NTNU
Norwegian University of
Science and Technology

Congestion Control for WebRTC Services

Maria Sørlie

Master of Telematics - Communication Networks and Networked Services
Submission date: June 2017
Supervisor: Min Xie, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

NTNU – Trondheim
Norwegian University of
Science and Technology

Congestion Control for WebRTC Services

Maria Sørlie

Submission date: June 2017
Responsible professor: Min Xie, ITEM

Norwegian University of Science and Technology
Department of Telematics

Title:	Congestion Control for WebRTC Services
Student:	Maria Sørlie
Course:	TTM4905, Master Thesis

Problem description:

Applications with audio- and media-mediated communication have increased over the years. WebRTC (Web Real-Time Communication) enables real-time multimedia services on the web. Since WebRTC services are delivered on web, it is challenging to offer smooth and satisfactory quality, especially for interactive real-time services such as video and audio conferencing. IETF RMCAT (RTP Media Congestion Avoidance Techniques) working group is working towards the demand of specifying congestion control mechanisms for RTP (real-time transport protocol) flows over UDP, used by WebRTC.

For better experience with real-time multimedia applications on the web and better utilization of the Internet. This study purpose is to implement a real-time multimedia service based on WebRTC, and a congestion control mechanism, developed using Java/Javascript. There will be conducted research of the WebRTC technology and studies of the various congestion control mechanisms before implementation. Student will analyze and give an evaluation of the congestion control mechanism performance.

Department:	Department of Telematics
Responsible professor:	Min Xie, ITEM

Abstract

Multimedia applications are increasing in popularity and using a big part of the Internet traffic. Web Real-Time Communication (WebRTC) is new technology which allows peer-to-peer communication in the browsers without any extra plugins. The focus of this report is to determine the importance of congestion control for the WebRTC-services.

In this thesis, I developed my own WebRTC-application. The developing process contained research about features, protocols, and technologies used. The final service was fully developed with audio and video features for multimedia conversations. In addition, instant messaging was added for assurance in case of communication problems.

Further, the developed WebRTC service were to conduct experiments. The Experiments conducted where divided into two phases. First phase tested the WebRTc service itself, to ensure the service was working properly. Second phase consisted of 10 participants using the developed service and give session feedback. I collected both session statistic data and giving feedback from each session.

The experiments focused on the Quality of Service (QoS) and the user perceived Quality of Experience (QoE), by looking at sent and received bits and packets, packet loss rate and jitter values. I got an indication of the QoS and if high values had an affect on how the user experienced the session. Results show that it was the audio interruptions were most heavily influenced by poor QoS.

Congestion control is a mechanism needed for transportation of data across the Internet, to promote fair usage and prevent congestion collapse. After looking at results from the experiments, a congestion control may be needed. I have evaluated two WebRTC congestion controls, Google Congestion Control (GCC) and Network Assisted Dynamic Adaption (NADA), to fully understand how they operate.

The findings from experiments and the evaluation indicate a need for a congestion control, and both GCC and NADA are found to be appropriate congestion controllers.

Keywords – WebRTC, Quality of Service, Quality of Experience, congestion control, GCC, NADA, experiments.

Sammendrag

Multimedieapplikasjoner øker i popularitet og bruker en stor del av trafikken på Internett. Web Real-Time Communication (WebRTC) er ny teknologi, som tillater peer-to-peer-kommunikasjon i nettlesere uten ekstra plugins. Denne rapporten handler om etterspørselen av congestion controls til WebRTC-tjenester.

I denne oppgaven utviklet jeg min egen WebRTC-applikasjon. Utviklingsprosessen hadde grunnlag fra forskning om ulike funksjoner, protokoller og teknologier som brukes mye i dag. Den endelige tjenesten ble fullt utviklet med lyd- og videofunksjoner for multimedia-samtaler. I tillegg ble direktemeldingsfunksjon funksjon lagt til, i tilfelle det ville oppstå kommunikasjonsproblemer.

Videre ble den utviklede WebRTC-tjenesten brukt til å gjennomføre eksperimenter. Eksperimentene som ble utført var fordelt i to faser. Første fase testet WebRTc-tjenesten for å sikre at tjenesten fungerte slik den skulle. Andre fase besto av 10 brukere som tok i bruk WebRTC-tjenesten og ga tilbakemeldinger om samtalen. Jeg samlet både statistiske data og tilbakemeldinger fra brukerne i hver enkelt samtale.

I eksperimentet ble det fokusert på Quality of Service (QoS) og brukerens oppfatning av Quality of Experience (QoE) og ved å se på sendte og mottatte bits og pakker, pakktap og jitter verdier. Jeg fikk en en bedre forståelse for hva god QoS var og om de høye verdiene hadde en innflytelse på hvordan brukeren opplevde samtalen. Eksperimentene viste at det var lyden som ble mest berørt av en dårlig QoS.

Congestion Control er en mekanisme som er nødvendig for transport av data over internett, for å fremme og forhindre overbelastning. Etter å ha sett på resultatene fra eksperimentene, kan det være nødvendig med en congestion controller. Jeg har vurdert to WebRTC- congestion controllers, Google Congestion Control (GCC) og Network Assisted Dynamic Adaptation (NADA), for å forstå hvordan de fungerer.

Funn fra eksperimenter og evaluering indikerer at det er behov for en congestion controller, og for dette er både GCC og NADA passende valg.

Nøkkelord - WebRTC, Quality of Service, Quality of Experience, congestion control, GCC, NADA, experiments.

Preface

This master thesis is an original and independent work by Maria Sørlie. The thesis is the final contribution to the Master's degree in Telematics at the Norwegian University of Science and Technology (NTNU).

The goal of this master thesis is to investigate the need of congestion control for WebRTC-services. The objective of my work is to develop a WebRTC service to use in experiments and evaluate congestion controls for WebRTC services.

I want to thank my responsible professor Min Xie for motivating me and giving me valuable feedback during this master thesis. I would also like to thank my family and friends for helping me through this period, supporting me and proof reading the master thesis.

Lastly, a special thanks to my dear Torgeir for supporting and helping me with this master thesis.

Contents

List of Figures

List of Tables

List of Acronyms

API Application Programming Interface.

CSS Cascading Style Sheets.

DCCP Datagram Congestion Control Protocol.

ECN Explicit Congestion Notification.

GCC Google Congestion Control.

HTML Hyper Text Markup Language.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IP Internet Protocol.

ITU International Telecommunication Union.

JSON JavaScript Object Notation.

NADA Network-Assisted Dynamic Adaptation.

NAT Network Address Translator.

NTNU Norwegian University of Science and Technology.

OS Operating System.

QoE Quality of Experience.

QoS Quality of Service.

RMCAT RTP Media Congestion Avoidance Techniques.

RTP Real-Time Transport Protocol.

RTT Round Trip Time.

SDP Session Description Protocol.

SIP Session Initiation Protocol.

SRTCP Secure Real Time Control Protocol.

SRTP Secure Real Time Protocol.

STUN Session Traversal Utilities for NAT.

TCP Transport Control Protocol.

TFRC TCP Friendly Rate Control.

TURN Traversal Using Relays around NAT.

UDP User Datagram Protocol.

VoIP Voice over IP.

W3C World Wide Web Consortium.

WebRTC Web Real-Time Communication.

WWW World Wide Web.

XML Extensible Markup Language.

XMPP Extensible Messaging and Presence Protocol.

Chapter 1
Introduction

1.1 Motivation

In the Internet, there is a big increase in popularity in use of real-time communication services, like Skype[1], and Viber[2]. These services are starting to consume big parts of the Internet traffic. However, a lot of these services need something extra, plugins or extra software to download, for it to work properly. Web Real-Time Communication (WebRTC) is considered a relatively new technology, which allows peer-to-peer browser communication without the extra plugin or software. It offers a more flexible and user-friendly way to communicate.

By not having to run additional software to get real-time communication with the browser. It makes it easier for a user with little or no computer-skills. Applications like this will be attractive for even more users to use these new services. The usage and popularity of these services are highly dependent on the Quality of Experience (QoE) and Quality of Service (QoS) in users encounters. Poor QoE or QoS will decrease the usage and popularity. Users will always want to have the best in terms of the Internet services. Entering a new web address should be executed as fast as possible with the best quality. This means there should be delivered minimum packet delays and packet loss.

WebRTC is considered a relatively new technology and there are challenges. Real-time applications have always been sensitive to packet loss and packet delays. If there are packet loss or packet delay, the WebRTC application would not be performing 100% and in some cases, be useless for the users, because of the interruption in the video or audio. There is a probability congestion occurred in the network.

The problem is about real-time media transport. WebRTC transports media using Real-Time Transport Protocol (RTP) over User Datagram Protocol (UDP).

[1]Skype: https://www.skype.com
[2]Viber: https://www.viber.com

RTP media transport is well defined and gives high performance. When RTP is in use, it will be deployed at very large scale and it has no professional network support. Therefore, with many real-time applications running on Internet, the applications can use a lot of bandwidth. There are also other applications running on the internet as well. Potential network congestion can easily occur.

In addition, UDP is one protocol that does not have a congestion control mechanism. UDP is the protocol suited for interactive real-time applications, because of less overhead in headers and can carry more data than most other protocols. Transport Control Protocol (TCP) is another transport protocol which has a congestion control implemented. The TCP congestion control algorithm causes high latency, because it is loss driven, which means it relies on queue overflow. For it to work properly buffers are needed to smooth abrupt changes in rate and match encoder output.

Although, with these problems and the expectations of the users regarding high quality of WebRTC service, there is a need to find a solution. One option is congestion control for WebRTC services. This master thesis will develop a WebRTC service for research purpose to evaluate the need of a congestion controller. Also, this master thesis will present an evaluation of two WebRTC congestion controllers.

1.2 Objectives

The project description of this thesis has two primary objectives:

- Develop a WebRTC service which include features like audio, video and instant messaging. With this WebRTC application, sessions related data will be collected to verify the importance of solving the existing congestion problem in real-time communication.
- Give an overview of the definition of congestion control and a evaluation of various congestion control mechanisms for WebRTC services. There are two recommended WebRTC congestion controllers by Internet Engineering Task Force (IETF) RTP Media Congestion Avoidance Techniques (RMCAT), this thesis will include an evaluation of two congestion control algorithms to understand how they work and if they are a solution to the congestion problem in real-time communication.

1.3 Structure of the report

This report is organized as follows: Chapter 2 provides background information to the work. Chapter 3 describes the methodology. Chapter 4 and 5 describes the design and implementation of the WebRTC application. Chapter 6 describes the experiment and presents the experiment results. Chapter 7 gives an analysis of two

WebRTC congestion controllers. Chapter 8 presents the discussion about the findings of the work. Finally, chapter 9 will give a conclusion and future work.

Chapter 2
Background and Related Work

This chapter presents the background information relevant to this thesis. This chapter covers the technical details about the Web Real-Time Communnication (WebRTC)-technology, as well as the concept of network congestion and congestion control. In the end you are given a brief description of the definitions of Quality of Service (QoS) and Quality of Experience (QoE).

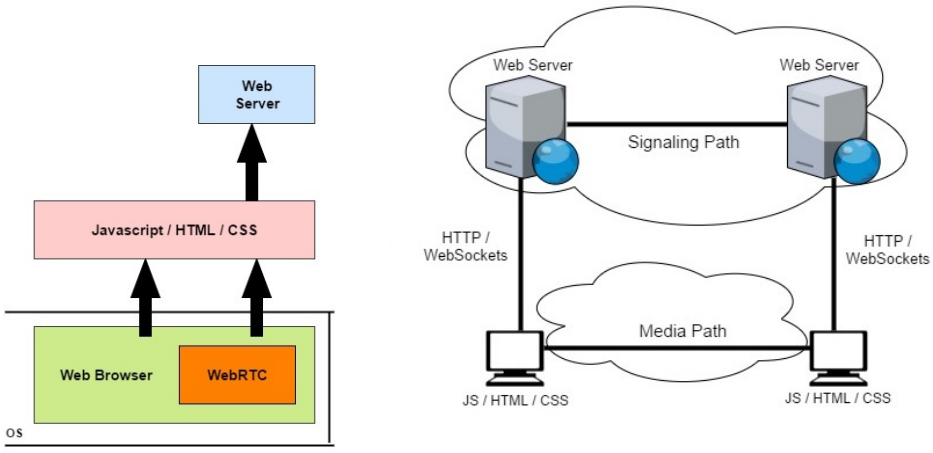
2.1 Web Real Time Communication (WebRTC)

WebRTC [42] is a free and opensource project supported by the World Wide Web Consortium (W3C) and IETF that provides browsers and mobile applications with real-time communications capabilities. Meaning that WebRTC enables browser-to-browser(peer-to-peer) communication. WebRTC is relatively new up and coming technology which does not require any extra plugin or software download. In the future this service will be very popular and may replace text chats.

The goal of WebRTC is "to enable rich, high-quality RTC applications to be developed for the browser, mobile platforms, and IoT devices, and allow them all to communicate via a standard set of protocols" [42]. Since WebRTC is not a standard yet, not all browsers support the WebRTC technology, but Google Chrome, Mozilla Firefox and Opera are examples of browsers that does.

2.1.1 WebRTC API

It is important to know that WebRTC is not one Application Programming Interface (API), but a collection of APIs. These APIs includes the fundamental components to build high quality RTC-based web applications. The fundamental components are audio, video, and data packets that are transported over peer-communication in WebRTC services. There are three main APIs which are needed to establish a full RTC connection. The main components of the WebRTC API are summarized as follows [4]:



(a) RTC in the browser (b) WebRTC communication and signaling process

Figure 2.1: WebRTC overview

– **MediaStream.**
MediaStream API represents synchronized streams of media. On the other hand, the MediaStream is responsible to give web browsers access to the camera and or microphone. The API manage the data streams such as displaying the stream's content, recording, or sending it to a remote peer.

– **RTCPeerConnection.**
The RTCPeerConnection API allow peers to connect and communicate directly, browser-to-browser, encryption and bandwidth management. Data transmission, session mechanism and other functions are encapsulated in this API.

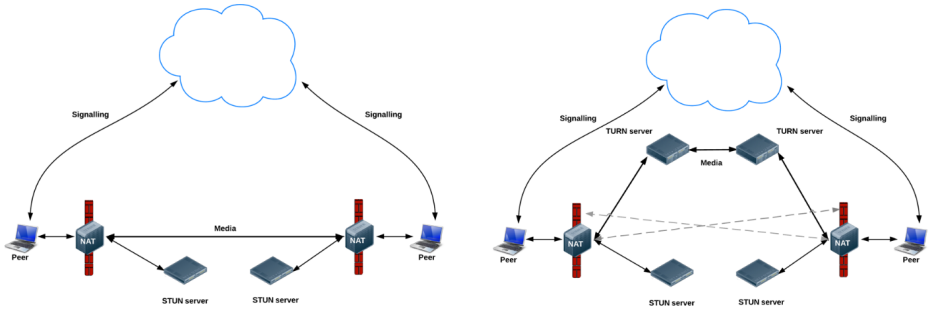
– **RTCDataChannel.**
The RTCDataChannel API represents a bidirectional data channel between two peers and enables exchange of data, with low latency and high throughput. The API has many potential uses, for example gaming, remote desktop applications, real-time text chat, and file transfer.

2.1.2 Architecture and Features

WebRTC is not a service that can run, but a technology one can use. The WebRTC package includes audio, video, and network components. You can access the WebRTC API, which are in the browser, through Javascript API and HTML5 from the webserver, like figure 2.1a shows. All needed functions will be embedded in WebRTC

in the web browser and no third-party plugins. WebRTC API makes it possible to use real time interactive audio and communication directly between browsers across the Internet, shown in figure 2.1b. In figure 2.1 you get an overview of the WebRTC model.

After all the session description are set, all communication can be sent between peers, as figure 2.1b shows. There are some technicalities WebRTC has to deal with, for example firewalls and Network Address Translator (NAT)s. Nowadays, nobody has a global Internet Protocol (IP) address and peers could connect directly to the Internet, but NATs hide the direct IP address and prevent a direct connection for security. WebRTC Session Traversal Utilities for NAT (STUN) server is designed to solve this problem and will find an external network address. This means that WebRTC service would get a publicly accessible address for itself. Then it is possible to pass the public address along to another peer via signaling, to set up a direct connection[43]. Figure 2.2a shows a look up with the STUN server.



(a) STUN server lookup (b) STUN and TURN server lookup

Figure 2.2: STUN and TURN server lookups in WebRTC[43]

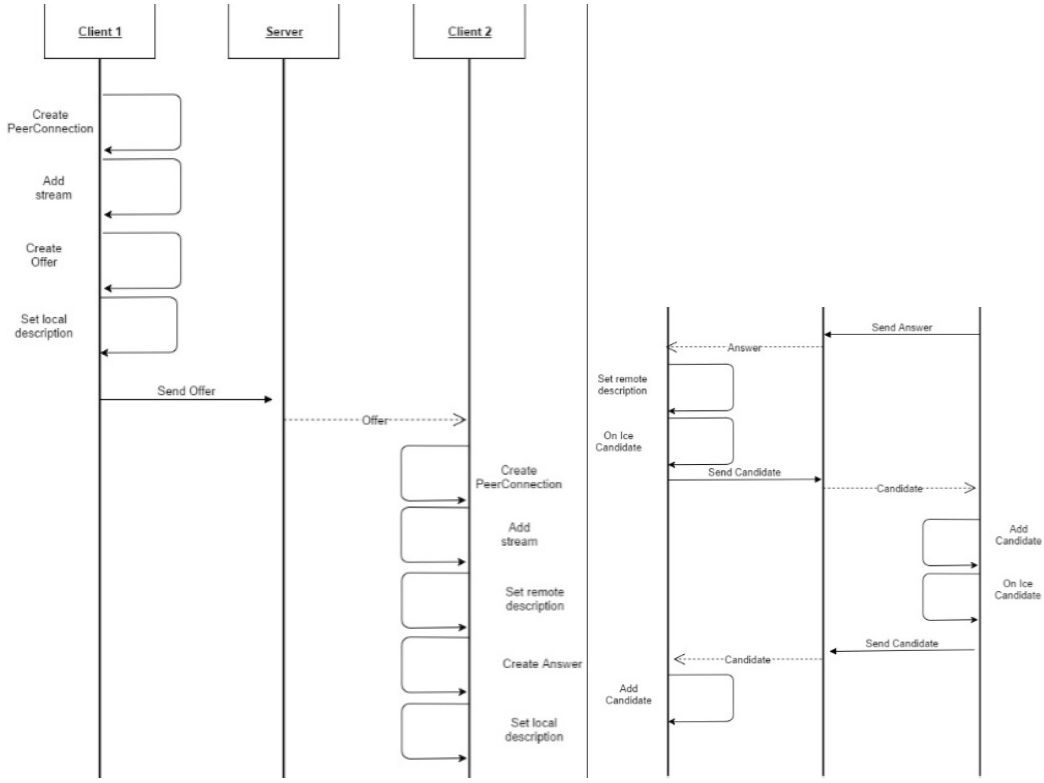
In other cases a users firewall can block the traffic sent directly from a client. Traversal Using Relays around NAT (TURN) servers are used as a fallback when STUN servers fail. TURN servers task is to relay data between different peers. The downside is that the TURN servers uses a lot of bandwidth which is not ideal. Therefore, STUN servers are required to go through first. Both STUN and TURN serves are needed to properly operate part of the WebRTC-infrastructure. Figure 2.2b shows when STUN servers fails and then falls back on relaying data through the TURN server.

2.1.3 Signaling

Signaling methods in WebRTC are not specified. WebRTC does not have any standard signaling protocol, because its purpose is to maximize compatibility with existing

technologies and to avoid redundancy. RTCPeerConnection API is responsible for finding another user to set up a connection. As soon as browsers know how to find each other over the Internet, they can exchange data about which protocols each of them support. This is called signaling, a process of connecting to the other user. Examples of signaling protocols that can be used by WebRTC are Session Initiation Protocol (SIP), Extensible Messaging and Presence Protocol (XMPP), and WebSocket.

The signaling channel is needed to exchange information between WebRTC peers, session description and network reachability information. There are three types of information. 1) Media session management, which describe how to set up and take down the communication, and report error. 2) Nodes' network configuration, which send network addresses and ports available for real-time data. 3) Nodes' multimedia capabilities which describe what kind of media is supported, available encoders/decoders, supported resolutions and frame rate.



(a) Part 1 (b) Part 2

Figure 2.3: WebRTC signaling [24]

It is not possible to transfer any data between WebRTC peers until all the information above has been exchanged and discussed. In the book, Real-Time Communication with WebRTC [24], it is explained in detail how signaling and message exchange between peers are handled and it is illustrated in figure 2.3.

First step is that client one, which is the initiator of the call, create a PeerConnection. Further on a MediaStream is added. MediaStream holds the video and audio stream. A Session Description Protocol (SDP) is a protocol describing the media initialization parameters. SDP defines the media characteristics of a call. After adding the media stream PeerConnection create an offer with SDP information. The PeerConnection set the created offer as a local description and sends it to client two. Client two receives client ones offer and creates a PeerConnection as well. Same as client one, it adds the media stream. After setting the offer as remote description to the PeerConnection, client two create an answer which contain the SDP information, same as the offer. Afterwards the answer is set as a local description to the PeerConnection on client two's side. Client two sends the answer to client one where client one set the answer as a remote description. After creating the PeerConnection and pass the available STUN and TURN servers, an event will be fired once the ICE framework has some "candidates" that will allow you to connect with a peer.

2.2 Network Congestion

Congestion is defined as a condition where one or more egress interfaces are offered more packets than are forwarded at any given instant [38]. Network congestion is when an increased transmission results in a smaller throughput. It is the same as when a network is congested, the more data one tries to send, the less data is successfully sent.

Network congestion is hard to define quantitatively, but everyone recognizes it when they see it. The users feel a reduced Quality of Experience (QoE) with slow video stream, bad Voice over IP (VoIP) communication, a poor web browsing experience and frustrating online gaming performance. For communication service providers, it means angry users and poor business.

Congestion collapse is the state in which congestion prevents or limits useful communication. Congestion collapse is when incoming traffic exceeds the outgoing bandwidth. Congestion is under control and in normal behavior when there is only one copy of the packet in transit. Once retransmission of packets starts and they do not get delivered, then congestion is a big problem. Typical effects of congestion collapse are queuing delay, packet loss or blocking of new communication.

2.2.1 Congestion Control

With congestion in the network, one would want a congestion control. Congestion Control is needed for transportation of data across the Internet, to promote fair usage and prevent congestion collapse. The requirements for real-time multimedia differentiate from requirements to transfer web pages. The real-time multimedia needs low delay and semi-reliable data delivery and because of the increased WebRTC traffic on the internet, there may be a need for a congestion controller.

IETF RMCAT (RTP Media Congestion Avoidance Techniques) working group is working towards the demand of specifying congestion control mechanisms for RTP (real-time transport protocol) flows over UDP, used by WebRTC. They have recommended two controllers, Google Congestion Control (GCC) and Network-Assisted Dynamic Adaption (NADA), which are described more in detail in chapter 4. First we look at other congestion control techniques and some challenges when it comes to providing a good congestion control mechanism.

Congestion Control Mechanisms
There are many different congestion control mechanisms out there. Here are just a selected few congestion control mechanisms described below.

- **TCP Congestion Avoidance**
  Connection-oriented protocols, such as TCP protocol, watch for packet errors, packet losses or delay to adjust the transmission speed. TCP congestion control [35] was first introduced by Van Jacobsen in 1986 for the Internet to avoid congestion collapse. TCP congestion avoidance mechanism is the basis for congestion control in the Internet [37]. The main operation of this avoidance mechanism is for each connection; TCP maintains a congestion window. The congestion window keeps track of the total number of unacknowledged packets that may be in transit end-to-end. The congestion window is maintained by the sender. Obviously, the TCP transmit window size must never be bigger than the congestion window, or that will cause network congestion.

- **TCP Friendly Rate Control (TFRC)**
  TFRC is a congestion control mechanism designed for unicast flows operating in an Internet environment and competing with TCP traffic [36]. The TFRC is designed for applications that use a fixed packet size, and vary their sending rate in packets per second in response to congestion. All the calculation of the congestion control information, i.e. loss and round-trip time, is calculated at the receiver. These parameters are than used to a model of TCP throughput. The expected throughput from the model is then used for the transmit rate of a TFRC flow. TFRC goal is to compete fairly with TCP traffic.

- **Datagram Congestion Control Protocol (DCCP))**
  DCCP is called a message-oriented transport protocol[8]. Like TCP, DCCP implements congestion control and serve as a general congestion control mechanism for UDP-based applications. The reason is, DCCP is more suitable for applications that transfer large amounts of data, which can benefit from control over the balance between delay and reliable delivery. Also, DCCP includes Explicit Congestion Notification (ECN) support, implements reliable connection setup, teardown etc.

2.2.2 Congestion Control Challenges and Requirements

These challenges and requirements of real-time media are defined by IETF at [5]:

Challenges:

- The media is usually encoded in forms that cannot be quickly changed to accommodate varying bandwidth, and bandwidth requirements can often be changed only in small, rather large steps.
- The participants may have certain specific wishes on how to respond when congestion is detected - which may not be reducing the bandwidth required by the flow.
- The encodings are sensitive to packet loss, because there is no time to retransmit real time data.

Requirements:

- The algorithm must provide low delay transit for real-time traffic, even within a very limited time window or faced with bottlenecks and competing flows.
- The congestion control should also deal well with routing changes and interface changes (WiFi to 3G data, etc) which may radically change the available bandwidth.
- The algorithm should be fair to other flows like TCP and other real-time flows. Not react to short-time burst like in a web session.
- The algorithm should merge information across multiple RTP streams between the same endpoints, whether or not they are multiplexed on the same ports, in order to allow congestion control of the set of streams together instead of as multiple independent streams. This allows better overall bandwidth management, faster response to changing conditions, and fairer sharing of bandwidth with other network users.

- The algorithm should rely on existing information about the incoming flows to provide feedback to the sender. Examples of this information are the packet arrival times, packet timestamps, packet sizes, packet losses. Not require any special support from network elements, e.g. ECN, etc.
- Since it is RTP streams used, then the backchannel should be RTCP, or header extension to RTP.
- The algorithm should quickly adapt to initial network conditions at the start of a flow. The initial adaption should be faster than adaption later in a flow.
- The algorithm should sense the unexpected lack of backchannel information as a possible indication of a channel overuse problem and react accordingly to avoid burst events causing a congestion collapse.

2.3 QoS and QoE

There are various ways to define and measure Quality of Service (QoS) and Quality of Experience (QoE). This following section will describe different ways to evaluate QoS and QoE involving WebRTC.

2.3.1 Quality of Service (QoS)

QoS is measured by evaluating the performance of the service. There are many ways of interpreting QoS. In [29] there are listed three different interpretations of the concept, which is described below.

1. *The delivery of a service in accordance with its specification.* This definition describes the service with its own QoS parameters with value. For example, a system with availability larger than 99.9%, a blocking probability less than 1%, and a setup time less than 200ms. The quality is depending on what extent these requirements are met.
2. *The end-user satisfaction with a service.* This definition is related to users experience with the service. If this description will be used in a context, it must be more detailed and concrete. Thus, this will be like the previous definition.
3. *The existence of mechanisms (in the network) for controlling the use of the different resources.* This definition is originally from using the Internet. The Internet is used to provide different services, including real-time services like live video stream. Those kinds of services make it necessary to control the transmission capacity, so they get enough bandwidth and small delay to be usable.

As is known, there are many definitions of QoS, but we stick to the definition stated by International Telecommunication Union (ITU), which is as follows:

> "Totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the users of the service." [19].

The definition of ITU describes to satisfy the needs of the user of the service, which only refers to the user using the service not the service itself. However, it is described in [29] that service is not necessarily a physical interface. A service is a set of functions that are offered on an interface between the user and the provider.

QoS depends on the actual service delivered and are often evaluated based on network statistics, such as jitter, bandwidth, loss and latency (required by some real-time traffic). On the other side, you have QoE which is measured by users experience of the service. When measuring QoS, QoE is heavily related. If QoS parameters like loss is high it will affect badly on how the user experience the service, known as the QoE.

2.3.2 Quality of Experience (QoE)

In comparison to QoS, QoE is measured with different parameters to the performance of the service. QoE is subjective and individual from various users. This means that even though QoS stays the same, QoE does not stand still, it varies depending of the users. Moreover, ITU has stated a definition of QoE and is as follows:

> “The overall acceptability of an application or service, as perceived subjectively by the end user.” [23].

This definition by ITU may confuse and get debatable. The ‘overall acceptability’ concept to measure may be unclear [23]. Based on the lack of the understanding, a new and more accurate definition of QoE was proposed by Qualinet [30]:

> “Quality of Experience (QoE) is the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and / or enjoyment of the application or service in the light of the user’s personality and current state.“.

In this definition, Qualinet, considers the personal factors in addition to system specific and context-related factors to evaluate the QoE. This means that measuring QoE is not evaluated based on only user’s expectations of the service, but also based on users’ feelings and how his/hers experience with the service change his/her emotions. QoE is subjective and it is hard and complicated to measure user’s feelings, expectations and personal relations. There are some factors influencing users QoE

when using WebRTC services. Those factors are human-related (personality and experience), the system (network conditions and application level aspects), service, and context [9].

2.3.3 QoS and QoE in Contex of WebRTC

The main feature of WebRTC is how easy it is for the user to use it, by not have to download extra software. For WebRTC services to be competing against well-established applications like Skype, it is important that the QoE and QoS are good in the WebRTC applications as the users are used to from other applications.

Although QoE and QoS are discussed as two different measurements, QoE is dependent on QoS. If the QoS is bad, the connection is expected to be poor, and then the user experience will likely decrease. There are other cases where the QoS parameters are fine, then the users presumptions will give different QoE-values. To help to improve and or stabilize the the different qualities A congestion controller can be vice to implement.

Chapter 3
Methodology

The methodology used in thesis project is divided in to four processes: literature study, development of service, experiments and evaluation of two algorithms. These processes are described in this chapter.

3.1 Literature Study

A literature study was conducted to research the topics, algorithms and technologies in this project to make better decisions carrying out the experiment and analysis. The literature study includes studying WebRTC, and technologies to set up a WebRTC-based video communication. Information about the different choices made about the technologies are described in chapter 4.

Further, there were a literature study of the concept network congestion and congestion control. Challenges and requirements of congestion control for real-time media were also looked at to get a better understanding when analyzing two different congestion control algorithms. Several articles, where IETF drafts were the baseline, were read and evaluated to to obtain better knowledge on the topic.

3.2 Development of Service

The development process of the WebRTC service is divided into three main phases.

- **System description.** First we define the system description. The system description contain the system architecture, functionalities, and technologies to be used.
- **Implementation.** Second, the implementation stage cover the actual development of the WebRTC application. This is the most time consuming part of developing a WebRTc service.

- **Testing.** Third, testing was conducted simultaneously and at the end of the implementation process, to make sure the service was working properly.

Also, more detailed description of the system and development process is described further in chapter 4 and 5.

3.3 Experiments

The primary goal of this thesis is to look at congestion control for WebRTC services and then the need for it. To verify a need for congestion control mechanisms, several Experiments With the implemented WebRTC Application were conducted. Two different studies were completed:

- A testing phase for the WebRTC service. The WebRTC application was tested on different computers, and laptops. It was also tested on different operating systems, and on different networks, from corporate networks to home networks and vice versa. All this to make sure the WebRTC service is working properly and is stable to use.
- After that, there was a group of users which used the application for real-time multimedia conversations. They gave feedback on their experience of the service and session, which then were collected as QoE data. While the session was going on, additional data was collected on perceived QoS.

Chapter 6 gives more details about the actual experiments with description of the tool used.

3.4 Evaluation of Two Algorithms

The evaluation of two algorithms were focused on IETF RMCATs recommended congestion control algorithms, GCC and NADA. These two were analyzed and compared against each other. The analysis was conducted by looking at the number of resources used by the algorithms and compared between the two algorithms. For maximum efficiency, the algorithm should minimize resource usage. The measurement parameters to measure resource usage are chosen based on parameters which measure the algorithm efficiency. This is considered most important to research. Here are the most known parameters with definitions:

- **Functionality.** Describes the quality of being suited to serve a purpose well. Is the algorithm working according to the requirements?

- **Architecture.** Describes how the algorithm is organized to get a better understanding of how the flow is in the algorithm.
- **Input data.** Describes the input data needed to do all the calculations in the algorithm. A lot of input data would slow down the algorithm computation.
- **Response time.** Gives an overview of how long it takes for the algorithm to complete. In addition, it gives an estimated response time of the algorithm. In association with real-time application, the algorithm must respond quickly.
- **Data storage.** Gives an overview of how much memory that is needed by the code, and the amount of memory needed for the data on which the code operates.
- **Implementation issues.** Explain implementation issues that can occur when implementing the algorithm. Implementation issues can also influence actual efficiency, for example the way in which the algorithm is coded.
- **Security issues.** Describes the security issues which can occur after implementation. These kinds of issues can have a big effect on the performance. If someone took advantage of the security issue, the algorithm would give false results or would not be working at all.
- **Total cost.** Describes the total cost of implementing an algorithm. Would the algorithm interfere with the rest of the service or computer, and if so, how it would interfere.

The evaluation was conducted to look at how the algorithms operate and the differences between the two mechanisms. Use of an inefficient algorithm can impact system performance. In time-sensitive applications, an algorithm taking too long to run can provide outdated or useless results. Also, an inefficient algorithm can require too much computing power or storage to run ,and again provide useless results.

Chapter 4
Design of the WebRTC Service

This chapter presents the design of the WebRTC application. It describes the service model and describes the software requirements specification. It also gives an overview of the technologies used in the service.

4.1 The WebRTC Service Model

My web application is designed like figure 4.1 is presenting. I have a WebSocket server as a signaling server for WebRTC. The signaling server is implemented with node.js. The peers connect to the server directly by a WebSocket handshake. When peers are connected to the signaling server and want to connect to another peer, the web server notifies the peer by pushing the message instantly. All contents of information to setup a connection between peers are transferred through the server. After the peers have agreed on the connection, media packets are transferred directly between the peers.

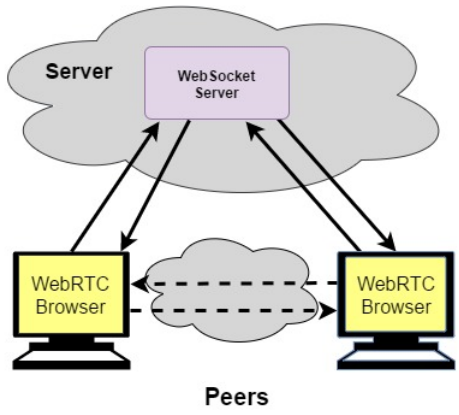


Figure 4.1: My WebRTC design model

4.1.1 Signaling

A WebRTC service does not have any standard signaling protocol, because the developers wanted to maximize compatibility with existing technologies and to avoid redundancy. Because of that there are many possible signaling protocols to use. A high Level study of Three protocols has previously been conducted by others, and has helped determine which protocol to use in this study.

SIP:
SIP is a signaling protocol designed to establish, modify and terminate multimedia session over the Internet. An example of a well-known technology that uses SIP is VoIP technology [31]. The advantages and disadvantages of using SIP[33] are listed below in table 4.1.

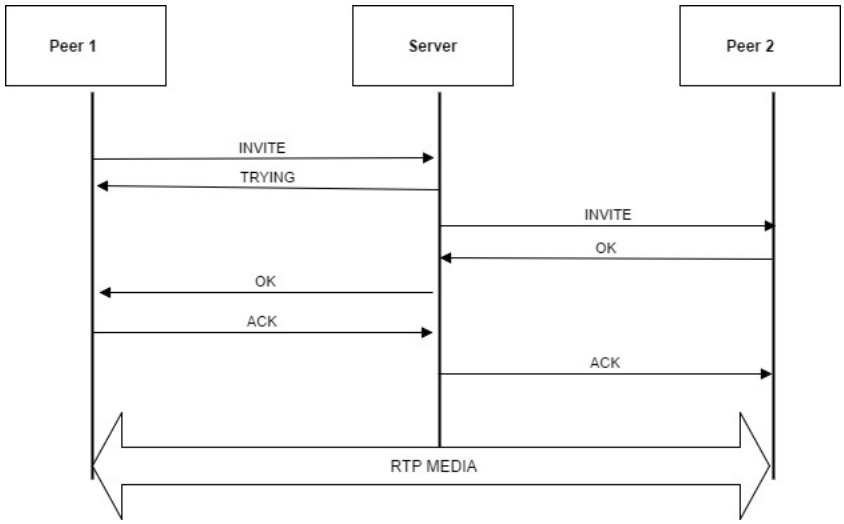


Figure 4.2: SIP procedure

SIP works as it is shown in figure 4.2 [32]. After the peers are registered in the server. Peer 1 sends an INVITE request to peer 2, while receiving information back from the server telling Peer 1 it is trying. The INVITE request contain the session description for peer 1. When client 2 accepts the INVITE, it sends back an OK message. The OK message contain peer 2's session description. Then peer 1 acknowledge the acceptance and the session is established. Both peers can now send media packets to each other.

Advantage	Disadvantage
SIP is new technology	SIP is mostly used for telecommunication
SIP is independent and flexible of the type of media used	SIP is used in more complex systems with proxy server, location server, registrar and user agent.
Messages in SIP is sent in clear text, easy to trouble shoot	Processing text messages with SIP can take a load on bandwidth
SIP can an accommodate multiple users with different capabilities. One user with video and audio, and another with only audio	The INVITE message in SIP contains a lot of information
SIP has a short session handshake	

Table 4.1: Pro and cons with SIP [33]

XMPP:
Another signaling protocol that can be used in development of WebRTC service is XMPP. XMPP is an open Extensible Markup Language (XML) protocol for real-time messaging and request-response services[11]. The advantages and disadvantages of using XMPP[46] [12] are listed below in table 4.2.

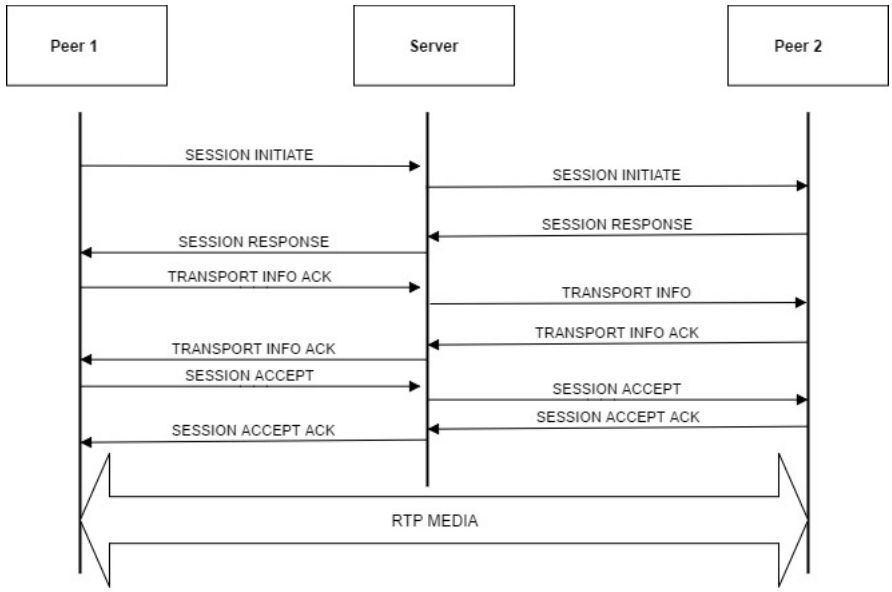


Figure 4.3: XMPP procedure

The XMPP protocol has a long process to establish a connection between peers. Figure 4.3 show the XMPP call flow. First, peer 1 initiate to start the session. Peer 2 respond with acceptance to start a session. Next, peer 1 sends transport information which include the IP address and port number etc. Further, peer 2 respond with a transport information acknowledgment. Then the session accept is sent from peer 1. Client 2 send back a session accept acknowledgement and finally RTP media can be sent between the two peers.

Advantage	Disadvantage
The protocol is free, simple and open	XMPP has a high network overhead, since it uses XML
XMPP can communicate with other protocols on different servers	The protocol is coded as a single long XML file. XML is text based.
XMPP is often used in instant messaging	XMPP is used in decentralized system
	The protocol has a long session handshake
	Limited scalability with XMPP . Cannot provide modification of binary data.

Table 4.2: Pro and con with XMPP[46] [12]

WebSocket:
WebSocket is a signaling protocol which enables two-way communication between a client and a server [45]. The WebSocket protocol is an independent TCP-based protocol. The protocol starts with an opening handshake, followed by basic message exchange, and then a data transfer data transfer. The advantages and disadvantages of using WebSocket[40] are listed below in table 4.3. Because the WebSocket is providing full-duplex communication channels, it is very attractive to use in transfer of real-time data.

Figure 4.4 shows how the call flow is with WebSocket protocol. It starts with a handshake between the peers and the server, which are relatively small compered to the other protocols. When that is done, the connection is established. After the peers have connected to the server, it makes it easier for the server to push the data to the peer almost immediately, after already received the data from the other peer.

After some consideration in looking at the different options, WebSocket was chosen to be the signaling protocol to use in this service. WebSocket has a lot of possibilities when implemented. It has the ability to define sub-protocols, messages

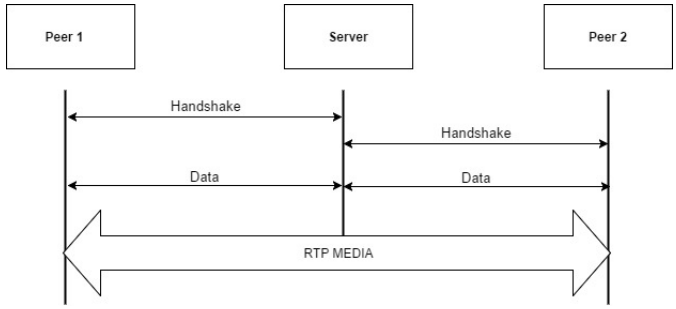


Figure 4.4: WebSocket procedure

Advantage	Disadvantage
The messages in WebSocket can be textual or binary	WebSocket considered new technology and not supported everywhere in browsers and web servers and proxies
WebSocket give fast messaging	WebSocket has problems with reconnections in services
WebSocket has scaling capabilities	
WebSocket has full duplex client-server communication. Delivers communication between the client and the server in both directions simultaneously	
WebSocket keeps the connection open on the servers for the duration of the time the user is interacting with the page.	
WebSocket has the ability to define sub-protocols, like XMPP	

Table 4.3: Pros and cons with WebSocket

can be textual or binary, and it has fast messaging. The other protocols had their advantages, but WebSocket appeared more attractive with its advantages.

WebSocket has disadvantages, same as the other protocols. It is not supported in every browser, web server, and or proxy, but there are ways to implement the WebSocket with a fallback option to solve this problem. Also, the other protocols had more complicated disadvantages which need to be addressed. WebSocket become the best choice for the signaling protocol in this WebRTC service.

4.2 Software Requirements Specification

Software requirements specification describes the functionalities that the system needs to be developed. Listed below is a set of requirements for this software system followed by the Institute of Electrical and Electronics Engineers (IEEE)'s standard [18]. These requirement specifications made designing the system easier and it helped increase the efficiency of the implementation process. The overall design of the system was simplified to make it user friendly.

4.2.1 Functional Requirements

Functional requirements describe the functionalities that the system require to perform. One of the main functions of the WebRTC-application is to provide peer-to-peer communication with video and audio.

In case there is lack of audio or video in the conversation, an instant messaging feature is required. It will contribute to the communication, if there are any problems to understand each other.

For security reasons, users should be able to determine who they communicate with. For that reason each user should be able to choose which room they will connect to. The two users that want to communicate have to write the same room name to connect to it.

4.2.2 Non-Functional Requirements

Non-functional requirements describe all the other requirements remaining, which are not included in the functional requirements. For example, the requirements which are not needed for the system to perform. One non-functional requirement is that the text in the system should be written in English. Another non-functional requirement is that the system shall at least be supported by Google Chrome web browser.

4.2.3 External Interfaces

The WebRTC application is a web interface developed in JavaScript and is accessible through a web browser. Since not all web browsers support WebRTC technology, the WebRTC service is only used on the web browser Google Chrome. The design of the WebRTC service is focused on personal computers, and not prioritized to be tested on tablets and smartphones.

4.2.4 Performance

Every functionality in the WebRTC-application is handled immediately, when requested. When a user goes to the web page, the browser and the server creates a

connection, for the user to be able to start using the service. First, the user have to create a room, which is done by enter a username and room name. Afterwards, an audio and video stream of the user appear. Then the user must wait there until another user entering the web-application address and enter the same room name. When the next user appear, an exchange of data happens right away to determine if the connection between both users is using the same signaling protocol and whether or not they can communicate with each other.

4.2.5 Attributes

The main characteristic of this type of system is that it has high availability. One is not dependent on location to use it and one can use the application whenever one want. When using the application with real-time multimedia conversations online, you are depending on it to be reliable. You hope that when you are in a video conference, you don't lose either video or audio, or both. This is critical for this kind of applications. Since there is no login page onto the application, the security of the WebRTC application has not been prioritized during implementation. Still, there is a security feature in the service, that only allows two peers into one conversation. To get in the conversation both peers need to write the same room name.

4.2.6 Design

The WebRTC application is designed using Hyper Text Markup Language (HTML)5 (described in more detailed in Section 5.3) and Cascading Style Sheets (CSS) (described in more detailed in section 5.3). The design of the WebRTC application was defined by myself.

4.3 Technologies in Use

Building this service, it takes different technologies in use, which figure 4.5 is presenting. One access WebRTC components with JavaScript APIs. The API in use are the MediaStream API which represent an audio and video data stream, and PeerConnection API. The WebSocket protocol enables two-way communication and is in this case used for signaling between client and server. Node.js ans socket.io is a JavaScript library used for enabling communication between client and server.

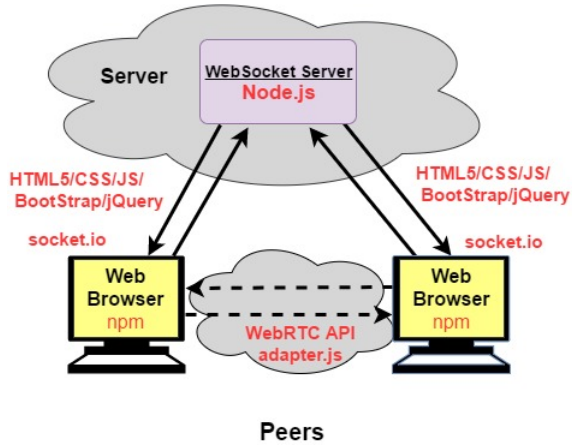


Figure 4.5: My WebRTC design model with technologies in use

JavaScript
JavaScript is an object-oriented and lightweight programming language, which is mostly used in developing web-pages. It is important to choose a language that has a good response time for the elements in the developing project. Advantages of using JS is that it is fflexible, and that is runs on both the client-side and the server-side. For this project, it is using both. Client-side JS means that the users web browser conducts all the computations and logic[20]. Server-side JS means that an application can communicate with an outside service like a database [20].

JS is perfect for changing HTML contents by showing and hiding elements, altering the styles of elements, and validation forms etc. On the other hand, JS is not recommended for handling security-sensitive data, for example, handling passwords. Since the WebRTC-application elements (audio, video etc.) require a fast response time and does not require password management, the WebRTC-application uses JS. JS will give a responsive web interface with dynamic functionalities.

WebRTC API
The WebRTC API is technology which makes it possible for web applications and pages to capture and optionally stream audio and/or video media. It also exchanges data between browsers without requiring a middleman. The WebRTC API makes it easier to share data and perform video conferencing peer-to-peer, without plugins, through the web browser. More technical details are described in section 2.1.1.

HTML5
JS, CSS and HTML are the main languages that build up a web page. It describes and defines the content of web pages [16], and create mobile- and web applications. HTML is the standard markup language used for organizing and presenting on the World Wide Web (WWW). It was in 2004 that W3C launched the latest version of HTML, HTML5, which as of 2017 is the current standard of HTML. HTML5 supports the multimedia and is supported in all modern browsers [17]. Since the WebRTC-application is a web-page, HTML5 is utilized.

CSS
CSS is one of the core languages of the WWW, and it describes the style of the content on a web page. CSS has been standardized by the W3C. In comparison to JS, CSS is a style sheet language and not a programming language. The style sheet can define the presentation of a document written in HTML or another markup languages [6] [7]. **jQuery**
jQuery is small, fast, and feature-rich JS library, as the jQuery motto says: "write less, do more" [21]. The purpose of jQuery is to make it much easier to use JS on web pages [22]. One feature of jQuery is that it handles cross browser issues. In this WebRTC-application, JQuery was used as an extra feature, so the page would load faster. Also, to decrease the time of fixing problems after implementing the WebRTC application, jQuery was used.

Node.js
Here the JS runs on the server-side. Node.js is an open-source and event-driven tool for developing easy and scalable server-side web applications and networking applications. [27]. With Node.js, it makes it easier for developers to implement a high-performance HTTP-server with customized behavior [24]. In the traditional way, HTTP requests and responses are handled as isolated events. Nevertheless, Node.js handles the requests in parallel, which allows the application to handle files faster [41]. This is beneficial when working on real-time audio or video encoding. For this reason, the WebRTC-application uses Node.js to achieve better performance.

socket.io
Socket.io is a JS library for real-time web applications. It is divided into two parts: a client side library which runs in the browser, and a server-side library [34]. Socket.io uses primarily the WebSocket protocol, but if needed can fallback to other methods, while providing the same interface [24]. Features that socket.io have is connecting multiple sockets with a server-side room, and the opportunity to store data associated with clients. Like Node.js, socket.io is event-driven. Since the WebRTC-application

is using WebSocket protocol for signaling, socket.io was the best alternative to use, especially since it has a fallback feature if older computer does not use the WebSocket protocol.

adapter.js
Adapter.js is a JS shim library, which creates a common API for WebRTC in the browser. For example, there are several ways to call getUserMedia() [44]. Mozilla Firefox can call the function from a file, but in Google Chrome and Opera, all that uses the function must be run from a server. Adapter.js is used to protect applications from specification changes and prefix differences [2]. For this reason, the WebRTC-application uses adapter.js to make it work in most web browsers without problems.

Bootstrap
A long with HTML, CSS and JS, Bootstrap is the most popular framework for developing responsive web sites [3]. Bootstrap is a free and open-source framework used in designing the front-end of web applications. The content of Bootstrap is HTML- and CSS-based design templates which contains the design for typography, forms, buttons, navigation etc. Instead of defining the design of buttons, forms, navigation etc. with CSS itself, programmers can easily include the Bootstrap library and save time. The WebRTC-application uses Bootstrap framework elements, such as input fields, buttons, drop downs etc.

npm
Npm makes it easy for JS developers to share and reuse code, and it is possible for others to update the code you are sharing [28]. This makes it easier to program because it is possible to get the code that has solved your problems.npm is a tool used to easily install socket.io to the WebRTC-application. For future developing, in the event of someone releasing a new version of npm projects, npm is an easy way to update the projects.

Chapter 5
Implementation of the WebRTC Service

This chapter will describe the implementation process and present the challenges during the process. It will also give a final description of the WebRTC application.

5.1 Iterative Development Model

The implementation process of the WebRTC service followed an iterative development model illustrated in figure 5.1. The whole implementation process is repeated numerous times to the get final product. Followed are the phases described in more detail.

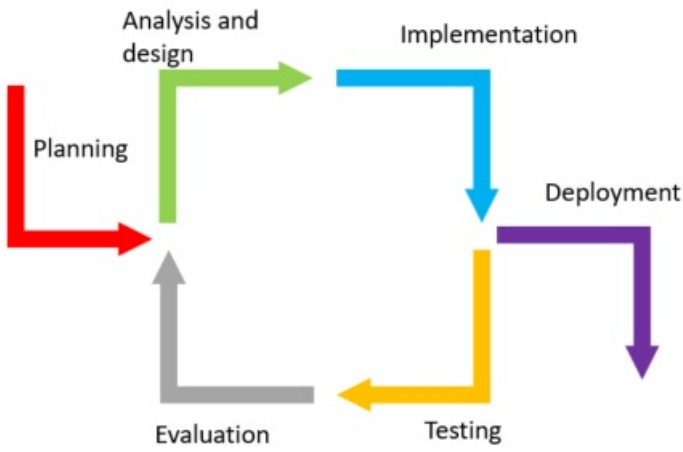


Figure 5.1: Iterative development model

1. **Planning**
   The first step is planning. I planned how this implementation process would be conducted, and how long each part took to carry out. Found out what kind of components were needed and did some last research on the technologies.

2. **Analysis and design**
   In this phase the design of the system was determined. I chose which kinds of technologies to use and how the system would look when completed. More details are described in chapter 4.

3. **Implementation**
   Implementation is where you transform the design into code. The whole system was divided into small implementation parts to make sure sure it worked fine, and that the whole system was fully implemented.

4. **Deployment**
   After implementation, the system gets deployed to see how it it worked online among other systems.It was also deployed to see the integrated parts work together. Each part needs to pass the next two phases to be considered fine.

5. **Testing**
   The testing phase is crucial because every part need to be tested to ensure that the integrated parts turn out well as part of the whole system. If there is any error detected during the tests, then that the part need to be traced back in the cycle to be re-implemented until the tests are a success.

6. **Evaluation**
   After a part of the system has been tested, there is an evaluation phase. This phase evaluates how the part fits in the whole system. The question asked here is does the part fit into the whole system? If it does not, then the cycle starts again at the analysis and design phase to redesign a new part to the system.

5.2 Detailed Description of the Implementation Process

This section describes in detail how the WebRTC service was implemented, step by step. First step was to implement the view of the application where the local media stream is showing. The view is what the end-user is presented and how the user interacts with the application. The user's actions and inputs are handled in the view. To access the local media stream, one have to access the MediaStream API via getUserMedia-method. Technologies used in the view are HTML5 and CSS.

In addition to access and managing local media stream in the browser, a test chat was implemented on its own, to learn the DataChannel API and how it works

together. All this was to make it more efficient later in the implementation process, because an instant messaging feature is required to be implemented in the application.

Next step was implementing the communication part, where media is allowed to be sent to and received from another browser. A mechanism is needed to coordinate the real-time communication, and control messages are exchanged between the peers. This is known as signaling. Before adding a server, the RTCPeerConnection API was implemented to get peer-to-peer behavior on a single machine. Calling a new RTCPeerConnection creates a RTCPeerConnection object, which is part of the communication channel between two users.

To make sure the WebRTC application is properly working, a real signaling channel and a server is required. The necessary software was downloaded to implement the WebSocket protocol, which handles the signaling. The server script was set up to handle the messages from the peers. After the main feature was working and signaling procedures were working correctly, the instant messaging feature was added.

The final design of the view of the application was applied as one of the last implementation part prioritized, to ensure that the functions were working correctly. Bootstrap was included to get a smoother look and more a responsive application.

The last step was deployment of the WebRTC service. It was to make sure the application was available and reachable. By having the application deployed it made it easier to test it and evaluate if everything was working properly.

5.2.1 Testing

During every part was implemented, testing was conducted. Every functional requirement (presented in Section 4.2.1) was tested and non-functional requirements were also tested. For example, testing the WebRTC service in a specific web browser (Google Chrome). Testing was a crucial part of the implantation process to assure everything was working properly.

5.2.2 Code Implementation

It may look easy to implement a WebRTC application, but it takes time to all details are implemented correctly and all components are communicating with each other. The development of the WebRTC service took two months and two weeks, and consists of about 565 lines of JS, HTML and CSS code.

5.3 Challenges and Decision Making During Implementation Process

At the beginning of the implementation process, it is not possible to identify all the challenges that will appear during the process. This section will present the most significant challenges that emerged during the implementation of the WebRTC-service and what decisions that were made to handle these challenges. Hopefully others will be aware of these challenges and learn from this to make their implementation of a WebRTC service smoother.

Not able to get video or audio in the browser
The main feature of WebRTC service is video and audio. Without video and audio stream the service is useless, because it wont be possible to use the service at all, if it does not serve its function as a multimedia communication tool.

To acquire the audio and video streams in the browser, two methods are used, getUserMedia() and createObjectUrl(). getUserMedia() asks the user for permission to use their webcam or other video or audio input. The createObjectUrl method instruct the browser to create and manage a unique URL associated with a media stream object.

This problem occurred because of lack of knowledge about the WebRTC APIs. The WebRTC API needed to be researched again to fully understand the technology and how everything was connected.

Separating the users of the call
Two users connect to a web server one by one. When they want to communicate to each other, it is important when implementing that the web server separate which user is starting the call and which user is joining. If not, the system will not work properly because the server does not understand that there is a room before a user wants to connect. Every user would start the call and establish a room with the same room name, and now they are connected to each other.

Both users are sending information to the server, but the server has nothing to distinguish the clients' messages and where to send them. Figure 5.2 shows the error in the communication between clients and the web server. When the server get a message from one client, it reacts with broadcast to both clients. Then both clients act on the message, which ensures chaos in the system and allow no communication between the clients.

To solve this communication problem with the web server, one have to make the web server understand who the initiater of the call is, and who is joining the call. By

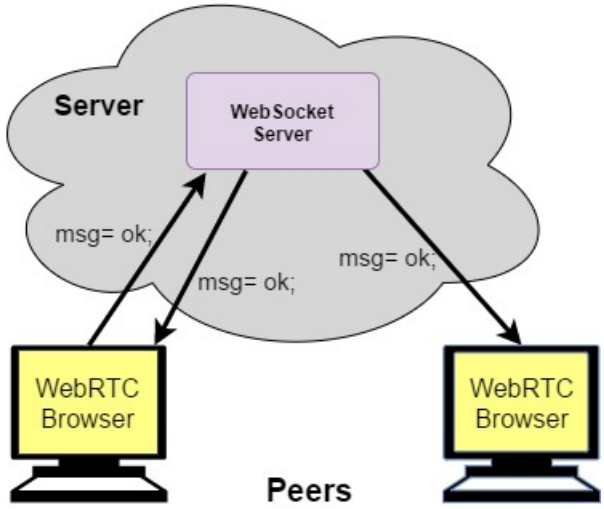


Figure 5.2: Communication chaos between server and clients

setting a Boolean parameter, isInitiator equals true for the first client to request to start the call and it is the first to establish a specific room, the server can separate the two clients. Next time a client want to establish a room with the same room name the web server check the parameter and if false the client is put in the already established room where the other client already is.

Signaling messages
Same as the problem above, when two users want to communicate with each other they first have to agree on how to set up the communication. This is done with signaling. Signaling is sending necessary messages between the users so they can agree on how to set up the connection.

In section 2.1.3 the signaling in WebRTC is described in detailed. Figure 2.3 shows all the necessary messages required to setup connection. During implementation process a crucial mistake was done. By not implementing the signaling messages in right order as in figure 2.3., the connection setup failed and the service are not working properly.

The important lesson here is to learn the system setup and not make fast decisions when you are uncertain. Be precise in the implementation process or extra work will appear and extra time will be lost.

Software version
Extra software packages needed to be installed to fully develop the system. When downloading these softwares, one usually chooses the newest version of the software, because one take it for granted that the newest version of the software is the best. Even though the newest software was fully tested before it was published, some errors can still appear after publishing.

During this implementation process the assumption mentioned above was made as well. The newest software was downloaded and used. When it was integrated with other components, error occurred. There was problem with integration between different software in the system. To solve this problem, an older version of the software was installed as part of the system.

Using getUserMedia() on insecure origins in Chrome
After deployment of the WebRTC service, the service was to be tested to make user it worked everywhere, different browsers etc. In the testing phase a big error was observed.

Some years ago Google Chrome decided that insecure web pages will not be fully supported in Chrome. To use the getUserMedia-method in the WebRTC API the web address needed to be certified and secure. One has to change the http protocol to the secure https protocol. All webpages from insecure origins would not properly appear in Google Chrome. This is a new feature decided by Google to implement in Chrome browser, and articles read about developing WebRTC services did not mention this obstacle.

From this day, there is no backdoor to avoid this feature in the Chrome browser. For this project, it is very important the WebRTC service is working properly in the Chrome browser, because of the tool being used for acquiring statistical data have a limitation to be web browser dependent where the web browser is Google Chrome. To make the WebRTC service work in Chrome browser, the service was deployed on a secure domain. Because of poor information of this feature, additional time was added to the implementation of this service.

5.4 Description of the Final WebRTC Service

This section describes the final WebRTC service with its advantages. A detailed description of the features and with an explanation of the benefits.

To be able to use the application, there is a need of a device which is connected to the Internet. On the device it is necessary to have an available web browser. In the web browser, enter the web address for the service and the service will appear

like figure shows 5.3. This gives easy access to the service, which will make it more desirable to use for all users.

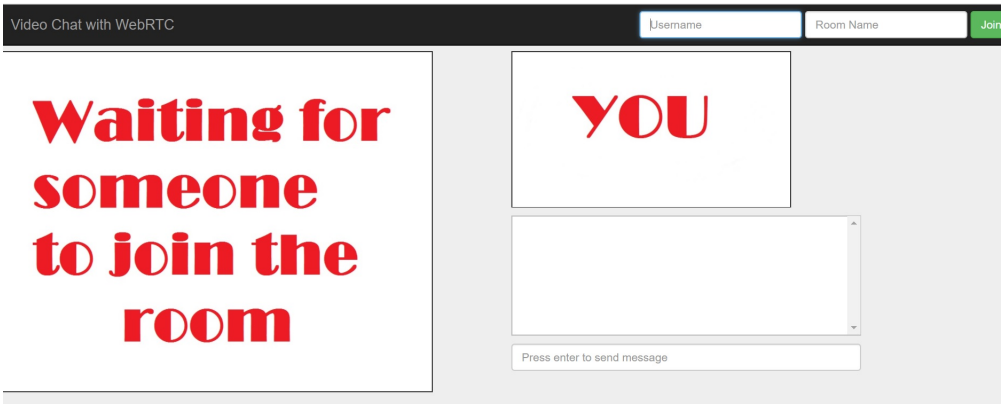


Figure 5.3: First page of the WebRTC service

You will not be able to anything in the application until you have entered a room. The room helps users to decide who they want to talk to, in a scenario where two users enter the same room name. This is a small security function to enable some privacy. To use the service, the first thing to do is to enter a username and a room name in the field above to the right, illustrated in figure 5.4. The username is for separating which user said what in the chat window.

After entering the required field entry, a local audio and video stream of yourself will appear to the right on the screen. With real-time multimedia features makes it simpler to communicate between the users.

Figure 5.4: Entering a username and a room name

To the left on the screen a picture with the writing; “Waiting for someone to join the room”, means there is one user in this specific room and the user are waiting for someone to join the same room. The second user, which want to communicate with the first one, also need to enter a username and it is very important to enter the same room name.

When another user has entered the same room name and after all the messages between the peers to set up the connection is done by the devices behind the users

eyes, a remote multimedia stream appear to left in the screen, shown in figure 5.5. Both users can easily communicate through audio and video.

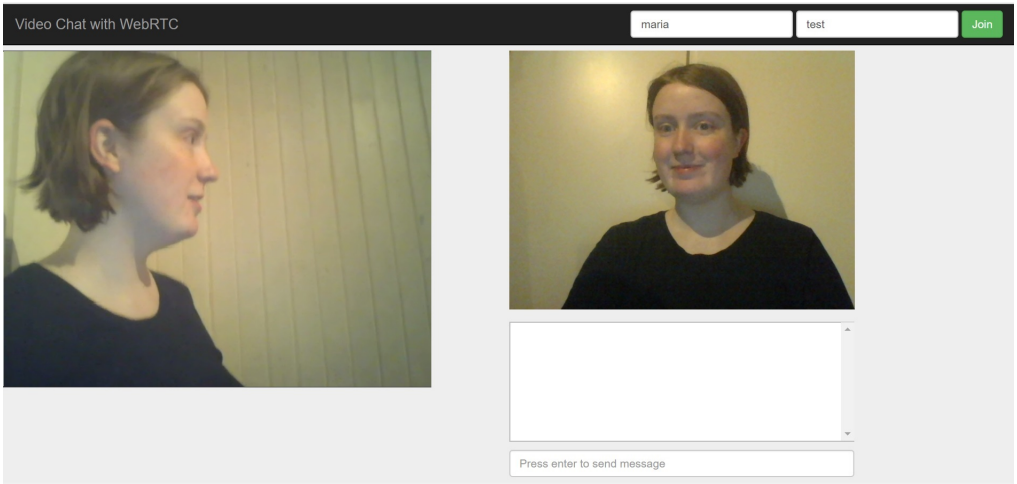


Figure 5.5: Peer-to-peer communication

In addition to multimedia communication, an instant messaging feature is added to the service. The instant messaging feature where added to make it easier for the users to communicate if audio and or video are not working properly. Figure 5.6 shows two different users texting each other in an easy way. This feature is placed below of the local stream of video.

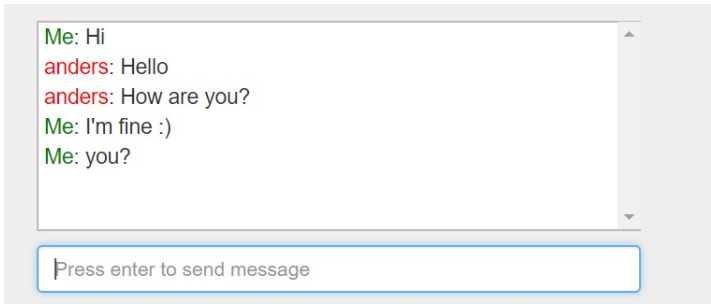


Figure 5.6: Instant messaging

5.4.1 Limitations

In this section, the limitations of the WebRTC service is presented. Because of the scope of this thesis and with the challenges, that emerge during implementation,

there was constrain on time. Followed are the limitations of the service covered.

Number of Clients
This WebRTC service has a limited number of possible clients, that can communicate in a specific room. If there was three clients who wanted to start a multimedia conversation using the WebRTC application, it would not be possible. For the future, it would be attractive to make the application available for multi-party conversations.

Security Limitation
There are some security mechanism implemented in the WebRTC service. First, the clients have to enter a username and a room name to be able to use the application. Second, clients who enter the same room name, can communicate with each other. This gives some security to the users, because they can choose who to communicate with. It would hence the security to the user to implement a control site to accept who is joining the room. this will make sure the right user is connecting to the right room. Also, this will ensure the privacy of the users.

Data Transfer Limitation
The WebRTc service includes an instant messaging function, which makes it possible to only send text to each other and the messages are displayed. The API applied in this function have the possibility to also implement file transfer. However, this is not a prioritized function needed to make the WebRTC work. This is only a add on to make the application more attractive.

GUI Limitations
GUI limitations are described in this section to highlight the limitations of the UI of the WebRTC service.

- **Limited Screen Size Adjustments.**
  The WebRTC application has two windows presenting the local and remote video streams. These windows are set to specific sizes. Multitasking when communicating in the WebRTC application, would make the user experience better with the possibility of adjusting these window sizes.
- **Limitation to Properly Adapt Different Devices.**
  When implementing a WebRTC application with audio and video features, the video stream need to be visualized at all times. When changing between devices and screens, the video streams window should automatically adjust to the new screen settings. This is not properly adjusted in this WebRTC service.

Chapter 6
Experiment and Results

This chapter covers the different experiments which were conducted during the thesis period. The experiments are presented in detailed in this chapter. Further, the results from the experiment are presented in this chapter. This section describes in more detail around the experiments. The experiments are separated into two part and both are described in this section.

6.1 Detailed Description of Experiment Phase One

The first phase is important for the rest of the experiments. The data collected later on need to be correct and reliable, therefor a set of tests were done to make sure the WebRTC service is working properly. About 36 different tests were conducted in different locations where the devices were. The tests were a combination of network types, devices, Operating System (OS), and web browsers, which is presented in table 6.1. Each test took about five minutes to carry out, and were conducted at the end of May.

Network Type	Device	OS	Web Browser
Wireless	Stationary computer	Windows	Google Chrome
Cable	Laptop	iOS	Mozilla Firefox
		Linux Mint	Opera

Table 6.1: Equipment for experiment phase one

To make sure the WebRTC service was available on different browsers. Browsers like Mozilla Firefox, Google Chrome, and Opera was tested. The WebRTC service was also tested on different operating system to be sure that it has nothing to say which OS serving the application. The OS tested were Windows 10, iOS, and Linux Mint. In addition, different computers were used during these tests. A variation of laptops and stationary computers. Because nothing can interfere with the application.

Last, the service was testes on different networks. Network is an important part of the WebRTC service for the application to be available at all time. There exists many network types, but there were only three network types chosen for this experiment.

6.2 Detailed Description of Experiment Phase Two

The second phase of the experiments is to use the WebRTC service to verify the need of a mechanism to make the sessions more stable and reliable. For this part 10 participants used the WebRTC service for multimedia conversations.

During and after the multimedia conversation, the participant need to fill out a form to feedback on the usability of the WebRTC-service and the quality of the session. Give feedback on their experience, and if there were occasions of slow movement in the video, no audio from the speaker, or if they lost their connection during the conversation. At the end of the session, all the participants sent the completed form back.

During each session, session related statistics were collected with Chrome WebRTC Internal Interface was used. Before closing the browser after a conversation, each user needed to manually download the statistical data. The statistical data related to the session is needed to look at the performance to the WebRTC service.

6.2.1 Technical Setup

To conduct these experiments there are variables which need to be in place. Followed are these different variables described.

Participants:
In this study I had a total of 10 participants. Almost all the participants are affiliated to NTNU. Seven were students, and three were adults working in different companies. Five of the participants were female and the other five were male. They were all between 20 and 52 years old. All the participants had different experience in using computers and real-time communication services prior to the experiment. Since WebRTC is user-friendly and easy to use, there were no requirements of knowledge or experience in using computers for the participants.

Session Information:
In table 6.2 a description of the sessions are presented with information about device, OS, location and network type. Each participant was paired up to carry out peer-to-peer multimedia conversation with the WebRTC application. The duration of each session was limited to approximately fifteen minutes. In the experiment the participants did not sit in the same location during the session. It was important

to make it as realistic as possible, so the participants were in their homes and work places. Before the conversation the participants were thoroughly instructed on how to download the data collected.

Session Id	Participant	Device	OS	Location	Network Type
1	A	Laptop	Windows	Trondheim	Wireless
	B	Stationary	iOS	Bergen	Cable
2	A	Laptop	Windows	Trondheim	Wireless
	B	Laptop	Windows	Trondheim	Wireless
3	A	Laptop	Windows	Trondheim	Cable
	B	Laptop	Windows	Trondheim	Wireless
4	A	Laptop	Windows	Vinstra	Wireless
	B	Laptop	Windows	Trondheim	Wireless
5	A	Laptop	Windows	Trondheim	Wireless
	B	Laptop	iOS	Trondheim	Wireless

Table 6.2: Information about each session

Hardware and Software Details:
For the first phase, the testing phase, I borrowed different laptops and stationary computers from friends and family to test the application. Also, on these computers there were required some different web browser to make sure the application was stable at all time.

For the study with the volunteers, there were expected that every participant had their own computer. It did not matter which kind of computer or system running on the computer. That is because the WebRTC application were thorough tested beforehand. When carrying out the multimedia conversation the participant were limited to only use Google Chrome web browser, because of the tool used to collect the data.

Network Conditions:
For the network conditions for the different conversations, I could not control. All to make it more realistic as possible. The network conditions were based on the participants own home or company network. In some cases there were additional people on the network during the conversation, which is a realistic scenario.

6.2.2 Network Parameters

Bandwidth
Bandwidth can be defined in numerous ways. For example, it defines the channel capacity, net bit rate, or the maximum throughput of a logical or physical communication path in a digital communication system. Bandwidth can we say is how much data per unit of time a network can handle. Bandwidth is usually measured by Mbps. Applications running on big bandwidth works perfect, but when the bandwidth decrease, the Application slowly start working more poorly, and a user would experience bad QoE.

Packet Loss
Packet loss is defined by the number of packets which doesn't arrive to its destination, in contrast to the number of packets sent. Also, packet loss is measured as percentage of packets lost, in respect of packet sent. Packet loss often happens when the network is congested. When packets containing video or audio file are lost, user would experience poor quality. There different ways causing packet loss, such as faulty networking hardware, faulty network drivers or a packet drop attack.

Delay
Delay is how long it takes for a packet to travel from source to arrival at its destination. The Round Trip Time (RTT) can also be used to describe the delay, by measuring how long it takes from a source to destination and back again. The delay will also influence the users QoE. For example, the user will experience slow video stream or long pauses in a conversation. Latency and delay are both used for the same meaning.

Jitter
Jitter is closely related delay, because delay is the difference in packet delay. Jitter is measuring the time difference in packet inter-arrival time. Video and image jitter can occur when the horizontal lines of the video image frames are randomly displaced because of interference during video transmission. Jitter is an important factor determining QoS of the performance of the network. Jitter is sometime imprecise, so the standard-based term most commonly used in computer networks, is packet delay variation.

6.2.3 Collecting Data from Sessions

There are different ways to collect data from multimedia sessions. This section gives an overview of the tools used. I distinguish between session related statistics, which uses Chrome's WebRTC Internal Interfaces, and user feedback gathered from forms filled out by participants in the experiment.

The session related data are used to verify that there was altercation in the

network during the sessions. User feedback is a helpful way of seeing how the user feel about the service and reacts when there are changes in the network. All these tools are needed to verify and analyze the data from a session.

Google Chrome's WebRTC Internal Interface

Google Chrome's WebRTC internal interfaces[1] was initially developed for WebRTC application developers to understand the features and functions of their WebRTC service. Chrome's *webrtc-internals* uses one the API in WebRTC, getStats. getStats gives a set of methods for collecting the peer connection or media stream track statistics. The statistics collected by the API are gathered from received RTP and RTCP packets.

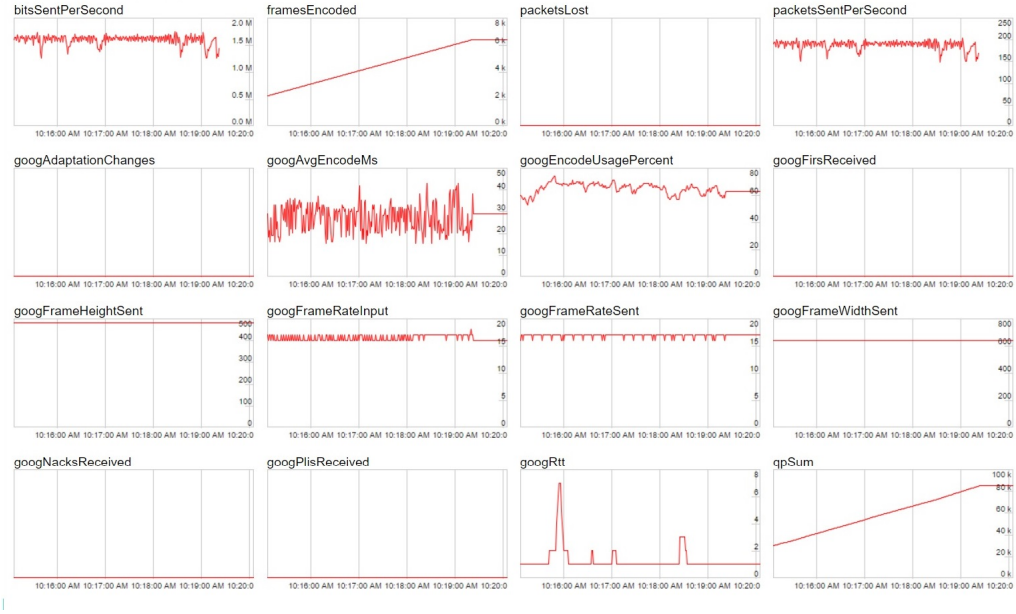


Figure 6.1: Screenshot of all graphs for receiving video (from *webrtc-internals*).

The tool offer a visualizing of statistics in real-time(illustrated in figure 6.1) and has the opportunity to download all the information of a dump file to perform post processing analysis.

To use the Google Chrome's internal interface, each participant must open the interface before starting the conversation. When the session is ended, not close the browser window. There is a necessity to only use the web browser, Google Chrome when using this tool. At the end of the conversation, each participant retrieve

[1] Adr.: chrome://webrtc-internals

the chrome statistics synchronously. It is important to download the statistics before closing the browser, otherwise the statistics are lost. The file downloaded are organized in a JavaScript Object Notation (JSON) format.

There is a big amount of session related statistics collected during a session. Everything from information about the WebRTC API, bits sent per second, packets lost, packets sent per second, and RTT etc. Not all of them are essential for this project. The most relevant session related statistics that are valuable for the project are found in table 6.3.

Parameter	Value	Media	Source
bytesReceived	Integer	audio, video	receive
bytesSent	Integer	audio, video	send
googJitterReceived	Integer	audio	send, receive
googRtt	Integer	audio, video	send
packetsLost	Integer	audio, video	send, receive
packetsReceived	Integer	audio, video	receive
packetsSent	Integer	audio, video	send

Table 6.3: List of statistics supported by Google Chrome's WebRTC Internal Interface

Google Chrome's WebRTC Internal Interface limitations. With every tool, there are limitiation and Google Chrome's WebRTC internal interface are no different[10]. In the following a description of the limitation of the design of the Google Chrome's WebRTC internal interface and the limitation of Chrome statistics.

- **Manual downloading of statistics:** The statistics in Google Chrome are fully visualized when observing in real-time, and can also be downloaded to perform post processing analysis [10]. However, Google Chrome's webrtc-internals requires the users to download the statistics immediately after a session. If the user closes the browser window before downloading, the statistics will be lost.
- **Limited number of sample points:** Google Chrome's webrtc-internals collects only the latest 1000 sample data. Older data will be lost.
- **Undocumented Chrome statistics extensions:** There will occur difficulties to in analyzing the downloaded statistics, because of lack of documented definitions of the WebRTC statistics attributes. This causes some challenges in reliably analyzing the collected data [10] of Chrome Statistics.
- **Imprecise sampling time:** The Chrome statistics are collected at each of the participant's web browsers. This means to be able to analyze the statistics,

the statistics from all browsers must be downloaded and manually combined and synchronized [9]. This implies that the statistics are recorded at the same time [10].

- **Web browser dependent:** To be able to use this tool, Google Chrome's WebRTC Internal Interface, you may only use Google Chrome browser.

- **Fixed sampling time:** Google Chrome's WebRTC Internal Interface uses fixed sampling time, which is one second. This setting cannot be changed.

Although, the Google Chrome WebRTC internal interfaces have some limitations, but according to [10] and [9] the statistics collected from this tool are useful, only if these limitations are known and handled accordingly.

QoE measurements
For this experiment, I wanted to collect feedback from users about the service and session. The feedback is gathered in a quantifiable way. It makes it easier to analyze and discuss, compared to having the participants describe freely how their QoE was. The questionnaire was inspired from articles written at the Department of Telematics at NTNU [9].

This experiment has a total of 10 users participating. This means there is an error margin, which need to be acknowledged. One cannot be too concluding when looking at the data collected from participants. However, it is possible to use the data to get a pointer on how this results are effecting or matching the problem statement in this thesis. To minimize the error more participants are needed, but it is too time consuming for my project to carry out. I choose to use the feedback collected as a proof-of-concept, since the goal is not to draw a definite conclusions.

The questionnaire is divided into two parts. First part is about the usability of the WebRTC service. Answering question about the WebRTC service and also rating the WebRTC service from one to five, where one is poor and five is excellent Second part of the form is about session feedback and is also split in two. First part needs to be answered during the session. Check off in the box if they experienced the alternatives listed. Last part is answered after the session is finished by rating the overall quality of the session.

6.3 Experiment Results

This section covers the results obtained from the experiments conducted during this project. Results from part one are first presented and discussed, and then results from part two are presented and discussed.

6.3.1 Results Experiment Phase One

Here are the results from phase one of the experiment where the different tests were preformed to make sure everything were working properly before conducting further research and experiments. Results over the tests are presented in table 6.4.

The application was tested in several combinations of the equipment and software, which can be used with the WebRTC service. The WebRTC application was tested with different network types and it works fine on each network type. The WebRTC service was tested on different types of devices, which went well. Also, with these devices had different OS, which passed the test. Also, the tests where the application was used in different browsers, showed them, all working fine..

Name	**Working? yes / no**	
Network Type:		
Wireless	x	
Cable	x	
Device:		
Stationary computer	x	
Laptop	x	
OS:		
iOS	x	
Windows 10	x	
Linux mint	x	
Web Browser:		
Google Chrome	x	
Mozilla Firefox	x	
Opera	x	

Table 6.4: Results from experiment phase one

During phase two of the experiment, the participants answered questions about the usability of the WebRTC service. On the question about if there was any problem to use the service, every users answered no. Also, everyone answered no to that they did not experience a service crash during the session.

In addition, when the participant rated the service, there were different output, which are shown in 6.2. Most of the users thought the service was easy to use, but the design was not the best. About the overall review of the service everyone rated the service was over average.

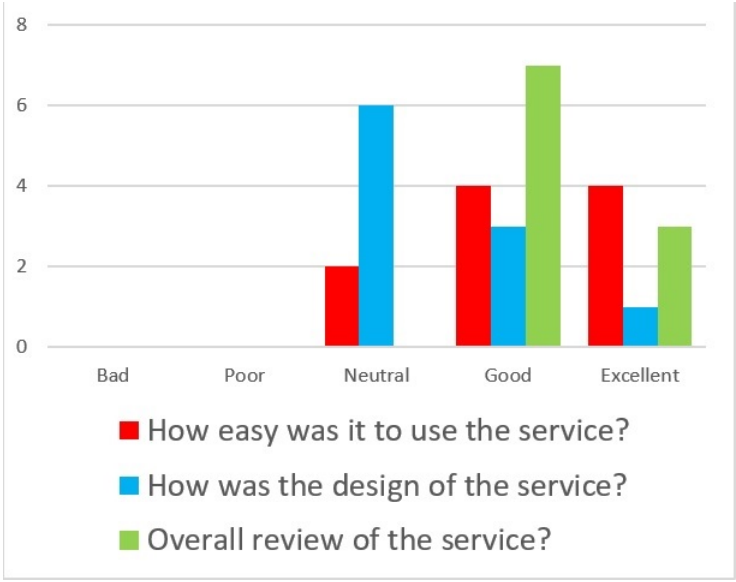


Figure 6.2: Feedback usability of WebRTC service

Phase one was conducted to ensure the stability of the WebRTC service. Verifying it was working properly with different network types, on different devices with different OS, and at last in different web browsers. Based on the results in in table 6.4 the service passed all the different requirements. In addition the user feedback was positive, even though the design was a little flawed and had improvements. This error is not big enough to make the WebRTC service not work properly. Further, with an appropriate amount of tests, one can conclude the WebRTC service is working properly enough to do further research and experiments.

There are a lot more specific browsers, operating systems, and networks out there. There was time limit on this project, the application could not be tested on all types. It is though a risk one should be aware of when moving forward. Also, not all types were available during the experiment period.

6.3.2 Results Experiment Phase Two

This section will present two types of results presented in form of feedback from users and session statistics from session. The session statistics will help to verify what the participant experienced.

QoE measurements during sessions
During the session, each participant had to check of in tables if they experienced interruptions in audio and video. Table 6.5 are presenting how many different times the users experienced the different interruptions. Looking at the table 6.5, no users experienced no video, black screen, no audio and no problem with the connection. Although there were small problems with video and audio. Not all participants experienced everything, but in some of the sessions there were few frozen images and more slow movement in the video.

About the audio, every session experienced audio disruption, some more than others. Slow audio was experienced as well to a lesser extent. In video, the users mostly experienced the audio disruption in almost every session. In fewer times they also experienced slow audio.

	Session #1 A / B		Session #2 A / B		Session #3 A / B		Session #4 A / B		Session #5 A / B		TOT
Video:											
No video	0	0	0	0	0	0	0	0	0	0	0
Frozen image	0	0	1	0	2	0	0	0	3	0	6
Slow movement	1	0	3	0	2	2	0	0	6	1	15
Black screen	0	0	0	0	0	0	0	0	0	0	0
Audio:											
No audio	0	0	0	0	0	0	0	0	0	0	0
Audio disruption	1	1	1	0	2	2	0	5	8	2	22
Slow audio	0	2	1	0	0	0	0	2	1	0	6
Connection:											
Can't connect	0	0	0	0	0	0	0	0	0	0	0
Lost connection	0	0	0	0	0	0	0	0	0	0	0

Table 6.5: Feedback during the sessions

QoE measurements after sessions

After session, the participant evaluated the overall quality of the audio, video and combined audio and video. There are different opinions out there and most of them seem to like the quality of the audio. the diagram in figure 6.3 shows the user feedback of the overall quality of audio. Eight out of ten participants gave the overall quality of audio a grade of good or excellent. The remaining two participants graded the audio neutral and poor, and no participant gave the lowest grade of Bad in their evaluation.

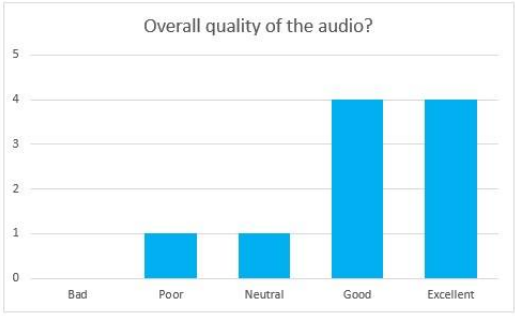


Figure 6.3: Feedback overall quality of audio

When it comes to the overall quality of video there have the participant different opinions. Diagram in figure 6.4 shows the majority of the participant rated excellent quality of video, while three participants gave a grade of neutral, and one even experienced poor quality of video.

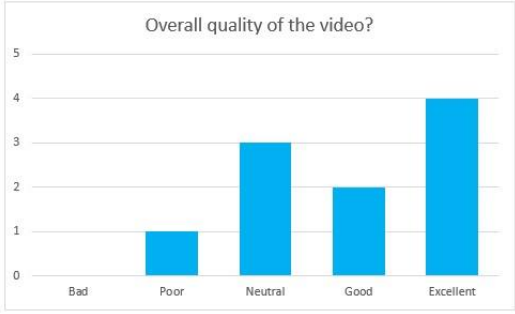


Figure 6.4: Feedback overall quality of video

In addition, the participant evaluated the overall quality of combined audio and video, to get a deeper understanding of how the user perceived the quality of combined audio and video. The diagram in figure 6.5 shows the majority of participants rated

neutral, but the others rated the quality of combined audio and video to be good and even excellent.

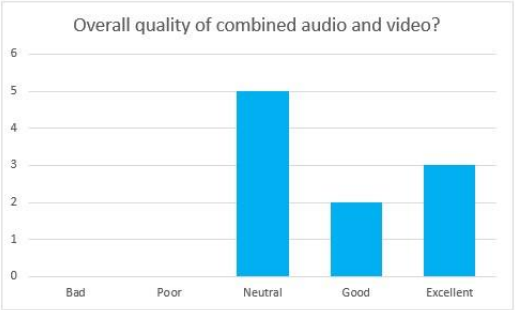


Figure 6.5: Feedback overall quality of combined audio and video

Session statistics

During experiments there were sessions statistics collected. Below is each session presented with session statistics in tables. Each sessions have to peers, A and B, with their own set of statistical data. Also, each peer have separated between audio and video stream as the tool visualized.

Session #1 have some differences between peer A and B. Table 6.6 presents the statistics from session #1. Peer B has a big sending bit rate for video stream, which may be because of a newer technology in video decoder. Also, peer B has higher packet loss rate than peer A.

	A		***B***	
Metric	**Audio**	**Video**	**Audio**	**Video**
Sending Bit Rate (kbps)	36.889	388.838	38.177	1433.226
Receiving Bit Rate (kbps)	37.816	1418.100	36.780	386.414
RTT (ms)	76.304	74.969	78.636	72.339
Packet loss rate (%)	0.260 %	0.294 %	0.999 %	0.611 %
Jitter Received per second	11.236		7.66	

Table 6.6: Session statistic from session #1

Session #2 is opposite of session #1. In table 6.7 it shows that it is peer A which has the high value in sending bit rate of video compared to peer B. The RTT is presenting higher values, which indicates there may have been delay in the session. In addition, peer A received higher amount of jitter than peer B.

	A		***B***	
Metric	**Audio**	**Video**	**Audio**	**Video**
Sending Bit Rate (kbps)	37.615	393.994	36.104	415.715
Receiving Bit Rate (kbps)	36.018	414.198	37.504	393.358
RTT (ms)	106.842	104.782	106.630	106.630
Packet loss rate (%)	0.286 %	0.186 %	0.244 %	0.207 %
Jitter Received per second	7.383		3.214	

Table 6.7: Session statistic from session #2

Session #3 presents the statistics in table 6.8. By looking at the table, the packet loss rate is higher at peer A than peer B. But both peers received a relatively high average number of jitter. However, the RTT value is small and fast considering.

	A		***B***	
Metric	**Audio**	**Video**	**Audio**	**Video**
Sending Bit Rate (kbps)	36.484	1213.552	35.512	428.765
Receiving Bit Rate (kbps)	35.459	426.165	36.222	1203.502
RTT (ms)	88.607	82.158	84.409	77.388
Packet loss rate (%)	0.744 %	0.608 %	0.286 %	0.381 %
Jitter Received per second	10.727		7.503	

Table 6.8: Session statistic from session #3

Session #4 seems to have low values of every metric presenting in table 6.9. Here the metrics are presenting low values, compared with the other sessions. This can indicate a good bandwidth during session. One can also say that the QoS is good in this session.

	A		***B***	
Metric	**Audio**	**Video**	**Audio**	**Video**
Sending Bit Rate (kbps)	28.357	493.002	37.843	510.709
Receiving Bit Rate (kbps)	37.823	510.075	28.345	500.901
RTT (ms)	88.550	87.233	87.441	88.550
Packet loss rate (%)	0.030 %	0.019 %	0.070 %	0.059 %
Jitter Received per second	2.742		3.529	

Table 6.9: Session statistic from session #4

Session #5 presents the session statistics in table 6.10. It presents a high RTT

value, which means the was a little delay during the session. In addition, both peers received a high number of jitter in the audio stream. Also, the packet loss rate in audio was a little higher at peer A than peer B.

	A		***B***	
Metric	**Audio**	**Video**	**Audio**	**Video**
Sending Bit Rate (kbps)	37.231	422.460	38.542	462.951
Receiving Bit Rate (kbps)	38.801	469.484	36.621	415.224
RTT (ms)	108.273	102.325	106.804	102.704
Packet loss rate (%)	0.560 %	0.560 %	0.469 %	0.155 %
Jitter Received per second	9.382		10.817	

Table 6.10: Session statistic from session #5

6.3.3 Correlate the QoE Scores with Session Statistics

When having users to participate in this experiment and answering a questionnaire, to verify what the user experienced was true, additional session statistics were collected. Each session lasted about 15 minutes, and this is where the statistical data was gathered.

In session #1 both peers experienced once audio disruption during the session and peer B experienced slow audio as well. By looking at the session statistics in table 6.6, it correlates with the QoE values. The value of the average jitter received per second explains the audio disruption. Peer B has a higher value of packet loss rate with the audio streams, which can explain the experience of slow audio. In addition, peer A experienced slow movement in video, which cannot be fully verified by looking at the session statistics. The RTT value looks to be stable and cannot be the reason.

During session #2 only one peer, A, experienced distortion in the audio and video streams. Peer A encountered a number of slow movement in the video and by looking at the packet loss rate for video, it is higher than at peer B. This can also explain that frozen image can occur. Also, peer A experienced audio disruption and slow audio, and by looking at the value of the average jitter received can be the explanation it occurred. In addition, the lower statistical values can explain the lack of interruption in the session.

In session #3 peer A encountered more interruptions in video stream than peer B. Frozen image and slow movement can be explained by the packet loss rate for video. The slow movement in video experienced by the peer B cannot fully be verified by the session statistics. Both peers experienced equal audio disruption during the

session, which by looking at the value of the average jitter received by per second can verify the interruption.

Out of all sessions, session #4 is the session with a mismatch between QoE values and session statistics. Peer B experienced more audio interruption than peer A. By looking at session statistics, it does not match. The session statistics are the same between peer A and B. But peer A experienced nothing interruption during the session and the session statistics correlate with this. One can discuss here that it is one own opinion of what can be interpreted as interruptions in audio.

Last, session #5 is the one session where there was a big number of interruptions in both audio and video, but for mainly one peer, A. The high number of slow movement in video, can be explained by looking at the packet loss rate value, which are higher than peer B's. About the high number of audio disruption experienced by peer A can be explained the average jitter received per second value. Peer B encountered smaller amount of audio disruption than peer A, but peer B has a higher value of jitter received. This hence the problem about hoe different users have different interpretation of what is audio disruption.

After discussing the correlation between the QoE scores and session statistics, one can see that in some sessions the QoE scores correlate with the session statistics, but on the other hand they do not. However, interruption in the audio and video did occur, and had an impact on the overall quality of combined audio and video, which is illustrated in 6.5. Most of the participant thought the overall quality of combined audio and video were neutral, which indicates improvements are needed.

So, based on these findings from the experiments, one can conclude there are still improvement to be made in multimedia conversations with WebRTC to make it more stable. To accomplish this implementation of a congestion control can be helpful.

6.4 Limitations of Results

In phase one of the experiments, the most known softwares and hardwares were selected. The WebRTC service could be tested in more detail on more specific software, systems, and networks. Part of the reason for not testing even more technologies, was due to the time limit of this project. The second part is that not all of the softwares and systems were readily available during the experiments.

Further the experiments were conducted on a small sample size, number of participants, which gives some uncertainty in the results. However, the results can be used as indications on the fact that QoE scores verify the importance of congestion controls. But one should not draw definite conclusions based on the results.

During experiments I experienced that the tool, Chrome's webrtc-internals, used to gather session statistics was lacking explanations and descriptions. It became necessary to spend some time looking at the graphs to understand the parameters. Still there were several parameters I did not understand, though they may not be relevant enough in the first place.

Chapter 7
Evaluation of Two WebRTC Congestion Controllers

This chapter will provide a detailed description of the two recommended algorithms Google Congestion Control (GCC) and Network-Assisted Dynamic Adaptation (NADA), based on IETF RMCAT draft [1] and [26]. It will also provide a comparison between both algorithms.

7.1 Google Congestion Control (GCC)

GCC is an algorithm recommended for WebRTC services. GCC is interesting since it is already implemented in Google Chrome, which is at the time of writing the most popular browser implementing the WebRTC framework[1].

GCC is combined with two controllers, loss-based controller and delay-based controller. It can be implemented in two ways; one where both controllers are placed at the sender-side, and one where the delay-based controller is placed on the receiver-side and the loss-based controller at the sender-side. Figure 7.1 shows a detailed architecture of GCC where the controllers are placed separately. The sender sends RTP packets to the receiver, where the delay-based controller is. At the delay-based controller a rate A_r is computed with statistics from the RTP packets. The rate A_r is sent back to the loss-based controller, at the sender, with RTCP reports. The loss based controllers calculates another rate A_s. Then the new rate for the encoder, A, is the minimum of A_r and A_s. The reason why is to avoid that A_s exceed A_r.

Below is a more detailed description of GCC given [1]. Also, a list of all the parameters in GCC with a description and calculation, are listed in table A.1, A.2 and A.3 in the appendices.

[1] https://www.w3schools.com/browsers/

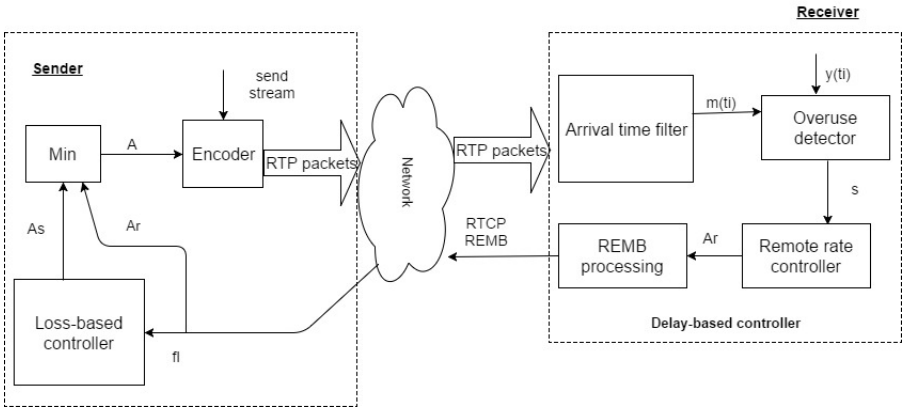


Figure 7.1: Detailed GCC Architecture

7.1.1 Delay-based Congestion Control Algorithm

The receiver-side controller is a delay-based congestion control algorithm, which take in the information from RTP packets sent from the sender. It monitors and processes the arrival time and gets the size of the incoming packets. The goal of the controller is to compute Ar, according to the following equation:

$$A_r(t_i) = \begin{cases} nj * A_r(t_{i-1}) & \text{Increase} \\ \alpha * R(t_i) & \text{Decrease} \\ A_r(t_{i-1}) & \text{Hold} \end{cases} \tag{7.1}$$

where t_i stands for the time the group of RTP packets carrying one video frame received. $nj \in [1.005, 1.3]$, $\alpha \in [0.8, 0.95]$, and $R(t_i)$ is the receiving rate measured in the last 500ms.

The whole controller is built up of five blocks, shown in figure 7.1. Followed is a detailed description of each of the five blocks.

– **Arrival-Time Filter:** The main goal is to estimate the queuing time variation $m(t_i)$ based on one-way delay variation, $d(t_i)$. The one-way delay variation is considered as the sum of three components [1]: 1)The transmission time variation $(L(t_i) - L(t_{i-1}))/C(t_i)$, 2) queueing time variation $m(t_i)$, and 3) network jitter $n(t_i)$. The following mathematical equation is proposed:

$$d(t_i) = (L(t_i) - L(t_{i-1}))/C(t_i) + m(t_i) + n(t_i) \tag{7.2}$$

where $L(t_i) - L(t_{i-1})$ is the video frame length and $C(t_i)$ is estimation of the path capacity. In [1] a Kalmanfilter is used to estimate the $m(t_i)$ based on

one-way delay variation measured, $d_m(t_i)$, which is calculated as follows:

$$d_m(t_i) = (t_i - (t_{i-1}))\breve{}(T_i\breve{}(T_{i-1})) \tag{7.3}$$

where t_i is the time of which the last packet was received and T_i is the time the last packet was sent.

- **Over-Use Detector:** The main purpose is to produce a signal s to determine the state of the rate, based on the estimated queuing delay, the arrival-time-filter calculated, and with an adaptive threshold proposed in [**gcc:8**]. If $m(t_i)$ is smaller than $y(t_i)$, then the network is underused and a signal is generated. If $m(t_i)$ is bigger than $y(t_i)$, the network is considered congested and an overuse signal is generated. Lastly, if $m(t_i)$ is equal to $y(t_i)$, the network is considered stable and a normal signal is generated.
- **Remote Rate Controller:** The main goal is to compute a rate, A_r, according to a signal computed by the over-use detector, which drives the finite state machine shown in figure 7.2. The goal of the finite state machine is to minimize the queuing delay in the buffers along the end-to-end way. When congestion has occurred, the $m(t_i)$ is positive. The over-use detector is triggered and generates an overuse signal. The overuse signal drives the machine into a decreased state. There the sending rate is decreased until the estimated queuing delay is negative. Then an underuse signal is triggered and drives the machine into the hold state. The machine will stay in the hold state until the bottleneck is gone, and $m(t_i)$ is close or equal to zero. The over-use detector then generates a normal signal, which drives the machine into an increased state.

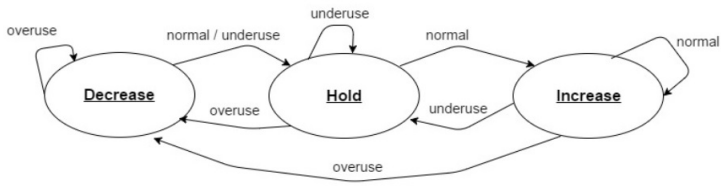


Figure 7.2: Remote rate controller finite state machine

- **Adaptive threshold:** The aim of this block is to adapt the sensitivity of the delay gradient based on network conditions [14]. As shown, the adaptive threshold is used by the over-use detector. There are two problems that may occur if the threshold is not adaptive: 1) the delay-based control may have no effect when the bottleneck queue along the path is too small. 2) the GCC flow may be starved by an existing loss-based TCP flow.
- **REMB processing:** This block notifies the sender with the calculated rate Ar through REMB messaging. The REMB messages are sent every 1s or right away when $A_r(t_i) < 0.97 * A_r(t_{i-1})$, when A_r has decreased 3%.

The output of the delay-based control algorithm is a new rate. With the loss-based controller rate and the delay-based controller rate, you have a new target rate for the encoder.

7.1.2 Loss-based Congestion Control Algorithm

Loss-based congestion control algorithm on the sender-side acts every time the RTCP report message it with fraction of packet loss information, arrives or every time glsremb message, which carries A_r, arrives at the sender. The REMB format is an extension of the RTCP protocol. A Loss-based controller takes loss rate measurements, feedback information and computes a target sending rate, A_s, based on the following equation:

$$A_s(t_k) = \begin{cases} A_s(t_{k-1}) * (1 \breve{} 0.5 * fl(t_k) & \text{fl(tk)} > 0.1 \\ 1.05 * A_s(t_{k-1}) & \text{fl(tk)} < 0.02 \\ A_s(t_{k-1}) & \text{otherwise} \end{cases} \tag{7.4}$$

$fl(t_k)$ is defined as a fraction of lost packets. If the fraction of lost packets is $(0.1 < fl(t_k) < 0.02)$ the rate A_s is small and constant. If the fraction of lost packets is considered high $(fl(t_k) > 0.1)$ the rate is decreased. Lastly, if the fraction of packets lost is considered small $(fl(t_k) < 0.02)$ the rate is increased. The threshold 0.1 and 0.02 have a big impact on decreasing and increasing of the encoder rate. By altering them to a higher or lower value, the rate would not change as often, while a congestion occurred. After A_s is computed the new sending rate A is calculated based on $min(A_r, A_s)$. The reason why is to avoid that A_s exceed A_r, because packet loss is affecting the service in a much larger degree.

7.2 Network-Assisted Dynamic Adaption (NADA)

NADA is another congestion control mechanism recommended by IETF RMCAT. NADA is able to contain queuing delays and provide reasonable fairness [47]. The NADA design benefits from explicit congestion control signals (e.g., ECN markings) from the network, yet also operates when only implicit congestion indicators, like delay and/or loss are available. NADA distinguishes from other controls because of the sender behavior.

The sender send media content in RTP packets over UDP to the receiver. The receiver calculates an aggerated congestion signal. The congestion signal is combined with both implicit (e.g. loss and delay) and explicit (e.g. ECN marking) congestion indicators from the network. The signal is sent to the sender in RTCP reports. Based on this signal, a new reference rate is calculated, R_n. Furthermore, the new encoder rate, R_v, is calculated based on the reference rate and buffer size. A new sending rate is also calculated, R_s, based on the reference rate and buffer size. On the sender-side

there is a rate shaping buffer which absorbs the mismatch between the output rate and sending rate. A large rate shaping buffer contributes to higher delay.

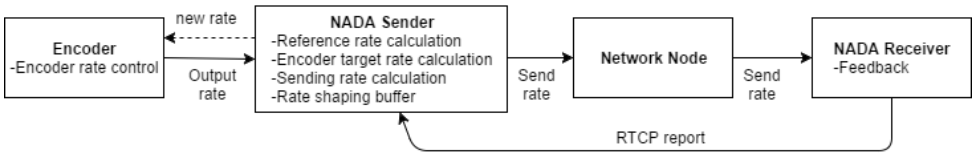


Figure 7.3: NADA System Overview

Figure 7.3 shows the system overview where NADA operates, and figure 7.4 shows a detailed overview of how the process works in NADA, based on [26]. Each block will be described below in detail to get a better understating on how NADA operates. Also, a list of all the parameters in NADA with description and calculation are listed in table A.4, A.5 and A.6 below in the appendices.

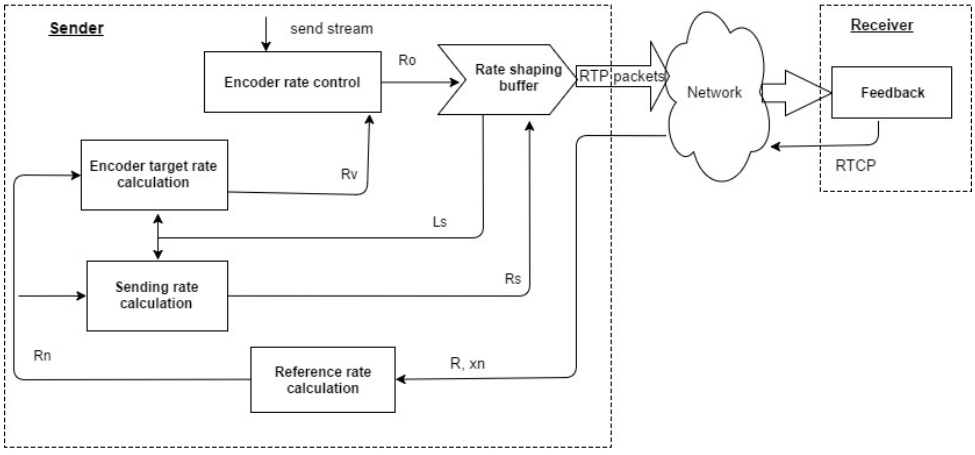


Figure 7.4: Detailed NADA architecture

– **Feedback.** On the receiver side the feedback is responsible for measuring and estimating delay, packet loss, ECN marking ratios and receiving rate of the flow. Feedback then calculates the gathered congestion signal, X_n, which is based on the parameters mentioned earlier, every time a RTP packet is received. X_n is calculated as follows:

$$X_n = d\_tihlde + p_m * DM + p_l * DL \tag{7.5}$$

d_tihlde is described as delay after non-linear warping. p_l is the fraction of packet loss measured in a time window. DL is the delay penalty for a loss

that is constant. p_m is the estimated ECN marking ratio, and then the DM is described as the delay penalty associated with ECN markings. ECN markings is extracted from the IP header [39]. Also, the receive side calculates the average rate, $R(t_i)$, received in the last 500ms. Based on the observed statistics of packets received, the receiver determines if the network is congested and then recommend rate adaptation mode for the sender. Congestion signal, average receiving rate and rate mode are put in the reports and sent back to the sender in RTCP packets.

– **Reference rate calculation.** The sender reacts based on a congestion signal from the feedback report from the receiver, and can operate in one of two modes.

  ◦ Accelerated ramp-up: when there are low bottlenecks, the rate is multiplicative increased with respect to the last rate which had a successful transmission. The new reference rate, R_n, is calculated as follows:

$$R_n = max(R_n, R(t_i) * (gamma + 1) \tag{7.6}$$

  $R(t_i)$ is the average rate measured in the last 500ms and gamma is the rate increase multiplier.

  ◦ Gradual rate update: the target rate changes according to the gathered congestion signal and its change in value. It calculates the reference rate as follows:

$$\begin{aligned} R_n = R_n - KAPPA * (delta/TAU) * (X_o/TAU) * \\ R_n \breve{} KAPPA * ETA * (X_d/TAU) * R_n \end{aligned} \tag{7.7}$$

  where there are some constants, like $KAPPA$ and ETA are scaling parameters for the specific model. TAU is upper bound of RTT, $delta$ is the observed interval between current and past feedback reports. X_o is the distance from X_n from reference congestion level. X_d is the change in congestion signal from previous value.

– **Encoder target rate calculation.** Based on the reference rate and the buffer size, L_s, a new target rate to the encoder is calculated.

$$R_v = R_n \breve{} BETA\_V * 8 * L_s * FPS(decrease) \tag{7.8}$$

FPS is here the frame rate of incoming video. $BETA_V$is a scaling parameter that can be tuned to maintain a small rate shaping buffer and can be deviating from the reference rate point. Then R_v is sent to update the encoder with a new target rate.

– **Encoder rate control.** The encoder encodes raw media frames into bit streams which is packed into RTP packets. The encoder also has rate control capabilities. It changes the rate into the new encoder target rate, R_v, which it received previously.

– **Sending rate calculation.** Based on the reference rate and the buffer size, L_s, a new sending rate is calculated.

$$R_s = R_n + BETA\_S * 8 * L_s * FPS(increase) \tag{7.9}$$

The calculation is almost the same as the encoder target rate calculation with the same parameters. FPS stands for the frame rate for incoming video and $BETA\_S$ is a scaling parameter for the sending rate. It tries to maintain the same goals as the encoder rate control.

– **Rate shaping buffer.** The goal is to have a small buffer. The task assignment for the rate shaping buffer is to immediately absorb any mismatch between the output rate and the sending rate. A large rate shaping buffer contributes to higher end-to-end delay, which harm the communication. Therefore, the sender try to prevent the rate shaping buffer from building up either by increasing the sending rate, as we see in function 7.9. Another option is to reduce the encoder target rate, see function 7.8, to limit incoming packets to the rate shaping buffer.

7.3 Evaluation of GCC and NADA

This section describes a comparison between GCC and NADA. The goal is to evaluate if both GCC and NADA satisfy the requirements defined in [5] i.e. low queuing, and relying on existing information about the incoming flows to provide feedback to the sender.

7.3.1 Functionality

GCC and NADA are two congestion control algorithms designed for real-time multimedia flows and recommended by IETF RMCAT. In [5] it says that the algorithm should depend on the information of the flows and provide feedback to the sender. This is true for both algorithms because GCC and NADA needs the flow information to do the computation at the sender. In addition, both algorithms uses RTP and RTCP for the feedback reports, which are another requirement.

Both algorithms should be fair to other flows and in [13] an experiment was conducted to verified that they provide intra-protocol and inter-protocol fairness. Also, the main goal is to adapt the sending rate to track the link capacity that was proven in [13].

7.3.2 Architecture

The architecture of GCC is more flexible than NADA. With GCC it is possible to choose to have all the computation on the sender side or divide the calculation on both sides, sender- and receiver-side, like in figure 7.1. It is up to the programmer to choose how they would like to implement GCC. There are pros and cons of both methods. If one wants all the calculations on the sender side, one will get full control of the algorithm, but it may take some time when transferring the data from the receiver. The data include all feedback information needed to do the calculation on the sender side. By splitting the calculation between the sides, data transfer will be faster because of less information needed on the other side. But then there is a loss of control over the algorithm.

The architecture of NADA has all the computations on one specific side, the sender-side, which is presented in figure 7.4. This helps to understand how the algorithm is working. By having all the computations on one side, the complete control over the algorithm is also there. NADA has a specific architecture, which makes it easier to implement than the flexible architecture of GCC.

7.3.3 Input Data

Both algorithms use RTP to transfer data from the sender to receiver and uses RTCP to transfer feedback from the receiver to sender. The input data GCC uses are timestamps, total packets received and a count of missing packets which are packets that does not follow the sequence number in RTP. RTCP reports sent from the receiver includes the fraction of lost packets. Also, GCC uses REMB messaging with RTCP reports, which include the new target rate.

The input data used in NADA on the receiver side are timestamps, total transmitted packets and number of missing packet as in GCC. The data sent in the RTCP report is the value of a congestion signal, the average received rate and the recommended rate adaption mode, which are a one-bit flag. Mode equals zero (mode=0) indicates the sender should operate in accelerated ramp up mode. Mode equals one (mode=1) indicates the sender should operate in gradual update mode.

7.3.4 Response Time

The time it takes for GCC to alter the rate depends on the congestion. The receiver side only sends the new rate, A_r, with REMB messaging every second or immediately when A_r has decreased 3%. First, the average rate is calculated in the last 500ms every time.

The time it takes for NADA to complete is based on when the feedback is received at the sender. The receiver sends back the RTCP messages periodically, for example around every 100ms. Also, first the average rate is calculated in the last 500ms every time at the receiver side.

7.3.5 Data Storage

Every algorithm need memory assigned to operate correctly. GCC is an algorithm which operates with old values already calculated [1]. This means that for the algorithm to work, it needs to store the old value to calculate the new value on the next computation. All the components in GCC have the need for memory to store important values for the next calculations. Based on this, GCC need data storage, but the exact amount of data storage needed is hard to define.

NADA is an algorithm designed with the need of a buffer, which means there is need for a data storage [26]. The exact amount of memory needed is not defined. Without the buffer it is a whole other algorithm. Same as GCC, the other components in NADA need small storage spaces to store the data required for later computations.

7.3.6 Computation

Computations done in GCC are straight forward, which means that when it has finished all the calculations, it alters the encoder with the new target rate. You must follow each calculation step to move forward. All computations are depending on each other. There are more calculations done in GCC than in NADA because of the adaptive threshold used. The threshold must be adaptive, otherwise two problems would occur, described in section 7.1.1.

On the other hand, NADA is more complex. NADA has less computation steps to get through. NADA also alters the target rate in two places. First, it changes the encoder target rate at the encoder, and then changes the sending rate at the rate shaping buffer.

7.3.7 Network

One requirement described in [5] says that algorithms should except routing changes and interface changes as well, like from WiFi to 3G data, etc. In GCC it considers the bandwidth capacity value when calculating the queuing time variation. In theory, GCC would work fine in every network. Based on research in [15] GCC is more robust over Wi-Fi, with respect to channels outages.

On the other hand, NADA has a rate shaping buffer which is making sure there is not any mismatch between the output rate and the sending rate[26]. If the bandwidth

is changing quickly, the buffer will increase and with it the packet delay will increase. If so, the algorithm would not be working to its full potential.

7.3.8 Implementing Issues

From today's date, GCC has been implemented in open-source WebRTC projects, and is being used by Google Hangouts. Also, the algorithm has been tested in a multi-party conference with a conference server [1].

Same ass GCC, Nada has been implemented in NS2 and NS3 simulation platforms, for example [26]. The algorithm has been implemented and evaluated in a lab setting by IETF RMCAT. Otherwise, NADA has not, from this date, been used for published applications, like appear.in.

7.3.9 Security Issues

GCC has some security issues to consider described in [**gcc:8**]. First, it is important to know that an attacker can interfere with the connection and insert or remove messages. This can make the algorithm either send a rate which is under utilizing the link capacity, or send a too high rate causing network congestion. Since the information used in GCC is carried inside RTP, it can be protected by using Secure Real Time Protocol (SRTP). Second, the timestamp used in RTP cannot be encrypted. But it is unlikely that an attacker would mount an attack based on timing information only.

Since NADA is using the same protocol, RTP, it is safe to conclude it has the same security issues as GCC. For the RTCP reports both GCC and NADA are using, there is a sister protocol, Secure Real Time Control Protocol (SRTCP), which can be used for security around the RTCP reports.

7.3.10 Total Cost

If GCC or NADA is implemented with a WebRTC service it will be noticed by the system, but the question is to what extent. Implementing an extra controller in a system is noticeable because of the extra memory stored and the rules to follow. The system and the controller need to be interconnected and work together. The prioritizing of packets would be evident, if the congestion controller is not correctly implemented to co-exist with the rest of the system. It is important that the congestion controller can work properly with other protocols like for example TCP. [25] and [14] gives proof of fairness achieved by GCC when coexisting with TCP.

After measuring the resource usage of GCC and NADA there are some costs to consider and advantages to implementing the algorithms. Exactly how big of a cost is depending on how the whole system is set up.

Chapter 8
Discussion

This chapter discuss the findings in this report with background and related work. Also, the limitations of the performance of the research method described in 3 will be discussed.

8.1 WebRTC Service

WebRTC API is considered up and coming new technology used in multimedia communications. It is a free and open source project. The WebRTC service enables peer-to-peer communication through the web browser with no extra plugins. This makes the service very attractive to use for people and an exciting topic to research deeper.

Because the service is easily accessible and easy to use, there will be an increase of traffic on the Internet along with the existing traffic from other applications. There will be a delay in transmission of packets and packet loss. Since this service is based in real-time, the WebRTC service is sensitive to delay and packet loss. If these things occur, the user expectation is not met and it can affect the value of QoE. It is important that the service is stable and reliable, and the QoS also needs to be accurate. Both QoE and QoS are dependent on each other.

Congestion is a normal state to the network, which occur when there is too much traffic for the network to handle. With an increase in traffic, the WebRTC service will not be performing properly and the congestion will become a big problem. To solve this problem there is a necessity to implement a congestion control.

The developed WebRTC service in this report focused on accessibility and simplicity. There were a lot of different technologies, protocols, and designs chosen to implement this kind of service. I made my choices after doing research, and there was a lot to research. It was not possible to go through all available research, but previous knowledge and skills factored into the decision making around the

service. In addition, time was a factor influencing my choices about the design and implementation of the WebRTC service as well.

8.2 Discussion of the Experiment

In this report, experiments were conducted. The experiments would be to use the new developed WebRTC service, to enhance the importance of a congestion controller with WebRTC services. By using my own new developed WebRTC service, I have full control over the service and know every aspect of it. There is no other third party included. Also, it does not give the participants expectations of grandeur and they can use and evaluate the service with an open mind. The negative side of using your own WebRTC service is that there may be errors in the service, that you are not aware of. Since this is a new developed WebRTC service, bugs may occur.

The experiments were separated into two phases, because it would verify the importance of what the results were. Phase one of the experiments were a set of tests conducted where the WebRTC service was tested on different network types, devices, OS and web browsers. The WebRTC service needs to work properly, so the data collected in the later phases are considered true and reliable. Not all network types, devices, OS and web browsers were tested, which is a downside.

Second phase was having participants use the developed service and collect additional data. The data collected was feedback from the users and session statistics during the sessions. By having two sets of data, one gets to look at the service in two ways, QoE and QoS, which is helpful to fully understand the service. But there were limitations in acquiring the data. There may be too few participants in the experiments which gave a small sample size. Because of the small sample size, no generalizing conclusion could be made. When collecting the session statistics, the tool Chrome's webrtc-internals that was used, had some limitations, which were made clear and are presented in section 6.2.3. Still, Chrome's webrtc-internals was recommended by different articles and was used in this experiment. A discussion of the QoE measurements are discussed below.

8.2.1 Discussion of QoE Results

This subsection discusses and analyzes the results from questionnaire. It discusses all three parts of the questionnaire and evaluates how the users were affected by comparing the answers. I must also consider that the participants may have a differentiating perception of what is considered slow movement or audio disruption. Some users have better tolerance than others.

When it comes to the usability of the WebRTC service, most of the users thought it was easy to use, but there were two users which were neutral to the service. Why there is a variation may be because they take their own computer skills into consideration. About the design of the service, six users where neutral. This gave me an idea of that the design needs more work, or is not that important. For the overall review of the service itself, seven users where satisfied above average about the service, and three users thought it was excellent service, all in all.

When looking at the user feedback during the session, there are variations. Some users experienced more slow movements and others experienced more audio disruption than others. Here one may question the interpretation the users have, because one user can have a higher tolerance of slow movements and audio disruptions.

Looking at the feedback after the session, when the user evaluates the quality, there are different opinions. The users that did not experience anything out of the ordinary during the session, automatically rated the quality excellent. On the other hand, the users who experience some audio disruption, rated the overall quality of the audio lower. So, the feedback during sessions correlates with the evaluation after the session.

However, it is important to question the integrity of the participants. Are the participants answering the questionnaire sincere? Since I am a student and people want the best for me, they may answer what they think are the best answers for my project. Also, the time was limiting and the participants were spread in different locations, which did not make it possible to record the sessions to verify the outcome of the form. I choose to believe in the participants' answers with only a sliver of doubt.

All in all, one should remember that the user feedback, is in fact subjective user feedback. What is considered acceptable differs from user to user, which makes it more challenging to interpret the results.

8.3 Discussion of Evaluating Congestion Controls

The results of the experiment conducted verified the need of a congestion controller with WebRTC service in some sessions. In these sessions, the users experienced different interruptions in audio and/or video during the sessions. These feedbacks correlates with the session statistics also collected during the session. Jitter was received, which alters the audio, and the packet loss rate was a little high, which can explain both interruptions in audio and video. Also, there may have been some delay in the session. The reason for this occurring during sessions can be because of congestion in the network.

In addition, there were sessions, for example session 4, which had participants not encounter any interruptions in neither audio nor video. The sessions statistics also present low values, which indicate there was a good bandwidth during these sessions.

The results from the experiments verified the importance of using a congestion controller. In this project, there was conducted an evaluation of two WebRTC congestion controls, GCC and NADA. Both GCC and NADA are recommended congestion controls for real-time communication from IETF RMCAT. The evaluation gave detailed description and a deep insight into two algorithms. These two congestion controls operate in two different ways. Both change the encoder rate in a real-time service, but the NADA can do it in two places, but GCC have only one way to make the change. This evaluation of GCC and NADA highlighted the benefits and disadvantages with both algorithms.

The evaluation gave findings, but there were some limitations since this was a theoretical evaluation of the two algorithms and not a practical one where the congestion controls are implemented in a service. It made it hard to evaluate how much memory is needed to implement the code and evaluate the total time for the code to run. There is a lack of more quantitative approach to the evaluation to make it count more.

All in all, if there is a necessity to add a congestion control to the WebRTC service, it is important to think about how this would affect the service and network. Because if users have a good network, why put the extra cost on the network. After looking at the evaluation of these two congestion controllers, there is no right choice of which one to implement. Both has its advantages and disadvantages, and it all comes down to a personal choice based on the system and skills to implement it. You have to choose the algorithm based on your own knowledge and understanding of the congestion controller.

Chapter 9
Conclusion and Future Work

This chapter presents a conclusion of this master thesis. The conclusion will present a summary of the thesis and its findings, followed by a description of the limitations in this thesis. At last, I will propose some directions for future work.

9.1 Conclusion

As WebRTC is evolving and with support of many different technologies, the QoS and QoE becomes important when utilizing these kinds of applications. If there is a lot of unnecessary interruptions during sessions, the popularity of the service will decrease.

This thesis discussed the importance of having congestion controls in context of WebRTC services. It was researched by conducting experiments with a developed WebRTC service. Also, this thesis looked at different WebRTC congestion controllers and evaluated them.

First, I developed a WebRTC service to use in experiments. The features, protocols and technologies were researched before implementation. The signaling was the biggest challenge to overcome during the development process.

The experiments conducted were separated in two phases. First phase was to ensure the stability of the WebRTC service, to make sure the WebRTC service was working properly to further conduct experiments. Results from this phase expressed no concerns about the service. Every software and hardware the WebRTC service tested on, gave positive results and it was possible to move forward to phase two.

The second phase were 10 users managing peer-to-peer conversations using the developed WebRTC service. The focus was on QoS and QoE to the WebRTC service. Feedback form the participants indicated there are disruptions in audio and video during the sessions which affected the QoE negatively. I was also able to find reasons

for this in the collected session statistics data. By looking at packets lost and jitter received, one could see high values.

In addition, I have evaluated two WebRTC congestion controllers, GCC and NADA. I achieved a deeper understanding of how they operate and how they can be a positive effect on real-time multimedia communication in WebRTC services. The evaluation of congestion controllers gave some findings but nothing revolutionary. GCC and NADA are two congestion controls which operates in two different ways. If you were to choose one of them, there would be no clear favorite. Both are well functioning congestion controllers and to choose one will be a personal choice.

The evaluation and experiment described in this report demonstrate only some parts of the truth, and it can be used for more extensive research both with deeper investigation of the WebRTC congestion controllers and implementations of them, as well as higher number of participants with deeper analysis of session statistical data. I concluded that there is a need for a congestion controller with the WebRTC service to satisfy the QoE and QoS to the users.

9.2 Limitations

9.2.1 Limitation of Evaluation of WebRTC Congestion Controls

When evaluating different congestion controllers, it is crucial to have enough data to research. With more concrete data to go by, a more insightful evaluation can be conducted. The two WebRTC congestion controllers are considered to be new technology, where there is not many written articles to explore. In other words, there has not been many experiments conducted. Almost all articles vetted in this thesis are dated from 2016, which says how new this topic is. There is so much more research to be conducted to get a deeper understanding of the congestion controllers.

9.2.2 Limitation in Experiment Setup

There are a lot of different factors that will influence the service and on the users acceded QoE, and some are out of the researchers control. It is therefore useful to properly manage the factors that actually can be controlled. One important factor is to control the environment during the experiments, so it decreases the risk of unreliable data.

For this reason, it would be helpful to screen record during the sessions in the experiments, to record the video and audio. This is possible, but one has to add this function into the experiments. Users must be aware of the screen recordings and the devices need to support the screen recorder. The screen recordings would

be useful when looking at the user feedback to fully understand what users consider acceptable or poor quality.

9.2.3 Limitation of Data

Conducting experiments and collecting data can be useful to test out the WebRTC service and obtain useful insights. But by increasing the number of participants, one would acquire more accurate results. The experiment in phase two had a total of 10 users that participated. The results gathered form the experiments are interpreted more as an indicator, than definite conclusion on how big a need there is for a congestion control by looking at the users QoE. The aim of this project is to look at the importance of a congestion control and by looking at the user feedback it will indicate the importance. In the future, conducting more extensive user studies could help determine how important a congestion control is, and if it is needed in all WebRTC services.

9.3 Future work

This project has contributed by looking at small parts of the research topic, and more investigation is needed to get more in depth knowledge about the topic. Therefore will the suggested future work, based on this thesis, involve going even further with this research and experiments.

9.3.1 Implementation of GCC and NADA

This thesis did a theoretical evaluation of two WebRTC congestion controls, GCC and NADA. It would be of interest to implement the congestion control algorithms and conduct experiments to further understand their behavior.

An example of implementing GCC is by using Google Chromium browser. One needs to have a good working knowledge of the developer browser and C#, a programming language. For collecting the statistical data from the congestion controller, you properly have to develop your own tool by using the WebRTC' statistics API.

9.3.2 Further QoE Testing

The results presented in this report were only indications of poor QoE without congestion control. It would be of interest to conduct similar experiments with a larger analysis of data, to see the importance of a congestion control in a bigger picture.

References

[1] *A Google Congestion Control Algorithm for Real-Time Communication draft.* URL: https://tools.ietf.org/html/draft-ietf-rmcat-gcc-02 (visited on 03/26/2017).

[2] *Basics about WebRTC.* URL: https://www.html5rocks.com/en/tutorials/webrtc/basics/ (visited on 02/14/2017).

[3] *Bootstrap.* URL: http://getbootstrap.com/ (visited on 05/18/2017).

[4] Dragan Samardzija Branislav Sredojev and Dragan Posarac. "WebRTC technology overview and signaling solution design and implementation". In: (2015).

[5] *Congestion Control Requirements For Real Time Media draft.* URL: https://tools.ietf.org/html/draft-jesup-rtp-congestion-reqs-00 (visited on 03/12/2017).

[6] *CSS developer guide.* URL: https://developer.mozilla.org/en-US/docs/Web/CSS (visited on 05/13/2017).

[7] *CSS guide.* URL: https://www.w3schools.com/css/default.asp (visited on 04/25/2017).

[8] *Datagram Congestion Control Protocol (DCCP).* URL: https://tools.ietf.org/html/rfc4340 (visited on 04/23/2017).

[9] Katrien De Moor Doreid Ammar et al. "Video QoE Killer and Performance Statistics in WebRTC-baced Video Communication". In: (2016).

[10] Min Xie Doreid Ammar Paul Heegaard, Katrin Demoor, and Markus Fiedler. "Revealing of the dark side of webrtc statistics collected by google chrome." In: *Qulaity of Multimedia Experience (QoMEX)* (2016).

[11] *Extensible Messaging and Presence Protocol (XMPP): Core.* URL: https://tools.ietf.org/html/rfc3920 (visited on 03/29/2017).

[12] *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence.* URL: https://tools.ietf.org/html/rfc6121 (visited on 03/29/2017).

[13] Cesar Ilharco Gaetano Carlucci Luca De Cicco and Saverio Mascolo. "Congestion Control for Real-time Communications: a comparison between NADA and GCC". In: (2016).

[14] Stefan Holmer Gaetano Carlucci Luca De Cicco and Saverio Mascolo. “Analysis and Design of the Google Congestion Control for Web Real-Time Communication(WebRTC)”. In: (2016).

[15] Stefan Holmer Gaetano Carlucci Luca De Cicco and Saverio Mascolo. “Making Google Congestion Control robust over Wi-Fi networks using packet grouping”. In: (2016).

[16] *HTML*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (visited on 04/23/2017).

[17] *HTML guide*. 2017. URL: https://www.w3schools.com/html/default.asp (visited on 04/25/2017).

[18] IEEE. “IEEE Recommended Practice for Software Requirements Specifications”. In: (1998).

[19] ITU. “Definitions of terms related to QoS”. In: (2008).

[20] *JavaScript*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visited on 05/23/2017).

[21] *jQuery*. URL: https://jquery.com/ (visited on 05/14/2017).

[22] *jQuery guide*. URL: https://www.w3schools.com/jquery/jquery_intro.asp (visited on 05/14/2017).

[23] Mollern Le Callet and Perkis. “European Network on Quality of Experience in Multimedia Systems and Services”. In: *Qualinet white paper on definitions of QoE* (2013).

[24] Salvatore Loreto and Simon Pietro Romano. *Realtime Communication with WebRTC: Peer-to-Peer in the Browser*. O’Reilly Media, 2014.

[25] Gaetano Carlucci Luca De Cicco and Saverio Mascolo. “Understanding the Dynamic Behaviour of the Google Congestion Control for RTCWeb”. In: (2013).

[26] *NADA: A Unified Congestion Control Scheme for Real-Time Media draft*. URL: https://datatracker.ietf.org/doc/draft-ietf-rmcat-nada/?include_text=1 (visited on 03/26/2017).

[27] *Node.js*. URL: https://nodejs.org/ (visited on 04/03/2017).

[28] *npm - building amazing things*. URL: https://www.npmjs.com/ (visited on 03/28/2017).

[29] B. E. Helvik P. J. Emstad P. E. Heegaard and L. Paquereau. *Dependability and Performance with Discrete Event Simulation*. 2011.

[30] Qualinet. URL: http://www.qualinet.eu (visited on 05/06/2017).

[31] *Session Initiation Protocol - Introduction*. URL: https://www.tutorialspoint.com/session_initiation_protocol/session_initiation_protocol_introduction.htm (visited on 03/25/2017).

[32] *Session Initiation Protocol (SIP) Basic Call Flow Example*. URL: https://tools.ietf.org/html/rfc3665 (visited on 03/20/2017).

[33] *SIP: Session Initiation Protocol*. URL: https://www.ietf.org/rfc/rfc3261.txt (visited on 03/20/2017).

[34] *SOCKET IO*. URL: https://socket.io/ (visited on 03/28/2017).

[35] *TCP Congestion Control*. URL: https://tools.ietf.org/html/rfc5681 (visited on 04/22/2017).

[36] *TCP Friendly Rate Control (TFRC): Protocol Specification*. URL: https://www.ietf.org/rfc/rfc5348.txt (visited on 04/22/2017).

[37] *TCP Slow Start, Congestion Avoidance and Fast Retransmit and Fast Recovery Algorithms*. URL: https://tools.ietf.org/html/rfc2001g (visited on 04/22/2017).

[38] *Terminology for Benchmarking Network-layer and Traffic Control Mechanisms*. URL: http://docwiki.cisco.com/wiki/Quality_of_Service_Networking (visited on 05/06/2017).

[39] *The Addition of Explicit Congestion Notification (ECN) to IP*. URL: https://tools.ietf.org/html/rfc3168 (visited on 03/23/2017).

[40] *The WebSocket Protocol*. URL: https://tools.ietf.org/html/rfc6455 (visited on 03/22/2017).

[41] *Top reasons to use Node.js for web application development*. URL: https://jaxenter.com/top-reasons-to-use-node-js-for-web-application-development-125144.html (visited on 04/06/2017).

[42] *WebRTC*. URL: https://webrtc.org/ (visited on 02/10/2017).

[43] *WebRTC infrastructure*. URL: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure (visited on 02/16/2017).

[44] *WebRTC samples*. URL: https://webrtc.github.io/samples/ (visited on 03/28/2017).

[45] *WebSocket - Overview*. URL: https://www.tutorialspoint.com/websockets/websockets_quick_guide.htm (visited on 03/22/2017).

[46] *XMPP*. URL: https://xmpp.org/ (visited on 03/29/2017).

[47] Xiaoqing Zhu and Rong Pan. "NADA: A Unified Congestion Control Scheme for Low-Latency Interactive Video". In: (2013).

Appendix A

Congestion control parameters

A1 Google Congestion Control

Name	Definition	Calculation	Extracted from?
DELAY-BASED CONGESTION CONTROL:			
Arrival-time filter			
t_i	Timestamp received		System clock
T_i	Timestamp sent		RTP timestamp
$R(t_i)$	Receiving rate measured in the last 500ms	Total size of packets arriving / time window when collecting packets, which is the last 500ms	Calculates at the receiver and extracts from RTCP
$fl(t_k)$	Fractions of lost packets	$ALPHA * p_{diff} + (1 - APLHA) * fl(t_k)$	Calculates at the receiver and extracts from RTCP
p_{diff}	Different between number of packet missing and sent packets	p_m/p_tot	
p_m	Missing packets	Count packets that does not follow the sequence	RTP sequence numbers
p_tot	Total packets sent	Count at the receiver	Receiver
ALPHA	Smoothing factor in exponential smoothing of packet loss	0.1	Constant

Table A.1: GCC parameters part 1

Name	Definition	Calculation	Extracted from?
$d_m(t_i)$	One-way delay variation measured	$(t_i - (t_{i-1}))$˘$(T_i$ ˘$(T_{i-1}))$	
$d(t_i)$	One-way delay	$(L(t_i) - L(t_{i-1}))$ $/C(t_i) + m(t_i) + n(t_i)$	
	Transmission time variation	$(L(t_i) - L(t_{i-1}))/$ $C(t_i)$	
$L(t_i)$	Video frame length		RTP
$C(t_i)$	Estimation of the path capacity	Constant	Look at service requirements
$n(t_i)$	Network jitter as Gaussian noise		
Adaptive threshold			
$y(t_i)$	Threshold	$y(t_{i-1}) + \Delta t*$ $k(t_i) * (\|m(t_i)\| - y(t_{i-1}))$	
$k(t_i)$		$k_d \|m(t_i)\| < y(t_{i-1})$ k_u otherwise	
k_d **and** k_u	Determines the speed at which the threshold is increased or decreased	$k_d = 0.00018$ $k_u = 0.01$	Constant
Over-use detector			
s	Signal	Underuse signal = $m(t_i) < y$ Overuse signal = $m(t_i) > y$ Normal signal = $m(t_i) == y$	Arrival time filter
Remote rate control			
nj		$\in$[1.005 , 1.3] = 1.05	
α		$\in$[0.8, 0.95] = 0.85	
$A_r(t_i)$	Delay-based new rate	1. $nj * A_r(t_i)$ 2. $\alpha * R(t_i)$ 3. $A_r(t_{i-1})$	Based on over-use detector signal 1 – Underuse signal = Increase state 2 – Overuse signal = Decrease state 3 – Normal signal = Hold

Table A.2: GCC parameters part 2

Name	Definition	Calculation	Extracted from?
REMB processing			
	Transmit new rate A_r to the sender	Every 1s OR $A_r(t_i) < 0.97 * A_r(t_{i-1})$, when A_r decreased more than 3%	Remote rate control
LOSS-BASED CONGESTION CONTROL:			
t_k	The time when feedback packet is received at sender		Sender
$A_s(t_k)$	Loss-based congestion controller	1. $A_s(t_{k-1}) * (1\check{}0.5 * fl(t_k)$ 2. $1.05 * A_s(t_{k-1})$ 3. $A_s(t_{k-1})$	$fl(t_k) > 0.1$ $fl(t_k) < 0.02$ other-wise
A	New target rate to the encoder	min(Ar,As)	REMB message sent from receiver with A_r and loss-based controller calculates A_s

Table A.3: GCC parameters part 3

A2 Network Assisted Dynamic Adaption

Name	Definition	Calculation	Extracted from?
Encoder rate control:			
R_o	Output rate for what the encoder can handle	$[R_{min}, R_{max}]$	Encoder
NADA SENDER			
Reference rate calculation			
- Accelerated ramp up:			
R_n	Reference rate based on collected congestion signal	$Max(R_n, R * (g + 1)$	
g	Rate increaser	$Min(g_max, (QBound/$ $rtt + DELTA + DFILT$	
$QBound$	Upper bound queuing delay	50ms	
rtt	Estimated round-trip-time at sender		
$DELTA$	Observed interval between current and past feedback reports	100ms	
$DFILT$	Filtering delay	120ms	
- Gradual update mode:			
R_n	Reference rate based on collected congestion signal	$R_n - KAPPA * (delta/TAU)$ $*(X_o/TAU)$ $*R_n \breve{} KAPPA * ETA$ $*(X_d/TAU) * R_n$	
$KAPPA$	Scaling parameter	0.5	
ETA	Scaling parameter	2.0	
TAU	Upper bound of RTT	500ms	
$delta$	Time difference	$t_i - T_i$	
X_o	Distance of X_n from $XREF$	$X_n - PRIO * XREF$ $*(RMAX/R_n)$	
X_d	Change in congestion signal from previous value	$X_n - X_p$	
X_p		X_{n-1}	

Table A.4: NADA parameters part 1

Name	Definition	Calculation	Extracted from?
$XREF$	Reference congestion level	20ms	
$RMAX$	Max rate of application	1.5Mps	
$PRIO$	Weight of priority of the flow	1.0	
Encoder target rate calculation			
R_v	Encoder target rate	$R_n \breve{} BETA\_V * 8$ $*L_s * FPS(decrease)$	
$BETA\_V$	Scaling parameter modulating outgoing rate	0.1	
FPS	Frame rate of incoming video	30	
Sending rate calculation			
R_s	Sending rate	$R_n + BETA\_S * 8$ $*Ls * FPS(increase)$	
$BETA\_S$	Scaling parameter modulating outgoing rate	0.1	
Rate shaping buffer			
	Mismatch between output rate and sending rate	$R_o - R_s$	
L_s	Buffersize		
NADA RECEIVER			
Feedback:			
$X_n(t_i)$	Congestion signal	d_tihlde $+p_m * DM + p_l * DL$	RTCP
d_tihlde	Equivalent delay after non-linear wrapping	d_q $Yexp(-0.5(d_q - Y/Y)$	if $d_q < Y$ otherwise
d_q	Estimated queuing delay	$d_f - d_b$	
d_f	Measured and filtered one-way delay	Current time – last time received or sending	
d_b	Estimated baseline delay	$Min(d_b, d_f)$	
Y	Delay threshold for non-linear warping	50ms	
«	Threshold for determining queuing delay build up at receiver	10ms	
p_m	Estimated packet ECN marking ratio	$ALPHA * p_i +$ $(1 - ALPHA) * p_m$	

Table A.5: NADA parameters part 2

Name	**Definition**	**Calculation**	**Extracted from?**
p_l	Estimated packet loss ratio	$ALPHA * p_i +$ $(1 - ALPHA) * p_l$	
DM	Delay penalty for ECN marking	200ms	
DL	Delay penalty for loss	1.0s	
$ALPHA$	Smoothing factor in exponential smoothing of packet loss and marking ratio	0.1	
p_i	The difference between number of missing packets over the total transmitted packets	Total packets transmitted – packets missing	
$d(t_i)$	One-way delay	$t_i - T_i$	
t_i	Timestamp received		RTP
T_i	Timestamp sent		RTP

Table A.6: NADA parameters part 3

Appendix B
Session Questionnaire

Quality of Experience of WebRTC-based application

Session ID:_____ **Date:**_____

This form gathers information on your experience with online multimedia conversations. The information is used in the report for describing QoE with a WebRTC service.

1) Usability of the WebRTC service:

Please answer these questions.

- **Was there any problem to use the service?** ☐ Yes ☐ No

  If yes, please specify: _____

- **Was there a service crash?** ☐ Yes ☐ No

Rate your experience with the service itself.

	Bad	Poor	Neutral	Good	Excellent
How easy was it to use the service?	◯	◯	◯	◯	◯
How was the design of the service?	◯	◯	◯	◯	◯
Overall review of the service?	◯	◯	◯	◯	◯

2) Session feedback:

a) During the session:

Check off in the boxes if you experience it during the session.

VIDEO:			
No video	Frozen image	Slow movement	Black screen

AUDIO:		
No audio	Audio disruption	Slow audio

CONNECTION:	
Can't connect	Lost connection

1

b) After the session:

Check off in the box that matches your overall experience.

	Bad	Poor	Neutral	Good	Excellent
Overall quality of audio	◯	◯	◯	◯	◯
Overall quality of video	◯	◯	◯	◯	◯
Overall quality of combined audio and video	◯	◯	◯	◯	◯