



Norwegian University of
Science and Technology

Quality of Experience of WebRTC based video communication

Eirik Fosser

Lars Olav D Nedberg

Master of Science in Communication Technology

Submission date: June 2016

Supervisor: Min Xie, ITEM

Co-supervisor: Doreid Ammar, ITEM
Katrien De Moor, ITEM

Norwegian University of Science and Technology
Department of Telematics

Title: Quality of Experience of WebRTC based
video communication

Students: Eirik Fosser and Lars Nedberg

Problem description

Internet video applications and services are taking up an increasing share of the Consumer Internet traffic. In this project we focus on Web Real-Time Communication (WebRTC), which is browser-to-browser (peer- to-peer) applications. They do not require download of additional third-party software (like Skype does). Avoiding installations allow the user to run applications more seamlessly. Their success and use are strongly influenced by the quality they provide and the experiences they offer for users. At the same time, however, the delivered quality and the experience for the user of video applications and services may be very negatively influenced by technical constraints such as bandwidth, and parameters like packet loss, delay and jitter. What actually causes degradation of the service is a hot research topic without any clear answers. Conducting experiments in a controlled environment in order to find the impact of these parameters is of great value for both users and application developers.

To support experiments in a controlled environment an experimental test platform is needed and in this project the focus is on implementing a testbed specifically for the WebRTC video conference application Appear.in. The testbed can apply different network limitations to different users (clients), to obtain a better insight on how different network properties affect the perceived Quality of Experience (QoE).

The main tasks for this project will be as follows:

- Briefly, overview the state of the art on most relevant Quality of Service (QoS) and QoE factors in the context of video conferencing (and in particular, WebRTC-based real-time video communication).
- Plan and develop a testbed useful to study the effect parameters like bandwidth, packet loss etc., has on the QoE of video-conferencing applications using WebRTC.
- Setting up and running a pilot study using the developed testbed in a controlled lab setting, data analysis and discussion of findings.

Responsible professor: Min Xie, ITEM
Supervisors: Doreid Ammar, ITEM
Katrien De Moor, ITEM

Abstract

Online video applications are growing in popularity and using an increasing share of the consumer Internet traffic. Web Real-Time Communication (WebRTC) is a new technology which allows browser-to-browser communications without any software downloads or user registration. The focus of this report is the Quality of Experience (QoE) in the context of WebRTC.

We have created a fully controllable testing environment, a *testbed*, where we can manipulate a network to perform under various conditions by altering the parameters packet loss rates, Mean Loss Burst Size (MLBS), delay, jitter, Central Processing Unit (CPU), and bandwidth. A testbed is of importance for testing of QoE services in general, and also for application developers because they can analyze their application's behavior in altered networks which can simulate real-world use.

We have used the WebRTC application *appear.in* for several different experiments where we altered the network conditions. We have collected both connection statistics and the subjective feedback from each participant.

Firstly, we conducted a pilot study consisting of two-party conversations of 12 participants, where our main focus was on packet loss and MLBS. After that, we conducted three-party conversations where we tested packet loss, MLBS, delay, jitter, and CPU.

We found in our experiments that the perceived quality of a specific packet loss rate depends also on the MLBS. Higher MLBS seems to result in an overall worse user experience, especially impacting the audio quality of the conversation. We also found that delay (<1 second) does not necessarily leads to a worse user experience, while jitter quickly impacts both audio and video quality. Finally, it seems that the CPU limitations seem to affect only the user with the reduced CPU-usage.

The experiments show that the testbed is working as specified, and can be used for more extensive research in the future.

Keywords - WebRTC, Quality of Experience, appear.in, testbed, pilot study, Mean Loss Burst Size.

Sammendrag

Nettbaserte videoapplikasjoner øker i popularitet og bruker en stadig større andel av den totale internett-trafikken. Web Real-Time Communication (WebRTC) er en ny teknologi som muliggjør nettleser-til-nettleser-kommunikasjon uten at ytterligere programvare eller brukerregistrering er nødvendig. Denne rapporten handler om Quality of Experience (QoE) i kontekst av WebRTC.

Vi har laget et fullstendig kontrollerbart testmiljø hvor vi kan manipulere et nettverk ved å endre parameterne pakketap, det gjennomsnittlige antall pakker som blir tapt av gangen (Mean Loss Burst Size (MLBS)), forsinkelse, jitter, Central Processing Unit (CPU) og båndbredde. Et testmiljø er et nyttig verktøy for å teste QoE-tjenester generelt, og også for applikasjonsutviklere fordi de får muligheten til å analysere hvordan applikasjonen deres fungerer i varierende nettverksforhold som kan simulere bruk over det åpne internett.

Vi har brukt WebRTC-applikasjonen *appear.in* i flere eksperimenter der vi har endret på nettverksparametere. Vi har samlet inn både tekniske data om nettverksforbindelsen og tilbakemeldinger fra brukerne.

Først gjennomførte vi en pilotstudie med 12 brukere, der hver samtale besto av to brukere. Hovedfokuset vårt for pilotstudien var på pakketap og MLBS. Deretter gjennomførte vi samtaler med tre samtidige brukere der vi testet ut pakketap, MLBS, delay, jitter og CPU.

Vi har oppdaget i våre eksperimenter at den opplevde brukerkvaliteten av en gitt verdi for pakketap er avhengig av hvor mange pakker som blir tapt rett etter hverandre (MLBS). Flere pakker som blir tapt på rad gir en samlet dårligere brukeropplevelse, spesielt med hensyn til lyd-kvaliteten. Vi fant også at forsinkelse (< 1 sekund) ikke nødvendigvis bidrar til en dårligere brukeropplevelse, mens jitter påvirker både lyd- og videokvaliteten betydelig. Til slutt fant vi at begrensninger av maskinressurser (CPU) ser ut til å kun påvirke personen med begrensede maskinressurser, og ikke de andre brukerne i samtalen.

Eksperimentene vi har gjennomført viser at testmiljøet fungerer som spesifisert, og at det kan bli brukt til mer omfattende undersøkelser i fremtiden.

Nøkkelord - WebRTC, Quality of Experience, appear.in, testbed, pilot study, Mean Loss Burst Size.

Preface

This thesis is original and independent work by Eirik Fosser and Lars Nedberg. The thesis is the final contribution to the Master's degree in Communication Technology at the Norwegian University of Science and Technology (NTNU). Our thesis is a part of the project "Quality of Experience and Robustness in Telecommunications Networks" which is a joint collaboration between NTNU and Telenor.

The goal of this thesis is to investigate user-perceived QoE in the online multi-party video application *appear.in*. The general objective of our work is to see how users respond to network alterations which impacts their ability to communicate with another party.

Thanks to our responsible professor Min Xie for valuable feedback and proof reading of the report. We would also like to thank Katrien De Moor and Poul Heegaard for showing interest in the project and helping with proof reading.

Special thanks to our supervisor, Doreid Ammar, for motivating and providing us with useful and constructive feedback throughout the project.

Contents

List of Figures	xi
List of Tables	xiii
List of Acronyms	xv
1 Introduction	1
1.1 Problem	1
1.2 Our Contributions	2
1.3 Disclaimer	2
1.4 Structure	2
2 Background and Related Work	5
2.1 WebRTC	5
2.2 Perceived User Quality	7
2.2.1 Quality of Service (QoS)	7
2.2.2 Quality of Experience (QoE)	8
2.2.3 QoS/QoE in the Context of WebRTC	9
2.3 Related Work	9
3 Methodology and Experimental Setup	13
3.1 System Description and Technical Setup	13
3.1.1 Appear.in	14
3.1.2 Relevant Parameters	14
3.1.3 Relevant Protocols	16
3.1.4 Markov Models	17
3.1.5 Network Emulators	20
3.2 Testbed	27
3.2.1 Testbed Topology	27
3.2.2 Hardware and Software Details	28
3.2.3 Configuration	29
3.2.4 Optimal Network Conditions	30
3.2.5 Synchronizing Machines in the Local Network	31

3.3	Acquiring Data from appear.in-sessions	32
3.3.1	Session-related Statistics	33
3.3.2	Screen Recordings	37
3.4	Questionnaires and Subjective Measurements	38
3.4.1	Pre-session Survey	38
3.4.2	Appear-in Feedback After Each Conversation	39
3.5	Background for Pilot Study	39
3.5.1	Participants	40
3.5.2	Discussion Topics	40
4	Experiments	43
4.1	Validation	43
4.1.1	Validation Tools	43
4.1.2	Validation of Packet Loss Function in NetEm	45
4.1.3	Validating Delay and Jitter Functions	46
4.2	Choosing the Appropriate Loss Model	46
4.2.1	Computing the MLBS	48
4.2.2	Testing the Simple Gilbert Model	48
4.2.3	Testing the Gilbert Model	48
4.2.4	Deciding on the Loss Model	49
4.3	Experiments Using the Testbed Controller	50
4.3.1	Early-phase Testing to Decide Parameters	51
4.3.2	Pilot Study	54
4.3.3	Three-party Conversations	57
5	Results and Discussion	61
5.1	Comparison Between Different Scenarios	61
5.2	Results from the Pilot Study	62
5.2.1	Scenario 1 (No network alterations)	62
5.2.2	Scenario 2 (PL = 10%, MLBS = 1.5)	63
5.2.3	Scenario 3 (PL = 10%, MLBS = 3.0)	63
5.2.4	Scenario 4 (PL = 20%, MLBS = 1.5)	63
5.2.5	Scenario 5 (PL = 20%, MLBS = 3.0)	63
5.3	Discussion of Pilot Study Results	64
5.3.1	Interpreting Subjective User Feedback	65
5.3.2	Comparing Low MLBS with High MLBS	65
5.3.3	Scenarios with High Deviation	68
5.3.4	Differences in Packet Loss Distribution	69
5.4	Results from Three-party Conversations	70
5.4.1	Packet Losses and MLBS Alterations	71
5.4.2	Delay and Jitter Alterations	71
5.4.3	CPU-limit Alterations	72

5.5	Discussion of Three-party Conversations Results	73
5.5.1	Packet Losses and MLBS	73
5.5.2	Effects of High Jitter Values	73
5.5.3	CPU Limitation	76
5.6	Limitations of Results	77
6	Concluding Remarks	79
6.1	Limitations	80
6.1.1	Limitation in Setup	80
6.1.2	Limitation of Data from Pilot Study	80
6.2	Future Work	80
6.2.1	Asynchronous Links	80
6.2.2	Further MLBS Testing	81
	References	83
	Appendices	
A	Configuration Scripts	iii
A.1	Controller configuration	iii
A.1.1	Configure forwarding table	iii
A.1.2	Redirecting traffic	iii
A.2	Static IP-address configuration	v
B	Simple Gilbert MLBS Data	vii
C	Conversation Task for the Pilot Study	ix
C.1	Survival task 1: Survival task in winter	ix
C.2	Survival task 2: Survival task at sea	xi
C.3	Survival task 3: Survival task on the moon	xiii
C.4	Survival task 4: Survival task in the desert	xv
C.5	Survival task 5: Survival task in the jungle	xvii
D	Questionnaires	xx
D.1	Pre-session questionnaire	xx
D.2	Session feedback from appear.in	xxiii
E	Pilot Study - NetEm Scripts	xxv
E.1	Script 1 - (10% PL and 1.5 MLBS)	xxv
E.2	Script 2 - (10% PL and 3 MLBS)	xxvi
E.3	Script 3 - (20% PL and 1.5 MLBS)	xxvi
E.4	Script 4 - (20% PL and 3 MLBS)	xxvii

List of Figures

2.1	Browser-to-browser communications through WebRTC [2].	6
2.2	STUN and TURN server lookups in WebRTC [6].	6
3.1	Relationship between RTP, RTCP, and underlying protocols, inspired by [8].	17
3.2	A simple Markov chain.	18
3.3	Existing packet loss models [35].	19
3.4	Linux queuing discipline [24].	22
3.5	Snippet from top command window, sorted on CPU-usage. Process 12880 is the <i>appear.in</i> conversation.	26
3.6	Snippet from Chrome Task Manager after limiting the <i>appear.in</i> tab to 30%.	27
3.7	Testbed topology for up to n clients.	28
3.8	Throughput for video in optimal conditions (from <i>getstats.io</i>).	31
3.9	Throughput for audio in optimal conditions (from <i>getstats.io</i>).	31
3.10	Illustration of not synchronized graphs before use of NTP (from <i>getstats.io</i>).	32
3.11	Screenshot of all graphs for sending video (from <i>webrtc-internals</i>).	35
3.12	Timeline for each survival task in the pilot study.	40
4.1	Example of ping request, displaying RTT and packet loss percentage.	44
4.2	Traceroute showing that traffic between clients goes through the testbed controller.	44
4.3	Ping simulation where client 1 continuously pings client 2, while the testbed controller alters the network link.	47
4.4	How MLBS and packet loss rate varies when p and r varies.	47
4.5	Simple Gilbert simulation results.	49
4.6	Gilbert simulation results.	50
4.7	Network topology for two clients.	55
4.8	Simple Gilbert loss model for scenario 2 (10% loss rate and 1.5 MLBS).	56
4.9	Simple Gilbert loss model for scenario 3 (10% loss rate and 3 MLBS).	56
4.10	Simple Gilbert loss model for scenario 4 (20% loss rate and 1.5 MLBS).	56
4.11	Simple Gilbert loss model for scenario 5 (20% loss rate and 3 MLBS).	56

4.12	Network topology for three clients.	57
5.1	Visualization of user feedback, indicating the average value \pm one standard deviation.	64
5.2	Packet loss video scenario 2 (10% loss rate and 1.5 MLBS.)	66
5.3	Packet loss video scenario 3 (10% loss rate and 3 MLBS)	66
5.4	Situation 1 scenario 3 (10% loss rate and 3 MLBS)	70
5.5	Situation 2 scenario 3 (10% loss rate and 3 MLBS)	70
5.6	Packet loss ratio for audio from scenario 11 (from <i>getstats.io</i>).	74
5.7	Jitter measures on audio from scenario 11 (from <i>getstats.io</i>).	74
5.8	Packet loss rate for video from scenario 11 (from <i>getstats.io</i>).	75
5.9	Jitter measures on video from scenario 11 (from <i>getstats.io</i>).	75
5.10	Delay measures for video from scenario 11 (from <i>getstats.io</i>).	76

List of Tables

3.1	Comparing properties in <i>getstats.io</i> and <i>webrtc-internals</i>	36
4.1	Scenarios applied in the pilot study.	55
4.2	Three-party conversation - delay and jitter values.	58
4.3	Three-party conversation - restricting CPU-usage.	59
5.1	Actual packet loss rates for each scenario in the pilot study (all values in percentages).	62
5.2	Statistics from user feedback for all scenarios (ratings for average, median and mode are given from 1 to 5).	62
5.3	How participants rated importance of different aspects of online video communications.	68
B.1	r-values with corresponding Mean loss burst size (MLBS)	vii
C.1	Items for surviving in the winter for participant 1	x
C.2	Items for surviving in the winter for participant 2	x
C.3	Items for surviving at sea for participant 1	xi
C.4	Items for surviving at sea for participant 2	xii
C.5	Items for surviving on the moon for participant 1	xiii
C.6	Items for surviving on the moon for participant 2	xiv
C.7	Items for surviving in the desert for participant 1	xv
C.8	Items for surviving in the desert for participant 2	xvi
C.9	Items for surviving in the jungle for participant 1	xvii
C.10	Items for surviving in the jungle for participant 2	xviii

List of Acronyms

API Application Programming Interface.

CPU Central Processing Unit.

DNS Domain Name System.

FAQ Frequently Asked Question.

FEC Forward Error Correction.

FIFO First In First Out.

GE Gilbert-Elliot.

HTB Hierarchical Token Bucket.

ICMP Internet Control Message Protocol.

IETF Internet Engineering Task Force.

IP Internet Protocol.

JSON JavaScript Object Notation.

Kbps Kilobits per second.

LAN Local Area Network.

Mbps Megabits per second.

MLBS Mean Loss Burst Size.

NAT Network Address Translator.

NTNU Norwegian University of Science and Technology.

NTP Network Time Protocol.

OS Operating System.

QoE Quality of Experience.

QoM Quantity of Movement.

QoS Quality of Service.

RTCP RTP Control Protocol.

RTP Real-time Transport Protocol.

RTT Round Trip Time.

SG Simple Gilbert.

SLA Service Level Agreement.

SSH Secure Shell.

STUN Session Traversal Utilities for NAT.

TC Traffic Control.

TCP Transmission Control Protocol.

TURN Traversal Using Relays around NAT.

UDP User Datagram Protocol.

VoIP Voice over IP.

WAN Wide Area Network.

WebRTC Web Real-Time Communication.

Chapter 1

Introduction

Real-time communications services, like Skype, Google Hangouts, and *appear.in*, are rising in popularity and taking over an increasing share of the consumer internet traffic. Many existing video communication services, however, require plug-ins or some software to be downloaded. Web Real-Time Communication (WebRTC) is a relatively new technology which allows browser-to-browser communications, without any downloads or plug-ins, offering a more seamless user experience.

Full-scale services that can run without any additional software downloads offer easy-to-use applications for users. Novice users with little or no computer knowledge may be attracted to use these new services, and experienced users do not have to spend time on challenging and time-consuming setups. The popularity and usage of these services are highly dependent on the Quality of Experience (QoE) the user perceives, as poor user experience quickly will decrease popularity and usage.

1.1 Problem

Obtaining and gathering QoE data about WebRTC is a complicated matter. The users' perceived QoE relies on a large number of network factors such as bandwidth, packet loss, and delays, but also hardware specifications such as processing power. QoE also includes non-technical aspects such as how the user subjectively is satisfied with the application performance. Real-life applications also run over the open Internet, which may be affected by variables outside the application developers control, and in situations and scenarios that can be hard or impossible to reproduce.

It is of interest to research how various network parameters affect the QoE of WebRTC applications. As many parameters and a broad range of values easily will reach an exponential number of scenarios, an important part of the research is to decide what kind of scenarios that are of greatest interest, and to narrow down these parameters as much as possible.

1.2 Our Contributions

The goal of this project is to create a fully controllable testing environment for browser-to-browser video communications. A testing environment, hereafter referred to as a *testbed*, gives the opportunity to manipulate several software and hardware parameters in the network. Our work is, to the best of our knowledge, the first work which investigates QoE in the context of WebRTC in a fully controllable testing environment.

Our main research area has been on how the combination of packet losses and the Mean Loss Burst Size (MLBS) affects the user experience. The fact that QoE is a highly subjective measurement also makes it problematic to draw clear conclusions on conducted experiments. Observing correlation between different network properties and user experience is, however, of great value, and is what we aim to achieve with our work. We have organized and conducted a pilot study using the WebRTC-application *appear.in*¹, and used our testbed to generate various faulty network scenarios. These scenarios were applied and tested by actual users, and we collected their subjective feedback on the perceived user experience by the use of questionnaires.

QoE in WebRTC is an important research topic because it offers useful insight for application developers in how the users experience the quality of their video applications as the network parameters change. The aim of this project is to verify that our testbed environment works correctly and that it can be used to conduct more extensive experiments on user experience in the future.

1.3 Disclaimer

Our pilot study was carried out on 12 participants. This number is far too limited to make any conclusive remarks about the user experience for various network scenarios, as the statistical uncertainty is too large for a small number of participants. Our work is intended as a proof-of-concept for a fully controllable testbed. We will, therefore, use and discuss the feedback from the participants as indicators of the perceived user experience rather than making definite conclusions.

1.4 Structure

This paper is organized as follows: Chapter 2 covers background information about WebRTC, QoE, Quality of Service (QoS) as well as relevant related work. Chapter 3 discusses background information directly related to our experiments, the methodology, and the experimental setup. Several conducted experiments are described in

¹www.appear.in

Chapter 4. Chapter 5 presents results and discussions from our experiments. Finally, conclusive remarks, limitations, and future work are discussed in chapter 6.

Chapter 2

Background and Related Work

This chapter covers background information relevant to this thesis. Technical details about the Web Real-Time Communication (WebRTC)-technology are discussed, as well as the concepts of Quality of Service (QoS) and Quality of Experience (QoE). The chapter is finalized by discussing relevant literature.

2.1 WebRTC

Internet communication applications, such as Skype, allow users to conduct audio and video multi-party conversations by downloading the software and registering to the service. WebRTC is the name of the emerging technology which allows browser-to-browser (peer-to-peer) communications, and does not require any third-party software download. The seamless and plugin-free nature of browser-to-browser communication is expected to be a popular service in the future which for example can be used to result in a better customer experience where WebRTC-conversations replace text chats or Frequently Asked Questions (FAQs).

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" [14]. WebRTC is currently supported by major browsers such as Chrome, Firefox, and Opera [3], and works on both Android and iOS-platforms¹.

It is important to note that WebRTC denotes a technology and not a service that can be run. WebRTC describes a set of necessary protocols, but purposely leaves some of the implementation details, such as initial signaling, to the developer. After the session details are set, all communication can be sent directly between peers, without any external access points, as illustrated in figure 2.1. The WebRTC protocols after that allow users to communicate directly from browser to browser, as

¹iOS and Android implementations as discussed on <https://tech.appear.in/>

6 2. BACKGROUND AND RELATED WORK

subsequent data traffic does not have to go through an external server. Following is a brief introduction about the technicalities of WebRTC.

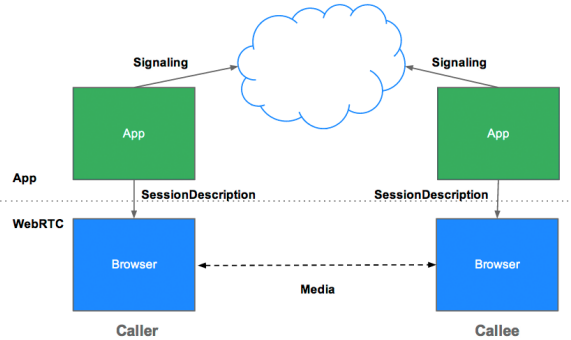
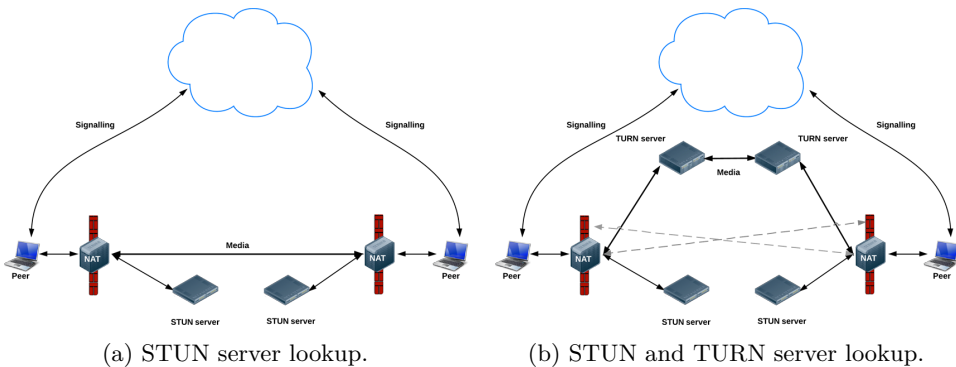


Figure 2.1: Browser-to-browser communications through WebRTC [2].

Dealing with firewalls and Network Address Translators (NATs), however, makes an actual implementation more difficult. If each endpoint had its own globally unique Internet Protocol (IP)-address, the peers could connect directly. However, NATs may hide the direct IP-address and prevent a direct connection for security reasons. WebRTC’s Session Traversal Utilities for NAT (STUN) server is designed to address this problem by retrieving an external network address, which is used for all subsequent messages. The requests made to a STUN server are computationally simple and are therefore used as a first option to connect peers. As much as 86% of all WebRTC calls are successfully connected using the STUN server [13]. Figure 2.2a illustrates a lookup using the STUN server.

In some cases, a users’ firewall can block all traffic sent directly from a peer.



(a) STUN server lookup.

(b) STUN and TURN server lookup.

Figure 2.2: STUN and TURN server lookups in WebRTC [6].

Traversal Using Relays around NAT (TURN) servers are used as a fallback solution if the STUN servers do not return successfully. TURN servers relay data between the different peers, which is not an ideal solution because they consume much bandwidth. For this reason, the STUN server is always queried first. STUN and TURN servers are an essential and required part of the WebRTC-infrastructure, which is needed to operate properly. Figure 2.2b shows an attempted STUN connection which fails, and then falls back on relaying data through the TURN server.

2.2 Perceived User Quality

User experience is an important aspect of all web applications. A service that is faulty or in other ways not working properly will quickly lose its popularity and customers. The literature today mainly uses two different measures for describing the user quality: QoS and QoE. QoS can typically be monitored by measuring individual network aspects such as delay and packet loss while the QoE is more about how the user experiences the video and audio quality of an application.

2.2.1 Quality of Service (QoS)

QoS is the most widely used way of measuring the performance of a service. QoS denotes quantifiable measurements of a network connection, accounting for bit rate, throughput, and delay, to name a few. The term QoS has several slightly different definitions, so we have chosen to use ITU's definition [29], stating that QoS is the:

"Totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service."

The QoS is easily quantified by measurements, and is therefore often used to quantify conditions in Service Level Agreements (SLAs) between providers and customers. QoS is therefore particularly pertinent for applications that require a given minimum network connection to operate properly, such as Voice over IP (VoIP), video conferencing and safety-critical applications which may all require a good end-to-end connection. The customers can complain to their service providers if they violate the QoS-levels stated in the SLA.

But the objective nature of measuring is not only positive as QoS only covers specific technical aspects concerning the application and the network. The objective feedback is problematic as the only measurement because the user experience also consists of several other aspects. It is, therefore, necessary to utilize other concepts to discuss the user experience.

2.2.2 Quality of Experience (QoE)

QoE is of interest because it covers other aspects than QoS. Even though QoE is not as easy to quantify, as it is more of a subjective manner, it also concerns the non-technical details of how a user experiences a given service. ITU's definition is as follows [28]:

"The overall acceptability of an application or service, as perceived subjectively by the end-user."

This definition is relatively vague, not offering any elaboration on what is meant by the word "acceptability", and it does not provide any detail about the "end-user". As QoE is the primary focus of our report, it is desirable to have a more accurate definition at hand. The Qualinet paper [31] proposes the following and more detailed definition:

"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 on the utility and enjoyment of the application or service in the light of the user's personality and current state."

Qualinet's definition goes into much greater detail about the users' expectations, and it also highlights the fact that the type of users could be vastly different. Tech-savvy users may expect a lot from a service while others might be happy with less. It is therefore of great importance to take user diversity into account when discussing QoE, and it should also be considered when conducting studies. In fact, QoE can be said to account for a vast amount of factors influencing the user experience:

- User characteristics. Young tech-savvy people versus older people with less computer experience may expect different performance values.
- The context of the video call, for example, whether the call is for business or pleasure. The user is possibly likely to expect more in a business setting than in a personal setting.
- Surroundings may also affect the expectations of a user. Factors such as room size and noise may change the user's mood and make the user more or less likely to have a good experience.

- The familiarity and quality of the equipment can also be important for the user experience.

Although QoE and QoS are now discussed as very different measurements, QoE is in fact highly dependent on QoS. If the QoS-parameters are bad, the connection is likely to be poor, and the user experience will also decrease. But if the QoS-parameters are acceptable, the users initial expectations may lead to different QoE-values. However, it will always be difficult to conduct QoE-research, because of its subjective nature and because experiments are hard or impossible to repeat as the test participants may have different preferences and opinions.

Even though we have discussed QoE as a highly subjective measurement, it is also possible to use QoE to obtain more objective data. While the individual feedback consists mainly of questionnaires and interviews, the objective feedback can consist of task scores or speech patterns [36]. Task scores can be used as a metric to see whether the user manages to solve a problem within a time limit, or even measure the number of successful attempts at a given case. Speech patterns can be used to measure speaking times and length of turns and pauses. We have, however, chosen to focus on the subjective measurements of QoE due to the time limitation of the project.

QoE is important because it says something about both technical and non-technical aspects of the network and application performance. In the pilot study described later on, we are interested in observing the QoE for the users when we manipulate the network to operate poorly. We will try to find out what combination of parameters the users think is the most important to conduct a proper video call. QoE is also of keen interest because it is much less researched than QoS. We will, therefore, try to contribute to some useful insights on QoE in the context of WebRTC.

2.2.3 QoS/QoE in the Context of WebRTC

One of the main features of WebRTC is the simplicity for the users, who do not have to download software nor register to a service. Plug-and-play applications like *appear.in* offer an alternative to the well-established Skype. It is therefore of great importance that the QoE and the QoS are as good in WebRTC applications as the customers are used to from other applications.

2.3 Related Work

Browser-to-browser communication, realized by WebRTC, is currently a popular research topic with broad academic interest. Several different research areas are

highly relevant for discussing QoE and WebRTC applications including network emulators, testing environments, and loss models.

A network emulator can be used to manipulate the network connection for a particular set of users. Nussbaum and Richard discuss different network emulators' features and problems [34]. The paper focuses on three main network emulators: Dummynet, NISTNet, and Linux Traffic Control Subsystem, which are all freely available to download. Several problems that may occur when using network emulators, such as system clock differences, are exposed.

The network emulator NetEm offers a simple interface for simulating different network connections by adding delay, packet loss, and jitter to name a few. The goal of NetEm is to "provide a way to reproduce long distance networks in a lab environment" [24], and is a simple and useful tool for varying network conditions in a testbed.

Cinar and Melvin use a black-box testing method for assessing the quality of WebRTC-sessions under varying network conditions by using a network emulator [21]. The report highlights potential problems and dangers of black-box testing, as it can lead to very misleading results if the testbed is not properly validated. Mainly they discuss issues of particular network architectures for the testbed. This is relevant to our project because it raises awareness of common pitfalls that might not be easy to detect, and confirms the importance of properly validating a testing environment.

While many papers discuss the minimal network requirements to obtain an acceptable QoE, Vucic and Kapov try to quantify the minimal hardware requirements for mobile devices to be used for video communications in WebRTC [39]. This paper discusses different smartphone configurations and suggests that a 2.5 GHz processor and 2 GB RAM are minimal requirements for three-party video conferencing. Hardware requirements are an important aspect of QoE, as poor hardware will lead to bad user experiences. Even though the mentioned article is mainly about mobile devices with generally weaker hardware specifications, it is still relevant restricting the Central Processing Unit (CPU)-power of powerful desktop computers as they can simulate less powerful devices.

Gunkel et al. discuss QoE in video conferencing under limited network conditions [23]. They conduct experiments on how a single participant with a limited connection can negatively affect the QoE for all other participants in the conversation. These experiments are done by altering the layout, video quality and network limitations of the conversations. Experiments on asymmetric connections are of high relevance because it pictures a real-life setting.

A QoE testbed is implemented and discussed in [36], allowing modification and

monitoring of network conditions in real-time. The ultimate goal of this approach is to create a controllable and reproducible testing environment for real-time video communications, which is quite similar to our ultimate goal. This report, however, says little about how the testbed is used to control network parameters, but instead focuses on building a single tool that can gather all the information required. They have, for instance, included the functionality for adding speech pattern analysis and questionnaires directly into their tool to collect all information in one place. They tested this configuration by conducting a study with around 50 people, with particular focus on the context of the conversation as well as the role of the user participating in the conversation.

Ammar et al. discuss performance statistics and how they relate to QoE in the context of WebRTC. Both the following reports describe Google Chrome’s *webrtc-internals* tool, which is used to present real-time connection statistics from ongoing WebRTC conversations.

A discussion on the usefulness of WebRTC-internals as a statistics gatherer despite its limitations is found in [18]. This paper highlights problems such as limitation in the number of sample points, imprecise sampling times, and that the tool lacks clearly documented definitions of the data it presents. Their conclusion is that the statistics can be very helpful for root-cause analysis, as long as the user is aware of its shortcomings.

Potential video QoE killers with respect to performance statistics are discussed in [17]. The authors try to relate the performance statistics to the users’ QoE. They ran a series of tests involving two-party conversations while collecting real-time session feedback by using the *webrtc-internals* tool, aiming to find specific network parameters which can be related to the perceived user experience. Their findings identify video freezes, which can be provoked by altering the mentioned network parameters, as a key QoE killer, and suggest a further in-depth investigation of other possible QoE killers.

Sunde and Underdal carried out a pilot study regarding QoE in Kahoot!², which is a cloud-based class response system [38]. Their goal was to see if the users’ perceived experience changed as they made the game unfair by varying the network delay. They conducted tests in a classroom with a large number of users and gathered user forms to get feedback from each participant. They managed to conclude that an unfair setting in a competitive environment enhances the feeling of annoyance among the affected users while it also increases the delight of the users not affected by the delay. Altering the delay can undoubtedly affect the QoE.

²<https://kahoot.it>

Apostopolous discusses how packet losses negatively affect the quality of different network applications [19]. But packet losses can come in several different shapes: isolated single-packet losses, burst losses which are (roughly consecutive) packets lost or a temporary outage. The effect these loss types have on the connection depends greatly on the application. Several error concealment techniques exist, such as the Forward Error Correction (FEC), which makes it possible to deal with the problem of single packet losses. Nevertheless, single packet losses have a relatively small effect on video streaming applications. Burst losses, however, are harder to deal with, mostly because error correction techniques become less effective when several consecutive packets are lost.

Liang et al. also investigate whether or not the average packet loss burst length matters for the perceived quality of video applications [33]. Several simulations run show that the loss pattern of the packets lost greatly affect the total distortion experienced by the user. These simulations are of great importance because they imply that two network links with the same packet loss percentage can describe two widely different user experiences, depending on the size of the bursts.

Chapter 3

Methodology and Experimental Setup

The primary goal of this thesis was to create a fully controllable testing environment for multi-party video conversations. We needed to create a testbed where we could control the different network and other performance related properties. Several experiments were carried out for this project, including deciding the network topology, configuration of the testbed and verification of correctness. After verification and validation, we conducted three different studies:

- A testing phase where we tried a broad range of parameters and values to decide on what we wanted to further investigate in the other experiments.
- After that, we conducted a pilot study where we used our testbed implementation on a small group of users to gain feedback on the perceived QoE. The parameters for the pilot study were narrowed down to the combination of packet loss and Mean Loss Burst Size (MLBS).
- Ultimately, we conducted a three-party conversations where we tested different parameters, such as packet loss combined with MLBS, delay combined with jitter and restricting the CPU-usage.

This chapter describes the methodology, a detailed explanation of the experimental setup and also includes background information about the tools we have used. Chapter 4 elaborates on details from the actual experiments.

3.1 System Description and Technical Setup

Following is a description of different applications, tools, network parameters and network protocols that are a part of our testbed.

3.1.1 Appear.in

Appear.in is a multi-party video application which implements WebRTC and is used without any software downloads or user registrations. A conversation has to be a part of a session, referred to as a *room* in *appear.in*, which is limited to a maximum of eight simultaneous users. Users can just enter an easy-to-remember URL with the specified room name and wait for other users to access the same room. The application also lets the users use a text chat and the possibility to lock the room to prevent others from joining.

The users in a particular conversation are connected using a mesh topology. *Appear.in* uses one link for audio and one for video. Therefore, a total of $2n(n - 1)$ connections (for an n -party call) are needed, which quickly increases as n grow larger. As *appear.in* is limited to eight people per room, the details on how WebRTC deals with signaling for a substantially greater number of endpoints is therefore outside the scope of this report. A paper on various network topologies and their effectiveness can be found in [27].

The reason why we chose to use *appear.in*, and not for example *Google Hangouts*, is that Norwegian University of Science and Technology (NTNU) hosts a research version of the *appear.in* server, which is beneficial because we gain access to additional features compared to other WebRTC applications. We can, for example, access data about the connection quality and control the questionnaire which is presented to the user at the end of a session. Therefore, *appear.in* offers a better total package than any other WebRTC application.

3.1.2 Relevant Parameters

Numerous controllable and non-controllable parameters may impact audio and video quality of a multimedia conversation and, therefore, affect the QoE. The following is an explanation of the different parameters used in our testbed.

Bandwidth

Bandwidth is a measure of the available bit-rate in the network, usually measured in Megabits per second (Mbps). Bandwidth will often vary from user to user depending on their internet connection. An application that works perfectly with a large bandwidth may operate poorly when the bandwidth decreases. Restricting maximum bandwidth for different client links can offer interesting results on how QoE varies depending on the available bandwidth.

Packet Loss

Packet loss measures the number of packets lost versus the actual number of packets sent, and is usually presented in the *percentage* of lost packets. One can look at packet losses in two ways: independent losses or burst losses, and the QoE may be different depending on the type of loss [32]. Later in this chapter, we will look at various loss models for introducing different types of packet losses to the network. According to [5], the number of packets lost, in WebRTC are defined as specified in RFC3550 [37] which says that the number of packets lost is the number of expected packets minus the actual received packets, including late and duplicate packets. Further, the number of packets expected is computed by using the highest sequence number received.

Mean Loss Burst Size (MLBS)

MLBS is directly dependent on the loss models discussed in chapter 3.1.4 but it needs a separate explanation. MLBS describes the correlation of packet losses by explaining how many consecutive packets that are lost on average each time a packet is lost. High packet loss correlation will result in a higher MLBS. The assumption is that loss events are to some extent correlated, meaning that the next packet transmission after a packet loss has a higher probability of being lost as well.

$$010101010101010101 \quad (3.1a)$$

$$00011111000001111100 \quad (3.1b)$$

An example is shown in listing 3.1, where 0 denotes a successful packet transmission and 1 denotes a packet loss. The listing illustrates (an extreme scenario of) two bit sequences with the same packet loss percentage, but with vastly different MLBSs. The first sequence has alternating successful and lost packets (MLBS = 1) while the second series loses five consecutive bits twice (MLBS = 5). The negative effects in the network caused by low MLBSs can be reduced by applying error correction techniques, but this is not possible for high MLBSs as the error correction techniques depend on a number of successful packets to find and correct single packet errors.

Delay

The delay is a measure of how long it takes to transmit a packet from its source to its destination. The Round Trip Time (RTT) can also be used to describe the delay, by measuring the time it takes from the source to destination and back again. The delay may typically influence the QoE by having late play-out on the receiving end,

which can cause long and unnatural pauses in a conversation. The two terms *delay* and *latency* are used interchangeably in this report.

Jitter

Jitter is related to the delay, and it is known as the variation of the delay between consecutive packets. Jitter may also lead to reordering of packets. WebRTC calculates jitter as defined in RFC3550 [37], according to [5]. RFC3550 states that the difference in arrival time, $D(i, j)$, between two packets i and j should be computed as follows.

$$D(i, j) = (R_j - S_j) - (R_i - S_i)$$

Where S_i and S_j are the time stamps for when packets i and j are sent, and R_i and R_j denotes the time of arrival for packets i and j . Further is the interarrival jitter for a packet i , computed as follows.

$$J(i) = J(i - 1) + (|D(i - 1, i)| - J(i - 1))/16$$

Where $J(i-1)$ is the inter-arrival jitter value of the previous packet, and $D(i-1, i)$ is the difference in arrival between packet i and the previous transmitted packet. The value $1/16$ is a noise reduction parameter.

A separate voice and video engine exists in WebRTC, each with its own jitter buffer. An explanation about the jitter buffer for audio is as follows: "*A dynamic jitter buffer and error concealment algorithm used for concealing the negative effects of network jitter and packet loss. Keeps latency as low as possible while maintaining the highest voice quality*" [12]. The jitter buffer in the video engine also mitigates the impact jitter and packet losses has on the video quality. The jitter buffer is useful because it can, to some extent, counteract the late arrival of a packet without the user noticing.

CPU-usage

Unlike the other parameters, CPU-usage has to be specified at each single client, as opposed to all other parameters which are controlled by the testbed controller. We want to restrict the CPU-usage below what is actually used during a normal conversation to see how the QoE is affected. It is of high relevance to see how different hardware specifications can affect QoE of a video chat as this can simulate the use of less powerful devices like mobile phones and tablets.

3.1.3 Relevant Protocols

Following is a short description of the Real-time Transport Protocol (RTP) and the RTP Control Protocol (RTCP) which are both used in WebRTC.

RTP is a protocol typically used in multimedia communication applications and resides at the application layer, running on top of the User Datagram Protocol (UDP). RTP supports services like identifying the payload type, sequence numbering and delivery monitoring of both audio and video packets [37]. In RTP, audio and video packets are separated and transmitted using different UDP ports.

RTCP is used together with RTP and is sent as separate packets to provide information about QoS parameters in an active conversation. Statistics presented in RTCP includes the fraction of packets lost, the highest sequence number received, the cumulative number of packets lost, the inter-arrival jitter, and information about the delay, to name a few. The delay is measured by the time since the last report was received.

Figure 3.1 shows the relationship between RTP, RTCP, and the underlying protocols UDP and the IP.

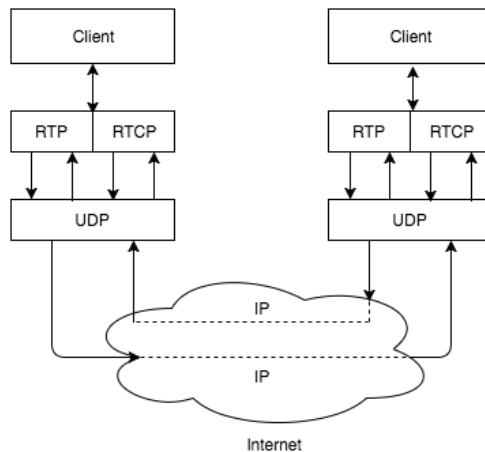


Figure 3.1: Relationship between RTP, RTCP, and underlying protocols, inspired by [8].

3.1.4 Markov Models

A large-scale packet-switched network can be hard to monitor because many factors affect the transmitted packets. We are interested in a quantitative measurement of lost packets, and not necessarily the reasons why the packets are lost. It is therefore of great value to be able to model an abstraction of the system, to gain some useful insight on packet loss estimates. A Markov model can be used to model an abstraction of the actual system.

A Markov model consists of a finite number of states and the transitions between them. Markov models hold the Markov property, which means that the future states depend only on the current state, and not how it got to the current state [40]. An illustration of a simple, two-state Markov model for packet losses can be seen in figure 3.2. G (the Good state) is the state where no packets are lost while all packets are lost in state B (the Bad state). p denotes the probability of a transition from state G to state B , and r indicates the probability of going from state B to state G .

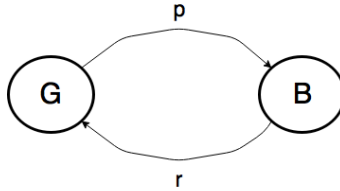


Figure 3.2: A simple Markov chain.

A Markov model can estimate the packet loss by calculating the probability of being in a particular state. From [22], we have that: "If a system in a steady state is observed over a long period of time $(0, \tau)$ where τ is large, the fraction of time the system is in state i is equal to $p_i \tau$."

Therefore, we have that the probability of being in the different states is given by [35]:

$$\pi_G = \frac{r}{p + r} \quad (3.2a)$$

$$\pi_B = \frac{p}{p + r} \quad (3.2b)$$

where π_G is the probability of being in the good state (G) and π_B is the probability of being in the bad state (B). These equations are valuable because they offer theoretical values which can be compared to the data from our practical approach further discussed in chapter 4.2. Following is a more detailed description of existing loss models from the literature.

Existing Loss Models

Figure 3.3 shows four different loss models, varying in complexity. It is important to choose a model that is valid for the task at hand, but not use a model that is more complex than necessary. Selecting a loss model should, therefore, be a trade-off between simplicity and controllability.

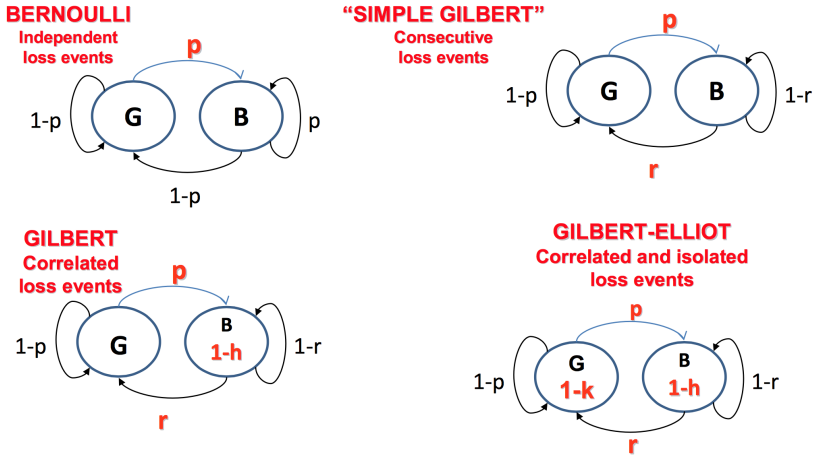


Figure 3.3: Existing packet loss models [35].

- The **Bernoulli model** only takes one parameter, p , which denotes the probability of the transition from state G to B, which corresponds to the packet loss rate for this model. Using only one parameter means that each packet is modeled independently from all other packets, and a bursty behavior is not possible. Modeling only independently dropped packets is too simplified to be applied to any real-world scenarios.
- The **Simple Gilbert (SG) model** takes the two parameters, p (transition from G to B) and r (transition from B to G), and supports the modeling of correlated loss events. Two parameters make it possible to model a simple burst behavior by specifying a separate parameter for a repeated packet loss, as opposed to only independent losses.
- The **Gilbert model** offers an enhancement to the SG model in the way that it provides another parameter, $1-h$, in addition to p and r . The $1-h$ parameter describes the *loss density* in state B, which allows modeling successful packet transmissions in state B. The loss density parameter is desirable because it allows fine tuning of the packet loss and the probability of error in a more accurate way than the SG model.
- The **Gilbert-Elliott (GE) model** introduces yet another parameter, $1-k$. The $1-k$ parameter describes the *loss density* in state G, which makes it possible to model isolated loss events in state G.

To better understand the difference between the four models, we can have a look at the following bit sequences where 0 denotes a successful packet transmission and

1 denotes a packet loss. The number of packets or packet losses in each case is not important but we want to illustrate the different models' application area.

$$\text{Bernoulli} : \overbrace{0000000000}^{\text{Good}} \overbrace{1111111111}^{\text{Bad}} \overbrace{0000000000}^{\text{Good}} \quad (3.3a)$$

$$\text{SimpleGilbert} : \overbrace{0000000000}^{\text{Good}} \overbrace{1111111111}^{\text{Bad}} \overbrace{0000000000}^{\text{Good}} \quad (3.3b)$$

$$\text{Gilbert} : \overbrace{0000000000}^{\text{Good}} \overbrace{1011110111}^{\text{Bad}} \overbrace{0000000000}^{\text{Good}} \quad (3.3c)$$

$$\text{Gilbert - Elliot} : \overbrace{0010100000}^{\text{Good}} \overbrace{1101111101}^{\text{Bad}} \overbrace{0100000100}^{\text{Good}} \quad (3.3d)$$

The listing in 3.3 illustrates important differences between the models. The first two bit sequences support only successful packet transmissions in state G, and only packet losses in state B. The third sequence, picturing the Gilbert model, allows successful packet transmissions in state B. The last sequence, picturing the GE model, allows both isolated loss events in state G and successful packet transmissions in state B.

It is not obvious how the different variables of the presented model affect the packet loss rate. The packet loss rate for the first two models corresponds to the time spent in state B, but it is more complicated for the last two models. A general error probability model valid for all models is as follows [35]:

$$p_{Error} = (1 - k)\pi_G + (1 - h)\pi_B \quad (3.4)$$

for $0 < 1-h \leq 1$ and $0 \leq 1-k < 1$.

If $1-h$ was 0, we would have no packet losses in state B, and if $1-k$ was 1, we would lose all packets in state G. For the models that do not contain these parameters, by default the parameter stating the loss density in state B is set to, $(1 - h) = 1$ and the parameter specifying the loss density in state G is set to, $(1 - k) = 0$, which is the case for the SG model. This also shows that the packet loss rate for the first two models corresponds only to the time spent in state B, as $1-k$ is zero.

3.1.5 Network Emulators

Network emulators which make networks operate poorly and slowly are of interest because they allow us to investigate different protocols to see how they perform under different circumstances. Adding latency to a Local Area Network (LAN) can,

therefore, imitate the behavior of a Wide Area Network (WAN). Limiting a users' bandwidth is of obvious importance because different users have different bandwidths. Congestion control in the Transmission Control Protocol (TCP) is an example of a protocol which should be tested in an imitated WAN network. The reason is that a buffer size working nicely for a low latency LAN may not apply to a longer latency because it may cause a significant number of re-transmissions and heavy congestion.

Note that it is of great importance to validate that the network emulator actually does what is specified, and this should be a part of any work involving network emulators.

Several different network emulators with varying perks exist, and their strengths and weaknesses are further discussed in the following paragraphs. We have had a closer look at Dummynet and NetEm which are both considered and compared in [34]. These two are chosen because they are open-source and in use by the research community.

Comparison of NetEm and Dummynet NetEm and Dummynet are network emulators which share many of the same capabilities but they do have some important differences. Dummynet is included in the operating system FreeBSD and OSX, but can with some effort be used in Linux and Windows [20]. NetEm is a network emulator available in most Linux distributions and is an addition to the Traffic Control (TC) tool in Linux. NetEm adds more functionality such as packet loss, delay, jitter, packet reordering and packet duplication into the network [24]. NetEm combined with the extensive TC tool in Linux provides countless opportunities of traffic shaping which is further discussed later in this chapter.

Both NetEm and Dummynet have the capability of adding delay to packets, but only NetEm allows adding jitter in addition. NetEm also includes more functionality regarding packet loss, particularly on packet loss with correlation, enabling packets to be dropped in bursts, and not just independently [34]. Dummynet only allows packet loss without any correlation, which was insufficient for the experiments in this project.

One of the main advantages with Dummynet was that it was possible to modify both incoming and outgoing packets [34]. All traffic in our testbed is, however, sent through a single entity where the network emulator resides. Whether traffic is shaped on the incoming or outgoing interface at this entity will not make any difference, so NetEm's capabilities are sufficient for our use.

NetEm also allows adding packet corruption, packet duplication, and packet reordering in a network. We chose to use NetEm, as it includes more capabilities than Dummynet relevant for the experiments in this project.

NetEm and Traffic Control

To understand how NetEm works it is important to know how the TC tool in Linux works. An introduction to the TC tool and how it was utilized in this testbed is needed before looking into how NetEm can be used to change the properties of a network.

Qdisc The *qdisc* is the scheduling component, or queuing discipline, in TC in Linux and is used to schedule incoming packets in the network [10]. The queuing takes place in the Linux kernel as shown in Figure 3.4. Note that the figure says that TCP is used, but the same applies for UDP. By default, the kernel uses a simple First In First Out (FIFO) queue, but it can be combined with other TC-components to create a more sophisticated scheduler for network packets. *qdiscs* can further be divided into *classless* and *classful*. The Hierarchical Token Bucket (HTB) is characterized as classful and is the one we have used in this testbed.

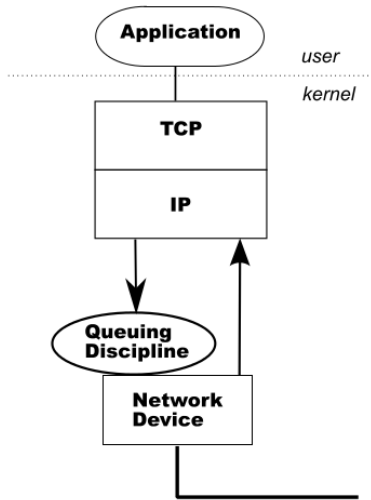


Figure 3.4: Linux queuing discipline [24].

Hierarchical Token Bucket (HTB) It was desirable to distinguish and control separate links between each party when controlling the network traffic with the testbed, for all variations of multi-party conversations. Because the topology of the testbed transmitted all the traffic on the same network link, this network link had to be divided into separate simulated links each having its properties. TC in Linux allows us to do this by using a component called HTB, which as the name suggests applies a hierarchical division of a network link [10]. The top of the hierarchy, the root, corresponds to the entire network link itself. It allows the top of the hierarchy to

be divided into several simulated connections in a parent-child manner. Furthermore, HTB allows each of the simulated links to have its traffic rate specified, thus enabling each link to have its bandwidth.

The following listing shows how to split a network link into two separate simulated links, each with a bandwidth of 24Mbps. It also specifies an identifier which can be used by NetEm and other TC components to distinguish the links.

```
$ tc qdisc add dev eth1 handle 1: root htb
$ tc class add dev eth1 parent 1: classid 1:1 htb rate 1000Mbps
$ tc class add dev eth1 parent 1:1 classid 1:11 htb rate 24Mbps
$ tc class add dev eth1 parent 1:1 classid 1:12 htb rate 24Mbps
```

Filter Another component in TC is called *filter*. Filters are used to classify packets based on their properties. The property we used in this testbed was the source and destination IP-address of the packet, by using the u32 [11] filter. Combining filters with the HTB allowed the specification of individual simulated links between all parties in a conversation.

The following listing shows how to apply a filter for two parties with IP-addresses *10.0.0.2* and *10.0.0.3*. The *flowid* parameter corresponds to an identifier of a simulated link and is used to reference the specified link.

```
$ tc filter add dev eth1 prio 1 u32 match ip dst 10.0.0.3 match \
  ip src 10.0.0.2 flowid 1:11
$ tc filter add dev eth1 prio 1 u32 match ip dst 10.0.0.2 match \
  ip src 10.0.0.3 flowid 1:12
```

Up until now, every component discussed has been components in the TC tool. We have shown how TC can be used to classify packets based on properties such as source- and destination IP-addresses and how this classification can be used to assign packets to specified simulated network links. What is left to describe is how to apply network properties to a link, which is where NetEm comes in.

NetEm - Rules Different network properties can be assigned to different links by specifying *rules* in NetEm. A rule can consist of one or more network properties, such as packet loss, delay, and jitter. The following will look at how packet loss, delay and jitter can be added to the network by using NetEm.

Packet Loss in NetEm

NetEm includes three different functions for introducing packet loss in an emulated network. The following section gives a description of the three functions and a justification for our choice of function for the testbed. The following listing illustrates the parameters available for the three different models:

```
LOSS := loss {random PERCENT [ CORRELATION ] |
             state p13 [ p31 [ p32 [ p23 [ p14] ] ] ] |
             gemodel p [ r [ 1-h [ 1-k ] ] ] } [ ecn ]
```

Note that *ecn* at the end of the listing makes it possible to mark a packet as lost, without actually dropping it. We did not use *ecn* for any of our experiments, as we wanted to actually drop the packets to impact audio and video quality of *appear.in*. The listing is taken from [7].

Packet Loss Functions The first and simplest way to introduce packet loss with NetEm is the function called *random*. Packet loss is added to the network by specifying a percentage value and optionally a correlation percentage, indicating to what extent a previously lost packet should affect the probability of a consecutive packet loss [7]. Packets will be dropped independently if the correlation is not specified. Below is an example of how to use NetEm to add a packet loss of 15%, and packet loss of 15% with 30 % correlation, respectively.

```
$ tc qdisc add dev eth1 root netem loss 15%
$ tc qdisc add dev eth1 root netem loss 15% 30%
```

Packet losses in NetEm can also be introduced by using the *state*-function, which uses Markov models with either two, three or four states [35]. The *state* function is used by specifying transition probabilities rather than the packet loss directly. A transition probability is a probability of transitioning from one state to another.

The third function in NetEm, denominated by *gemodel*, is an implementation of the GE model previously described. Using two parameters results in using the SG model, as illustrated in the following listing.

```
$ tc qdisc add dev eth1 root netem loss random gemodel 10% 70%
```

In the example above the transition probability p , from state G to state B, is 10%, while the transition probability r , from state B to state G is 70%, which adds

up to a total packet loss rate of 12.5% (corresponding to the time in state π_B by using equation 3.2).

Delay in NetEm

The following listing will give a brief overview of the different capabilities NetEm has on adding delay to a network link.

```
DELAY := delay TIME [ JITTER [CORRELATION] [DISTRIBUTION] ]
```

As shown in the listing above, the delay function in NetEm can take up to three parameters. The only mandatory parameter is the delay value denoted as "TIME", specified in milliseconds. As with packet loss, further functionality can be added by specifying more optional parameters. Jitter and the correlation percentage of the jitter can be added by specifying these values. A statistical distribution¹ can also be specified when applying delay and jitter. The following is an example of how delay can be added by either determining the delay value, delay with jitter, delay with jitter and correlation, and delay with jitter using a *normal* distribution, respectively.

```
$ tc qdisc add dev eth0 root netem delay 100ms
$ tc qdisc add dev eth0 root netem delay 100ms 50ms
$ tc qdisc add dev eth0 root netem delay 100ms 50ms 20%
$ tc qdisc add dev eth0 root netem delay 100ms 50ms distribution normal
```

Cpulimit

*Cpulimit*², as the name suggests, makes it possible to limit how much of the CPU that can be used by a single process. The restrictions are done on the client side and has to be done for each client, which is different from all other parameters which are controlled by the testbed controller.

An active *appear.in* two-party video-conversation uses around 70-80% of the CPU on one single core on our clients. We restricted the maximum CPU-percentage this browser tab is allowed to use by issuing commands with *cpulimit*, which alters the maximum allowed CPU-usage by issuing *SIGSTOP* and *SIGCONT* signals to the specified process [4].

SIGSTOP pauses the defined process in its current state and is applied when the actual CPU-value exceeds the specified limit. The *SIGCONT* is used to resume

¹Distribution can be specified for other NetEm functions also, but we have only used it for the delay function.

²<http://cpulimit.sourceforge.net/>

the execution of the specified process at where it was when it was stopped. Since *cpulimit* uses *SIGSTOP* and *SIGCONT* signals to control the total CPU-usage, the actual usage will vary around the specified limit, and not be exactly equal to the threshold set.

We then conducted several experiments by using our testbed to find various values for CPU-usage by issuing commands such as:

```
$ cpulimit -l 30 -p 12880
```

which sets the maximum CPU-usage for process 12880 to 30%.

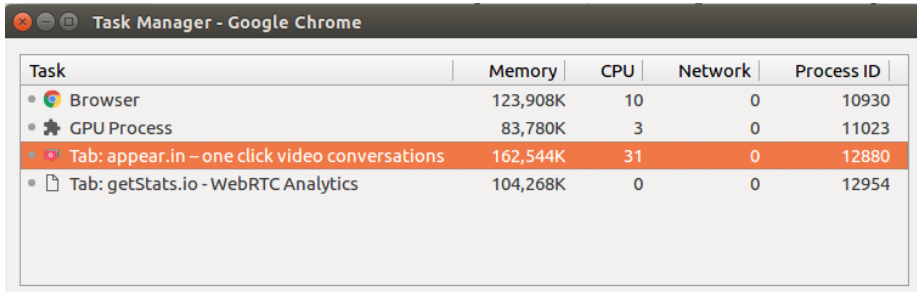
Running processes can be monitored by using the *top* command, as illustrated in Figure 3.5, which shows a cropped version of a screenshot. While most web browsers run in one single process, Chrome runs one process for each browser tab and extension, so it is necessary to find out which process that runs the *appear.in*-session. Luckily, Chrome has its own task manager, which shows the CPU-usage of each tab, as can be seen in Figure 3.6.

```
top - 15:52:10 up 4:40, 5 users, load average: 0,50, 0,80, 0,86
Tasks: 269 total, 3 running, 262 sleeping, 4 stopped, 0 zombie
%Cpu(s): 10,5 us, 2,5 sy, 0,0 ni, 85,1 id, 1,8 wa, 0,0 hi, 0,2 si, 0,0 st
KiB Mem: 12238404 total, 5826940 used, 6411464 free, 526048 buffers
KiB Swap: 12515324 total, 0 used, 12515324 free. 2152896 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
13718	lars	20	0	1140104	97080	36412	S	32,9	0,8	0:00.99	shutter
12880	lars	20	0	1096020	237724	82156	T	32,2	1,9	9:53.98	chrome
1298	root	20	0	848248	311320	127416	R	9,0	2,5	12:43.86	Xorg
2185	lars	20	0	1777976	408888	103228	S	7,6	3,3	7:16.00	compiz
10930	lars	20	0	1778448	245744	122500	S	7,0	2,0	8:42.06	chrome
2051	lars	9	-11	581480	12724	10232	S	6,3	0,1	6:06.42	pulseaudio
2024	lars	20	0	516436	46960	22228	S	3,3	0,4	0:11.36	unity-panel-ser
4468	lars	20	0	1036344	149144	80056	S	2,3	1,2	5:42.22	spotify
1993	lars	20	0	771072	77588	44184	S	2,0	0,6	0:08.80	hud-service
11289	lars	20	0	994692	252072	61328	S	2,0	2,1	0:19.88	chrome
4621	lars	20	0	665504	39888	24688	S	1,7	0,3	0:30.75	gnome-terminal

Figure 3.5: Snippet from *top* command window, sorted on CPU-usage. Process 12880 is the *appear.in* conversation.

Monitoring the CPU-usage is of great importance because it can be used to find minimum hardware requirements that result in an acceptable QoE. Finding minimum hardware requirements is especially relevant for tablets and phones that have much more restricted hardware specifications than desktop computers, which is important because more and more people use tablets and phones for more demanding operations than earlier. Limiting the CPU-usage is an important part of fully controlling our testbed environment.



Task	Memory	CPU	Network	Process ID
Browser	123,908K	10	0	10930
GPU Process	83,780K	3	0	11023
Tab: appear.in - one click video conversations	162,544K	31	0	12880
Tab: getStats.io - WebRTC Analytics	104,268K	0	0	12954

Figure 3.6: Snippet from Chrome Task Manager after limiting the *appear.in* tab to 30%.

3.2 Testbed

This section includes an overview of the different components of the testbed. Firstly, an outline of the testbed topology is provided, before the details of the hardware and software of the testbed are given. Finally, the configuration of the devices in the testbed are explained.

3.2.1 Testbed Topology

The testbed topology is illustrated in figure 3.7. The testbed consists of a number of clients, a switch and a testbed controller (hereafter referred to as "controller"). *Appear.in* allows a maximum of eight simultaneous users while the switch we have used have seven network interfaces available. If another WebRTC application and a different switch was used, however, the number of participants would not have been limited in the same way. Nevertheless, our intention was not to test the maximum number of connections possible for the testbed.

All clients are connected through the switch which is connected to the controller. Clients are not directly interconnected; instead, all traffic is redirected to the switch. Redirecting to a single point is desirable because it allows a centralized point of configuration. Each link, both uplink, and downlink from each client can be manipulated independently.

All traffic from the clients will pass by the controller, and over the Internet and eventually reach the *appear.in* test server. We used a separate computer to remotely run scripts on the testbed controller by issuing Secure Shell (SSH) commands. Administering the testbed controller remotely offers flexibility in the way that we do not have to be at the same physical location .

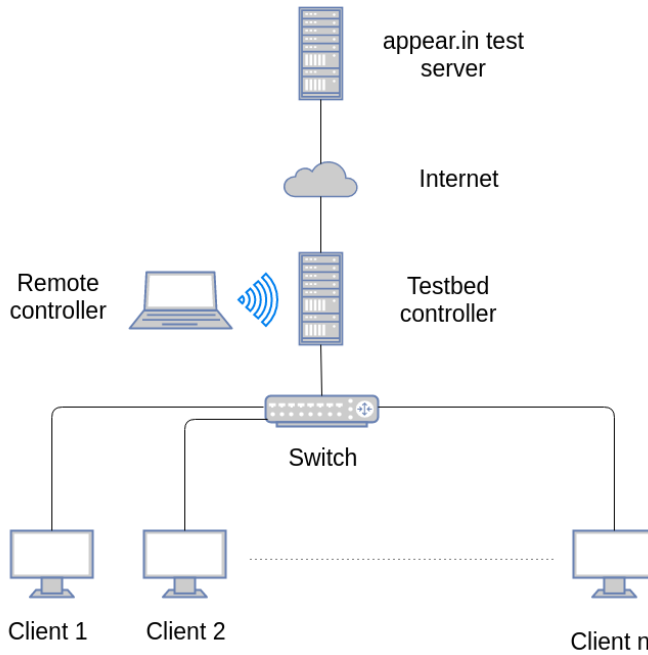


Figure 3.7: Testbed topology for up to n clients.

3.2.2 Hardware and Software Details

Desktop Specifications All the desktop computers used in the testbed, clients and controller, have the same specifications:

Computer HP Compaq Elite 8100 SFF

Processor Intel® Core™i7 CPU 860 @ 2.80GHz x 4

Memory 2 x DIMM DDR3 Synchronous 1333 MHz - 2GiB

The clients were configured with *Ubuntu 14.04*, and the controller was configured with *Debian 8.3*. Debian was chosen for the controller due to performance issues with NetEm in Ubuntu, which is further elaborated in the next chapter.

Switch The switch used in the testbed was a D-Link Gigabit switch³, with the following specifications:

³<http://www.dlink.com/uk/en/business-solutions/switching/unmanaged-switches/desktop/dgs-1008d-8-port-10-100-1000mbps-gigabit-switch>

Interfaces 8 x 10/100/1000 Gigabit LAN ports

Data Transfer Rates

- Ethernet
 - Half Duplex 10Mbps
 - Full Duplex 20Mbps
- Fast Ethernet
 - Half Duplex 100Mbps
 - Full Duplex 200Mbps
- Gigabit Ethernet
 - Full Duplex 2000Mbps

Transmission Method Store and Forward

The additional equipment used for the experiments was as follows:

Headphones Koss SB45

Webcam Microsoft LifeCam Studio

Software Details *appear.in* can be used with several different browsers, for example Firefox, Opera and Chrome. We are restricted to using Chrome because we use a built-in function in Chrome to collect connection statistics.

3.2.3 Configuration

Configuration scripts were needed for the testbed to work properly. Following is an explanation of the setup required for the different entities.

Testbed Controller

The testbed controller was configured as a proxy by forwarding traffic between all clients. The IP-address of the controller was set to *10.0.0.1*. Sending all traffic via the controller made it possible to add restrictions from a single point on different network parameters on all the traffic from the different clients. The controller can be set up as a proxy for forwarding traffic from the interfaces connected to the switch. The necessary commands can be found in Appendix A.1.1.

We wanted all the traffic from one client to another to go via both the switch and the controller. The controller, however, will by default notice that the path going through it is unnecessarily long, and will automatically try to shorten down the route by sending a Internet Control Message Protocol (ICMP) redirect message. It was crucial that all the traffic between the clients went through the controller. The commands required to redirect all traffic through the controller is listed in Appendix A.1.2.

Clients

All clients in the testbed was configured to route all traffic through the testbed controller, achieved by modifying the routing table on each client. The following is an example of how to use the controller as a default gateway for all Internet traffic, and how to route traffic to two other clients with IP-addresses *10.0.0.3* and *10.0.0.4*. This configuration is done from the client with IP-address *10.0.0.2*.

```
$ ip route add default via 10.0.0.1
$ ip route add 10.0.0.3 via 10.0.0.1
$ ip route add 10.0.0.4 via 10.0.0.1
```

3.2.4 Optimal Network Conditions

It was important that the network conditions were optimal when no alterations were added to the network. That is, no packet losses, negligible delay and jitter, and high bandwidth. The optimal network conditions were essential to have a good reference point when conducting QoE experiments using the testbed, and to be able to compare results from different network conditions. It was also important to make sure that the actual network conditions did not impact the desired network conditions specified in the testbed.

Connecting the clients and the controller via a wireless connection would have made the testbed more mobile and easier to configure than using physical links and a switch. But since the setup done on each client is minimal, we decided to use a wired connection between the entities in the testbed, because a wired connection is more reliable than a wireless connection.

Data we have gathered shows that the RTT of a two-party conversation with optimal network conditions had an average of 2.1 milliseconds which is negligible in this context. The packet loss under optimal network conditions is measured to 0%. We can see from Figure 3.8 and 3.9 that the throughput for video is stable around 1.7 Mbps and the throughput for audio is stable around 42 Kilobits per second (Kbps) for optimal network conditions.

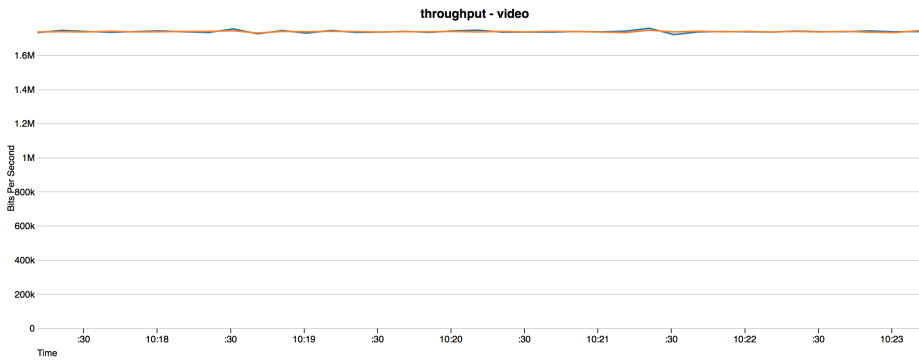


Figure 3.8: Throughput for video in optimal conditions (from *getstats.io*).

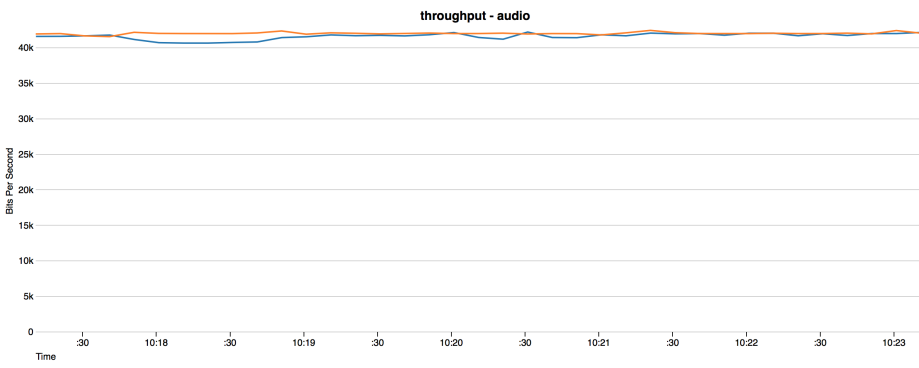


Figure 3.9: Throughput for audio in optimal conditions (from *getstats.io*).

3.2.5 Synchronizing Machines in the Local Network

We experienced that the system clock on the different clients was not synchronized even though they were set to retrieve the time automatically from the Internet. The lack of synchronization led to problems when analyzing the statistics from an *appear.in*-session. Even though the conversation was started simultaneously on different clients, statistical data from *getstats.io* showed that the curves were shifted and plotted according to their local clock. Figure 3.10 illustrates the shift (of ≈ 10 seconds) in the graphs for video latency. The same shifting applies to all the other graphs in *getstats.io* as well.

The Linux program *Network Time Protocol (NTP)*⁴ is a TCP/IP protocol for synchronizing time in a network. NTP requires that one machine in the local

⁴<https://help.ubuntu.com/lts/serverguide/NTP.html>

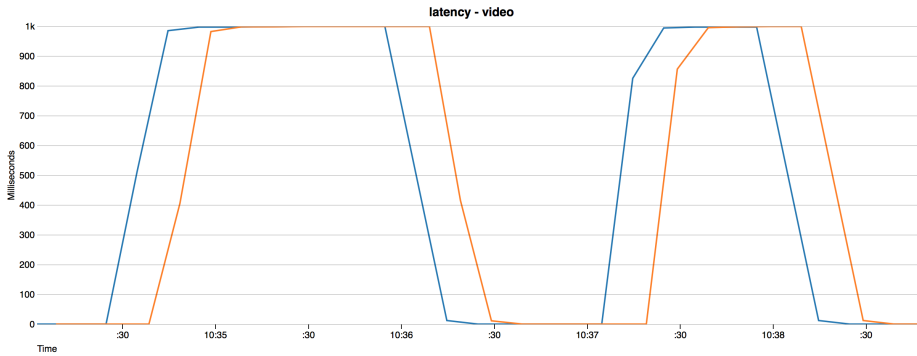


Figure 3.10: Illustration of not synchronized graphs before use of NTP (from *getstats.io*).

network is set up as the time server while all other machines retrieve the time from this machine. We used our testbed controller as the time server, and after that synchronized all the clients by requesting the time from the controller.

NTP is "designed to mitigate the effects of variable network latency and can usually maintain time to within tens of milliseconds over the public Internet. The accuracy on local area networks is even better, up to one millisecond"⁵. The configuration required to set up NTP is minimal, and leads to easier analyzing of the results in *getstats.io*, and was therefore used for all our experiments.

We experienced no further issues regarding shifts in the graphs after using NTP.

3.3 Acquiring Data from *appear.in*-sessions

There are several possible ways of collecting information regarding the connection quality from an *appear.in*-session. This section gives a brief overview of the tools we have used. We differ between session-related technical statistics, which is using the tools *getstats.io* and Chrome's *webrtc-internals*, and user feedback gathered from forms filled out by actual users. We also recorded the audio and the video of all conversations for analysis purposes.

The session-related statistics are mainly used to verify that our testbed reacts when we run scripts to manipulate the connection. User feedback is a useful way to see how users respond to the different network alterations, and the most important aspect of the QoE. Screen recordings are mostly necessary for analyzing all the data

⁵https://wiki.archlinux.org/index.php/Network_Time_Protocol_daemon

in retrospect. All these three tools are useful in both verifying and analyzing the data from a session, and is, therefore, important to use in combination with the testbed.

3.3.1 Session-related Statistics

Both *getstats.io* and Chrome's *webrtc-internals* use the WebRTC "getStats" Application Programming Interface (API) (not to be confused with the webpage *getstats.io*), which offers a set of methods for downloading peer connection statistics. Both tools show statistics real-time, and offer to download all the info as a dump file. Most of the statistics collected by the API are gathered from received RTP and RTCP packets. Since *getstats.io* and *webrtc-internals* use the same API to collect data. They, therefore, hold all the same information initially but choose to present their data slightly differently. For this reason, it is necessary to use them both. Following is a discussion on how they differ.

getstats.io

*Getstats.io*⁶ generates one statistic overview per session, collecting data about each user. *Getstats.io* offers information about the following parameters, presented in the user interface:

- The time the conversation started and how long it lasted.
- Type of web browser for all clients.
- The geographic location of all clients.
- Events such as client connecting, client disconnecting, microphone and camera on and off are illustrated for all users, marked with a time stamp.
- CPU-usage for all clients.
- bandwidth limited resolution for all clients.
- Latency, throughput, packet loss and jitter are plotted for all clients with two graphs: one for audio and one for video.

The user interface allows the user to zoom in on different sections of the conversation. It is also possible to look at only a subset of the participants at a time, which can be useful for finding details on specific users. A JavaScript Object Notation (JSON)-object containing all data used to plot the graphs can be downloaded at any time. The JSON-object also reveals information about the IP-address and what Operating System (OS) is used by each client.

⁶<https://getstats.io>

Access to the data in *getstats.io* is restricted, and we are not allowed to look at session statistics for the *appear.in* production version for privacy reasons. Throughout this project, we have therefore used a test version of *appear.in* which is located at NTNU's servers. Our access in *getstats.io* is limited to the conversations made on the *appear.in* test server⁷.

webrtc-internals

Chrome offers a user interface for gathering statistics during a WebRTC-call, named *webrtc-internals*⁸, as can be seen in Figure 3.11. The user interface provides a lot of different data and also separates the audio and video statistics. Some of the graphs, however, are hard to interpret and understand and therefore not so useful. Some of the information offered by *webrtc-internals* is the following:

- Bits sent per second
- Packets lost
- Packets sent per second
- Encoding data
- Information about the frame rate
- RTT

webrtc-internals also offers to download a statistics file which includes session metadata and real-time updates about delays, packet loss, and several more parameters for both audio and video. This file is, however, limited to 1000 samples, sampling every second. If the file exceeds 1000 samples, it will replace the oldest samples, and hold the 1000 most recent samples. The limitation in the number of samples limits *webrtc-internals* as a statistics gatherer, as the dump file needs to be downloaded in the middle of the conversation if it is to last longer than ≈ 16 minutes. The log file must be downloaded while the conversation is active, as it is not possible to retrieve at a later instance. *Webrtc-internals* only gathers information about the local computer, and therefore has to be downloaded separately for each client.

Using *webrtc-internals* offers challenges to the user as it is poorly documented. Some of the graph names are cryptic and difficult to understand, and several of the axes are without designation. Some of the graphs, such as *packetsLost*, are presented in the accumulated value of packets lost, instead of showing packet loss percentages.

⁷Test server accessible at www.appear.item.ntnu.no

⁸<chrome://webrtc-internals/>, accessible from Chrome web browser



Figure 3.11: Screenshot of all graphs for sending video (from *webrtc-internals*).

The accumulated value can be used for retrospect analyses purposes, but is less convenient when looking at the graphs in real-time. See [18] for a study on the limitations of *webrtc-internals* and the consequences they lead to.

Sampling Times of *getstats.io* and *webrtc-internals*

The JSON-object downloadable from *getstats.io* contains all statistical data accompanied by a time stamp. Samples are made every 10 seconds in *getstats.io*.

Webrtc-internals does not include any time stamps in the dump file. It does not even say anything about how often samples are collected. Our observations indicated that it samples once every second, but [18] indicated that *webrtc-internals* sometimes samples at uneven time intervals, and we wanted to be certain of the actual sampling time. *webrtc-internals* does not have any time information in the file, but each sample value is put in different lists, one list for each network property. We can, therefore, check the number of elements in a list after a given amount of time. Since the dump file is limited to 1000 samples, we conducted several conversations lasting 14 minutes (well under the limit of 1000 samples). We then checked the number of elements in the list, to see if it varied between the different conversations. The results were that a sample was taken every 0.9964 seconds on all three conversations. These variations are so small that they can come from small delays when starting

Table 3.1: Comparing properties in *getstats.io* and *webrtc-internals*.

	getstats.io	webrtc-internals
OS information	yes	no
Device information	yes	no
Information about connection type	yes	no
Browser information	yes	no
Start time, end time, duration	yes	yes
Information from conversation	for all users in session	only for single user
Events (connect, disconnect, video on/off, microphone on/off)	yes	yes
CPU-usage	yes	no
Bandwidth	yes	yes
Latency	yes	yes
Throughput	yes	yes
Packet loss	yes	yes
Jitter	yes	yes
Sampling period	every 10 seconds	every 1 second
Download data from conversation	from whole conversation	limited to 16 minutes
Downloading connection stats	automatically and at any time	manually and only when conversation is active

up the application or when downloading the dump file at the end. We, therefore, conclude that the sampling time of *webrtc-internals* is actually one second, and that we did not experience the same problems as mentioned in [18].

The fact that the sampling period in *getstats.io* is so long may lead to situations where large-valued short-lasting deviations are not accounted for at all. This is an example of where it is useful to use *webrtc-internals* instead. We will use both *getstats.io* and *webrtc-internals* to produce graphs in this report. We have used the JSON-object from *webrtc-internals* to retrieve the statistics, and then plotted them ourselves. Graphs from *getstats.io* are screenshots directly from the user interface.

Comparing Session-related Statistics

Getstats.io and *webrtc-internals* offer much of the same information, through slightly different user interfaces. It seems that *getstats.io* provides a more user-friendly interface, where the graphs are more easily interpreted than in *webrtc-internals*. Table 3.1 compares the two, to see what information they offer.

webrtc-internals and *getstats.io* should produce many of the same results, given that they both collect information using the same API. However, they are both interesting because they offer different perspectives. Statistics gathered can then be analyzed and compared across the two tools. Ideally, individual user comments can be connected with peaks visualized in the graphs, so that we can link these observations with our network parameters, linking QoE and QoS together.

Getstats API - Recording Jitter

Obtaining strange recordings of jitter values from *getstats.io* motivated an investigation to figure out what kind of data is computed and gathered in terms of jitter. Analyzing the JSON-files collected by *webrtc-internals* from a session revealed that three different parameters were collected with regards to jitter: *googJitterBufferMs*, *googPreferredJitterBufferMs* and *googJitterReceived*. Another observation made from the JSON files was that all three parameters were collected for audio, but only the *googJitterBufferMs* was collected for video. No documentation could be found about the three parameters, so an examination of the source code of WebRTC [15] was done to understand what the parameters represented.

googJitterBufferMs The source code showed that the parameter *googJitterBufferMs* describes the size of the jitter buffer for both video and audio.

googPreferredJitterBufferMs This parameter indicates the preferred jitter buffer size, which is indicated as the optimal buffer size for each sample.

googJitterReceived Only present in the data gathered for audio in a session, *googJitterReceived* contains information about jitter value of RTP packets, collected from the RTCP statistics, sampled every second.

3.3.2 Screen Recordings

A screen recorder was used during the tests to record the video and audio for the duration of the pilot study. The screen recordings were useful when looking at the user feedback from the pilot study, and to see what users considered as acceptable or poor quality. It was also helpful to use the screen recordings when the session-related statistics were analyzed in hindsight. The tool used for the screen recordings was *SimpleScreenRecorder*⁹, available in most Linux distributions.

⁹SimpleScreenRecorder: <http://www.maartenbaert.be/simplescreenrecorder/>

3.4 Questionnaires and Subjective Measurements

We wanted to collect information about users and the feedback from the different conversations in a quantifiable way. Concrete feedback is much easier to discuss, rather than having users describe freely how their QoE was. The questionnaires were inspired from previous studies conducted at the Department of Telematics at NTNU.

Several statistical method should be used to analyze the results. The certainty of the results and the ability to come with any useful conclusions is heavily dependent on the sample size, N , which in our project corresponds to the number of users. Martínéz-Mesa et al. discuss how the probability of error on the results are reduced as the sample size increases [16]. An estimate of the error margin for obtaining a 95% confidence interval is suggested as $\frac{1}{\sqrt{N}}$ by [9].

Our pilot study consisted of a total of 12 users, which indicates that we have an error margin of $\approx 29\%$. The large error margin means that we should not be too conclusive when looking at the data we have gathered from the participants, as there is a relatively large error probability. We could, however, use the data to get an idea about the results as it is still useful to look at correlations, and see whether it matches our hypothesis. A much larger sample size is needed to reduce the error probability ($N = 100$ gives $p_{error} = 10\%$) and is therefore too time-consuming for our study. We choose to use our testbed and the feedback gathered as a proof-of-concept, and could, therefore, allow a higher error probability, as the goal is not to draw any definite conclusions.

3.4.1 Pre-session Survey

We divided the pre-session survey into three different parts:

- The first three questions are about the participants familiarity with video communication tools in general. The participants familiarity gives us an insight into what kind of applications they know, and how often they have used these applications recently. They were also asked to quantify how important they regard video and audio quality and the audio-video synchronization on a scale from "Very unimportant" to "Very important".
- The next three questions were aimed specifically at the participants familiarity with *appear.in*. We wanted to get an idea on how often and how many times they have used the application, and rate how their previous experiences with *appear.in* have been.

- The remaining questions are demographic characteristics, which we could use in our analysis, and included age, gender, profession and the participants knowledge of the field of video and audio communication applications.

The complete questionnaire can be found in Appendix D.1

3.4.2 Appear-in Feedback After Each Conversation

The *appear.in* feedback form was presented to the users after a conversation when they pressed "Leave" within the ongoing conversation. They were asked to rate the overall audiovisual quality, video quality and audio quality on a scale from "Excellent" (5) to "Bad" (1). They were also asked to describe which quality-related issues they experienced. Finally, the participants were asked if they would consider quitting the session because of quality-related issues, and if they perceived any reduction in their ability to interact with the other party. The session feedback form can be found in Appendix D.2.

3.5 Background for Pilot Study

Before conducting the pilot study, the number and type of network scenarios for the pilot study were prepared. The testbed is configured such that a wide range of parameters can be configured to alter the network properties and allows up to seven clients to work simultaneously. We had to make a decision on the number of scenarios that should be tested, the number of simultaneous parties, and what network parameters that should be tested. The pilot study was limited to 12 people, and this put restrictions on the number of scenarios that could be applied. Many different parameters with a broad range of values is clearly not feasible for 12 participants.

Given the number of participants, we decided to conduct the study with two-party conversations only. Further, to get more data from each scenario applied, we also decided to use synchronous links between each party. Synchronous links mean that we use the same network alterations from client 1 to client 2, as from client 2 to client 1. We decided that each pair of participants should go through a total of five different scenarios. The time of the entire pilot study, with introductory phase and conversations with feedback, would be limited to at most 60 minutes per pair. We chose to have each survival task (conversation) last for 4 minutes leaving enough time for preparation and enough time for the participants to answer the surveys. Out of the 4 minutes, we chose to have the first and last 30 seconds to have no network alteration, as shown in Figure 3.12. We did this to allow the participants to get settled in the beginning, and we had experienced that it took about 30 seconds for the impact the network alterations had to wear off.

We designed the scenarios to look at the impact MLBS has on the QoE. We used the loss models to create scenarios with different packet loss rates and MLBSs.

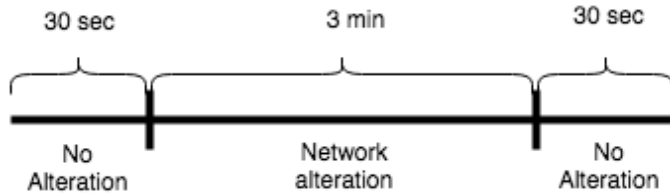


Figure 3.12: Timeline for each survival task in the pilot study.

3.5.1 Participants

This pilot study was conducted with a total of 12 participants. All of the participants are connected to the Telematics Department at NTNU: eight were students, and four were employees. Two of the participants were female, and ten were male. They were all between 21 and 35 years old. Two of the participants noted that they work or study in the field of audio/video quality, multimedia processing or a related field.

It was relevant to the experiment that the participants did not sit in the same room during the conversations to make it as realistic as possible. We controlled the testbed remotely from a separate location, and we were only present in the rooms with the participants during the setup and collection of data in between the sessions, to avoid interfering with the study.

3.5.2 Discussion Topics

It is important to simulate a realistic situation when conducting studies on user experience. A realistic situation is typically that all participants of the conversations are active at some point and as natural conversation flow as possible. As many of the participants did not know each other, we needed to make sure that the conversation was going so that they could have some meaningful contributions to the quality of the video conversation. It was important that the discussion tasks were easy and quick to learn and that they simulated a real-life conversation. It was also important that the tasks did not take the participants' attention away from the video conversation. Several possible discussion topics with different characteristics are proposed by ITU-T [30]. ITU-T recommends the *survival task*, as it offers the best compromise between "visual attention, number of speech turns, naturalness, and satisfaction".

The survival task is created so that 2-3 participants have to agree on a decision-making problem. For each survival task, the participants have presented a survival situation based on an accident as well as a list of objects to help them survive. The

participants each has different lists of 5-7 objects, adding up to a total number of 15 objects. They have to present their objects and cooperate with the other participants to choose at most six objects they would pick in order to survive. This task is beneficial because it is easy to learn, and requires that the participants will justify their choices through conversation.

The survival task would not, however, blindly be the best discussion task for all kinds of experiments. The most important factor of the survival task is the audio quality: It is possible to complete the task in a satisfactory way as long as the audio quality is sufficient. Video problems can, however, be harder to notice when the discussion task does not require any physical movement during the conversation. A participants movement on the screen can be referred to as the Quantity of Movement (QoM), indicating how much of the frame that needs to be rendered for each image. A setting with two participants sitting still does not have to render the image from scratch each time, and will suffer less from packet losses than for example sports events with a high QoM. Situations with a low QoM may, therefore, not experience the same problems as situations with a high QoM. We found that it is important to be aware of the QoM, but that it was not as important for our experiments, as the QoM is fairly low, and that the survival task is, therefore, suitable for our experiments.

We used a total number of five survival tasks for our pilot study, where each survival task corresponds to one accident location. For a full description of the different survival contexts and object lists, see Appendix C.

Chapter 4

Experiments

This chapter covers the different experiments we have conducted during our lab work. A validation of the tools we have used is followed by how we determined which loss model to use. We then describe the early-phase experiments using the testbed, before describing the pilot study. We finalize the chapter by describing the three-party conversations experiment.

4.1 Validation

Both network and hardware parameters may affect the connection quality of a video call. This section discusses the different parameters we can alter, how they are validated to assure correctness, and how we have gathered information from the video conversations.

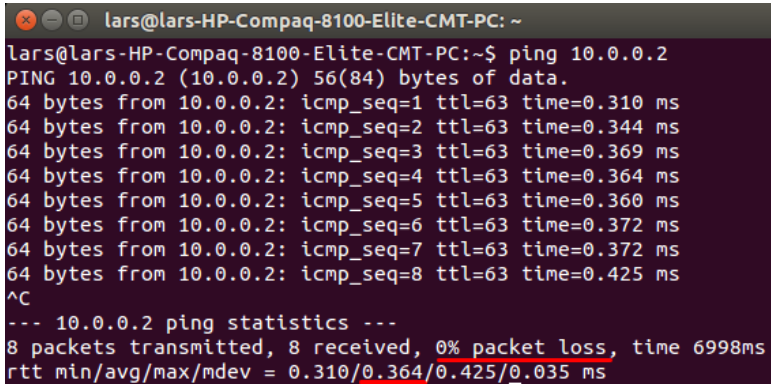
4.1.1 Validation Tools

A testbed is used to create a fully controllable environment where it is possible to recreate and reproduce experiments. The concept of QoE entails many variables. It is therefore especially important that the parameters we can control are thoroughly validated, to understand and manipulate the system in a controlled manner. The different network parameters we have altered by the testbed requires different validation tools to ensure everything is working correctly. The following includes a brief overview of the tools we have used.

Ping

Ping sends ICMP echo request packets to the specified destination and waits for the response packet. Unless specified, Ping sends one request packet each second, but can be configured to send packets much more frequently. Ping can be useful to check your Internet connection and to gather information quickly about the average RTT (which gives an indication of delay and jitter) and the packet loss percentage, as both

are highlighted in Figure 4.1. We used ping mainly as a first step to quickly verify commands regarding delay and packet loss issued through NetEm. The MLBS was also calculated by analyzing the results of a ping command, counting packet losses and calculating the mean length of a burst.



```

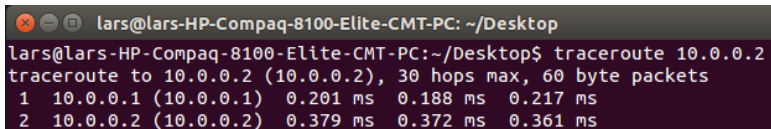
lars@lars-HP-Compaq-8100-Elite-CMT-PC: ~
lars@lars-HP-Compaq-8100-Elite-CMT-PC:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=63 time=0.310 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=63 time=0.344 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=63 time=0.369 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=63 time=0.364 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=63 time=0.360 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=63 time=0.372 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=63 time=0.372 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=63 time=0.425 ms
^C
--- 10.0.0.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 6998ms
rtt min/avg/max/mdev = 0.310/0.364/0.425/0.035 ms

```

Figure 4.1: Example of ping request, displaying RTT and packet loss percentage.

Traceroute

Traceroute is similar to ping in the way that it retrieves the measured delay. Traceroute also returns one line for each access point along the way the packet traverses from its origin to destination, which can be an effective way for troubleshooting a network connection. The illustration in Figure 4.2 shows that the packet is first transmitted to *10.0.0.1* (the testbed controller) before it reaches its final destination at *10.0.0.2* (client 1). We used Traceroute to verify that the routing tables of our clients were set up correctly, ensuring that all traffic goes through the controller. This traceroute is relatively small because it only finds access points within our LAN. A traceroute to an external website, such as *www.google.com*, would result in a much larger traceroute.



```

lars@lars-HP-Compaq-8100-Elite-CMT-PC: ~/Desktop
lars@lars-HP-Compaq-8100-Elite-CMT-PC:~/Desktop$ traceroute 10.0.0.2
traceroute to 10.0.0.2 (10.0.0.2), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.201 ms  0.188 ms  0.217 ms
 2  10.0.0.2 (10.0.0.2)  0.379 ms  0.372 ms  0.361 ms

```

Figure 4.2: Traceroute showing that traffic between clients goes through the testbed controller.

iPerf

iPerf¹ offers simple commands for active measurements of a bandwidth of an IP-network, which is useful to verify and validate that bandwidth throttling works. iPerf is also useful for recording packet loss and jitter in the network and can use both UDP and TCP.

The following command sets up a server which can receive UDP-packets and is scheduled to report the estimated bandwidth, packet loss, and jitter, every 20 seconds.

```
$ iperf -s -u -i 20
```

The following command creates a client which connects to the specified IP-address for a total of 60 seconds.

```
$ iperf -c -u 10.0.0.2 -t 60
```

We used iPerf to validate bandwidth, packet loss, and jitter alterations.

4.1.2 Validation of Packet Loss Function in NetEm

We discovered some performance issues for some of the functions in NetEm. The *random*-function was working as expected when specifying independent packet losses without any correlation between lost packets. Introducing *correlation* with the packet loss, however, drastically decreased performance, especially for higher correlation values. We noticed the reduction in performance by detecting that the actual percentage of lost packets was much lower than specified. Similar observations were also detected in [35]. This incorrect behavior introduced from the correlation values was problematic, because it meant we had no good way of testing our correlated packet losses.

NetEm is dependent on the Linux kernel which affects the performance of the functions. Problems occurred with both the *state* and the *gemodel* functions when using an Ubuntu distribution of Linux. The problems occurred either by a considerable decrease in performance as with the *random* function, but also by the failure to start the tool altogether. As a step to mitigate the problems encountered, a different Linux distribution, Debian, with a different kernel was tested. Using a Debian distribution instead proved useful, and both the *state* and the *gemodel* was working as expected.

¹iperf.fr

The *random* function with correlation, however, had the same performance issues in Debian as in Ubuntu.

For this reason, we chose to use Debian for the testbed controller and the *gemodel* for modeling losses. Ubuntu was still used for the clients, as they were unaffected by this problem.

4.1.3 Validating Delay and Jitter Functions

Due to the limited time of this project, our main priority was conducting experiments regarding packet loss and MLBS. Our main focus was, therefore, validating and experimenting with NetEm packet loss functions. We have, however, done some testing of functions regarding delay and jitter in NetEm.

Delay without any jitter specified was working as expected and did not suffer any performance issues. We did, however, experience some performance issues when specifying both delay and jitter, but not as severely as the issues experienced with packet loss and correlation. Some further testing revealed something strange about the delay and jitter functions in NetEm. We found that specifying both delay and jitter, with the *normal* distribution, provided better performance with respect to the average jitter that was applied to the network, compared to only specifying delay and jitter. This was strange because the documentation for NetEm [7] says that the default distribution is the *normal* distribution when no distribution is specified. The testing conducted with *iPerf* did, however, reveal a difference in performance of the delay and jitter function, with and without, the normal distribution specified. We, therefore, chose to specify the *normal* distribution when applying delay and jitter in the experiments.

4.2 Choosing the Appropriate Loss Model

Four different loss models were described in chapter 3.1.4, the GE model and its three varieties, Bernoulli, Simple Gilbert and Gilbert, all implemented in NetEm in the *gemodel* function. We deemed the Bernoulli model too simple, and the GE model unnecessarily complicated for our experiments. We have, therefore, conducted experiments by using the SG model and the Gilbert model, to see which one would serve its purpose best in the experiments.

The experiments of choosing the appropriate loss model consisted of two clients, the switch, and the testbed controller. The setup is depicted in Figure 4.3. Ethernet connects the clients to the switch, which lets the controller set various parameters for the different network links. Client 1 is configured to transmit ping messages continuously to client 2. The controller is configured to vary the network parameters

between the two clients, to see how different values for the loss models affect the actual network connection.

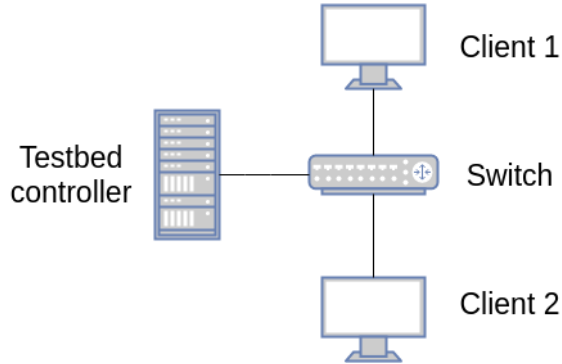


Figure 4.3: Ping simulation where client 1 continuously pings client 2, while the testbed controller alters the network link.

We needed to find out how we could create scripts with the desired values for both packet loss rate and MLBS. Figure 4.4 shows an overview of how different p and r -values in the SG model result in different values for packet loss and MLBS. A more detailed explanation on how to compute the MLBS follows.

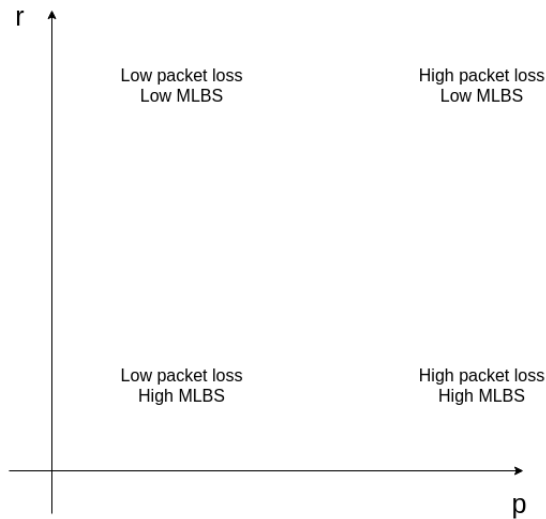


Figure 4.4: How MLBS and packet loss rate varies when p and r varies.

4.2.1 Computing the MLBS

Computation of the MLBS is less straightforward than the other network parameters. As mentioned earlier, the MLBS describes how many packets that are lost on average each time a packet is lost. The *getstats* API used by *webrtc-internals* gives samples indicating packets lost each second, but this is not sufficient to correctly compute the MLBS (*getstats.io* is obviously not better with a sample time of 10 seconds). Losing x number of packets within a second does not reveal the actual burst sizes, as a given number of packets lost in one second can either have come in one long burst or in many small bursts, resulting in different MLBSs. Our statistic tools can, therefore, not be used in hindsight to compute the average burst size.

Instead, we have to run *ping* between the clients while the network conditions are active and save all the ping messages to a text file. After that, we iterate through all entries in the text file and use the *ICMP sequence number* to identify packet losses. This gives us the burst sizes at the packet level which is what correctly depicts the MLBS.

4.2.2 Testing the Simple Gilbert Model

The SG model was run with $p = \{10,50,90\}$, and r -values ranging from $[25,100]$ with an interval of five (25,30,35,...,100). R -values below 25 are not included, as it results in higher MLBS than we intended to use for the experiments later on.

We decided to do a large number of iterations to flat out statistical peaks, and to obtain as correct values as possible. We ran through a total of 70 repetitions for each combination of p and r , each iteration lasting 20 seconds, and recorded the average packet loss and MLBSs. Figure 4.5 shows a plot from Matlab of the recorded MLBS for all combinations of p and r , clearly indicating that the MLBS is independent of the p -value, as the lines representing different p -values correspond closely.

4.2.3 Testing the Gilbert Model

As we showed that p is independent of the MLBS in Figure 4.5, we only tested one p -value for the Gilbert model. The Gilbert model, however, introduces the parameter $1-h$, which allows further fine tuning of the packet loss behavior, by adding the possibility of successful packet transmission in state B. We ran the simulations with $r = \{20,35,50\}$ and $1-h$ values $\in [0, 100]$. The simulations were executed with lower r -values than in the previous simulation because we wanted to see to what extent the $1-h$ value reduces the MLBS. As lower r -values give higher MLBS, the impact the $1-h$ value had on the MLBS would become clearer. The number of combinations to test was much larger, but we ran every scenario 50 iterations and recorded the average values for packet loss and MLBS. Each simulation was run for

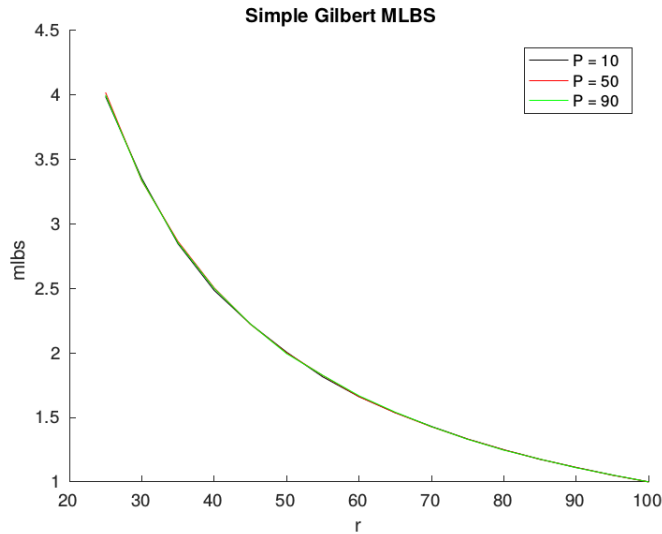


Figure 4.5: Simple Gilbert simulation results.

a total of 20 000 packets instead of running it for a given period, because significant packet loss rates would otherwise affect the total number of packets received and give potentially misleading results. Figure 4.6 shows a Matlab plot of the MLBS for different combinations of r and $1-h$.

Different r -values are illustrated by different graphs. We can easily see how smaller r -values result in a higher MLBS, and that a lower $1-h$ value reduces the MLBS. It is important to notice here that burst density $1-h$ value also will impact the total packet loss. The experiments with the Gilbert model were done to see if more accurate results on MLBS were seen than with the SG model.

4.2.4 Deciding on the Loss Model

We discovered throughout the experiments that the SG model falls short for high packet loss rates ($>50\%$) combined with low MLBSs. Intuitively this makes sense. For instance, consider a scenario with a packet stream of several thousand packets, and assuming a packet loss ratio of 80%. This situation is impossible with an MLBS of 1, as it would imply that every single lost packet is followed by at least one successfully transmitted packet, resulting in a packet loss percentage of at most 50%. Naturally, the same applies for the Gilbert model, even though we saw from the simulations that the $1-h$ value could be used to reduce the MLBS, this will also impact the total packet loss. The results from the two simulations showed that the

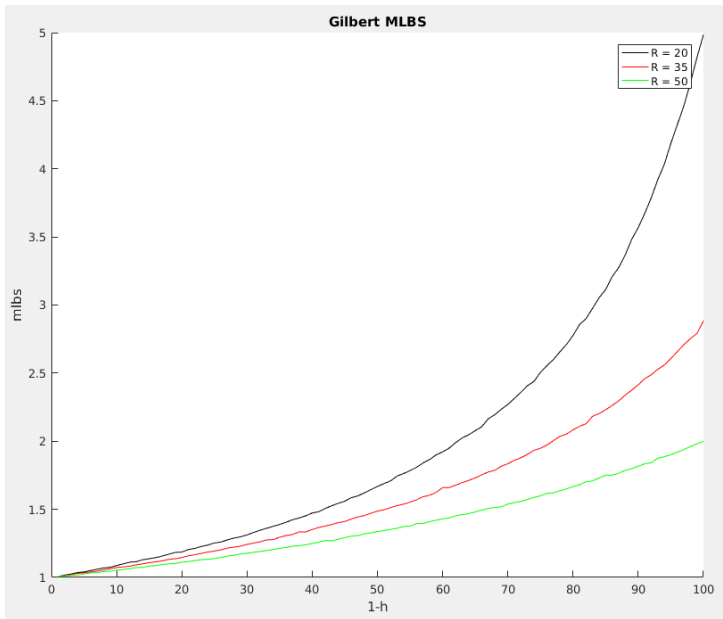


Figure 4.6: Gilbert simulation results.

two loss models had the same capabilities in applying packet loss rates and MLBSs needed for the experiments later in the project. The main difference between the two loss models, from its intended use in this project, is that the Gilbert model is more complicated to obtain the combination of parameters to get the expected packet loss rate and MLBS.

By the Ockham's razor theory, we have that: "Ockham's Razor belongs to the category of logical rules which indicate how to process experimental facts. It shows the way to the best fit of observables to the least complicated possible interpretation." [25]. We have therefore chosen to use the SG model, and not the Gilbert model.

The data gathered from the simulations with the SG model in NetEm, shown in Figure 4.5, was used as a reference point for selecting r -values for all subsequent experiments involving MLBS. Appendix B includes the data from the SG loss model simulations, as well as an example of how to use the p and r -values to get specific packet loss and MLBS combinations.

4.3 Experiments Using the Testbed Controller

Once all validations and verification were done, we could begin the actual experiments involving participant conversations. We needed to find the testing parameters to

use in the pilot study and later in the three-party conversations. Following is an explanation of how we picked these parameters.

4.3.1 Early-phase Testing to Decide Parameters

Initially, we needed to test a broad range of values before we could decide on parameters. We first tried all the parameters (bandwidth, packet loss rate, MLBS, delay, jitter, and CPU) one at a time with a broad range of values to get a picture of how they were working separately. We wanted to find values with significant differences within each parameter so that they could be perceived differently by users, which would be of value for our later experiments.

After getting an intuition on how the parameters worked separately, we started grouping parameters that are somewhat related. The packet loss rate was combined with the MLBS as they both model aspects of packet losses. We also combined delay with jitter. These parameters are tightly coupled as jitter describes the difference in the packet inter-arrival delay also known as the variation of the delay. As for the bandwidth and the CPU limitation, we experienced that they affected other parameters as they were introduced. The bandwidth led to an increased delay and packet loss rate while the CPU-usage limitation had an impact on both delay and jitter. For this reason, we did not want to combine bandwidth and CPU limitation with any other parameters. Following is a description of how we chose the different parameters.

Applying Bandwidth Limitations

We wanted to include some scenarios in the pilot study where we limited the bandwidth to fully utilize the functionality of the testbed. But we did not want to have dedicated scenarios where we only tested the limitations of bandwidth because the number of scenarios we could fit into the pilot study was already limited. We quickly found that introducing bandwidth limitation in combination with other parameters led to further difficulties for us when analyzing the data in retrospect. The reason for this is that lowering the bandwidth also impacts other network parameters, such as the delay and packet loss rate, which made it problematic for us to distinguish if the packet loss rate and the delay were due to the limitation of the specific parameters or due to the limitation in bandwidth.

We conducted some testing regarding limitation of bandwidth for all the network links in a conversation and found some indicators which could affect the QoE. We concluded that the throughput is stable around 1.7 - 1.8 Mbps for a two-party conversation with no alterations, and that we did not experience a significant reduction in the quality of the conversation before the throughput dropped to less than 1.0 Mbps. Further experiments revealed that a bandwidth of under 1 Mbps led to

increased delay for both audio and video. As the bandwidth was further reduced all the way down to 350 Kbps, we recorded an increasing delay from 2000 milliseconds up to 16 000 milliseconds. For values below 350 Kbps, our *appear.in* conversation suffered a timeout.

Even though we did detect a reduction in the quality of both video and audio when limiting the bandwidth, it only occurred when we severely limited the bandwidth. For reference: the average connection speed was 16.4 Mbps, and 88% of Internet subscribers had over 4 Mbps in Norway in 2015. Globally the average bandwidth was 5.1 Mbps [1].

We did not do any further experiments with bandwidth limitations.

Packet Loss Rate and MLBS

A packet loss rate describes the average number of packets lost over a given time interval. We have that the packet loss rate can behave differently from one scenario to another since the packet loss is run by statistical probabilities, as long as the total number of packets lost over time corresponds to the average packet loss rate. Introducing an MLBS can increase the statistical differences for two scenarios with the same packet loss rate and different MLBSs. A low MLBS will give small variations in the actual packet loss rate while a larger MLBS will create bigger fluctuations and have more significant deviations compared to the specified average.

For further experiments, we were interested in several values of both packet loss rate and MLBS. After testing a wide number of values for each, we found that we were able to distinguish between a packet loss rate of 10% and 20% and that they could offer differences in the number of video freezes and audio reductions.

Several testing sessions with different MLBSs combined with the different packet loss rates revealed that the perceived quality was noticeably degrading as the MLBS increased. We did not want the difference between the MLBSs to be huge, but enough to notice a change in the perceived quality. We ended up with choosing MLBSs equal to 1.5 and 3.

Delay and Jitter

After that we wanted to test different combinations of delay and jitter values. All real-life network connections have some delay, as the delay is directly dependent on the physical distance between the communicating clients. Due to our connection setup where all clients are connected directly to the switch, we have achieved a negligible delay of less than one millisecond. We wanted to see how it affected the connections when we substantially increased the delay.

Researching delay independently revealed that we did not perceive any real difference in the perceived quality for (one-way) delays < 500 milliseconds. Delays up to 1000 milliseconds did not disturb the conversation quality much, but delays exceeding 1000 led to annoyance when taking turns to talk and unintentionally speaking simultaneously. We set 500 and 1000 milliseconds as threshold values for the delay.

Jitter led to a clear difference in the perceived quality at 300 milliseconds. We experienced further reduction in the conversation quality for a jitter value equal to 500 milliseconds. We also chose not to alter the jitter value, so that we could see the effect of delay independently.

Note that a jitter value of 300 milliseconds added in combination with a delay of 500 milliseconds means that the delay will fluctuate in the range 200 to 800 milliseconds which is a quite broad range.

Deciding Parameter Combinations for Further Experiments

Because of the difficulty in combining bandwidth with other parameters, and the fact that the required bandwidth for maintaining a proper conversation is well below the facts from Akamai [1], we, therefore, chose to focus on other parameters than the bandwidth for the rest of the studies. Following are the parameters we did choose to focus on:

- The effects of limiting the CPU-usage is of particular interest because limiting CPU can be compared to using less powerful devices.
- The packet loss and MLBS parameters were chosen because they offer an interesting approach saying that the packet loss rate alone is not enough to determine the perceived QoE, as it can also vary with different MLBSs.
- Testing delay in combination with jitter is interesting because we find it difficult to understand the impact of jitter. For this reason, we have chosen to conduct further experiments with delay and jitter as well.

Due to the small number of participants and that the numbers of scenarios increase quickly as more parameters are included, we had to narrow down the parameters we would like to experiment with for the different experiments. For the two-party pilot study, we focused on how packet losses and MLBSs affect the network connections. As the three-party experiments were conducted by the two authors and our supervisor, it required less planning and administrative work. We were, therefore, able to test several other aspects of our testbed in addition to the pilot

study. The three-party conversations had a wider range of testing values, but less extensive testing with regards to the number of participants. The testing parameters for the three-party conversations were packet losses and MLBS, the effects of delay and jitter combinations, and finally, how restricting the CPU-usage influence the QoE.

4.3.2 Pilot Study

The pilot study focused on two-party conversations only. Each pair of participants conducted five different survival tasks, where four of the five survival tasks were manipulated with different network conditions and one had no network alterations. Because the number of possible scenarios to run was limited, we had to focus on only a subset of the parameters that we can control with our testbed. Our choice of parameters for the pilot study was therefore limited to packet loss and MLBS.

We randomized the order of the scenarios for all of the groups because we wanted to remove the fixed order from 1 to 5 which would go monotonically from (what we anticipated as) best to worst. Randomization is also interesting because we can see how the participants perceive the contrast of going from a bad network connection to the ideal scenario and back again to bad conditions. As mentioned earlier, the network alterations were applied synchronously between all participants. Even though asymmetric links may be a better approximation to the real world, we chose synchronous links because we wanted to reduce the number of factors that affect the user experience for analysis purposes. We, therefore, decided to use synchronous links due to the relatively small number of participants. Figure 4.7 illustrates the network topology for the pilot study.

Table 4.1 shows the parameters which are used in both the pilot study and the three-party conversations. Note that the first scenario does not have anything to do with packet loss and MLBS, as it is a no-alteration scenario for our reference. It is of interest to see how the users rate the ideal conditions and compare it to how they would rate poorer conditions. Our main hypothesis across the experiments was that a high MLBS greatly can affect the perceived user experience, and we wanted to investigate whether the participants were able to differ between combinations with the same packet loss rate but different values for the MLBS.

Our hypothesis is that a packet loss rate of 20% will be significantly worse than a packet loss rate of 10%. We also believe that a high MLBS will lead to a greater reduction of the perceived quality than a lower MLBS for both audio and video. The MLBS hypothesis comes from our testing from the early phase, and the fact that error correction techniques become less effective when consecutive packets are lost, as already discussed in section 3.1.2. A 10% increase in packet loss rate should be worse for obvious reasons. We are also interested in a potential difference between

Table 4.1: Scenarios applied in the pilot study.

Scenario	Packet Loss	MLBS
1	0	0
2	10	1.5
3	10	3
4	20	1.5
5	20	3

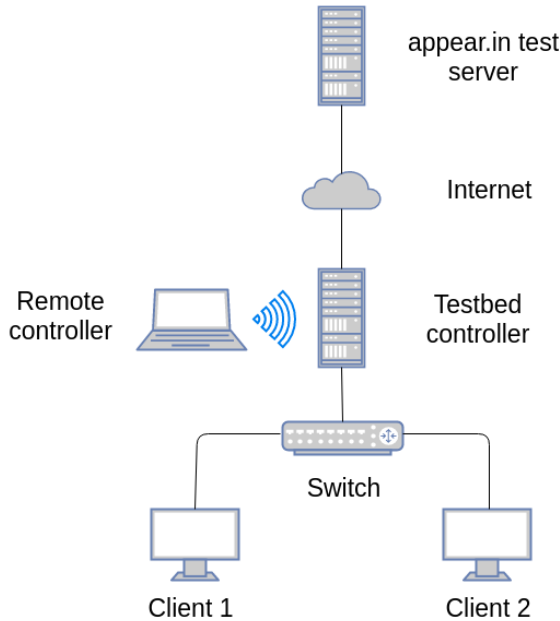


Figure 4.7: Network topology for two clients.

the two-party and three-party conversation, and to see if the perceived quality is affected by the number of simultaneous users.

Here we will present the exact numerical values we have put in the SG loss model for the different scenarios 2 through 5. Figure 4.8 depicts the values we used to create scenario 2. The transition probability (p) from state G to state B is 7.33%, and the transition probability (r) from state B to state G is 66%. Similarly, Figure 4.9, 4.10 and 4.11 show the SG loss model used for scenario 3, 4 and 5 respectively.

Recall that r controls the MLBS, and p is computed to give the desired packet loss based on the r -value. Note also that the transition p happens more frequently in Figure 4.8 than in Figure 4.9 but the time spent in state B will by probability be

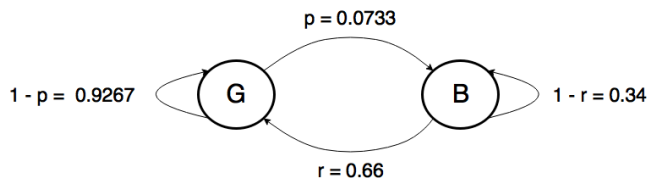


Figure 4.8: Simple Gilbert loss model for scenario 2 (10% loss rate and 1.5 MLBS).

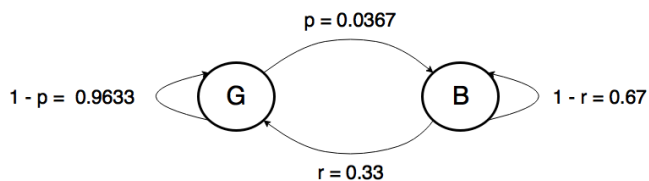


Figure 4.9: Simple Gilbert loss model for scenario 3 (10% loss rate and 3 MLBS).

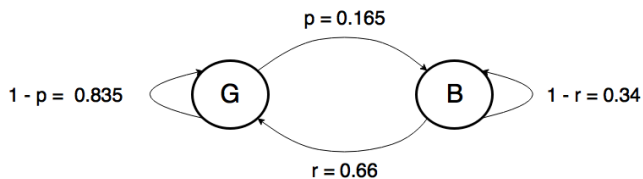


Figure 4.10: Simple Gilbert loss model for scenario 4 (20% loss rate and 1.5 MLBS).

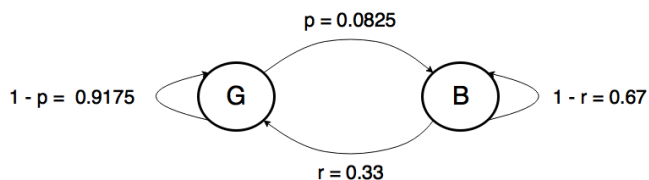


Figure 4.11: Simple Gilbert loss model for scenario 5 (20% loss rate and 3 MLBS).

shorter, hence a smaller MLBS. We can see from the models that the MLBS is only dependent on the r -value by comparing scenario 2 with scenario 4, and scenario 3 with scenario 5. The scripts run by the testbed controller uses the values we have presented here, and can be found in Appendix E.

4.3.3 Three-party Conversations

As mentioned, *appear.in* supports up to eight simultaneous users in the same session. Conducting experiments, however, becomes more challenging as the number of participants increases. Finding discussion topics that include all participants can also be problematic and requires a much longer time for each network scenario for all participants to be active. The most significant shortcoming, however, is the practical limitation. Our testbed setup requires that all the participants are connected to the same switch, and all participants are put in a separate room to make the setting as realistic as possible. Testing with a larger number of participants is possible but requires much more planning and other practical difficulties such as a separate room for each. The maximum number of users, however, is not what we aim to investigate in this report. We have therefore limited our multi-party experiments to three participants. The setup would have been the same for more simultaneous participants, but we choose to focus on the data from a smaller number of users.

Figure 4.12 illustrates the network topology for a three-party conversation in our testbed.

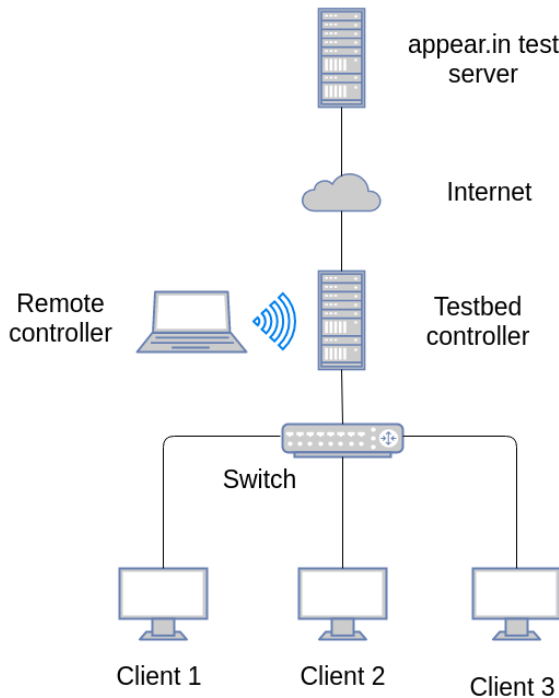


Figure 4.12: Network topology for three clients.

Packet Losses and MLBS

The parameters used for packet losses and MLBS were the same for these experiments as for the pilot study. Please refer to Table 4.1 for details about the different scenarios regarding packet losses and MLBSs.

Delay and Jitter

Table 4.2 holds the various combinations of delay and jitter used in the experiments regarding delay and jitter (scenarios 6-11).

Our hypothesis was that adding reasonable amounts (less than 1000 milliseconds) of delay would not affect the perceived quality, but could lead to annoyance because the flow of the conversation would be interrupted. The effects of adding network jitter were somewhat harder to predict. A high enough jitter value may cause enough packet reordering and packet losses to affect both video and audio quality negatively.

Table 4.2: Three-party conversation - delay and jitter values.

Scenario	Delay [ms]	Jitter [ms]
6	500	0
7	500	300
8	500	500
9	1000	0
10	1000	300
11	1000	500

Restricting CPU-usage

We noticed that a regular two-party conversation used about 70% - 80% of one CPU core. A three-party conversation was, however, clearly higher with values around 90% - 100%. A higher CPU-usage makes sense, as the computer needs to use more resources when processing more video and audio data. For testing purposes, we would decrease the limit gradually to see when we experienced a reduction in the quality.

The focus on limiting the CPU was on how the users would perceive the QoE as the CPU-usage decreased. An important difference compared to all other parameters is that the CPU limitations needs to be applied on the client side and for each single client participating in the conversation. The CPU limitation is the only parameter we cannot control centrally from the testbed controller.

The values used in the experiment for restricting the CPU-usage can be seen in Table 4.3, as scenario 12 and 13. The two different values were chosen so that we could clearly see the differences between them, where one was close to normal conditions and the other one way worse.

Table 4.3: Three-party conversation - restricting CPU-usage.

Scenario	CPU-usage [%]
12	80
13	60

Our hypothesis on restricting the CPU-limit is that both audio and video are substantially negatively affected, and will lead to a bad experience for all involved participants, even for small changes in the CPU-usage.

Chapter 5

Results and Discussion

This chapter covers the results we have obtained from the pilot study and the three-party conversations. The results are followed by a discussion concerning our findings. We begin the chapter by a comparison of the scenarios created by the testbed for the pilot study.

5.1 Comparison Between Different Scenarios

We wanted to ensure that the testbed was working correctly at the time of the pilot study before we looked at the feedback from the participants. We, therefore, conducted a statistical analysis of the packet losses, to verify that our scripts resulted in the specified network link values. As mentioned earlier, each scenario consisted of 30 seconds with no network alterations, followed by a three-minute period with network modifications, and ended with 30 seconds where all network alterations were removed. We use data only from the three-minute period in our discussion, as this is the time interval of interest with regards to the testbed. Table 5.1 shows the statistics calculated from the measured packet loss values in the pilot study for all participants combined. We see that average of the total packet loss rates are very close to the values we specified with the testbed, with a negligible deviation between the different scenarios. We can, therefore, conclude that the testbed was working as specified during the pilot study and that each participant experienced the same network conditions with respect to the total packet loss rate in each scenario.

As mentioned in Chapter 4.2, the MLBSs were computed actively when the network alterations were applied, and not with the data gathered by either *getstats.io* or *webrtc-internals*. Therefore, active testing was conducted with the pilot study setup to ensure that the MLBSs were as expected for each scenario.

Table 5.1: Actual packet loss rates for each scenario in the pilot study (all values in percentages).

	scen #1	scen #2	scen #3	scen #4	scen #5
Expected	0.0	10	10	20	20
Average	0.0	10.15	9.90	20.18	20.09
SD	0.0	0.47	0.58	0.63	0.82
Median	0.0	10.12	9.97	20.08	20.09

5.2 Results from the Pilot Study

After controlling the scenarios from the pilot study, we could start analyzing the results. Table 5.2 holds all the participant feedback, and the results from this table will be elaborated further in the following subsections where we discuss each scenario from the pilot study.

Table 5.2: Statistics from user feedback for all scenarios (ratings for average, median and mode are given from 1 to 5).

		scen #1	scen #2	scen #3	scen #4	scen #5
Average	Overall	4.50	3.64	3.55	3.09	2.75
	Video	4.42	2.50	3.36	2.09	2.33
	Audio	4.50	4.27	3.82	3.55	3.00
Variance	Overall	0.45	0.25	0.67	0.69	0.93
	Video	0.63	0.67	0.85	1.09	1.15
	Audio	0.45	0.42	0.36	0.87	1.27
Standard dev.	Overall	0.67	0.50	0.82	0.83	0.97
	Video	0.79	0.82	0.92	1.04	1.07
	Audio	0.67	0.65	0.60	0.93	1.13
Median	Overall	5	4	4	3	3
	Video	5	3	3	2	2
	Audio	5	4	4	3	3
Mode	Overall	5	4	4	3	3
	Video	5	3	4	2	2
	Audio	5	4	4	3	3

5.2.1 Scenario 1 (No network alterations)

The first scenario had no network alterations and was the highest rated scenario for both video, audio, and overall audiovisual quality. The majority of the participants

agreed that this was the best scenario with relatively small variations in the ratings. No apparent freezes were found in the screen recordings from scenario 1. The audio quality is stable, and the video is clear and without any blurriness.

5.2.2 Scenario 2 (PL = 10%, MLBS = 1.5)

The overall quality rating for scenario 2 was, as expected, lower than for the first scenario. Several participants noted that they experienced some reduction in the video quality. Some reported that the video suffers from blurring throughout the session, but no incidents of video freeze. The audio quality seems to be little impacted if any. These findings were supported by the video recordings.

5.2.3 Scenario 3 (PL = 10%, MLBS = 3.0)

Scenario 3 was rated higher than scenario 2 concerning the video quality but was rated lower for both audio and overall audiovisual quality. The screen recordings indicate that scenario 3 suffers less from blurring compared to scenario 2 for most of the duration. There are, however, smaller periods of time where the video quality is poor, but no apparent video freezes were found.

5.2.4 Scenario 4 (PL = 20%, MLBS = 1.5)

Scenario 4 was the scenario with the overall lowest rated video quality by the participants. Some complained about the video quality for most of the conversation and that they considered quitting the session. Most participants said that the audio quality was fair, but some experienced audio problems. Screen recordings support that the video quality was poor for most of the session and that there were three specific incidents where the video froze, lasting 2-3 seconds each. The audio quality seemed to have been relatively unaffected.

5.2.5 Scenario 5 (PL = 20%, MLBS = 3.0)

We assumed that scenario 5 would be the worst scenario due to the high packet loss rate and MLBS. The participants' overall rating of this scenario agreed, but the video quality was rated higher than scenario 4. Scenario 5 was, however, the scenario with the lowest audio quality rating, and where most participants noted they perceived a reduction in their ability to interact with the other party. Some participants described that the sound occasionally was lost and that some parts of the conversation had to be repeated due to unintelligible sound. The screen recordings show some blurriness throughout the entire session but appears to be less severe than in scenario 4. Only one incident of video freezes was recorded, lasting less than a second.

Figure 5.1 is a Gnuplot visualization of the same data as in Table 5.2. The points in Figure 5.1 represent the average rating for each scenario and the solid lines show the standard deviation. Note that for scenario 1, the upper limit of the standard deviation has been truncated, so it did not go above 5 (which was the maximum rating). The dotted lines are added to better visualize the differences between the scenarios with the same packet loss rates and different MLBSs.

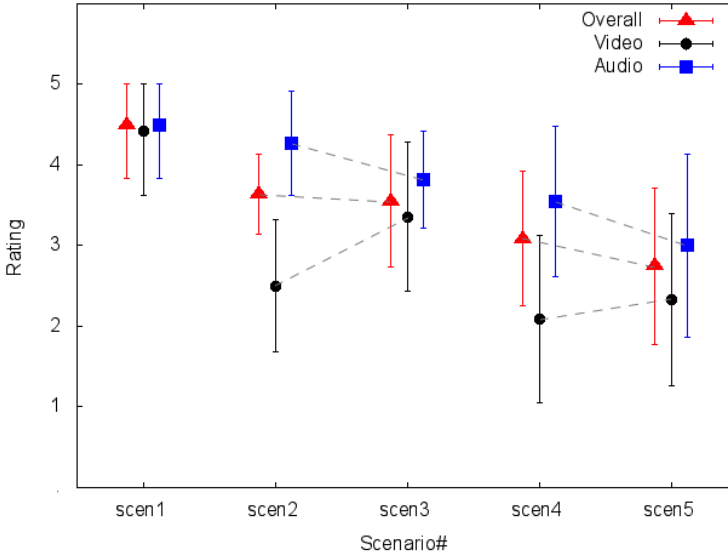


Figure 5.1: Visualization of user feedback, indicating the average value \pm one standard deviation.

5.3 Discussion of Pilot Study Results

This section discusses and analyzes the results from the pilot study. We will consider how the users were affected by the different network alterations and see the feedback across the various groups in the pilot study. We also compare the scenarios that have the same packet loss rate to see how the MLBS impacted the QoE.

The order of the scenarios was randomized between the different groups as mentioned earlier. Randomization was done so the network conditions should not monotonically decrease, but rather be experienced as variations throughout the study. The order of the scripts is likely to affect how participants perceive the different scenarios. If the worst scenario comes right after the scenario with no network alterations, the user is more likely to give a worse rating because he will compare the two most recent scenarios.

We have chosen to compare scenario 2 with scenario 3, and scenario 4 with scenario 5 when we discuss the scenarios in the pilot study, as these scenarios have the same packet loss rate. The reason why we chose to compare the scenarios with the same packet loss rates, and not the same MLBSs, is that our main focus was on finding the effect of different MLBSs. Also, our early-phase testing indicated that the difference between a packet loss rate of 10% and 20% was enormous, and that the participants could easily differ between these two. For this reason, we have not compared scenario 2 with scenario 4, and scenario 3 with scenario 5.

5.3.1 Interpreting Subjective User Feedback

Some users gave an overall high or low rating which deviated from the statistical mean. One user gave 3-3-3 (for overall audiovisual-, video- and audio rating respectively, from 1 to 5 where 5 is best) to the scenario with no network alterations. This user is likely to give a relatively low rating to all scenarios. The opposite goes for a user which consequently provides feedback which is higher than the statistical mean.

Another important factor is that the actual perceived quality of a single scenario may be different between the pilot study groups. Even though the network alterations are set to specific values, a different number of video freezes can be experienced by different users due to statistical variations. Users may therefore actually experience differences in the same scenarios, which makes analyzing the data even more challenging.

Finally, another factor is that the subjective user feedback is, in fact, subjective. What is considered acceptable differs from person to person, which can make interpretation of the results more challenging.

5.3.2 Comparing Low MLBS with High MLBS

Video The scenarios with a total packet loss of 20% were, as expected, rated lower with respect to video quality than the scenarios with a total packet loss of 10%. What was interesting was the difference the MLBS had on the video quality in each pair of scenarios with 10% packet loss and 20% packet loss. The analysis done after the pilot study, shown in Table 5.1, shows that scenario 2 and 3 had the same total packet loss rate of 10% and that scenario 4 and 5 had a total packet loss rate of 20%. However, the participants on average rated the scenario with the lower MLBS worse, with regards to video quality compared to the higher MLBS case. It seems, therefore, that the different values of MLBS did somehow impact the video quality.

It was difficult to see from the collected data why the lower MLBS gave the worst perceived video quality. For instance, the throughput (bits received per second) was equally low in both cases. Figure 5.2 and 5.3 show the packet loss rates for video in

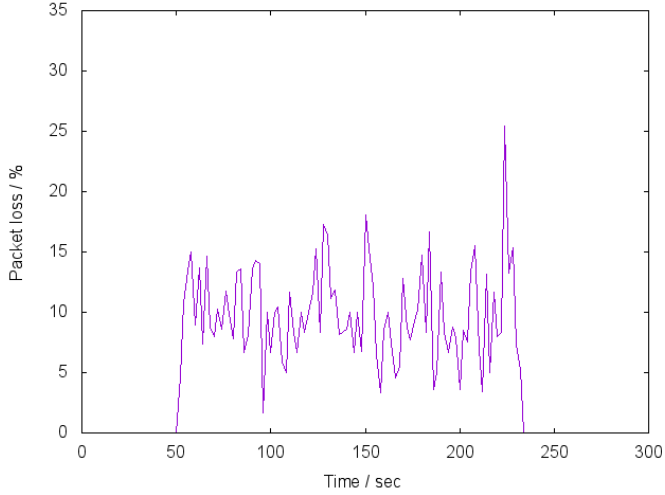


Figure 5.2: Packet loss video scenario 2 (10% loss rate and 1.5 MLBS.)

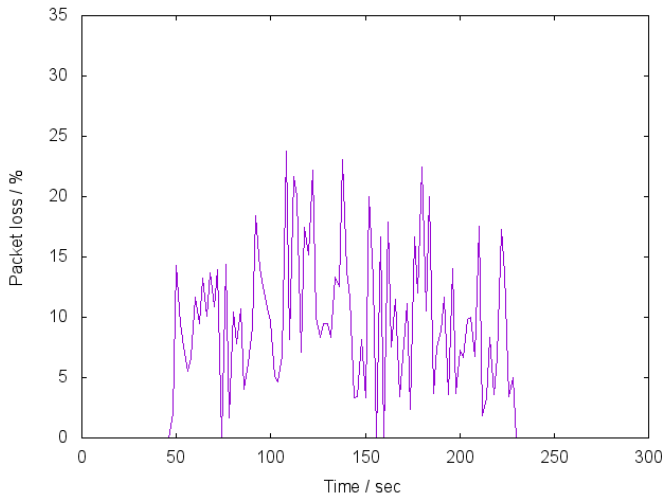


Figure 5.3: Packet loss video scenario 3 (10% loss rate and 3 MLBS)

a conversation in scenario 2 and scenario 3, respectively¹. Differences between these graphs can be seen in how the packet losses are distributed, where high MLBS leads to bigger fluctuations. One possible explanation for why the video quality was lower when the MLBS was lower could be that difference in how the packet losses were distributed during the conversation.

¹Figures 5.2- 5.5 were plotted with Gnuplot with the data collected from *webrtc-internals*.

Another way to look at this is with the figures from Chapter 4.3.2, which illustrate the transition probabilities between the states in the SG model used to apply packet loss rates in the network. The figures show that transition p happens more frequently in scenario 2 than in scenario 3, and more frequently in scenario 4 than in scenario 5. Since the transition p happens more frequently in scenario 2, this could lead to the video quality being relatively poor for the entire conversation. Similarly could the smaller number of transitions in scenario 3 lead to larger time intervals with fair conditions, and that this scenario therefore was perceived as the "better" one with regards to video.

The difference between the rating of the video is smaller between scenario 4 and scenario 5, than for scenario 2 and scenario 3 in the feedback from the participants of the pilot study. When experiments were conducted prior to the pilot study, we did notice this difference when applying packet losses. The difference in video quality between for instance 10% and 15% was clearer than for 20% and 25%. That is, the video was already so blurry and suffering from freezes at 20%, that the increase to 25% packet loss was not that prominent. Similarly, it is not unlikely that the impact the different MLBSs had on 20% packet loss, was less apparent than with the 10% packet loss, especially with the relatively small difference of 1.5 in the MLBS.

Audio It seems that audio the quality was less affected than the video quality for the pilot study in general. However, the scenarios with the most complaints and lowest rating regarding audio quality were the scenarios with the higher MLBS, which is the opposite of what the feedback on the video quality indicated. The audio quality feedback corresponds better to the hypothesis we made prior to the pilot study, that the quality would decrease as the MLBS increases.

These differences in audio quality due to the higher MLBS could be because of more consecutive lost packets which lead to less efficient error concealment techniques. It is not apparent why audio seemed more impacted than video by higher MLBS as WebRTC uses FEC for both. Part of the reasons for the recorded variations could be related to differences in error correction techniques utilized in WebRTC between audio and video. Error correction techniques for video in WebRTC are discussed in [26]. Video in WebRTC uses multi-frame FEC for mitigating the effects of burst losses, while audio only uses single-frame FEC². These differences could be an explanation for why audio appears to be more impacted than video for a larger MLBS.

Overall Quality Table 5.2 shows that the rating of the overall quality of the conversation decreased as both the packet loss and MLBS increased. This makes

²WebRTC Forward Error Correction Requirements draft-ietf-rtcweb-fec-03(Work in progress): <https://tools.ietf.org/html/draft-ietf-rtcweb-fec-03>

sense considering the scenarios that had the lowest perceived video and audio quality, compared to the recorded answers about the importance of video and audio quality presented in Table 5.3. Nine participants said that audio quality was *Very Important*, while only one participant said the same about video quality. As many as five participants said that they were *Neutral* when asked about the importance of video quality, while all considered the audio quality as either *Important* or *Very Important*. Considering these answers, it makes sense that the overall rating of the scenarios with the higher MLBS was lower than the scenarios with the lower MLBS, because the higher MLBS had the most issues regarding audio quality.

Table 5.3: How participants rated importance of different aspects of online video communications.

	Very unimportant	Unimportant	Neutral	Important	Very important
Audio quality	0	0	0	3	9
Video quality	0	0	5	6	1
Audio-video synchronization	0	0	0	5	7

5.3.3 Scenarios with High Deviation

Here we will take a closer look at some of the scenarios where the feedback deviated the most compared to the rest of the data. Ratings regarding video and audio quality had standard deviation values over 1 in both scenario 4 and scenario 5. It is not surprising that the data gathered from the participants had some deviation, as QoE is a subjective matter and the number of participants was limited. However, we decided to look closer at the situations that had much higher standard deviation values than the rest to see if it was possible to detect some particular differences.

Scenario 4 - Video Quality Out of the twelve persons participating in the pilot study, all but two gave scenario 4 a video rating of either 1 or 2. Two participants gave the video quality of scenario 4 a rating of 4. Changes in the video quality can be seen in the screen recordings from those two conversations. In both cases, the video was blurry more or less the entire duration of the network alterations. We saw three incidents of video freezes each lasting for about 1-2 seconds when analyzing the screen recordings. Further, the screen recordings from the two most deviating cases from scenario 4 was compared to the screen recording from another conversation with scenario 4 rated closest to the average. We found the same amount of video freezes with approximately the same duration regardless of the rating from the participants.

The data recorded of the packet losses from these conversations did not give any clear indications to why this scenario was rated so differently between the participants. When looking at the questionnaire the participants answered before the pilot study,

however, it shows that the participants giving the highest rating of the video quality of scenario 4 had responded that they were neutral to the importance of video quality in a video communication service. The participants who rated the video quality of scenario 4 as high also regards video quality as less important for a video communication service which might be one reason why the rating is deviating.

Scenario 5 - Video Quality We also noted some deviations with scenario 5. From the screen recordings, it appears that both participants who gave the worst rating had the same number of video freezes and that there were no apparent differences in the video compared to those who gave a high rating. Both the participants who rated the video quality of scenario 5 as *good*, also said in the pre-session questionnaire that they did not regard video quality very important in a video communication service. We also noticed that the video conversation for one group of the participants lasted longer than intended, such that scenario 5 was ended with several minutes of *perfect* conditions, which may have impacted the rating the participants gave for this specific scenario.

Scenario 5 - Audio Quality The feedback from the participants on the audio quality of scenario 5 had the highest standard deviation out of all the ratings. We took a closer look at the data from the maximum and minimum rated conversations. From the screen recordings, we found that a noticeable reduction in audio quality was detected for both the parties. The audio sounds metallic, and some small freezes in audio occur several times during the conversation in both situations. The freezes do not, however, last longer than approximately 0.5 seconds. Both the participant who rated the audio quality high and the participant who rated the audio quality as low had previously answered that audio quality was of great importance when using a video communication service.

5.3.4 Differences in Packet Loss Distribution

Differences in the distribution of packet losses occur with use of the SG model, as it is probabilistic. We have seen that the total packet loss rate has been the same in the different situations when the same scenarios were applied, but at the same time, individual differences in how the packet loss was distributed between different situations have been recorded. For instance Figure 5.4 and 5.5 show the packet loss for two different situations, but with the same network alterations (scenario 3).

Both situations have a total packet loss rate of 10%, but we can see clear differences in how the packet losses are distributed. It is not unexpected that such differences occur since packets are dropped or transmitted based on certain probabilities, and statistical variations are bound to happen. It is, however, important to be aware of such differences, because the distribution of packet losses might impact the overall

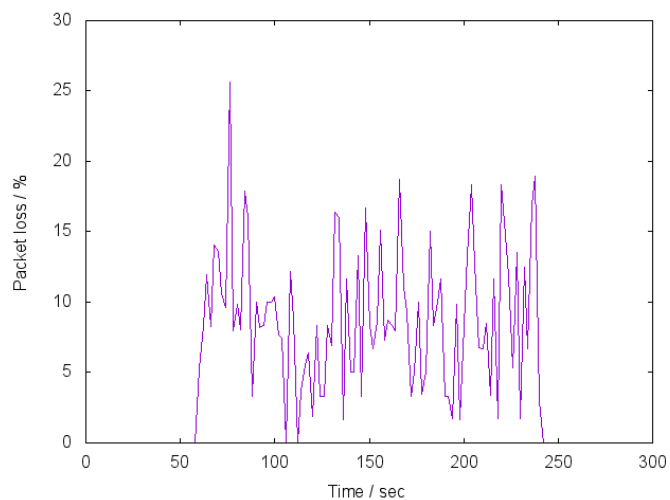


Figure 5.4: Situation 1 scenario 3 (10% loss rate and 3 MLBS)

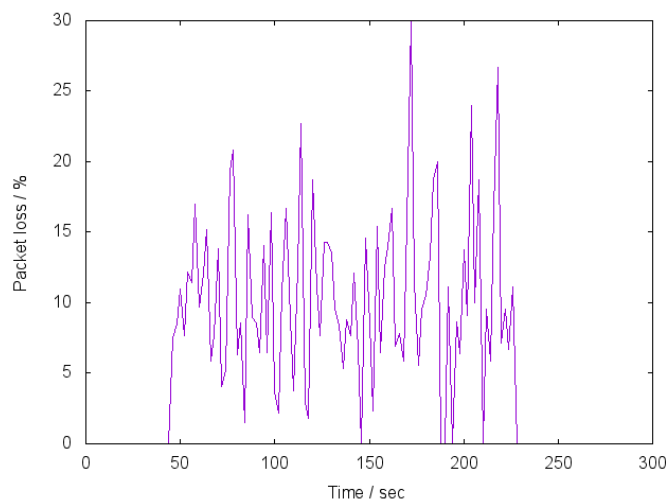


Figure 5.5: Situation 2 scenario 3 (10% loss rate and 3 MLBS)

perceived quality. Given the differences observed regarding the distribution of packet losses, it shows the importance of having multiple experiments to even out the statistical differences.

5.4 Results from Three-party Conversations

The experiments on the three-party conversations were conducted with all participants in separate rooms. The network alterations were the same for all network links for all the scenarios, also known as symmetric links. Each participant answered the same feedback form as was used in the pilot study, to rate the most recent scenario (see Appendix D). The data gathered from these forms describes the rating of audio- and video quality, as well as a description of potential problems experienced. The feedback from the three-party conversations comes only from a total of three users and is, therefore, hard to come up with any useful statistics. The results, however, are used as a pinpoint, as they can give a good indication on the perceived QoE.

Each scenario is discussed briefly in the following section. Note that scenario 1 was a testing scenario with no network alteration and is therefore not further discussed.

5.4.1 Packet Losses and MLBS Alterations

The scenarios for packet loss and MLBS are the same for the three-party conversation as for the pilot study, as described in chapter 5.2. We did not experience any distinct difference whether there were two or three parties in the conversation, so the details about the perceived user quality of scenarios 2 to 5 are, therefore, not described again here.

5.4.2 Delay and Jitter Alterations

Scenario 6 - Delay = 500 ms and Jitter = 0 ms

The participants noted that they experienced an almost ideal scenario with an overall high rating and little annoyance for users. No occurrences of video freezes or degraded audio.

Scenario 7 - Delay = 500 ms and Jitter = 300 ms

This scenario was rated in the mid-range. Some annoying disturbances from time to time, which were bad enough that the users said they would consider quitting the session in a normal setting. The users experienced troubles with both audio and video.

Scenario 8 - Delay = 500 ms and Jitter = 500 ms

This scenario was rated very poorly by the participants. Severe problems with both audio and video resulted in difficulties in understanding the other participants and maintaining a conversation with good flow. All participants noted that they

considered quitting the session and that they perceived a reduction in their ability to interact because of poor network conditions.

Scenario 9 - Delay = 1000 ms and Jitter = 0 ms

Increasing the delay to 1000 ms and removing the jitter resulted in a good rating of both audio and video quality for all users. The users did not experience any reduction in their ability to interact with the other users. One user noted that there was a slight problem with the synchronization, but not enough to cause any annoyance.

Scenario 10 - Delay = 1000 ms and Jitter = 300 ms

Adding 300 ms of jitter once again led to problems with both the audio- and video quality. Several video freezes and problems with audio resulted in users who said they considered quitting the session.

Scenario 11 - Delay = 1000 ms and Jitter = 500 ms

This scenario was, as expected, rated the overall worst scenario concerning delay and jitter with severe audio- and video problems. All users said they perceived a reduction in the ability to interact with the others, and they all considered quitting the session. One user also noted a problem with audio-video synchronization.

5.4.3 CPU-limit Alterations

What we experienced for the different CPU limit experiments was that one of the computers had much higher CPU-usage than the other two ($\approx 130\%$ versus $\approx 90\%$) while maintaining the connection to the two other clients. This difference made the links from that user asymmetric compared to the other users' links.

Scenario 12

Limiting the CPU to 80% led to difficulties for the user who initially used the most CPU, who reported that he had problems with hearing the other users and experienced a stuttering video. The two other users noted that the CPU limitation was an annoying factor but not problematic to maintain a conversation.

Scenario 13

A CPU of 60% affected both audio and video of the connection. The user with originally high CPU-usage could not participate in the conversation at all while the two others said they experienced some problems in understanding the other parties.

5.5 Discussion of Three-party Conversations Results

5.5.1 Packet Losses and MLBS

Packet losses and MLBS have been discussed in great detail in Section 5.2. Analyzing the same scenarios in a three-party setting served as a confirmation of what we found with regards to the pilot study as we experienced the same amounts of video freezes and quality reductions.

5.5.2 Effects of High Jitter Values

We noticed that the delay values we have used alone did not seem to affect the QoE. We also observed that the same jitter values for different delays are rated very similarly. We also noticed slight audio-video synchronization issues when increasing the delay to 1000 ms. The synchronization issues were noticeable but did not cause a great deal of annoyance.

Scenario 8 and 11 both had a jitter value of 500 milliseconds but had different delay values (500 ms and 1000 ms, respectively). Our experience was that a jitter value of 500 ms led to a terrible user experience, regardless of the delay being 500 ms or 1000 ms. We made some interesting observations when analyzing the graphs from *getstats.io* with large differences between the audio and video graphs. Note that neither the script of scenario 8 or scenario 11 introduces any packet losses into the network. As the graphs are fairly similar for both scenarios, we picked scenario 11 to discuss further. Figure 5.6³ and figure 5.7 are graphs for audio, showing the packet losses and jitter, respectively. The graphs from both figures have high fluctuations. The packet loss average is around 7 %, with peaks varying from 0 % to 14 %. The jitter average is around 400 ms with values ranging from 0 ms to 800 ms, and is too low compared to what we specified in the scenario. A discussion of weird jitter recordings in *getstats.io* follows later in this section.

An interesting part is the fact that packet losses are present. As we have not introduced any packet losses, it has to come from heavy jitter bursts. The extra delay introduced by jitter and potentially also the packet reordering results in packet losses. It seems that the jitter bursts has the most effect on the audio quality, but that the story is different for the video graphs.

³For all graphs from *getstats.io*: each color corresponds to a one-way connection from one client to another. The number of connections are given by $n(n - 1)$ for audio and video separately, where n is the number of users.

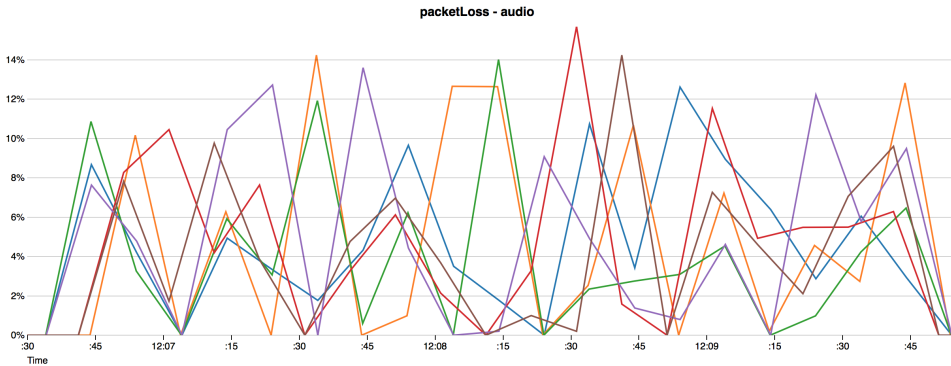


Figure 5.6: Packet loss ratio for audio from scenario 11 (from *getstats.io*).

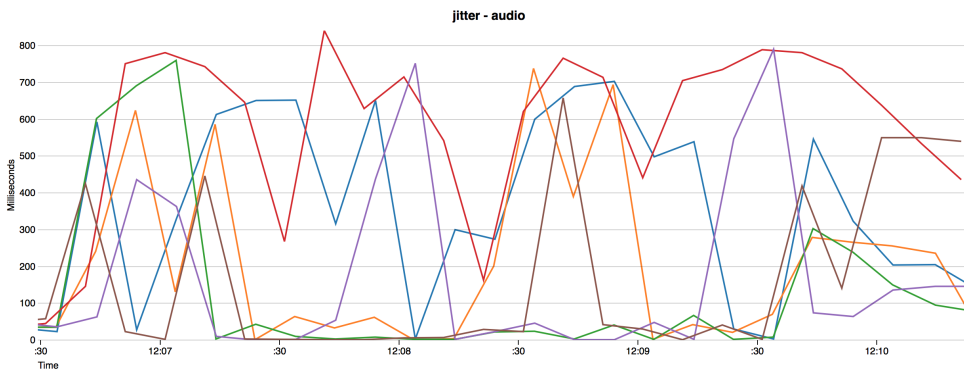


Figure 5.7: Jitter measures on audio from scenario 11 (from *getstats.io*).

Figure 5.8 shows the packet losses for video. The packet losses hit peaks at approximately 80 - 90 % to begin with, and then drops down and stabilizes around 30 %. Figure 5.9 illustrates the jitter of the video connection. The graphs peak at relatively high values at the beginning and in the ending (top peak equal to 4 seconds), but is 0 for most of the conversation.

It is of great interest how jitter can introduce packet losses for both audio and video. A peculiar observation is that both packet losses and jitter are present and variate for the duration of the call with respect to the audio graphs. It seems, however, that the jitter value concerning the video graphs seems to drop down to zero, while the packet loss rate drops down to around 30 %.

We looked closer at the data gathered by both *getstats.io* and *webrtc-internals* to see if we could find an explanation for this strange behavior. When discussing the session-related statistics of WebRTC earlier, we saw that *webrtc-internals* collected

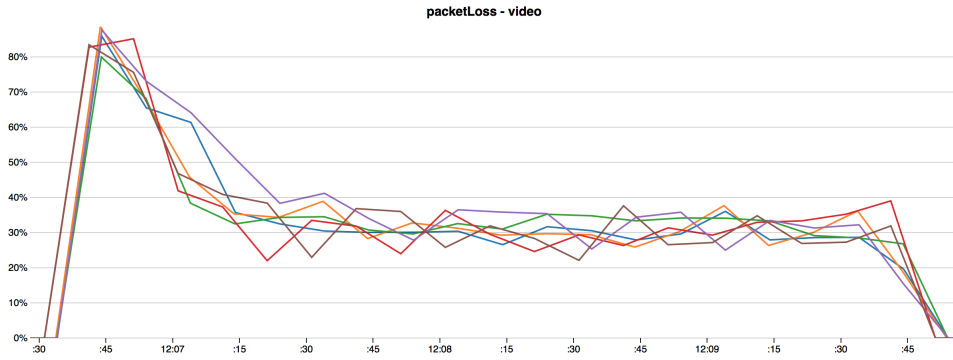


Figure 5.8: Packet loss rate for video from scenario 11 (from *getstats.io*).

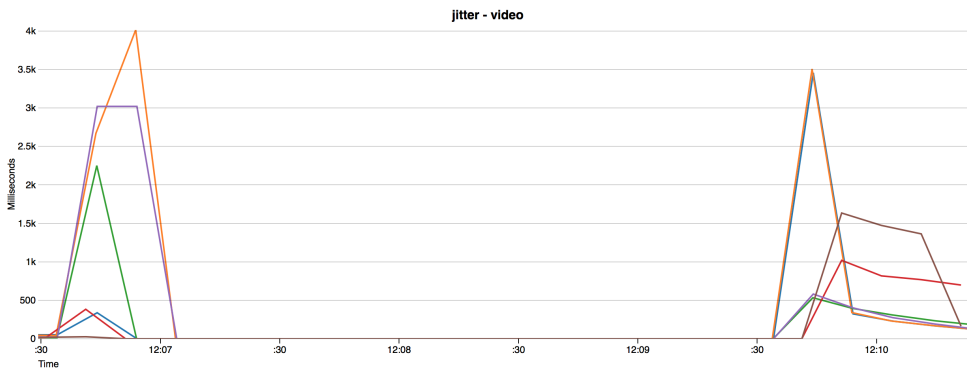


Figure 5.9: Jitter measures on video from scenario 11 (from *getstats.io*).

three different parameters for jitter. The only parameter collected regarding jitter for the video was the *googJitterBufferMs* parameter, which gave information about the jitter buffer size. Further investigation of the data gathered by *getstats.io* revealed that only one type of value was collected for the jitter, denoted simply by *jitter*. Comparing the data entries for the *jitter* value from *getstats.io* and *googJitterBufferMs* from *webrtc-internals* showed that the data was the same. This suggests that the jitter graph in *getstats.io* shows the size of the jitter buffer and not the actual jitter values recorded at the receiving end. We have not been able to confirm if this is, in fact, the case, as we did not have any documentation of *getstats.io*. Considering that *webrtc-internals* only recorded the jitter buffer size for video and that both *webrtc-internals* and *getstats.io* uses the same API, it does make sense that also *getstats.io* records the jitter buffer size.

If *getstats.io* only recorded the jitter buffer size, it still does not explain why there was a difference between audio and video, and why the graph dropped down to

zero for jitter for video. In the source code [15], it says that the video jitter buffer is *flushed* if the number of consecutive old packets exceeds 300 packets. This would explain why there first is a peak in Figure 5.9 before it drops down to zero, but it does not explain why it remains zero. The differences between audio and video could simply be that the jitter buffers for audio and video behave differently and that fewer audio packets are transmitted. Without any proper documentation, this question remains unanswered.

The graphs of the latency from the same scenario, shown in Figure 5.10, show that there definitely is jitter in the network, that is, variation in the latency between samples. The latency is stable around 2000 ms most of the time, but with some peaks.

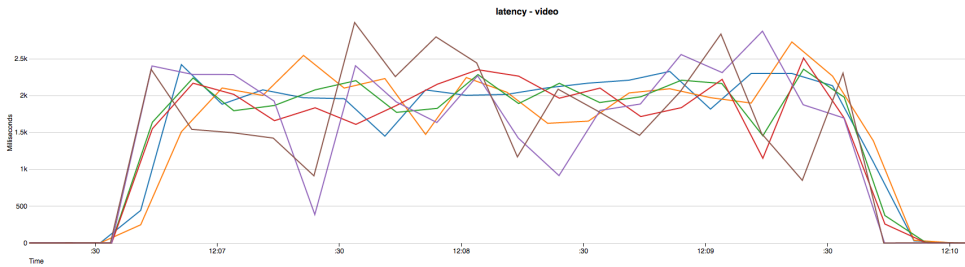


Figure 5.10: Delay measures for video from scenario 11 (from *getstats.io*).

5.5.3 CPU Limitation

The three-party conversations on CPU limitations indicated that both audio and video are negatively affected whenever the maximum CPU-usage is restricted. The data from *getstats.io* and *webrtc-internals* did not, however, give any clear indication to what impact the CPU limitation had. We believe this is because there are no alterations in the network but instead, the limitation of the CPU impacts the processing of audio and video packets on the clients. An increase in delay was recorded from the session statistics, and we believe this increase in delay is due to an increase in packet processing time. No packet loss or reduction in throughput was recorded.

Some interesting results we did find, was that the two clients who used the computers with the lowest CPU-usage had a much better user experience than the third user with high CPU-usage. It seems, therefore, that lower hardware requirements for one user do not necessarily affect other users who have sufficient hardware capabilities. To further investigate this a simple two-party experiment was conducted, where only one party had limitations on the CPU. No considerable

reduction in the quality could be seen at the party with no alterations, other than a slight increase in the delay.

For this reason, we conducted the two-party asynchronous experiment with CPU alterations as well to investigate the findings further. We set up a conversation between two clients and limited the CPU of only one of them. We experienced that the user who had no CPU alterations did not perceive any reduction in either audio or video quality while the user with the limited CPU experienced problems with both audio and video. At the client who had CPU limitations, a clear reduction in both video and audio quality could be perceived. As before, no packet loss or reduction in throughput could be seen.

The two-party asynchronous experiment was a little sidetrack because it is the only experiment which was purposely done with asymmetric conditions. We felt, however, that it was of interest to investigate further the findings of CPU limitation. The experiment is also of interest because it illustrates further capabilities of the testbed. Asymmetric limitation of CPU-usage is an interesting future research topic.

5.6 Limitations of Results

The testing is conducted on a relatively small sample size, N , which leads to uncertainty in the results. Despite the uncertainty, the results can be used as indications on factors that affect the QoE, but one should not draw definite conclusions based on the material.

Some of the trends we have found, such as why audio and video is impacted so differently for different MLBSs are difficult to explain. More thorough investigations regarding the connection data and research in literature suggest that the MLBS does have an impact on the perceived quality [19], [32]. These findings reveal no absolute truth about the MLBS, but rather that this should be more extensively researched in future studies.

WebRTC is a relatively new technology. Implementation details regarding the WebRTC API are well documented but we experienced that the tools for gathering session statistics were deficient with regards to explanations and descriptions, especially Chrome's *webrtc-internals*. We needed to search through the source code to find meaning in some of the graphs because we did not understand several parameters and how they were implemented. We also struggled with finding details about algorithms and error correction techniques used in WebRTC which would be helpful for our discussion. Some of the topics we found were work-in-progress by the Internet Engineering Task Force (IETF) but not yet published.

Chapter 6

Concluding Remarks

This thesis has discussed QoE in the context of WebRTC. We have created a testing environment and conducted experiments to see how the user experience is affected as different network alterations are applied.

We conducted a pilot study with 12 users, which consisted of two-party conversations and focused on the combination of packet loss rates and the Mean Loss Burst Size (MLBS). Feedback from the participants indicates that a higher MLBS tends to affect the overall perceived QoE negatively, especially with regards to the audio quality. The video and audio appeared to be impacted differently by an increase in the MLBS. We were unable to find any definite reasons for this difference with the collected session statistics data.

Further experiments on three-party conversation tested both delay and jitter, as well as limiting the CPU-usage on the clients. We found that delay alone has to be high (> 1 second) to cause any annoyance for the users. Jitter, however, quickly led to disturbances in both audio and video and led to a clear reduction in the QoE. Further research also revealed that high jitter values (> 300 milliseconds) introduce packet losses, which can help explain why both audio and video is negatively affected.

The experiments regarding CPU-restrictions showed that both the audio and video quality is impacted as the maximum allowed CPU-usage is reduced. The feedback from the users did, however, indicate that only the user who had limited resources was affected, while the other users were seemingly unaffected.

The experiments described in this report demonstrate only some of the capabilities of the testbed, and it can be used for more extensive research both with respect to the number of simultaneous participants as well as a broader range of parameter combinations. We conclude that the testbed was working as specified as an experimental test platform.

6.1 Limitations

6.1.1 Limitation in Setup

A lot of different factors will have an effect on the users' perceived QoE, some which are out of the developers control. It is, therefore, desirable to properly manage the factors that can be controlled. One important factor is the users' familiarity with the equipment used during the experiments, as it might feel more comfortable to use equipment you already know.

For this reason, it would be desirable if the users could use their computer for the experiments. This is possible, but the configuration necessary on the clients will be different depending on the OS used, and we have only provided documentation for how this is done on *Linux Ubuntu*. Our setup also requires that all clients are connected to the testbed controller via Ethernet, so currently, all devices needs to have support for an Ethernet connection.

6.1.2 Limitation of Data from Pilot Study

Conducting experiments and collecting data can be useful to test out a system and obtain useful insights. But certainty in statistics increases as the number of participants grow large. Our pilot study consisted of a total of 12 users, so the results are interpreted more as indications, rather than definite conclusions on how QoE is impacted by the different scenarios we have applied to the network. The aim of this project was not to come up with definite conclusions on the users' QoE, but rather to prove that the testbed can be used to generate a wide number of experiments and scenarios. Conducting more extensive user studies in the future could help determining with a greater certainty on how the users' QoE changes as the network is altered.

6.2 Future Work

The testbed made in this project is capable of applying a vast range of different scenarios to a network, well beyond the experiments conducted in this thesis. Therefore, the suggested future work based on our thesis involves utilizing the testbed to conduct further experiments.

6.2.1 Asynchronous Links

As a part of greatly limiting the number of parameter combinations, we used synchronous links for all experiments. It would be of interest to conduct experiments using asynchronous links to see how different link conditions in a conversation impacts

the overall QoE for all users as asynchronous network links are a better approximation to the real world.

More specifically, a real-life approximation could be illustrated by applying different delay values between different participants in a conversation, simulating that they are located at different geographical locations.

6.2.2 Further MLBS Testing

We saw in Chapter 5 how MLBS to some extent impacted the overall perceived QoE. It would, however, be of interest to conduct similar experiments where the differences in MLBSs are more significant. We believe that the impact of a larger MLBS will have a clearer effect on the perceived QoE.

References

- [1] Akamai state of the internet q3 2015 report. <https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/akamai-state-of-the-internet-report-q3-2015.pdf>. Accessed: 2016-05-22.
- [2] Basic image of WebRTC. <http://www.html5rocks.com/en/tutorials/webrtc/basics/jsep.png>. Accessed: 2016-02-15.
- [3] Browser support scorecard. <http://iswebrtcready.com/>. Accessed: 2016-04-07.
- [4] CPU usage limiter for Linux. <http://cpulimit.sourceforge.net/>. Accessed: 2016-03-15.
- [5] Identifiers for webrtc's statistics api. <http://w3c.github.io/webrtc-stats/>. Accessed: 2016-05-24.
- [6] Image of WebRTC with STUN and TURN. <http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/turn.png>. Accessed: 2016-02-15.
- [7] NetEm manual. <http://man7.org/linux/man-pages/man8/tc-netem.8.html>. Accessed: 2016-05-16.
- [8] Rtp, rtcp, and rtsp - internet protocols for real-time multimedia communication. <http://www.cse.wustl.edu/~jain/books/ftp/rtp.pdf>. Accessed: 2016-05-25.
- [9] Sample size: How many survey participants do i need? http://www.sciencebuddies.org/science-fair-projects/project_ideas/Soc_participants.shtml. Accessed: 2016-05-12.
- [10] Traffic control manual. <http://man7.org/linux/man-pages/man8/tc.8.html>. Accessed: 2016-05-29.
- [11] Universal 32bit traffic control filter. <http://man7.org/linux/man-pages/man8/tc-u32.8.html>. Accessed: 2016-05-29.
- [12] WebRTC architecture. <https://webrtc.org/architecture/>. Accessed: 2016-05-16.
- [13] WebRTC in the real world: STUN, TURN and signaling. <http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/>. Accessed: 2016-02-24.

- [14] WebRTC organization. <https://webrtc.org/>. Accessed: 2016-03-09.
- [15] WebRTC source code repo. <https://chromium.googlesource.com/external/webrtc/>. Accessed: 2016-05-25.
- [16] Sample size: how many participants do i need in my research? *Anais Brasileiros de Dermatologia*, 2014.
- [17] D. Ammar, K. De Moor, M. Fiedler, and P. Heegaard. Video QoE killer and performance statistics in WebRTC-based video communication. *IEEE Sixth International Conference on Communications and Electronics (ICCE)*, 2016.
- [18] D. Ammar, P. Heegaard, M. Xie, K. De Moor, and M. Fiedler. Revealing the dark side of WebRTC statistics collected by google chrome. *Quality of multimedia experience (qomex), eighth international conferences, Lisbon*, 2016.
- [19] J. Apostolopoulos. Reliable video communication over lossy packet networks using multiple state encoding and path diversity. In *Visual Communications and Image Processing, 2001*, December 2000.
- [20] M. Carbone and L. Rizzo. Dummynet revisited. May 2009.
- [21] Y. Cinar and H. Melvin. WebRTC quality assessment: Dangers of black-box testing. In *Digital Technologies (DT), 2014 10th International Conference on*, pages 32–35, July 2014.
- [22] P. Emstad, P. Heegaard, B. Helvik, and L. Paquereau. *Dependability and performance in information and communication systems*. Tapir uttrykk, 2011.
- [23] S. Gunkel, M. Schmitt, and P. Cesar. A QoE study of different stream and layout configurations in video conferencing under limited network conditions. In *Quality of Multimedia Experience (QoMEX), 2015 Seventh International Workshop on*, pages 1–6, May 2015.
- [24] S. Hemminger et al. Network emulation with NetEm. In *Linux conf au*, pages 18–23. Citeseer, 2005.
- [25] R. Hoffmann, V. Minkin, and B. Carpenter. Ockham’s razor and chemistry. *HYLE- International Journal for Philosophy of Chemistry*, 3, 1997.
- [26] S. Holmer, M. Shemer, and M. Paniconi. Handling packet loss in WebRTC. In *ICIP*, pages 1860–1864, 2013.
- [27] T. K. Husøy. Topology in WebRTC services. Master’s thesis, Norwegian University of Science and Technology, 2015.
- [28] ITU-T. Amendment 1: New appendix i - definition of Quality of Experience (QoE). *International Telecommunication Union*, 2006.
- [29] ITU-T. Definitions of terms related to Quality of Service. *International Telecommunication Union*, 2008.

- [30] ITU-T. Subjective quality evaluation of audio and audiovisual multiparty telemeetings. *International Telecommunication Union*, 2012.
- [31] P. Le Callet, S. Möller, and A. Perkis. Qualinet white paper on definitions of quality of experience. *European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003)*, 2012.
- [32] Y. J. Liang, J. G. Apostolopoulos, and B. Girod. Analysis of packet loss for compressed video: does burst-length matter? In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, volume 5, pages V-684-7 vol.5, April 2003.
- [33] Y. J. Liang, J. G. Apostolopoulos, and B. Girod. Analysis of packet loss for compressed video: Effect of burst losses and correlation between error frames. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(7):861-874, July 2008.
- [34] L. Nussbaum and O. Richard. A comparative study of network link emulators. march 2009.
- [35] S. Salsano, F. Ludovici, A. Ordine, and D. Giannuzzi. Definition of a general and intuitive loss model for packet networks and its implementation in the netem module in the linux kernel. August 2012.
- [36] M. Schmitt, S. Gunkel, P. Cesar, and P. Hughes. A QoE testbed for socially-aware video-mediated group communication. In *Proceedings of the 2Nd International Workshop on Socially-aware Multimedia, SAM '13*, pages 37-42, New York, NY, USA, 2013. ACM.
- [37] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, RFC Editor, July 2003.
- [38] M. Sunde and A. Underdal. Investigating QoE in a cloud-based classroom response system. Master's thesis, Norwegian University of Science and Technology, 2014.
- [39] D. Vucic and L. Skorin-Kapov. The impact of mobile device factors on QoE for multi-party video conferencing via WebRTC. In *Telecommunications (ConTEL), 2015 13th International Conference on*, pages 1-8, July 2015.
- [40] X. Yu, J. W. Modestino, and X. Tian. The accuracy of markov chain models in predicting packet-loss statistics for a single multiplexer. *IEEE Transactions on Information Theory*, 54(1):489-501, Jan 2008.

Appendix

Configuration Scripts



A.1 Controller configuration

A.1.1 Configure forwarding table

The following listing shows how to enable *ip forwarding* between the two interfaces eth0 and eth1 on the testbed controller. This is necessary so the clients connected to the controller can send and receive data from outside the LAN, created by the testbed.

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward
$ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
$ iptables -A FORWARD -i eth0 -o eth1 -m state --state \
RELATED,ESTABLISHED -j ACCEPT
$ iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
```

A.1.2 Redirecting traffic

The following listing shows how to disable ICMP redirect messages on the controller. The testbed controller will not send redirect messages to the clients when it discovers that there exists a shorter path between the clients if executing these commands.

```
$ echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects
$ echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
$ echo 0 > /proc/sys/net/ipv4/conf/all/secure_redirects
$ echo 0 > /proc/sys/net/ipv4/conf/default/accept_redirects
$ echo 0 > /proc/sys/net/ipv4/conf/default/send_redirects
$ echo 0 > /proc/sys/net/ipv4/conf/default/secure_redirects
$ echo 0 > /proc/sys/net/ipv4/conf/eth1/accept_redirects
$ echo 0 > /proc/sys/net/ipv4/conf/eth1/send_redirects
```

```
$ echo 0 > /proc/sys/net/ipv4/conf/eth1/secure_redirects
```

A.2 Static IP-address configuration

Both the clients and the controller needs to be configured with static IP-addresses. The following listing shows the commands for how this was done.

```
In /etc/network/interfaces add the following:
```

```
$ auto eth1
$ iface eth1 inet static
$ address 10.0.0.2
$ netmask 255.255.255.0
$ dns-nameserver 8.8.8.8 8.8.4.4
```

The configuration was added to all the entities in the testbed, and the interface name was specified to the name of the interface on the entity. The address field must be specified for each entity and all the entities must be on the same subnet. We chose that the testbed controller used the IP-address *10.0.0.1*, and the clients would use IP-addresses *10.0.0.2* to *10.0.0.x*, depending on the number of clients. The Domain Name System (DNS)-nameserver is specified so the clients can make DNS lookups when they are configured with static IP-addresses. The DNS name server is configured to use Googles' DNS server.

Appendix **B**

Simple Gilbert MLBS Data

The data in table B.1 is part of the results from the experiments in chapter 4, and shows how different r -values in the SG loss model corresponds to different MLBSs. The values in the table can be used to apply specific MLBSs to packet loss rate in the scenarios by choosing the appropriate transition probability, p . For instance, a scenario with 30% packet loss and MLBS = 2, would require a r -value of 50, and the p -value would be computed as follows.

$$p = \frac{\pi_B * r}{1 - \pi_B} = \frac{0.30 * 0.50}{1 - 0.30} = 0.214 = 21.4\%$$

Table B.1: r -values with corresponding Mean loss burst size (MLBS)

r	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
MLBS	4.00	3.33	2.85	2.50	2.22	2.00	1.82	1.66	1.54	1.43	1.33	1.25	1.17	1.11	1.05	1.00

Appendix

Conversation Task for the Pilot Study

The four first survival tasks, both textual description and object lists, are taken directly from [30]. We made the fifth survival task concerning the jungle. For each survival task, each participant is presented a brief description of the context of the situation they are in, as well as a list of objects. The textual description is the same, while the list of objects is different for the two participants. For brevity, we have cut out the textual description for participant 2 in all the following scenarios.

C.1 Survival task 1: Survival task in winter

You have just crash-landed in the North of Canada. The small plane in which you were traveling has been completely destroyed except for the frame. The pilot and co-pilot have been killed, but no one else is seriously injured.

You are in a wilderness area, snow-covered and made up of thick woods broken by many lakes and rivers. The pilot announced shortly before the crash that you were eighty miles northwest of a small town that is the nearest known habitation. It is mid-January. The last weather report indicated that the temperature would reach minus twenty-five degrees in the daytime and minus forty at night. You are dressed in winter clothing appropriate for city wear – suits, pantsuits, street shoes and overcoats. While escaping from the plane, your group salvaged the items listed below.

x C. CONVERSATION TASK FOR THE PILOT STUDY







Ball of steel wool	
Extra shirt and trousers for each survivor	
A little axe	
A strong sheet (6 m x 6 m)	
Newspaper (one per person)	
Compass	

Table C.1: Items for surviving in the winter for participant 1







Loaded .45-calibre pistol	
Sectional air map made of plastic	
Margarine in a big iron box	
Quart of 85-proof whiskey	
Cigarette lighter without the fluid	
Family-sized chocolate bar (one per person)	

Table C.2: Items for surviving in the winter for participant 2

C.2 Survival task 2: Survival task at sea

You are drifting in a private yacht in the South Pacific. A fire with unknown origin has destroyed much of the yacht, notably navigational and radio equipment. After having controlled the fire, you realize that the boat is sinking little by little. Your best estimate is that you are many hundreds of miles from the nearest landfall. You and your friends have managed to save 15 items, undamaged and intact, after the fire. In addition, you have salvaged a four man rubber life craft and a box of matches.






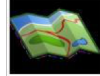

A sextant	
A small transistor radio	
A shaving mirror	
20 square feet of opaque plastic sheeting	
A quantity of mosquito netting	
A map of the Pacific Ocean	
2 boxes of chocolate bars	

Table C.3: Items for surviving at sea for participant 1








A 20 litre container of water	
One bottle of 160 per cent proof rum	
A case of army rations	
15 feet of nylon rope	
A can of shark repellent	
A floating seat cushion	
A fishing kit	

Table C.4: Items for surviving at sea for participant 2

C.3 Survival task 3: Survival task on the moon

You are a member of a space crew originally scheduled to rendezvous with a mother ship on the lighted surface of the moon. However, due to mechanical difficulties, your ship was forced to land at a spot some 200 miles from the rendezvous point. In addition to your space suit, your crew has managed to save items left intact and undamaged after landing. Your task is to take the items which allow you to reach the mother ship.








Food concentrate	
Stellar map	
50 feet of nylon rope	
One case of dehydrated milk	
Portable heating unit	
First aid kit	
A torch	

Table C.5: Items for surviving on the moon for participant 1








Magnetic compass	
Two signal flares	
Box of matches	
Parachute silk	
Solar-powered FM receiver-transmitter	
A 100 lb. tanks of oxygen	
A .45-calibre pistol	

Table C.6: Items for surviving on the moon for participant 2

C.4 Survival task 4: Survival task in the desert

You have just crash-landed in the Sonora desert in the south-west of United States. The pilot and co-pilot have been killed in the crash. However, the pilot announced that before impact you were approximately 110 kms off the course of the flight plan. He also indicated that that you were 113 km southwest of a mining camp which is the nearest known habitation. The surrounding desert is made up of sand dunes and seems dry except for some cactus. The last weather report indicated that the temperature at the ground level will be about 45°C. All of you are dressed in light clothes – cotton shirts, trousers, socks and soft shoes. Before the crash, your group was able to save some items.








Torch with 4 battery-cells	
Bottle of 1000 salt tablets	
Folding knife	
1 litre of water per person	
Air map of the area	
First-aid kit	
2 litres of 180 proof liquor	

Table C.7: Items for surviving in the desert for participant 1








Plastic raincoat (large size)	
A cosmetic mirror	
Magnetic compass	
Sunglasses (for everyone)	
A book entitled 'Desert animals that can be eaten'	
.45-calibre pistol	
Overcoat (for everyone)	

Table C.8: Items for surviving in the desert for participant 2

C.5 Survival task 5: Survival task in the jungle

You are on a guided trip in the Taman Negara jungle, on the Malaysian peninsula. You and your friend fell into a river and was taken down stream for several hours, and are now separated from the rest of the group. You have no idea how far away from civilization you are, and it will soon be sunset. You need to prepare yourself to spend the night in the jungle. Most of your equipment was lost in the river, but you have the following left:








Machete	
Toilet paper	
Snake bite kit	
Headlamp	
Plastic raincoat (large size)	
Satellite phone	
Fire steel	

Table C.9: Items for surviving in the jungle for participant 1

Tent	
Tree climbing gear	
Additional pair of socks	
Shotgun (loaded)	
Magnetic compass	
Life jacket	
Book: Bear Grylls - Mission survival	

Table C.10: Items for surviving in the jungle for participant 2

Appendix **D** Questionnaires

D.1 Pre-session questionnaire

Quality of Experience of WebRTC-based application appear.in

This form gathers information on your familiarity with online video conversations as well as some general questions about yourself. The information is used in the report for describing the reach of our test participants

First, some general questions about how you use online video communication tools and services

1. **Which services and applications for online video conversations have you used during the last month (approximately)?**

Check all that apply.

- Skype
- Google Hangouts
- Appear.in
- Facetime
- Firefox Hello
- Tiny chat
- Viber
- Professional or semi-professional video conferencing service
- Other, please specify:

2. **How often have you participated in online video conversations, using any of the above (or other) applications, during the last month (approximately)?**

Mark only one oval.

- Never
- Once
- 2 to 3 times
- Around once a week
- Several times a week
- Daily

3. To which extent do you consider the following aspects as (un)important when you are using online video communication service or application?

Mark only one oval per row.

	Very unimportant	Unimportant	Neutral	Important	Very important
Good audio quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Good video quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Good audio-video synchronization	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

And now some questions specifically related to your use of appear.in

4. When did you use appear.in for the first time?

Mark only one oval.

- I have never used it before
- Less than one month ago
- Between 1-3 months ago
- Between 3-6 months ago
- Between 6-12 months ago
- More than a year ago

5. How often did you use appear.in during the last month (approximately)?

Mark only one oval.

- Never
- Once
- 2 to 3 times
- Around once a week
- Several times a week
- Daily

6. (If you have used appear.in before) To which extent are you (dis)satisfied with appear.in when it comes to the following aspects?

Mark only one oval per row.

	Very dissatisfied	Dissatisfied	Neutral	Satisfied	Very satisfied
Good audio quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Good video quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Good audio - video synchronisation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Finally, a few questions about yourself

7. What is your birthyear? (please use 4 digits)

.....

8. What is your gender

Mark only one oval.

Male

Female

9. What is your profession / main occupation?

Mark only one oval.

Student

Employee / civil servant

Blue collar worker

Executive

Selv-employed / free profession

Pensioner / retired

Unemployed / job seeker

Other, please specify:

10. Are you studying or working in the field of audio / video quality, multimedia processing, or a related field?

Mark only one oval.

Yes

No


11. What is your email address

.....


12. Roomname

.....

D.2 Session feedback from appear.in



3



Thank you for using appear.in.

[Return to room](#)

	5- Excellent	4- Good	3- Fair	2- Poor	1- Bad
How would you rate the overall audiovisual quality of the session (the overall combined audio and video quality)?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
How would you rate the video quality of the session?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
How would you rate the audio quality of the session?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Which quality-related issues have you experienced during the session? Several answers are possible.

Audio problems: bad audio or no audio at all
 Video problems: bad video or no video at all
 Bad synchronization between audio and video
 Not applicable (never experienced any problems)
 Other, please specify:

Have you considered quitting the session because of quality-related issues?

Yes
 No

Did you perceive any reduction in your ability to interact with the other party (parties) during the session?

Yes
 No

If Yes, specify the problem if you could:

Your text here

[Send](#)

Appendix **E**

Pilot Study - NetEm Scripts

The following includes all the NetEm scripts used in the testbed for the pilot study. The only difference between the scripts is the part where the packet loss rate is applied by using the *gemodel* in NetEm. Note that the bandwidth is only specified so that TC and NetEm does not impose any restrictions on the bandwidth. The bandwidth is, therefore, set to an arbitrary high value and the only restriction on the bandwidth is the bandwidth initially available in the network.

E.1 Script 1 - (10% PL and 1.5 MLBS)

```
1  #!/bin/bash
2  Client1IP=10.0.0.2
3  Client2IP=10.0.0.3
4  duration=180s
5  pause=30s
6
7  tc qdisc del dev eth1 handle 1: root htb
8  tc qdisc add dev eth1 handle 1: root htb
9  tc class add dev eth1 parent 1: classid 1:1 htb rate 1000Mbps
10 tc class add dev eth1 parent 1:1 classid 1:11 htb rate 100Mbps
11 tc class add dev eth1 parent 1:1 classid 1:12 htb rate 100Mbps
12
13 function finish {
14     tc qdisc del dev eth1 handle 1: root htb
15 }
16
17 trap finish EXIT
18 sleep $pause
19 echo 'Using Packet-loss: 10 and MLBS: 1.5'
20
21 tc qdisc add dev eth1 parent 1:11 handle 10: netem loss gemodel 7.33% 66%
22 tc qdisc add dev eth1 parent 1:12 handle 11: netem loss gemodel 7.33% 66%
23
24 tc filter add dev eth1 prio 1 u32 match ip dst $Client1IP match ip src $Client2IP flowid 1:11
25 tc filter add dev eth1 prio 1 u32 match ip dst $Client2IP match ip src $Client1IP flowid 1:12
26
27 sleep $duration
28 tc qdisc del dev eth1 handle 1: root htb
29 sleep $pause
30
31
```

E.2 Script 2 - (10% PL and 3 MLBS)

```

1  #!/bin/bash
2  Client1IP=10.0.0.2
3  Client2IP=10.0.0.3
4  duration=180s
5  pause=30s
6
7  tc qdisc del dev eth1 handle 1: root htb
8  tc qdisc add dev eth1 handle 1: root htb
9  tc class add dev eth1 parent 1: classid 1:1 htb rate 1000Mbps
10 tc class add dev eth1 parent 1:1 classid 1:11 htb rate 100Mbps
11 tc class add dev eth1 parent 1:1 classid 1:12 htb rate 100Mbps
12
13 function finish {
14     tc qdisc del dev eth1 handle 1: root htb
15 }
16
17 trap finish EXIT
18 sleep $pause
19 echo 'Using Packet-loss: 10 and MLBS: 3'
20
21 tc qdisc add dev eth1 parent 1:11 handle 10: netem loss gemodel 3.67% 33%
22 tc qdisc add dev eth1 parent 1:12 handle 11: netem loss gemodel 3.67% 33%
23
24 tc filter add dev eth1 prio 1 u32 match ip dst $Client1IP match ip src $Client2IP flowid 1:11
25 tc filter add dev eth1 prio 1 u32 match ip dst $Client2IP match ip src $Client1IP flowid 1:12
26
27 sleep $duration
28 tc qdisc del dev eth1 handle 1: root htb
29 sleep $pause
30
31

```

E.3 Script 3 - (20% PL and 1.5 MLBS)

```

1  #!/bin/bash
2  Client1IP=10.0.0.2
3  Client2IP=10.0.0.3
4  duration=180s
5  pause=30s
6
7  tc qdisc del dev eth1 handle 1: root htb
8  tc qdisc add dev eth1 handle 1: root htb
9  tc class add dev eth1 parent 1: classid 1:1 htb rate 1000Mbps
10 tc class add dev eth1 parent 1:1 classid 1:11 htb rate 100Mbps
11 tc class add dev eth1 parent 1:1 classid 1:12 htb rate 100Mbps
12
13 function finish {
14     tc qdisc del dev eth1 handle 1: root htb
15 }
16
17 trap finish EXIT
18 sleep $pause
19 echo 'Using Packet-loss: 20 and MLBS: 1.5'
20
21 tc qdisc add dev eth1 parent 1:11 handle 10: netem loss gemodel 16.5% 66%
22 tc qdisc add dev eth1 parent 1:12 handle 11: netem loss gemodel 16.5% 66%
23
24 tc filter add dev eth1 prio 1 u32 match ip dst $Client1IP match ip src $Client2IP flowid 1:11
25 tc filter add dev eth1 prio 1 u32 match ip dst $Client2IP match ip src $Client1IP flowid 1:12
26
27 sleep $duration
28 tc qdisc del dev eth1 handle 1: root htb
29 sleep $pause
30
31

```

E.4 Script 4 - (20% PL and 3 MLBS)

```
1  #!/bin/bash
2  Client1IP=10.0.0.2
3  Client2IP=10.0.0.3
4  duration=180s
5  pause=30s
6
7  tc qdisc del dev eth1 handle 1: root htb
8  tc qdisc add dev eth1 handle 1: root htb
9  tc class add dev eth1 parent 1: classid 1:1 htb rate 1000Mbps
10 tc class add dev eth1 parent 1:1 classid 1:11 htb rate 100Mbps
11 tc class add dev eth1 parent 1:1 classid 1:12 htb rate 100Mbps
12
13 function finish {
14     tc qdisc del dev eth1 handle 1: root htb
15 }
16
17 trap finish EXIT
18 sleep $pause
19 echo 'Using Packet-loss: 20 and MLBS: 3'
20
21 tc qdisc add dev eth1 parent 1:11 handle 10: netem loss gemodel 8.25% 33%
22 tc qdisc add dev eth1 parent 1:12 handle 11: netem loss gemodel 8.25% 33%
23
24 tc filter add dev eth1 prio 1 u32 match ip dst $Client1IP match ip src $Client2IP flowid 1:11
25 tc filter add dev eth1 prio 1 u32 match ip dst $Client2IP match ip src $Client1IP flowid 1:12
26
27 sleep $duration
28 tc qdisc del dev eth1 handle 1: root htb
29 sleep $pause
30
31
```