes in the network;
if $k == 2$ or $n/k <= 3$ then
    find supernode with free space for another node;
    assign current node to that supernode;
else
    make the new node supernode;
    k++;
    connect to other supernode;
end
```

Algorithm 2: Server side algorithm for clients connecting in naive implementation.

server. A disconnecting node can either be a supernode or a regular node. If it is a supernode it might either have children, or it might due to a variety of circumstances not have children. If a supernode that currently has children leaves the conference, its children need to be alerted to the change. The algorithm used to handle a node disconnection is provided in Algorithm 3. Therefore a new signal needs to be added to the signaling channel in order to tell the nodes that they should connect to a new supernode, and who that supernode is.

```
let k be the disconnecting node;
if k is a supernode then
    if k has children then
        pick one of children of k, let this be j;
        de-register j as a child of k ;
        make j a supernode ;
        transfer all of k's children to be children of j ;
        attach j to the other supernode if there is one; use signal to inform
        all affected nodes of a new supernode ;
    else
        disconnect k from other supernode;
        remove k from the network graph;
    end
else
    remove the node from the network graph;
end
inform all nodes of the disconnect;
```

Algorithm 3: Server side algorithm for clients disconnecting in naive implementation.

Note that Algorithm 2 and 3 run on the server to set up and change the network as needed. There is also an algorithm needed on the client side to setup the conversation according to the instructions received from the server. When a node connects, its supernode needs to update his local cached version of the network graph with the new information. Furthermore the supernode needs to set up forwarding of this nodes stream to all other nodes it is responsible for. This is described in Algorithm 4.

```
let s be the signal received;
let k be the node described in s;
if s is about a new supernode then
    set up connection with all current streams to k;
    if this node is a supernode then
        set up relay of all streams received from k to all children;
    end
end
if s is about a disconnecting client then
    remove clients video feed on the screen;
    if this node is a supernode then
        remove all streams received from k from all other connections;
        dispose of all old connections with k;
        start renegotiation of all changed connections;
    end
end
if s is about a connecting client then
    if this node is a supernode then
        if k has been assigned as our child then
            setup connection with all current streams to k;
            set up relay of streams received from k to all other connections;
            start renegotiation of all other connections;
        end
    end
end
```
Algorithm 4: Client side algorithm for naive implementation.

6.4 Opt-in supernode implementation

This Opt-in implementation relies on clients opting in to being supernodes. This implementation was chosen for several reasons, the first being to evaluate solutions in which all nodes are not viable supernodes. Further, with this implementation, clients themselves have the power to decide who wants to be or not be supernode. The clients should be able to pick the most suitable supernodes and the people who do not want to be supernode can make sure that they are not. The option is presented to the clients as a button. The codebase of appear.in already has a small menu that will open as seen in Figure 3. This solution will have to be able to function both with and without supernodes, and be able to switch between the two mid conference. If there are no nodes who opt-in to being a supernode, the functionality is exactly the same as the original fully connected graph, while if all nodes opt-in, the implementation will be as the one described in Section 6.3 as all nodes are eligible to be supernodes. Another reason for choosing and opt-in implementation is that it is believed that there may not be a suitable supernode candidate in every conversation. Therefore a solution which can work both with and without supernodes is explored to form a basis for future solutions.

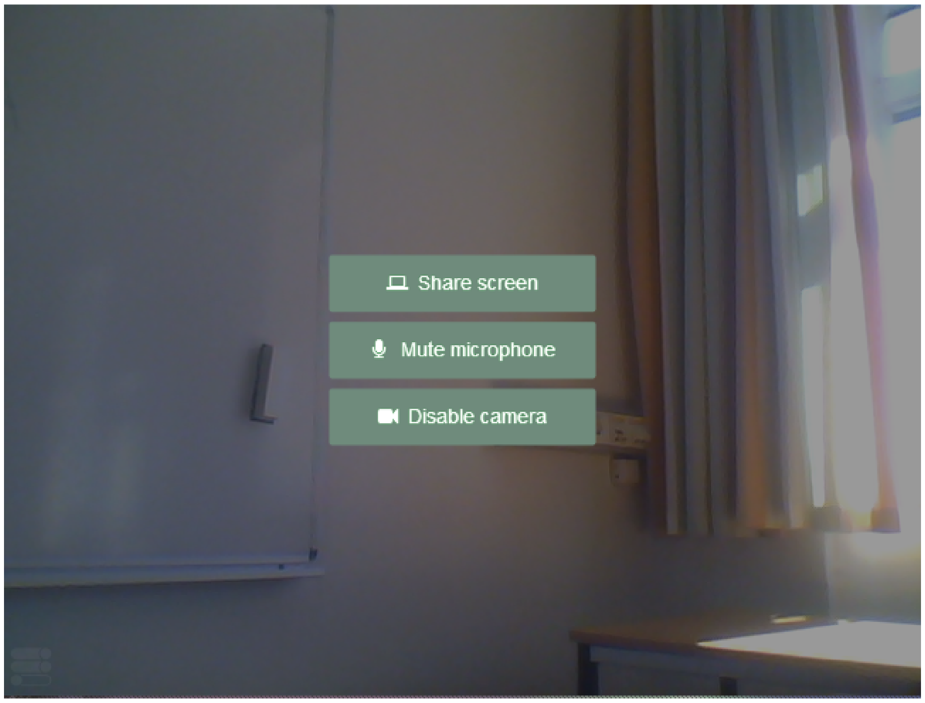

Figure 3: The appear.in options menu in which a supernode consent button will be inserted.

When changing between topologies in the middle of a conversation there needs to be another signal added, a signal telling the client to go back to a fully connected topology. When the network changes from a tree topology to a fully connected one and vice versa all nodes need to reconnect to each other. The difficulty with WebRTC's offer and answer session negotiation is that if both nodes try to connect to each other they both might end in an unexpected state, receiving an offer when they are expecting an answer. The best way to handle this situation is for the signaling channel to tell the nodes how they should connect to each other, because if this is decided by a single authority then there can be no conflicting information. So by opting for the signaling server to decide it can be ensured that the new network is connected without problems.

The solution is also a step towards other solutions where more advanced heuristics could be used to determine if a node is suitable to be a supernode or not. In solutions that do not see all nodes as suitable supernodes a few special cases arise when a tree needs to be rebuilt into another topology to work. This can happen under a few different conditions. When a supernode leaves and there are not any suitable supernodes to replace it or too many nodes connect and there are not enough suitable supernodes to build a functioning tree topology network. This behaviour is split into two algorithms. When a node connects it can either be put into the existing graph, or force a change in network topology. This behaviour is specified in Algorithm 5.

```
let k be the node connecting if conference is currently in tree topology
mode then
    if there are enough viable supernodes to support k then
        put k into the network and signal nodes as needed;
    else
        change to a fully connected topology;
        tell Nodes to change topology;
        let a be set of nodes in conference foreach node n in conference
        do
            remove n from set a;
            tell n to connect to all nodes in set a;
        end
    end
else
    since all nodes are considered non eligible from the start we just add
    it;
    connect k to network and tell others of new node;
end
```

Algorithm 5: Server side connection algorithm for opt-in implementation.

When a node disconnects it can either be simply removed as a regular node, or force a rebuild of the network if it is a supernode. This behaviour is specified in Algorithm 6. There is also need for a signal when a node opts-in to being a supernode. This can also cause a rebuild in the network, which however is not included in Algorithm 5 as it is a trivial build of a tree network.

```
let k be the node disconnecting;
if conference is currently in tree topology mode then
    if k is a supernode then
        if there are enough supernodes, or eligible supernodes to rebuild
        the tree then
            rebuild the tree;
            signal the nodes affected by the change;
            signal all nodes about disconnection;
        else
            change to a fully connected topology;
            tell nodes to change topology;
            let a be set of nodes in conference;
            foreach node n in conference do
                remove n from set a ;
                tell n to connect to all nodes in set a;
            end
        end
    else
        remove k from the network;
        signal all nodes about the disconnection;
    end
else
    remove the node from the network;
    inform all nodes of disconnection;
end
```

Algorithm 6: Server side disconnection algorithm for opt-in implementation.

On the client-side there are issues of dealing with changing topology in the middle of the conversation. This change is preformed by a signaling handling algorithm on the client side.This is described in Algorithm 7 and is however made simpler by keeping the logic for how nodes will connect to each other on the signaling server. This way the nodes only need to concern themselves with connecting to nodes they are told to connect to. Therefore Algorithm 7 is similar to the client side algorithm of Algorithm 4, except for the aforementioned addition(which is reflected in the last else-branch of Algorithm 7).

```
let s be the signal received;
let k be the nodes described in s;
if s is a signal about a new supernode then
    set up connection with all current streams to k;
    if this node is a supernode then
        set up relay of all streams received from k to all children;
    end
end
if s is a signal about a disconnecting client then
    remove clients video feed on the screen ;
    if this node is a supernode then
        remove all streams received from k from all other connections;
        dispose of all old connections with k;
        start renegotiation of all changed connections;
    end
end
if s is a signal about a connecting client then
    if this node is a supernode then
        if k has been assigned as our child then
            setup connection with all current streams to k;
            set up relay of streams received from k to all other connections;
            start renegotiation of all other connections;
        end
    end
end
if s is a signal about switching to fully connected then
    remove local representation of tree network;
    foreach node in k do
        if this node is already connected to k then
            remove all streams that are not produced locally from
            connection;
        else
            initiate connection with k with all local streams;
        end
    end
end
```

Algorithm 7: Client side algorithm for opt-in implementation.

6.5 Architectural changes

In WebRTC a signaling channel is needed to transport all sorts of information and this information needs to reach the correct node and the correct connection on that node. Since a signaling channel needs to be set up, it is also sometimes used to transport other information about a stream, such as status updates. When supernodes are used, it is not certain that a stream is arriving to a client from its actual origin. To be able to handle signalling messages, concerning streams, originating from a client, a mapping between clients and streams is required to be kept on each client.

In an application with supernodes it is easy to keep a list of the streams that the node is supposed to forward so that they can easily be accessed if a new clients joins. Since in WebRTC there is no special event for capturing when a stream stops, it is easiest to use the signaling channel for alerting clients that a stream has ended. When a conference only has one supernode, dealing with stopped streams becomes trivial as the supernode only needs to remove that stream from the stream list. However, in a conference with more supernodes, two distinct cases appear. When the disconnecting node is not a supernode the stream can simply be removed from the list of all supernodes. When the disconnecting node is a supernode however, all streams received from that supernode need to be removed, not only the ones originating from that node. This is because when a new tree is constructed, the old supernode will try to relay streams it does not actually have to the supernode. Therefore a table of which streams are received from which nodes is also needed.

When the tree construction algorithms select a new supernodes in an already active conference, multiple nodes may try connecting to this node. Depending on the latency to all nodes these connection attempts might come in rapid succession. When a connection is established and a stream is received over this connection, the supernode will relay this stream over all other connections. This is a problem since if all connections started at a similar time some might still be in the offer/answer sequence of session negotiation. Therefore, adding streams to those connections will result in a bad state since the session would now differ from the session being negotiated. This race condition can be resolved by only adding streams to connections that are established while waiting until other connections are established to add them.

6.6 Benchmarking

When implementing Algorithms 1-7, benchmarking solutions have also been evaluated to see if benchmarks such as connection speed or CPU could be used as a heuristic for supernode selection in an implementation for this project. It has however been found that performance benchmarking from a browser is a benchmark of the browsers Javascript engine and not of the computer. Therefore similar results were found from different computers.

For benchmarking connection speeds a program could count packets being sent and what rate they are being sent at to calculate upload speeds. For accurate measurements however, packets should be sent over a longer time such that an average result can be statistically significant.

7 Results

This chapter presents the results from the survey and benchmarking of the implementations. The results are compared and analyzed. The particulars and effects of drop out nodes, supernode selection, and ad-hoc topology changes are discussed.

7.1 Background survey

One important parameter in video-conferences is how many people are involved in the average conference. When this question was posed to 50 people, in our survey there were options between two and ten people to choose from. The answers are presented in Table 2 below. What can be seen is that the answers are quite confined to the lower part of the scale: 35 answers stated that only two people take part of the average video conference. Further, the average of all the obtained answers was 2.76 participants.

Members	2	3	4	5	6	7	8	9	10
Answers	35	6	5	0	2	0	1	0	1

Table 2: The answer rate for the question "How many people usually participate in your video-conferences".

Furthermore, the respondents were asked what usage pattern is common when using a video-conference service. In Table 3 scenario A means that everyone enters the conference when it starts and leaves the conference when it ends. Scenario B described the situation when the conference members enter and leave at different times. Scenario C is similar to scenario B with the difference that the conference host never leaves. As can be seen from the answers most conferences have a starting and ending point where people come and go.

Scenario A	45
Scenario B	2
Scenario C	1
Other	2

Table 3: The commonality of different usage scenarios when using video-conferences.

In Section 6.4 an implementation using a button to opt-in to becoming a supernode was discussed and implemented into a solution. To expand on this reasoning the respondents were asked how many of them would understand what being a supernode implied without any further explanation. 19 people responded that they would understand, while the majority, 31 people responded that they would not understand if such an option was offered to them. When an explanation was provided as to what it entailed to be a supernode for a video-conference, 39 people answered that they would opt-in to being a supernode while 11 would not.

The final question of the survey was about the upload and download speed of the respondents Internet connection in megabits per second. As can be seen below in Table 4, 40 and 38 percent respectively were not aware of their upload or download speed. This table also adheres quite well to that node characteristics are separated by a large margin as described in Section 4.1.1. The answers regarding download speeds are also in general higher than the upload speeds, making most of these connections asymmetric in speed with the download speed being higher. From the online speed measurement site Ookla [19] it is viewable that the average connetion speed of people who measure their broadband with Ookla is a download speed of 18.3 Mbps and upload speed is 8.2 Mbps as of the writing of this report.

Speed in Mbpss	Upload	download
<0.5	3	0
0.5	0	0
1	6	2
2	2	
3	0	2
4	1	1
5	0	1
7	0	0
10	8	4
15	0	0
20	0	2
25	1	1
40	0	1
50	0	6
100	8	9
Do not Know	20	19

Table 4: The upload and download speed of respondents.

7.2 Tester interview

After all tests had been concluded, the tester were asked for their opinion on the overall service quality and service usability of the reference, naive and opt-in implementations that they had tried compared to each other. While there were some varied answers on the usability and quality of the reference service all the opinions about the naive and opt-in implementations were quite unanimous. The service usability stayed the same through all three implementations as there was nothing changed from a users points of view. However, all testers agreed that the quality on the tree topology implementations had suffered a little, and the reason given for this was increased latency between the clients in any conference with more than two participants. As audio and video are sent as a part of the same stream however there will never be a latency difference between audio and video.

7.3 Appear.in results

To further substantiate how common different conference sizes are, data was extracted from the live service of appear.in. This data was gathered by looking at the time spent in conferences of different sizes. By looking at data for the months of March and April 2014 that is presented in Table 5 it is noted that more than 76% of all conferences are conducted with 2 participants while only 15.5% are conducted with three. From appear.in it is also seen that almost 30% of connections made via the service have to be relayed via a TURN server.

Conference size	Percentage
2	76.9%
3	15.5%
4	5.5%
5	1.5%
6	0.6%

Table 5: The relative number of participants in a video-conference on the appear.in service.

7.4 Reference results

The first data that was collected for the reference implementation was bandwidth data in order to observe the tendencies of bandwidth consumption as well as how our test systems could cope with the demands put on them. To be able to observe tendencies it was only deemed necessary to go as high as four simultaneous participants in the conference. The results are presented in Table 6. As can be seen the increase in bandwidth from one to two streams is as high as 77%. However, from two to three streams the bandwidth usage stays almost the same.

Conference size	# of streams	Average upload speed(Bytes/s)
2	1	237570
3	2	420964
4	3	410378

Table 6: The average bandwidth usage in a video-conference on the reference implementation.

It is described in Section 4.2 that the resolution on streams can scale automatically. As can be seen in Table 7 a conference with two participants always maintains its original resolution in the tests performed with this implementation. For three participants 90% maintain the original resolution while a smaller fraction has its resolution downsized. However, for four participants these change by a large margin. For four participants only 58% of the streams maintain their original resolution while the rest are downsized, which explains the decrease in average bandwidth from three participants in Table 6.

Conference size	640x480	480x360	320x240
2	100%	0%	0%
3	90%	7%	3%
4	58%	29%	13%

Table 7: The resolutions distributions for different sized conferences on the reference implementation.

According to the statistical report it is possible to get a limited stream resolution. It can either be limited because there is not enough available bandwidth to use for sending or receiving, or because there is not enough processing power to encode a stream at a higher resolution. In Table 8 the reason for downscaling is listed for each resolution as well as conference size. The three columns with a scaling reason display the percentage of streams that had this specific restriction. As can be seen none of the streams for the two participants conferences are downscaled. However, of the streams downscaled to 480x360 resolution a majority of them are downscaled due to bandwidth restrictions while all of the streams downscaled to 320x240 resolution are downscaled due to bandwidth restrictions. A stream can only be reported as limited to it's current resolution by one factor, and that is the factor that caused the downsizing. So even though there may not have been enough bandwidth nor CPU for a 640x480 resolution stream, the factor that first caused the downsizing is reported as the limiting factor of the resulting 480x360 resolution stream.

7.5 Tree topology implementations

After the implementation with an opt-in button had been tested and confirmed to be working correctly, the data from it was combined with the data from the naive tree implementation. This is because when the naive and opt-in

		Limitation		
Conference size	Video resolution	None	CPU	Bandwidth
2	640x480	100%	0%	0%
3	640x480	100%	0%	0%
3	480x360	0%	38%	62%
3	320x240	0%	0%	100%
4	640x480	100%	0%	0%
4	480x360	0%	43%	57%
4	320x240	0%	0%	100%

Table 8: The limitation on a streams resolution on the reference implementation.

implementations both use supernodes they work in the same way and as such it was deemed unnecessary to have two separate sets of results as they would be largely similar. As the larger data set given by the two together provides a better indication of the characteristics of networks using supernodes.

For the tree topology implementations the average bandwidth is presented in Table 9. When the conference only has two participants there is as before a case where both participants send one stream to each other. When there are three nodes there is however a difference, where the regular nodes use almost the same bandwidth as for the case with two participants but the supernodes use close to double the bandwidth. The same can be seen in conferences with four nodes where normal nodes use less and supernodes use more bandwidth. The streams column shows the number of uploaded streams including the number of relayed streams.

Conference size	# of streams	Average upload speed(Bytes/s)
2	1	234717
3	1	215995
3	4	496746
4	1	182627
4	9	785398

Table 9: The average bandwidth usage in a video-conference on the tree topology implementation.

With tree topology networks, conferences with two participants only yielded 640x480 pixel resolutions as can be seen in Table 10. For conferences with three participants 92% of the conferences retained their earlier resolution while the remaining were downsized to 480x360 resolution. When there where four participants in the conferences the numbers who had to be downsized however rose quite sharply. 24% were downsized to 480x360 resolution while 7% were downsized all the way to 320x240 resolution. The rest of the streams maintained their original resolution.

Conference size	640x480	480x360	320x240
2	100%	0%	0%
3	92%	8%	0%
4	69%	24%	7%

Table 10: The resolution distributions for different sizes of conferences on the tree topology implementations.

The streams in conferences with two participants were as can be seen in Table 11 never limited by anything and as such were never scaled. For three participants 47% of the 480x360 resolution streams were limited by CPU and the rest by bandwidth. However, for four participants 61% of the 480x360 resolution streams were limited by CPU and the rest by bandwidth, while all streams at an even lower resolution were limited by bandwidth.

Conference size	Video resolution	None	CPU	Bandwidth
2	640x480	100%	0%	0%
3	640x480	100%	0%	0%
3	480x360	0%	47%	53%
4	640x480	100%	0%	0%
4	480x360	0%	61%	39%
4	320x240	0%	0%	100%

Table 11: The limitation on a streams resolution on the tree topology implementations

7.6 Validation

The implemented solutions have been manually validated using log files and chromes built in WebRTC monitoring site chrome://webrtc-internals to monitor connections between clients and make sure the correct topology is reached without any unnecessary connections being made. Data on the solutions have also been gathered from the browser Google Chrome by using the built in *getStats* API and thereby getting the access to the values calculated internally in the browser about connections and streams.

7.7 Analysis of experimental results

The following sections provide an analysis of the results gathered from the reference and the tree topology implementations.

7.7.1 Reference results

From the data gathered both from the answer from the survey that is presented in Table 2 and the data gathered from appear.in that is presented in Table 5 it can be seen that the majority of users only use video conferences with four or less participants. In fact, in the appear.in case 97.9% of all conferences have less than five members. This may indicate that video conferences are used mostly for personal conversation and not as much for meetings between multiple individuals. It may however be that most conferences have less than five participants because clients cannot handle conferences larger than this or that larger video conferences feel impractical. This fact solidifies that data only being presented for up to four users is representative of how video conference systems are used.

In Table 6 it is shown how the bandwidth consumption increases as the conference increases in size. What is of extra interest in Table 6 is that the bandwidth requirement for three participants is very similar to the requirement for four participants. Once Table 6 is compared to Table 7 and Table 8 it can be seen that this is because with four participants many of the streams have reduced video resolution because of both CPU and bandwidth limitations. This accounts for a non increase in total bandwidth since lower quality streams use less bandwidth each.

There is evidence that even for conferences with only four participants some clients start to reach their limit as either the CPU struggles or the network cannot send the amount of information required. This causes a downscaling of resolution which leads to a decrease in service quality. The speed of 410378 bytes per second as recorded with four participants also equates to 3.28 Mb/s which is similar to the global average connection speed as discussed in chapter 3.1. From this information it is apparent that larger conversations will have a decline in quality for most clients as the required bandwidth and CPU usage will exceed what is available and thus causing streams to scale down as well as placing a lot of strain on the computer reducing its ability to perform other simultaneous tasks.

7.7.2 Tree topology implementations

With the tree topology tests it can be seen in Table 10 that for two and three participant conferences there is not a big difference between the supernode implementations and the reference implementation. The biggest difference is that there are no 320x240 resolution streams in the supernode version, as these were all caused by bandwidth limitations as can be seen in Table 8. This shows an increase in resolution due to less bandwidth required by most nodes in the tree topology implementations.

As can be seen in Table 9 the upload speed required by the supernodes for conferences with three participants is quite similar to the requirements for every node in the reference case. However, when a supernode is connected to two nodes, it is uploading four streams which shows a slight reduction in stream quality since a similar amount of bandwidth is used for three streams in the reference case. When a supernode is connected to three nodes it requires an upload of nine streams, which does not show a doubling in the amount of data sent, but still an increase. This points to more reductions in quality than in the three participant case. The nodes that are not chosen as supernodes maintain a similar amount of upload bandwidth usage regardless of conference size. This is also evident when looking at Table 10 where it is noticeable that, for four participant conferences, more streams now retain a higher resolution than they did for the reference implementation while still having a substantial increase from the three participant case. If mobile devices are used in a conference as non-supernodes this will be of great help as they do not have the same resources as computer for processing and uploading multiple streams. To always keep them at a single stream to upload it should enable better performance with mobile devices.

In the three participant conferences a slight decrease in 480x360 resolution streams is detected. As can be seen in Table 11 more streams of this resolution are limited by CPU than in the reference case while bandwidth limited streams have gone down. When the conference reaches a size of four clients a larger increase in original quality streams is recorded over the reference case. With the streams downscaled to 480x360 resolution 61% of streams are downscaled due to CPU limitations. Since the amount of streams encoded and sent on most clients is decreased from three to one this is seen as either a difficulty for certain nodes to receive, decode and display all the streams in the conference or too much load placed on supernodes. Because there were no bandwidth limited streams in the reference solution with two participant conferences it is unlikely that non-supernodes in the tree topology implementations have bandwidth limitations due to upload speed since they use the same amount of upload. They could however not have enough download for three full resolution streams and thus cause a downscaling. The bandwidth limited streams are also caused by the increased bandwidth requirements put on the nodes that have been selected as supernodes in these solutions. This is however not surprising since there were bandwidth limited streams for the cases where a node sent two or three streams in the reference case as well. However, the amount of reduced streams as well as the portion of the reductions caused by bandwidth help alleviate bandwidth issues for nodes that previously faced them.

As has been shown the upload bandwidth required was decreased for most nodes in the conference. The download bandwidth will however not be changed since all nodes still need to download all the streams that are to be shown. But as can be seen from Table 4 as well as in data from the online broadband measurement service ookla [19], on an average connection there is asymmetric behaviour between download and upload speed, with connections often having higher download than upload speeds. Therefore it is of greater importance to reduce the amount of upload speed needed than the amount of download speed.

The proper selection of supernodes is however imperative in the creation of tree topology networks, while the naive selection methods implemented in this report will not always be successful. In the tests conducted, the testers have always joined with the best node first and only opted in to being a supernode with the most suitable nodes. If an unsuited node was chosen to be a supernode, all streams relayed via that node would have reduced resolution, as it could not handle the increased load. This reduction would propagate through the entire network leading to a diminished user experience for all participants. Because of this, a very naive solution, such as the one described in Section 6.3, should only be used for comparisons and exploration. In a live service a better supernode selection scheme will be required.

As can be seen by the answers to the interviews mentioned in Section 7.2 one point that was noticeable was the increased latency in the implementations with tree topology networks. However this is a problem that will always occur with tree topology networks as the stream is sent from node A to node B and finally to node C instead of being sent from node A to node C directly. This will always add latency, although the latency increase could probably be minimized through improved selection of supernodes.

There will always be some CPU power needed to encode the streams emanating from a client itself and there is no way to reduce that. There will also be the need for download speed as well as the CPU needed to decode and display streams. The two latter can be improved, but only by downscaling streams or mixing them together into a single stream that is smaller than all the other streams together. As such there will always have to be some information loss to improve these parameters.

7.7.3 Drop out nodes

Previously in Table 3 it is shown that almost all users stay for the duration of a conference. With this knowledge the consequences of drop out nodes is already reduced since all nodes will leave at the same time and as such will not experience any consequences.

However, there will always be a problem with nodes disconnecting during the middle of a conference e.g. due to network issues. The naive and opt-in implementations have been implemented with this in mind and represent two categories of network. The naive implementations represent the networks that are always tree topology and the opt-in implementation represents hybrid networks were a supernode are only used if there are eligible choices for supernodes.

In all discussed algorithms, the consequences of a regular node drop out are the same, namely the expected event that this node is disconnected from and will have to manually reconnect later. In networks that are always tree topology the consequences of a supernode dropout will result in a disconnection of all the supernode's children. In the case of a conversation with more than one supernode this is only a subset of the clients. The clients will then experience a brief pause in the service when a new supernode is selected and all affected nodes are re-connected to the new supernode. In a solution that always requires a supernode this can also bring a decline in quality if none of the remaining nodes are well enough suited to being a supernode.

7.7.4 Super node selection

In the data gathered in this report it is indicated that tree topology networks reduce the load on the majority of nodes while not adding any significant load to any central server, only a few added signals and lightweight algorithms. However, due to improper selection of supernodes, the backend server load could actually increase. This would happen if a node that needs to use TURN for all of its connections is chosen as supernode because then even more streams will be relayed via a TURN server which will create more traffic over the TURN server. On the other hand, with proper selection of supernodes, less TURN server usage can greatly help scalability of the service by reducing the amount of streams that need to be relayed via a TURN server.

7.7.5 Ad-hoc topology change

As can be seen in Tables 2 and 5, most conferences only consist of two members and will, as a consequence, not benefit from the assignment of supernodes as no load can be decreased. By trying to connect from both nodes, the TURN server usage can be minimized. Therefore a solution could use tree topology networks only in conferences with more than three members and where a problem is detected that could be fixed by changing topology in the network. This means a hybrid network as described by the opt-in implementation in this report.

A change in network topology can be performed first when problems in the network arise. This approach reduces any unnecessary load caused by supernode selection by only executing the associated algorithms and signaling when needed. For example a potential problem is discovered when a node discovers unnecessary TURN usage, reduced resolution or other problems defined by a service.

This solution will enable clients to use the original service until an option that could improve their experience is discovered. Then there would be a brief pause while re-connecting the new network which will hopefully lead to an improved experience.

8 Discussion

There are many additional evaluations that can be conducted as future work and that would be of interest in this field. For example, further investigations are needed to determine if a node can handle the additional load of being selected as a supernode. Also, dynamically adding more supernodes is an interesting task for future work, whenever the current supernode is starting to show evidence of being overloaded.

In Section 7.2 it is described how testers thought the added latency of supernodes decreased the quality of conferences. There are several techniques for supernode selection with latency as a concern today, see for example [17]. In Section 1.3 a bachelor project which used an algorithm called popCorn with vivaldi coordinates was used to implement supernode networks with an aim at low latency. Evaluating this algorithm for use with WebRTC video conferences is of interest to see what effect this brings to the added latency of tree topology networks compared to the standard fully connected networks and tree topology networks without this improvement.

Section 6.2 describes how the implementations evaluated in this report use a centralized approach to supernode selection as opposed to the distributed approaches described in Section 4.1.2. Implementing a different selection scheme and evaluating the effect on set up time of a conference as well as re-build time of the network to determine which selection scheme offer better results is of interest for future implementations of tree topology networks in WebRTC video conferencing.

The reduction of TURN server usage is a big step in reducing required backend infrastructure and therefore increasing scalability for service owners. Currently, almost 30% of all connection made via appear.in are relayed via a TURN server as described in Section 7.3. In the work of [10], it is described how approximately 8% of all clients require a TURN server in order to achieve connection. The 22 percentage point difference between the two numbers is quite big, hence it would be of interest to test supernode selection techniques to reduce TURN server usage in order to evaluate how much this can be reduced and how close to the described limit of 8% a WebRTC video conferencing service can come.

In order to minimize the impact of drop out nodes a redundant, non-active connection could be set up and not used until needed. When a supernode drops out, such connections could be activated to quickly re-establish the conference again. Exploration of such methods would be of interest to see how much the impact of drop out nodes can be reduced, and if setting up redundant network is always viable or if there are cases where the network cannot handle it.

At the moment WebRTC is still a relatively new and changing standard. Therefore implementing solutions in WebRTC often take longer than expected and it is easy to set goals too high before starting to work. While this project has tried to follow the standard as published by the W3C there are still many features that are not implemented, or differ in their implementation from the standard. As the standard and implementations grow more mature it is however the hope that they will be more consistent, as this will make development easier.

There are features from the standard that remain unimplemented or undocumented. Therefore a significant amount of time has to be spent finding workarounds around these issues and finding solutions to bugs that occur, since the implementation or how it works is undocumented or the documentations often is hard to find. Through these difficulties this report however aims to be general about WebRTC from the point of the W3C specification and not mention workarounds needed to combat non compliant implementations as these are believed to be changed in the future.

9 Social and Ethical Aspects

Society today is changing at a rapid pace and globalization is also moving forward. Technology is emerging in markets that have previously not seen it. Reducing what is needed by a service owner to start a service makes it easier for new actors to emerge onto these new markets and bring the technology to them. If the service requires more initial investment and hardware for it to work, emergence on these markets can be severely delayed or even not happen as small actors might not be able to set up the service.

The quality of Internet connections and computers tend to vary across different parts of the worlds. Reducing the requirements to partake in video conferences could enable people who otherwise would have been unable to use the technology to do so and communicate with friends and family around the world. This could also easily, with some small modification or without modifications be used for educational purposes. With a teacher having a reliable computer and connection he or she could be able to connect to all students. Students, on the other hand could see and could be seen by the teacher on a much lower quality setup.

The emergence of video conferences also improves the conditions for employees and companies, where previously, employees would have to travel to physically attend a meeting. Now meetings can be held virtually and/or physically by connecting two meeting rooms via a video-conference. This will still allow for face to face discussions which is important during meetings, but save companies time and money by not having its employees travel and stay at hotels. Since employees would not have to travel as much, there will also be the added benefit of reduced environmental impact.

While the social implications of using supernodes are mostly positive there are also ethical implications to take into consideration. If supernodes are just used as in the naive model implemented in this project, then it has to be asked if it is justifiable to use someones computer and bandwidth without informing them, or asking for their consent. It might perhaps be the case that most clients would accept more CPU and bandwidth usage in order to help others in need. When it comes to bandwidth however, there are users who pay for what they use, or have a limit to how much they can use. In such cases, it would not be justifiable to use all the bandwidth they have available since it's their money on the line. There are many aspects to consider when choosing a supernode, not only from the technical point of view, but also from the ethical.

Even if the issue above was not present and there were unlimited resources, there are still other ethical aspects to consider. Even though a client has allowed that the service use the amount of resources needed to conduct the conference, is it really justifiable to use more than absolutely necessary so that others might be helped? This question is a part of the reason why in Section 6.4 there was a solution that needed consent in order to use additional resources thus leaving the choice up to the client. While some service owners might worry that not enough people would use such an option, or that they might forget, we feel that it is also ethically justifiable to use an opt-out approach, as long as it is properly explained, as the choice is still left up to the users. As can be seen in Section 7.1 there might not be many nodes that opt-in unless an explanation is provided. Therefore service owners have to be careful to provide a good explanation as to what opting in means. If a service cannot do this but feels it is important to get as many supernode candidates as possible, they might wish to use an opt-out approach instead.

In all implementations of supernodes in WebRTC considered in this report clients only get access to information that would otherwise be available to them, as they only relay video and audio that they are already receiving and displaying themselves. If this is however not the case and clients in other conferences were used as supernodes, there could be leaks or theft of information as it is handled by clients who should not otherwise have access to it. This might even lead to others being able to eavesdrop and peek on a conference. Due to the sensitivity of the service, as it involves private conversations between two or more people, the information should never be sent to a third party, even if it is believed to be safe. With eavesdropping being an important topic in today's society, it is our opinion that the information in a conference should never leave the conference enviroment, thus minimizing the possibilities for eavesdropping and peeking.

A large responsibility rests on the shoulders of the service owners though, as they are granted a right to use extra resources from a clients computer. This is so,because in most cases the clients will not know how much extra the service will have to use to accomplish what it needs. What is then to stop the service owners from also diverting a bit of resources from the clients computer for their own purposes. This would usually not be visible to clients, but could be very dangerous as their computer and/or connection could be used for anything from calculations to illegal activities.

10 Conclusions

This thesis investigated supernode selection schemes, as well as how tree topology networks influence the quality of WebRTC based video conferencing.

Tree topology networks are feasible in WebRTC based video conferences as an alternative to the standard fully connected mesh networks. As described in the discussion tree topology networks lower the load on most nodes and thereby also increase the overall resolution of streams in the conference. Since the load on most nodes in a conference becomes lower in a tree topology network, the amount of clients that can participate in the conference increases since more clients are needed to experience the same load as before. This assumes that a client has enough download bandwidth, as the download speed will not be reduced by changing topology, it will only be reduced by video scaling. It can also decrease the usage of a TURN server which will decrease cost from the service owner and thereby increase scalability for the service owner as well as the user.

The present thesis shows that the quality of conferences was slightly lowered by tree topology networks because of increased latency. This is however contrasted by that the actual resolution of streams was improved from the fully connected case. Higher latency is unavoidable when adding an intermediate step in a connection, it is however believed that future efforts could reduce the extra latency to points were the benefit of increased stream resolution will outweigh the added latency.

In a standard fully connected implementation of a WebRTC video conference the upload and download speed required are the same or very similar while in tree topology solutions they are asymmetric with a lower upload speed. Therefore, the thesis concludes that tree topology networks are seen as a better for most users since the average connection has lower upload speed.

Supernode selection in WebRTC is believed to be very subjective to the service and the purpose of their selection. Some services may want to select supernodes to minimize TURN server usage while others care more about minimizing added latency. This area would however benefit from further research and comparison to investigate the differences between these approaches. No matter if latency, reduced TURN usage or something else is chosen as main criteria for choosing a supernode the most important is still that a supernode will be able to handle the increased load. No matter how many supernodes are selected as a means to distribute the load there will always be root node in a tree that will have a greater load than the others as it will have to connect all other supernodes to each other.

Drop out nodes do not pose any major problem as addressed in the discussion. It only requires a very brief reconnection and this pause can also be made shorter by having redundant connections that are disabled until they are needed. However in our opinion this adds unnecessary complexity to the network. It is however our opinion that with regard to drop out nodes hybrid networks preform better since they will not put the increased load on any that cannot handle it but instead distribute the load on the entire network if there is no suitable supernode.

References

[1] A. Bergkvist ,et. al. WebRTC 1.0: Real-time Communication Between Browsers. [online] , http://www.w3.org/TR/2013/WD-webrtc-20130910/ (Accessed: 20 January 2014)

[2] J Abbink, et. al, Final report : Dynamic peer to peer game networks using WebRTC. [Online] ,http://repository.tudelft.nl/assets/uuid:04082eb3-bb8c-4077-9ec2-0855da97a645/final_report_webrtc.pdf (Accessed: 5 april 2014)

[3] How it works : Overview of Licode and its components [Online], http://lynckia.com/licode/architecture.html (Accessed: 7 april 2014)

[4] M.H, Willebeek-Lemair, et. al, On multipoint control units for videoconferencing, in Local Computer Networks, 1994. Proceedings., 19th Conference on, Minneapolis, MN, USA, 2-5 October, 1994, pp 356-364

[5] J.P Rodrigue, Graph Theory: Definition and Properties in The geography of transport systems, 3rd edition, New york, Routledge, 2013, Appendix

[6] L. Virginia, et. al, Scalable Supernode Selection in Peer-to-Peer Overlay Networks. [Online], http://csis.pace.edu/ marchese/CS865/Papers/lo_scalable-supernode-selection-in.pdf (Accessed: 21 February 2014)

[7] S. A. Baset and H Schulzrinne, An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, Computer science department, Columbia University, New York, 2004

[8] J. Liang, et. al, Understanding KaZaA, Polytechnic University, Brooklyn, NY

[9] M. Handley, et. al. SDP: Session Description Protocol. [online] ,http://tools.ietf.org/html/rfc4566.html (Accessed: 4 February 2014)

[10] I. Grigorik , "WebRTC" in High Performance Browser Networking, 1st edition. Sebastopol , O'Reilly , 2013, ch. 18

[11] W3Counter(2014, March) March 2014 Market Share, [Online] , https://www.w3counter.com/globalstats.php (Accessed: 5 April 2014)

[12] J. Rosenberg , et. al. Session Traversal Utilities for NAT (STUN) [online] , http://tools.ietf.org/html/rfc5389 (Accessed: 7 February 2014)

[13] R. Mahy, et. al. Traversal Using Relays around NAT (TURN) [online] ,http://tools.ietf.org/html/rfc5766 (Accessed: 7 February 2014)

[14] J. Rosenberg, et. al. Interactive Connectivity Establishment (ICE) [online] , https://tools.ietf.org/html/rfc5245 (Accessed: 10 February 2014)

[15] J. Lennox, et. al. Source-Specific Media Attributes in the Session Description Protocol (SDP) ,avalible : http://tools.ietf.org/html/rfc5576 (Accessed: 19 February 2014)

[16] Statista, [Online] , http://www.statista.com/statistics/204954/average-internet-connection-speed-worldwide/ (Accessed: 28 March 2014)

[17] J. Sacha, Exploiting Heterogeneity in Peer-to-Peer Systems Using Gradient Topologies, Ph. D Dissertation, Comp. Sci, Trinity College, Dublin, 2009.

[18] D. C. Burnett, Et. Al, Media Capture and Streams. [Online] , http://dev.w3.org/2011/webrtc/editor/getusermedia.html (Accessed: 4 February 2014)

[19] Net index, Global Broadbad. [Online] avalible : http://www.netindex.com/download/allcountries/ (Accessed: 9 May 2014)