EVALUATING NMP QUALITY OF SERVICE
Experiment with JackTrip regarding Latency versus Packet Jitter/Dropouts with High Quality Audio via LAN and WAN

UTVÄRDERING AV QUALITY OF SERVICE VID NMP
Experiment med JackTrip angående Latens kontra Jitter/Tapp av Paket med Högkvalitetsljud via LAN och WAN

Bachelor Degree Project in Information Technology
G2E 22.5 hp
Spring term 2018

Daniel Müntzing, a15danmu@student.his.se

Supervisor: Dennis Modig
Examiner: Jianguo Ding
Abstract
This study has developed a method to create an, to a big extent, automated testing system for NMP (Networked Music Performance) communication over LAN and WAN to be able to benchmark the UDP streaming engine JackTrip using a client-server model. The method is not locked into using JackTrip only, it could be used to do experiments with other engines too. The study tried to answer the question if latency correlates to amount of correctly aligned audio, and to what extent the audio is correctly aligned in respect to tolerated latency (based on earlier research) when at least two musicians remote-conducting musical pieces together. There were 13 different buffer settings tested, which used no redundancy and redundancy of 2, and which were sent through 4 different LAN/WAN-scenarios. A big dataset was produced, with about 82 minutes’ worth of audio per test. To post-process the data a phase cancelling method was used to measure correctly aligned audio, while the latency was measured by counting the number of samples from the start of each audio file to the first sample that were not null or not under a certain threshold. The results showed clear correlations of buffer sizes impact of latency and amount of correctly audio sent over the network. If the buffer sizes are greater, it will produce higher latency and increase the amount of correctly aligned audio, and on the opposite side, if using less buffer, it will produce lower latencies and less correctly aligned amount of audio. The study also showed that there was very little impact of using higher redundancies, both regarding latency and amount of correctly audio. When analysing the amount of correct data when respecting the tolerated level of latency, the study showed a support for correctly aligned amount of streamed audio up to 65% when using JackTrip.

Keywords: NMP, Networked Music Performance, WAN, LAN, benchmark, JackTrip, Jack, QjackCtl, buffer, latency, correct audio, jitter, packet dropouts, high quality audio
**Sammanfattning**


*Nyckelord: NMP, Networked Music Performance, WAN, LAN, prestandatest, JackTrip, Jack, QjackCtl, bufferts, latens, korrekt ljud, jitter, pakettapp, högkvalitetsljud*
Table of contents

1 INTRODUCTION ........................................................................................................................................................................ 1

2 BACKGROUND ........................................................................................................................................................................... 2

  2.1 DEFINITION AND EXPLANATION OF KEY CONCEPTS .................................................................................................................. 2
  2.1.1 NMP ................................................................................................................................................................................... 2
  2.1.2 JACK Audio Connection Kit, QjackCtl and JackTrip .............................................................................................................. 2
  2.1.3 Digital Audio ........................................................................................................................................................................ 3
  2.1.4 Latency .................................................................................................................................................................................. 6
  2.1.5 Jitter, Clock Synchronization, Buffers and Glitches/Dropouts ................................................................................................. 7
  2.1.6 WAN-Analyser and WAN-Simulator ..................................................................................................................................... 7
  2.2 RELATED WORK ............................................................................................................................................................................ 7

3 PROBLEM DESCRIPTION ................................................................................................................................................................. 9

  3.1 LATENCY, JITTER AND DROPOUTS ............................................................................................................................................ 9
  3.2 UDP OR TCP – PROS AND CONS ............................................................................................................................................... 9
  3.3 MOTIVATION .................................................................................................................................................................................. 9
  3.4 RESEARCH QUESTION ............................................................................................................................................................... 10
  3.5 OBJECTIVES ................................................................................................................................................................................ 10
    3.5.1 Collecting Requirements ..................................................................................................................................................... 10
    3.5.2 Development of the Examination Model ............................................................................................................................... 10
    3.5.3 Collection of Data ............................................................................................................................................................... 10
    3.5.4 Analysis of Data ............................................................................................................................................................... 10

4 METHODOLOGY ............................................................................................................................................................................. 11

  4.1 FULFILLING THE OBJECTIVES ................................................................................................................................................ 11
    4.1.1 Collecting Requirements ..................................................................................................................................................... 11
    4.1.2 Development of the Examination Model ............................................................................................................................... 11
    4.1.3 Collection of Data ............................................................................................................................................................... 11
    4.1.4 Analysis of Data ............................................................................................................................................................... 11
  4.2 THREAT OF VALIDITY ............................................................................................................................................................... 12
  4.3 ETHICS ........................................................................................................................................................................................ 12

5 DESIGNING THE EXPERIMENT ...................................................................................................................................................... 13

  5.1 THE HARDWARE/VIRTUALISATION DESIGN ............................................................................................................................... 13
  5.2 THE SOFTWARE/SCRIPT DESIGN ............................................................................................................................................ 13
    5.2.1 Scripts to Automate ............................................................................................................................................................. 13
  5.3 AUDIO FILES ................................................................................................................................................................................ 14
  5.4 WAN-SIMULATION .................................................................................................................................................................... 15
    5.4.1 Collecting Samples ............................................................................................................................................................. 15
    5.4.2 Simulating WAN on Interface ............................................................................................................................................. 15
  5.5 BUFFERS AND WAN ................................................................................................................................................................. 16
  5.6 POST-PROCESSING OF THE DATA ...................................................................................................................................... 17

6 DATA ANALYSIS AND RESULT ..................................................................................................................................................... 19

  6.1 CHARTS ....................................................................................................................................................................................... 19
    6.1.1 Tests without Redundancy ................................................................................................................................................... 19
    6.1.2 Tests with Redundancy of 2 ............................................................................................................................................... 23
    6.1.3 Comparison of the Redundancies ...................................................................................................................................... 27
APPENDIX A. - Graphical overview of scripts
APPENDIX B. - Count_inf.sh (Linux Bash-Script)
APPENDIX C. - daemon_cli.sh (Linux Bash-Script)
APPENDIX D. - jacktrip_client1_clientstart.sh (Linux Bash-Script)
APPENDIX E. - jacktrip_client1_serverstart.sh (Linux Bash-Script)
APPENDIX F. - jacktrip_client2_clientstart.sh (Linux Bash-Script)
APPENDIX G. - jacktrip_client2_serverstart.sh (Linux Bash-Script)
APPENDIX H. - Latencyscript_Ecasound.sh (Linux Bash-Script)
APPENDIX I. - onedrive.sh (Linux Bash-Script)
APPENDIX J. - qjackctl-buffer.sh (Linux Bash-Script)
APPENDIX K. - Scriptloop.sh (Linux Bash-Script)
APPENDIX L. - snapshot_client1.sh (Linux Bash-Script)
APPENDIX M. - threshold.sh (Linux Bash-Script)
APPENDIX N. - tmux.sh (Linux Bash-Script)
APPENDIX O. - trim_invert_merge-script.sh (Linux Bash-Script)
APPENDIX P. - Client1.xml (XML-File for QJackCTL-Settings)
APPENDIX Q. - Audiosort.ps1 (Windows Powershell-Script)
APPENDIX R. - Chain-creation.ps1 (Windows Powershell-Script)
APPENDIX S. - CreateTests.ps1 (Windows Powershell-Script)
APPENDIX T. - GatherSampledata.ps1 (Windows Powershell-Script)
APPENDIX U. - Sortback.ps1 (Windows Powershell-Script)
APPENDIX V. - Testiterations.csv (Example File)
APPENDIX W. - Sample-Data (Short Snippet of Example File)
APPENDIX X. - Index File for Sorting Back Sample-data Files (Example File)
APPENDIX Y. - Latency and Amount of Correctly Aligned Audio - LAN (No Redundancy)
APPENDIX Z. - Latency and Amount of Correctly Aligned Audio - “Netherlands” (No Redundancy)
APPENDIX AA. - Latency and Amount of Correctly Aligned Audio - “Sweden” (No Redundancy)
APPENDIX AB. - Latency and Amount of Correctly Aligned Audio - “USA” (No Redundancy)
APPENDIX AC. - Latency and Amount of Correctly Aligned Audio - LAN (Redundancy of 2)
APPENDIX AD. - Latency and Amount of Correctly Aligned Audio - “Netherlands” (Redundancy of 2)
APPENDIX AE. - Latency and Amount of Correctly Aligned Audio - “Sweden” (Redundancy of 2)
APPENDIX AF. - Latency and Amount of Correctly Aligned Audio - “USA” (Redundancy of 2)
APPENDIX AG. - Latency and Amount of Correctly Aligned Audio – Redundancy Comparison With Buffers of 2048 Samples and 4096 Samples
APPENDIX AH. - Traceroute
1 Introduction
The amount of audio communication over WAN, such as the Internet, will most likely increase in the future. Using real time uncompressed high-quality audio over LAN and WAN is an interesting topic and could be useful for both performing and practicing music (“Networked Music Performance”) and for reducing costs regarding expensive analogue cable solutions. This paper is about a study where evaluation, research and benchmarks of a high-quality uncompressed audio streaming engine was conducted. The analysis of the benchmarks tried to find the lowest latency and jitter free settings as possible. The focus of the experiment was for musicians remotely connected via a server who plays music together.

For system administrators and network technicians this research is useful for exploring the world of latency sensitive services via network and servers. As a user, such as musicians or for those who wants to communicate via low latency high quality audio over LAN/WAN, this research is useful to know how to tune audio and network settings for optimal results and know what to expect out of the limits of such service. The work can also be used as a part of the foundation for research about Internet communication services, such as Discord and Skype, but for musicians and/or other latency sensitive communications. For a developer and/or programmer who are developing an engine for streaming high-quality audio this study can be used to compare how well her engine works.

This study has developed a complex system to be able to automate the testing and processing of the data to be able to achieve a big dataset. The method for this could be used to make experiments with even bigger datasets and/or other streaming engines, which is useful for developers who wants to test their own engines in similar manner.
2 Background

Using high quality audio communication over packet switched network introduces some key concepts that needs to be clarified to be able to understand this study and its purpose. This section describes those key concepts and defines an overview of the scientific area and its related work.

2.1 Definition and Explanation of Key Concepts

In this topic the most important key concepts will be explained, for the sole reason to easier understand this paper.

2.1.1 NMP

NMP is short for Networked Music Performance and according to Rottondi, Chafe, Allocchio, & Sarti (2016) consists of real-time synchronous audio systems, for remotely interacting performances over geographic distances for musicians.

2.1.2 JACK Audio Connection Kit, QJackCtl and JackTrip

JACK Audio Connection Kit (hereafter written as Jack) is a free (GNU GPL and LGPL license) low-latency audio server, for several platforms, such as Linux, OS X, Solaris, FreeBSD and Windows. It allows third party software to send audio and data to and from each other via the audio interface (Davis, Letz, O’Quinn, & Hohn, 2014). According to the official site there should be no latency for audio input or output using Jack (Jackaudio.org, n.d.)

QJackCtl is a desktop application to control the status of Jack audio server, such as buffers, audio quality and audio connections (Cepela, n.d.).

JackTrip is a system that plugs in to Jack and uses it as a host audio server and is developed by The SoundWIRE group at the Center for Computer Research in Music and Acoustic. JackTrip connects a client and a server to each other via network communications such as LAN and WAN, and streams, via UDP, none compressed audio in chosen format, with or without redundancy. Redundancy means that a packet redundancy algorithm in JackTrip are applied, where two identical copies of the header and audio are packed into a bigger UDP packet. At the receiving end, the algorithm then reads the UDP packet and checks if it has not in forehand received the extra copy, and if it has it will discard it, otherwise it will be used as replacement for the lost packet. This method has a higher cost of bandwidth requirement (Cáceres & Chafe, 2010b).
2.1.3 Digital Audio
Since all traffic over packet switched networks needs to be digital, the same rule applies to audio sent over the medium. To record, store and playback audio in that way, the analogue signal must be converted, or digitized, to a stream of numbers. (Li, Drew, & Liu, 2014)

![Amplitude vs Time](image)

*Figure 1: The dotted lines represents the time dimension of sampling*

The digitizer takes samples of the analogue audio (sent in voltage from e.g. a microphone) in the dimensions of time and amplitude. This will happen in intervals that are evenly spaced in time, called *sampling*. The rate of how often this will happen is called *sampling frequency* and is measured on the horizontal axis, see Figure 1. This rate is directly connected to how high frequencies that can be represented in the audio stream. According to the *Nyquist Theorem*, the sampling frequency must be twice the frequency to what should be represented in the digital audio stream. The reason for this is because if sampling happens at the exact same rate, it will only create snapshots at constant values. Sampling rate at 1-2 times the analogue audio frequency will create false digital audio frequencies, while a sampling rate exactly 2 times the analogue audio frequency will create the correct digital audio frequencies (Li et al., 2014). Figure 3 shows a graphical representation of this process.

A dataset of samples can be converted to time by using the formula \[
\frac{\text{samples}}{\text{frequency rate}} = \text{time}.
\]
For example, using when using 44100 Hz frequency rate, 10000 samples is \[
\frac{10000}{44100} \approx 0.226s \text{ or } 226ms
\]
(Shambro, 2017).
In the dimension of amplitude, the sampling of the levels is called quantization. The quantization rates are measured on the vertical axis, see Figure 2, where 8-bit rate divides the axis in 256 levels, and 16-bit in 65,536 levels. Using lower bitrate will get lower sampling precision which will introduce roundoff errors. The roundoff errors causes quantization noise, measured in signal-to-quantization-noise ratio (SQNR). That means that this will give a certain dynamic field from the noise floor created by the errors and the signal maximum peak level. Each bit will give about 6dB of resolution, which means that 16-bit rate will have an SQNR of 96dB, while 8-bit rate will have 48dB SQNR. (Li et al., 2014).

*Figure 2 The dotted lines represents the amplitude dimension of sampling*
Figure 3 shows how frequencies is preserved according to the Nyquist Theorem when being digitized, where $t$ is the time intervals between the sample snapshots digitized. Blue line represents the original audio source, while red line represents the actual digitized sound.

In the domain of sending audio data over packet switched network, there is a sense of making the quality not higher than the somewhat subjective term “high enough”, otherwise there is a risk there will be huge overhead and waste of bandwidth. Since 16-bit audio gives an SQNR of 96 dB, and the sampling rate of 44100Hz will give frequencies audible up to 22050hz (Li et al., 2014), higher bitrates and sample rates will be left out of this discussion.
The bandwidth with bytes per second for uncompressed audio is calculated with the formula of $\frac{\text{bits per sample} \times \text{samples per second} \times \text{channels}}{8} = \text{bytes/second}$ (Audacity(R), 2018b).

Using 16-bit uncompressed audio quality at 44110Hz on one channel, means that the byte size per second to transfer over the network will be $\frac{16 \times 44100}{8} = 88\,200$ bytes/second or 88.2 kilobytes/second.

2.1.4 Latency

Simply defined, latency is the time it takes from the input into a system to the wanted outcome of it. The term might be a bit different interpreted depending on its circumstances. In the context of wide area networks, such as the Internet, the time for a packet to travel from its sender to the receiver and back again (called round-trip time) is the term of latency. In the context of digital audio, latency is the time from when a sound is created to when it is heard (Rouse, 2016).

Latency, or delay, or rather the absence of it, is a key aspect about successfully delivering real time audio communication over a packet switched network. In the domain of NMP it is important to keep it as low as possible as otherwise it will impact the synchronicity of performing and/or practicing musicians. (Rottondi, et. al, 2016)

When streaming audio over WAN, several contributors subdivide the total one-way end-to-end-delay, from source to ear. According to Rottondi, et. al (2016) there are 12 elements that subsequently affects the latency:

1. the air propagation from source to microphone
2. an insignificant time transducing the acoustic wave to electric signal
3. an insignificant time the electric signal travels from microphone to the analogue-to-digital converter
4. analogue to digital conversion (and, if any, encoding) and internal buffering in the transmitters soundcard
5. machine processing for packetization of the data
6. the time it travels through the networking, including propagation, transmission and routing delay
7. machine processing of depacketization of the data (and, if any, decoding) on the receiver’s side
8. buffering delay for the application in use on the receiver side
9. driver buffering and digital to analogue conversion in the soundcard on the receiver side
10. an insignificant time for transmission of electrical signals to an acoustic membrane
11. an insignificant time for transducing the electrical signal to acoustic sound waves in the acoustic membrane
12. the air propagation from acoustic membrane to ear
2.1.5 Jitter, Clock Synchronization, Buffers and Glitches/Dropouts

Jitter is a result of packets, in a packet switched network, arriving to the receiver at varieted unexpected time, and causing faulty audio result. To resolve this problem, timestamped packages and a buffer to place them in correct synchronization and order is necessary. The higher buffer used, the more capacity there is to fix jitter, but that will also increase the overall latency (Davidson, Peters, Bhatia, Kalidindi, & Mukherjee, 2006).

In an ideal situation all of the audio clocks on all transmitting and receiving machines match. The absence of synchronized clocks is specifically a problem over WAN and it causes buffer overrun or underrun conditions, meaning that the buffers will be either full or empty (Cáceres & Chafe, 2010b).

According to Cáceres & Chafe (2010b) and Rottondi et al. (2016) buffer over- and underrun conditions can cause mainly two glitches. Packets will be skipped (overrun condition) which causes unexpected result, or lost (underrun) which, depending of application, send zeros (causing short period of silence) to the process call back or forces the buffer to resend the last available packet to the process call back.

In Jack, buffers are divided in two parts, buffers and periods. There are not much written about it in the official documentation, but according to Britton (2013) local latency is calculated with

\[
\text{latency} = \frac{\text{buffer}}{\text{samplerate}} \times \text{periods}.
\]

2.1.6 WAN-Analyser and WAN-Simulator

A WAN-analyser will analyse the traffic between two points in a WAN and collect information regarding it. Data that can be gathered are latency, bandwidth, jitter and packet loss. A WAN-simulator simulates the typical characteristics of a WAN by using the information gathered by the WAN-analyser, or by using arbitrary data. In that way a LAN-setup can simulate what would happen to the packets out on the Internet (TATA Consultancy Services (TCS), 2008).

2.2 Related Work

There have been some research of latency and jitter regarding real-time audio communication over WAN, but most of them are targeted at IP-telephony, and not low latency high quality audio services. There are little studies to be found evaluating the relationship between latency and dropouts/jitter regarding high quality audio.

Bouillot & Cooperstock (2009) compares different solutions for NMP streaming, to see which one that results in best quality of service and still affectting smallest amount of latency with one hour of recording per engine, measuring the latency every 10 minutes. The four streaming engines they used were JackTrip, jack-tools, Soundjack and nStream. To analyse the latency, they manually compared the waveforms in a multitrack audio editor zooming enough to see where corresponding audio should match. To count the number of glitches they used a bandpass filter to extract the clicks caused by amplitude variations and micro silences. In that way they counted the number of glitches using an algorithm. The four different stream engines had clear connections between latency and number of glitches, where the one (JackTrip) with lowest latency had the greatest number of glitches and the one (jack-tools) with the highest latency caused the least number of glitches.

Cáceres & Chafe describes in two papers their own implementation application for streaming real-time low latency audio over network, called JackTrip. In JackTrip: Under the hood of an Engine for Network
Audio (2010b) they describe how their network audio package functions and compares it to other solutions. JackTrip is designed to keep decent audio with low latency, using different methods, strategies and mechanisms to minimize and cure jitter when using WAN connections. In JackTrip/SoundWIRE Meets Server Farm (2010a) they describe how their solution can be used as a multithreaded server to connect two or more clients together with the server as the spider in the web.

Queiroz, Schiavoni, & Wanderley (2013) describes an own solution testing audio streaming using four different transport layer protocols, TCP, UDP, SCTP and DCCP. They tried five different scenarios: localhost (one computer), cross over cable (peer-to-peer), cable switch (using a router), direct wireless (peer-to-peer) and access point wireless. The four different protocols resulted in different measured values, resulting in both pros and con, depending on reliability or latency and scenario.

Rottondi et al. (2016) reviews studies that are ‘psycho-perceptual’, which identifies latency tolerance thresholds when using NMP. They provide an overview of, in 2016, available technologies for setting up an NMP session.
3 Problem Description

When streaming high quality and low latency real-time audio communication over packet switched network, one must differentiate between LAN and WAN in terms of Quality of Service. While normally a LAN setup will most likely cause less trouble, a WAN will have several problems to bring the stream from end to end in a fluent way. There are several factors that will influence the result.

3.1 Latency, Jitter and Dropouts

When communicating with audio over network, latency might become a huge threshold for a successful session, e.g. when musicians interact with each other. Depending on the pace of the interacting musicians, the generally tolerated level of delay (latency) is up to 50ms. Between 50-150ms, the musicians have difficulty keeping the pace and their interpretation of the music becomes compromised. If the delay is over 150ms, it’s basically impossible to play together (Chew, Zimmermann, Sawchuk, & Tanoue, 2005).

Latency and jitter often go hand in hand in the sense of that when adding buffer (i.e. increasing latency) one will get less jitter, and vice versa (Davidson et al., 2006). In these scenarios it is a compromise between receiving full audio quality or receiving the sound in time.

See topic 3.2 about dropouts and the pros and cons with UDP or TCP.

3.2 UDP or TCP – Pros and Cons

There is always a consideration of what type of transport protocol that is best for a certain application over the Internet, and both UDP and TCP has its pros and cons. According to Cáceres & Chafe (2010a) the pros for TCP is that it is reliable since it will resend any lost packet and fix those that are out of order, but it will also increase the delay. Even worse, TCP might make the delay elastic since it is hard, if not impossible, to foresee when or if a packet will arrive at the receiving end. UDP has its cons that it is less reliable, since it misses the feature of acknowledging the transceiver that a successful packet arrived. On the other hand, UDP causes less overhead (i.e. causes less delay), and does not suffer from the elastic problem, with possibly lost packages as consequence.

3.3 Motivation

Earlier work, mentioned in chapter 2.2, theorizes and technically describes different solutions of NMP applications. Bouillot & Cooperstock (2009) benchmarked four different engines but did a lot of manual work when analysing the audio, and they did only check for latency every ten minutes in a total stream of 60 minutes. Their experimentation was only conducted over LAN.

This study is important to increase the knowledge to be able to use and develop the concept of NMP for both users and developers. There are few benchmarks with big datasets conducted, and even fewer tested over WAN. Benchmarks to see how this type of streaming service functions is therefore important.

This study tried to benchmark latency and amount of correctly transmitted audio with JackTrip over LAN and WAN using different 13 different buffer settings, with and without redundancy, with about 82 minutes’ worth of audio per test. It checked the latency every 22 or 17 seconds depending on audio material used. The experiment was also automated to a big extent to be able to handle big amount of data.
3.4 Research Question
In this project the following research questions were used as guide lines:

1. *Are there correlations between latency, dropouts and jitter regarding the performance of NMP?*

By analysing the data, it is possible to cross refer latency and jitter/dropouts with the corresponding buffer settings used in the experimentation environment, and then comparing the latency results with the amount of correctly aligned audio data.

2. *What impact do redundancy have on the performance?*
   
   a. *Will it achieve significant better results regarding the amount of correctly aligned data?*
   
   b. *Will it cause any other drawbacks such as increased latency?*

This is achieved by cross referring the results of latency and the amount of correctly aligned audio data when comparing redundancies.

3. *Based on acceptable latency for musicians declared by Chew, et al. (2005), how jitter free/correct streaming is it possible to achieve without passing that limit in a LAN or WAN environment?*

By setting a threshold latency in respective LAN/WAN scenario, the results can be fetched from the corresponding buffer to see how well the audio was obtained on the receiving machine.

3.5 Objectives
To be able to answer the questions there were objectives that needed to be fulfilled.

3.5.1 Collecting Requirements
A plan had to be developed to be able know what scenarios and preferences that was required to be included to answer the research questions.

3.5.2 Development of the Examination Model
A model of how the experimentation were going to be achieved was designed to be able to perform the required examination. The model had to be able to simulate LAN/WAN scenarios in an automated way to collect enough data for statistical analysis.

3.5.3 Collection of Data
A plan and a model were developed to record data and process it to be able analyse it in a correct way.

3.5.4 Analysis of Data
To be able to analyse the result a spreadsheet program of some kind had to be used to draw statistical conclusions of the recorded and processed data.
4 Methodology
This section describes how the objectives were fulfilled to answer the research questions. The type of study and technique that this project implemented was an experimental one, followed by the guidelines described in the book Thesis Projects (Berndtsson, Hansson, Olsson, & Lundell, 2008).

4.1 Fulfilling the Objectives
The following sub-subheadings will go through the objectives declared in former chapter.

4.1.1 Collecting Requirements
The main goal of the experiment was to explore independent variables in an examination model of dependent values that simulated conditions of wide area networks and, as a baseline, a local area network. The hypothesis is that certain independent variables should affect dependent variables, such as buffer/latency (independent value) versus jitter/audio quality (dependent value), therefore this experiment was designed and performed to falsify or support them.

The requirements were gathered to fulfil the scope of the project, where all demanded scenarios were tested to answer the research questions. That means that there were several variables to include and that they had to be tested in all possible combinations.

4.1.2 Development of the Examination Model
To develop an examination model, software was chosen to set up a testing system. It was conducted by studying earlier research. After searching for papers around the subject, the work presented by Cáceres & Chafe (2010a, 2010b) about JackTrip were most akin to this project.

Since this test was limited to the HP Z400 Workstations of the NSA lab of University of Skövde/Sweden, there were virtually three hardware compatible choices of operating system kernels to use, Windows, Linux or BSD. For the ease of the experimentation process, decision was made that the operating system should be lightweight, boot fast, be well documented and easily maintained. The kernel also had to be compatible with JackTrip to be able to function properly.

The testing machines had to be easily managed, cloned and deployed, while still being isolated from each other’s. To speed up and avoid the impact of the human factor, automation had to be conducted. A simulation of WAN had to be implemented into the testing model.

4.1.3 Collection of Data
Since the tests are automated, the collection of data had to be collected in an automated way too. As far as possible, data collected should not be affected by external circumstances during the collection. The data files also had to be distinguished and sorted in a good way and treated and processed so statistic conclusions could be drawn from them.

4.1.4 Analysis of Data
The data collected and processed were compared to each other by using a phase cancelling method where audio files that are sample exact will be silent (null or under a certain threshold in dB). The silent samples are then counted and gives a rate of correct audio that went through the packet switched network. To measure the latency, the number of samples from sample 0 to first sample that were not null or under a threshold in dB were counted in the unprocessed recorded audio file. The data were
then gathered in a spreadsheet program to draw statistical conclusions, such as average, median, standard deviation, minimum and maximum values.

4.2 Threat of Validity
The following validity threats, based on conclusions made by Wohlin, et al. (2012) were taken under consideration while planning this project:

Conclusion validity threat was present in this project as it was merely a model, via virtualisation and LAN- and WAN-simulation of a real scenario. This means that errors might have occurred that would not exist if same tests had been made on real, as in physical, equipment, likewise, errors that would occur on real physical equipment might not be present in the virtualized equipment. Consideration of that had to be taken in the analysis and conclusions.

Since using packet switched networks with UDP transport, the variety of incidents could have been large, which means that it had to be a quite big dataset to be able to draw undoubttable conclusions. That means that there was a risk in the validity of the statistical power. To aid this, a big dataset was used.

A big threat to the internal validity was that applications, which were key factors of the tests, and which were programmed by third parties, did not function as they were documented to do. In other words, because of bugs and incorrect documentation. There were also risks that mistakes were done during the design and development of the scripts used in the project. To aid this, all results’ plausibility had to be considered in the analysis and conclusion.

Each type of test was done during one session between server and client, which played and recorded a sound file several times in serial. That could lead to problems with the latency test if the sound from the earlier transmitted sound was still received, since the processing of the latency test did not check if the sound was correct or not, only if there were any sound at all. That should not have occurred very often though, since there were always at least a half of a second pause before next sound in the series were played. The plausibility of the results had to be considered during the analysis and conclusion.

4.3 Ethics
Most of the software used in this study were uncommercial (as in free of charge) software, although each program might have a copyright holder of the software. None of the software were reverse engineered or changed in any way. The result might not reflect the whole truth for all situations and should be seen as a test under a certain context, and not for all circumstances. The publication of the results and conclusions from this study was in no way intended to do any harm to anyone affiliated with the software used for it.
5 Designing the Experiment
This section describes how the experiment were conducted.

5.1 The Hardware/Virtualisation Design
The system used consisted of one server, HP Z400 Workstation (Intel(R) Xeon(R) CPU W3550 @ 3.07GHz with 4 physical cores (Intel, n.d.), with 24 GB DDR3-ram), running VMware ESXi 6.5 hosting two virtualised clients with Lubuntu 17.10 using the LXDE desktop. Both clients were allocated with 8gb of ram, but client 1 were allocated with only one core while client 2 were allocated with 2 cores. The reason behind this decision was that client 2 had to deal with both playing and recording the audio, while client 1’s only task was to send back the network audio stream, with WAN-simulation in ¾ of the tests. The rest of the resources were spared to the VMM (Virtual Machine Manager) itself. The clients were connected internally with a dedicated virtual switch, to be able to isolate them completely from external network resources. See Figure 4 for a graphical overview of the testing system.

![Diagram of test setup](Image)

Figure 4: A graphical overview of the test setup

5.2 The Software/Script Design
The software used was based on the Linux audio package Jack as audio API, and JackTrip for connecting the two clients together via network. Both clients used a server/client solution, where they used one server instance and one client instance of JackTrip each. This means that Client 2 connected its client instance to Client 1 server instance, and Client 1 connected its client instance to Client 2 server instance. Client 2 acted both as audio transmitter and receiver, started audio player and audio recorder virtually at the exact same time, using Ecasound, which plugs in to Jack (Vehmanen, 2014). Each audio test lasted about 39 seconds and were iterated 125 times per test of each configuration, multiplying it up to about 82 minutes’ worth of audio per test. For example, one test had the buffer size of 2048 samples times 4 periods.

Each test was tested with no redundancy and with redundancy of 2 (see section 2.1.2).

5.2.1 Scripts to Automate
Several bash scripts in Linux/Lubuntu were developed to be able to automate the tests and make sure that all settings and tests were achieved. See APPENDIX A for a basic graphical overview of the scripting system.
See APPENDIX N for script for main loop on the Lubuntu management computer, which controls all tests. The main information for the tests is collected from a CSV file, see APPENDIX V for an example of such file, which is created with a script to get all possible combinations (APPENDIX S). APPENDIX C displays the daemon script used to start and kill QjackCtl on each client since it only supports to be launched from the desktop. It needs to be killed to be able to get its buffers adjusted. APPENDIX J displays the script for adjusting the buffers. APPENDIX D, E, F and G shows the JackTrip client and server start-up scripts on each virtual machine (client 1 and client 2), and the audio connections of client 1 is controlled with the script in APPENDIX L. Ecasound is started via the script displayed in APPENDIX K and H. The latter script also transfers all audio files to the management computer and starts the script in APPENDIX O which trims, inverts and mixes the inverted file with the original audio file into a new audio file. All audio files are synced to OneDrive each half of an hour via the script in APPENDIX I which is executed from the Crontab.

When the audio files were finished being recorded, they were transferred to a Windows machine, where they were sorted into series of numbers with the script in APPENDIX Q, which also saves all folders in an index file, see APPENDIX X for an example of such file. This is since Audacity only converts one directory at the time, and names them with the pattern of ‘filename,’ where ‘n’ is a number added if the option are included that files should not be overwritten. After initial tests, it was discovered that Audacity, because of a bug, fails to convert more than about 750 files in a chunk, which means that a gather script was needed to be able to sort them back to correct series number again, see APPENDIX T. When all files from a test iteration (see APPENDIX V) were converted and gathered, a script was used to sort them back to their correct folders and series number, using the previously created index file, see APPENDIX U for that script. There was also a script written to create chain files (batch conversion settings) for Audacity, which was used when converting the files, see APPENDIX R.

When all audio files from a test iteration were converted, gathered and sorted, they were transferred back to the Lubuntu management computer, where two scripts were used to either calculate latency (APPENDIX B) or amount of correctly aligned audio (APPENDIX M).

In chapter 5.6 the method of processing the data are explained in greater details.

5.3 Audio Files
The audio files used in the experiment consisted of two files, one that were 30 seconds (although only about 22 seconds were used in the test) long which were a musical piece, and one that were 17 seconds long and consisted of a rhythm from drums beating every quarter note in a raising tempo, based on the most popular tempos found in music (Moelants, 2002). See Table 1 for the tempos used in the rhythm audio file. The audio files were mono and had the specifications of 16bit in resolution and 44.1khz in sample rate.

| Table 1: Tempos used in the rhythm audio file |
|------------------|------------------|
| 85 bpm           | 100 bpm          |
| 120 bpm          | 120 bpm          |
| 125 bpm          | 130 bpm          |
| 140 bpm          | 160 bpm          |

14
5.4 WAN-Simulation

To simulate WAN, two steps were taken.

5.4.1 Collecting Samples

To collect samples of different WAN scenarios, WANalyzer was used (TATA Consultancy Services (TCS), 2008) which tests the connection between a chosen remote destination on the Internet and the computer where it is started from. In that way three different locations were analysed to gather data such as jitter, latency, dropouts and bandwidth. The remote locations were servers in USA, Netherlands and Sweden.

5.4.2 Simulating WAN on Interface

To simulate WAN in the testing environment, TC/NetEm (for latency, jitter and loss) with Token Bucket Filter (for bandwidth) was used on outgoing and incoming traffic on Client 1’s interface. The applications/processes are incorporated in Linux’s kernel (The Linux Foundation, 2016). See Figure 5 for a graphical overview of the testing circuit. A full test of a LAN-environment was also made to be able to compare the results with more optimal conditions. A local test, using only the audio API, was also made to see how much latency the audio services added to the result.

![Figure 5: A graphical overview of the testing circuit](image)

In Table 2 parts of the Bash script for starting the WAN-simulation is written out, see APPENDIX N for full script. There are in total four different network setups, as stated in chapter 5.5.
Table 2: Parts of the Linux Bash script to start the WAN-simulation using TC/Netem

```bash
wan[5]="0.0724ms"       # "Sweden" Latency
wan[6]="1.658255%"      # "Sweden" Loss
wan[7]="29897kbit"      # "Sweden" Rate

# Adding rules for outgoing traffic on client 1
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ens192 root handle 1: htb default 1'
ssh daniel@10.208.15.177 'sudo tc class add dev ens192 parent 1: classid 0:1 htb rate '${wan[0]}kbit'
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ens192 parent 1:1 handle 10: netem delay '${wan[1]}ms' 'distribution normal loss '${wan[2]}

# Starting virtual interface for incoming traffic on client 1, will use same rules as outgoing traffic
ssh daniel@10.208.15.177 'sudo modprobe ifb'
ssh daniel@10.208.15.177 'sudo ip link set dev ifb0 up'
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ens192 ingress'
ssh daniel@10.208.15.177 'sudo tc filter add dev ens192 parent ffff: protocol ip u32 match u32 0 0 flowid 1:1 action mirred egress redirect dev ifb0'
```

5.5 Buffers and WAN

Buffers used are 1024 samples, 2048 samples and 4096 samples. These were multiplied with number of periods. The buffers were chosen after initial tests showing that using lower buffers will render audio with too much dropouts to be considered meaningful. In total there are thirteen tests with different buffer settings.

The following (Table 3) buffers of samples were used in the tests:

Table 3: Buffers used in the experiment

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Periods</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>2</td>
</tr>
<tr>
<td>1024</td>
<td>3</td>
</tr>
<tr>
<td>1024</td>
<td>4</td>
</tr>
<tr>
<td>1024</td>
<td>5</td>
</tr>
<tr>
<td>1024</td>
<td>6</td>
</tr>
<tr>
<td>2048</td>
<td>2</td>
</tr>
<tr>
<td>2048</td>
<td>3</td>
</tr>
<tr>
<td>2048</td>
<td>4</td>
</tr>
<tr>
<td>2048</td>
<td>5</td>
</tr>
<tr>
<td>2048</td>
<td>6</td>
</tr>
<tr>
<td>4096</td>
<td>2</td>
</tr>
<tr>
<td>4096</td>
<td>3</td>
</tr>
<tr>
<td>4096</td>
<td>4</td>
</tr>
</tbody>
</table>

The following (Table 4) WAN simulations were used in the test (LAN simulation is not listed here):

Table 4: WAN settings used in the experiment

<table>
<thead>
<tr>
<th>WAN-name</th>
<th>Jitter</th>
<th>Latency</th>
<th>Loss</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Netherlands&quot;</td>
<td>17.6ms</td>
<td>18.65085ms</td>
<td>0%</td>
<td>16210kbit/s</td>
</tr>
<tr>
<td>&quot;Sweden&quot;</td>
<td>9ms</td>
<td>9.0724ms</td>
<td>1.64255%</td>
<td>29897kbit/s</td>
</tr>
<tr>
<td>&quot;USA&quot;</td>
<td>10ms</td>
<td>18.2931ms</td>
<td>0%</td>
<td>26864kbit/s</td>
</tr>
</tbody>
</table>

In APPENDIX A, Table 7, Table 8 and Table 9, shows the routing information of how to get to the remote servers that the WAN simulations are based on. The source was sent from Skövde in Sweden, and one quite surprising detail was that the remote destination to Sweden, although it got less latency,
resulted in most hops (14) which might explain why there were packet losses. Also, Netherlands got more hops than USA (11 versus 10).

5.6 Post-processing of the Data
The audio data were post-processed in two different manners. Audacity were used to export the audio files as sample-data represented as decibel into text-files, which printed out one line per sample (Audacity(R), 2018a). A script were used (see below) to count exactly how many samples there were between the starting point of the audio files and their first sample that did not have null value (which was represented as “[ inf ]”) or were not under the threshold of -90dB.

The other process was to use Sox to trim the silence in the beginning of each audio data file, invert the rest of the sound and mix it with the original sound file (Bagwell, 2013) to phase cancel out all samples that are aligned on exactly same spot. This is done by simply summing the values out, i.e. if a sample that is on a certain positive offset over the zero line (middle line) would be mixed with a sample that is on the exact opposite side of the zero line (e.g. has the same offset but with negative value), they would cancel each other out and create silence (e.g. 200 + (−200) = 0, −200 + 200 = 0) (Audio-Technica, 2017). See Figure 6 for a graphical representation for phase cancelled audio, where silence (just a middle line) is the phase cancelled audio and therefore are correctly aligned audio.

Figure 6: Waveform representation of one audio file in Audacity showing phase cancelled audio as silence.

After the audio was phase cancelled, they were used with Audacity to export out the sample data to text-files. A silent track were represented as “[ inf ]” or as -90dB or lower. All sample data over the threshold of -90dB were counted as jitter, misaligned or erroneous in some way.

The threshold was chosen after analysing phase cancelled audio of locally re-recorded sound, where -90dB and lower were represented as silence.

To get statistics from the data, two scripts looped through all sample-data files, where it used the Linux built in commands Grep (Free Software Foundation, Inc., 2010) and WC (Rubin & MacKenzie, 2010) to count the matching lines. See Table 5 for parts of the Bash script to count latency lines, see APPENDIX B for full script. The latency was measured by counting all lines of silence in the first 50000 samples (~1.1337 seconds) of each audio file.
The amount of correctly aligned samples was counted in a similar manner as with latency, but with this method the lines of silence were counted in the files that were nulled out with the mixing of the phase reversed audio with the original file. The first 1000000 samples were counted (~22.67 seconds), since that was the limit for Audacity to output sample-data files. See Table 6 for part of the Bash script for counting correctly aligned samples, see APPENDIX M for full script. See APPENDIX W for a short snippet of an example of a sample-data file.

<table>
<thead>
<tr>
<th>Table 5: Parts of Linux Bash script for counting lines of latency in the sample-data files</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf[$count]=`cat $y/sample-data$count$txt</td>
</tr>
<tr>
<td>threshold[$count]=`grep &quot;-$9[0-9]|-$10[0-9]|-$11[0-9]|-$12[0-9]|-$13[0-9]|-$14[0-9]|-$15[0-9]|-$16[0-9]|-$17[0-9]|-$18[0-9]|-$19[0-9]|-$20[0-9]&quot; $y/sample-data$count$txt</td>
</tr>
<tr>
<td>latency[$count]=$(({inf[$count]} + {threshold[$count]}))</td>
</tr>
<tr>
<td>((count++))</td>
</tr>
</tbody>
</table>

After the counted lines were gathered in CSV files, they were combined in Microsoft Excel to be able to calculate average, median, standard deviation, minimum and maximum values.

<table>
<thead>
<tr>
<th>Table 6: Part of Linux Bash script for counting lines with correctly aligned sample data</th>
</tr>
</thead>
<tbody>
<tr>
<td>inf[$count]=`grep inf $y/sample-data$count$txt</td>
</tr>
<tr>
<td>threshold[$count]=`grep &quot;-$9[0-9]|-$10[0-9]|-$11[0-9]|-$12[0-9]|-$13[0-9]|-$14[0-9]|-$15[0-9]|-$16[0-9]|-$17[0-9]|-$18[0-9]|-$19[0-9]|-$20[0-9]&quot; $y/sample-data$count$txt</td>
</tr>
<tr>
<td>zero[$count]=`(${inf[$count]} + {threshold[$count]}))</td>
</tr>
<tr>
<td>((count++))</td>
</tr>
</tbody>
</table>

---

18
6 Data Analysis and Result

This section provides analysis and result of the data that was gathered during the experiment. The result origins from a network scheme of client-server model, which means that latency and jitter/dropouts will be added on two links. Although this study did not do any experiment with peer-to-peer connection, JackTrip supports that too, and such setting should be able to produce better results with both latency and amount of correctly aligned audio since there would be one link instead of two.

6.1 Charts

The following chapters analyse the results and present charts with the latency (in milliseconds) and the amount of correctly aligned audio (in percent). Since there were a limited amount of time per audio file, the percent is calculated by comparing every sound files length with the amount of audio that was correctly aligned.

This subchapter is divided in three sub-subchapters: tests without redundancy, tests with redundancy of 2 and finally a comparison between the redundancies.

The charts of every individual network are presented in the APPENDIX Y, Z, AA, AB, AC, AD, AE and AF, and all redundancy comparison charts are presented in APPENDIX AG.

6.1.1 Tests without Redundancy

This sub-subchapter presents the analysis of the tests without redundancy, network by network, and finally comparing all thirteen buffer settings over all networks.

The charts for average and median results for the baseline LAN network without redundancy are displayed in APPENDIX Y.

Chart 9, which represents latency in the LAN network without redundancy, shows that there were clear differences between low buffers and high buffers in latency, although higher periods did not seem to affect the total latency in greater amounts. Lowest average latency was at ~51.5ms, which had the buffers of 1024×2 samples, which achieved an average of ~42.5% amount of correctly aligned audio. The highest average latency was ~243.7ms with the buffer of 4096×4 samples, which achieved an average of ~91% amount of correctly aligned audio.

The standard deviation of the average latency was quite high and ranged ~20.6-21.9ms (lower sized buffers), ~39.7-40.5ms (mid-sized buffers) to ~69.4-76.7ms (higher sized buffers), which shows that the variation was large.

The lowest latency achieved was around ~21ms for buffers of 1024 samples (all periods), ~43-44ms for buffers 2048 samples (all periods) and ~91ms for buffers of 4096 samples (all periods). The highest latency was ~92-116ms for buffers of 1024 samples, ~185ms for buffers of 2048 samples and ~371ms for buffers of 4096 samples.

Chart 10, which represents the average and median amount of correctly aligned audio in the LAN network without redundancy, shows that there were clear correlations between high buffers and amount of correctly aligned audio, and vice versa. As with the latency test, the number of periods did only affect the result slightly. The best average result gave the buffers of 4096×4 samples with ~91% correctly aligned audio, which had the average latency of ~243.7ms, which was also the one with
longest average latency. The worst average result of amount of correctly aligned audio gave the buffers of 1024×3 samples with ~41.4% correctly aligned audio which had the average latency of ~51.6ms, about the same result as the best achieved average latency result (see above).

The standard deviation for average correctly aligned amount of audio was quite high, and ranged from ~23-25% (lower and higher sized buffers) to ~35% (mid-sized buffers), which shows that the variation was large.

All buffer types managed to achieve at least one audio stream that had 100% correctly aligned audio (22s or 17s long). The lowest amount of correctly aligned audio was as low as 0% up to ~2.5% (22s or 17s long).

The charts for average and median results for the “Netherlands” network without redundancy are displayed in APPENDIX Z.

Chart 11 which represents latency in the “Netherlands” network without redundancy, shows that there were clear differences between low buffers and high buffers in latency, although higher periods did not seem to affect the total latency in greater amounts. Lowest average latency was at ~120.8ms, which had the buffers of 1024×6 samples, which achieved an average of ~30.9% amount of correctly aligned audio. The highest average latency was ~299.3ms with the buffer of 4096×3 samples, which achieved an average of ~87.7% amount of correctly aligned audio.

The standard deviation of the average latency was quite high and ranged from ~16.6ms (lower sized buffers) to ~79.5ms (higher sized buffers), which shows, as with the LAN network, that the variation was large.

The lowest latency achieved was around ~67-69ms for buffers of 1024 samples (all periods), ~89-92ms for buffers of 2084 and 4096 samples (all periods). The highest latency was ~92-116ms for buffers of 1024 samples, ~232-275ms for buffers of 2048 samples and ~460-464ms for buffers of 4096 samples.

Chart 12 which represents the average and median amount of correctly aligned audio in the “Netherlands” network without redundancy, shows that there were clear correlations between high buffers and amount of correctly aligned audio, and vice versa. As with the latency test, the number of periods did only affect the result slightly. The best average result gave the buffers of 4096×2 samples with ~88% correctly aligned audio, which had the average latency of ~289.3ms, which was about 10ms less than the longest achieved average latency (see above). The worst average result gave the buffers of 1024×3 samples with ~30.616% correctly aligned audio which had the average latency of ~121.5ms, about the same result as the best achieved average latency result (see above).

The standard deviation for average amount of correctly aligned audio was quite high, and ranged ~11-12% (lower sized buffers), ~31-34% (mid-sized buffers) and ~25% (higher sized buffers) which shows, as with the LAN network, that the variation was large.

The buffer types of 1024 samples managed to maximally achieve 55-64% correctly aligned audio, the mid-sized buffer types of 2048 samples managed to maximally achieve ~99-100% correctly aligned audio, and all buffers of 4096 samples managed to maximally achieve 100% correctly aligned audio, all
of which were either 17s or 22s long. The lowest amount of correctly aligned audio was as low as ~0.5% up to ~3.5% (17s or 22s long).

The charts for average and median results for the “USA” network without redundancy are displayed in APPENDIX AB.

Chart 15, which represents latency in the “USA” network without redundancy, shows that there were clear differences between low buffer and high buffer in latency, although higher periods did not seem to affect the total latency in greater amounts. Lowest average latency was at ~88.6ms, which had the buffers of 1024×6 samples, which achieved an average of ~35.9% amount of correctly aligned audio. The highest average latency was ~270.3ms with the buffer of 4096×4 samples, which achieved an average of ~89.7% amount of correctly aligned audio.

The standard deviation of the average latency was quite high and ranged ~19.2-20.6ms (lower sized buffers), ~37.6-41.3ms (mid-sized buffers) to ~82.4-88.3ms (higher sized buffers), which shows, as with the other networks, that the variation was large.

The lowest latency achieved was around ~43-45ms for buffers of 1024 and 2048 samples (all periods), ~91ms for buffers of 4096 samples (all periods). The highest latency was ~116ms for buffers of 1024 samples, ~228-232ms for buffers of 2048 samples and ~371ms for buffers of 4096 samples.

Chart 16, which represents the average and median amount of correctly aligned audio for the “USA” network without redundancy, shows that there were clear correlations between high buffers and amount of correctly aligned audio, and vice versa. As with the latency test, the number of periods did only affect the result slightly. The best average result gave the buffers of 4096×4 samples with ~89.7% correctly aligned audio, which had the average latency of ~270.3ms, which was also the longest achieved average latency (see above). The worst average result gave the buffers of 1024×6 samples with ~35.9% correctly aligned audio which had the average latency of ~88.6ms, about the same result as the best achieved average latency result (see above).

The standard deviation for average amount of correctly aligned audio was quite high, and ranged ~18-27% (lower sized buffers), ~33-34% (mid-sized buffers) and ~23-25% (higher sized buffers) which shows, as with the other networks, that the variation was large.

The buffer types of 1024 samples managed to maximally achieve ~86-100% correctly aligned audio, the mid-sized buffer types of 2048 samples managed to maximally achieve 100% correctly aligned audio, and all buffers of 4096 samples managed to maximally achieve 100% correctly aligned audio, all of which were either 17s or 22s long. The lowest amount of correctly aligned audio was as low as ~0% up to ~3.9% (17s or 22s long).

The charts for average and median results for the all networks without redundancy are displayed below.

Chart 1 which represents overall latency, sums up as with the individual networks with no redundancy, that there were clear differences between low buffers and high buffers in latency, although, as with
the individual networks, higher periods only affected the latency very slightly. Lowest average latency was at ~87.3ms, which had the buffers of 1024×6 samples, which achieved an average of ~36% amount of correctly aligned audio. The highest average latency was ~266.5ms with the buffer of 4096×3 samples, which achieved an average of ~88.3% amount of correctly aligned audio.

The standard deviation of the average latency was quite high and ranged ~33.9-35ms (lower sized buffers), ~50-52ms (mid-sized buffers) and ~81-85ms (higher sized buffers) which summed up as with the individual networks, that the variation was large.

The lowest latency achieved was around ~21ms for buffers of 1024 samples (all periods), ~43ms for buffers of 2084 samples (all periods) and ~91ms for 4096 samples (all periods). The highest latency was ~159-162ms for buffers of 1024 samples, ~232-275ms for buffers of 2048 samples and ~460-464ms for buffers of 4096 samples.

Chart 2, which represents the overall average and median amount of correctly aligned audio for the networks with no redundancy, shows that there were clear correlations between high buffers and amount of correctly aligned audio, and vice versa. As with the latency test, the number of periods did only affect the result slightly. The best average result gave the buffers of 4096×4 samples with ~89% correctly aligned audio, which had the average latency of ~255.5ms, which was about 11ms less than the longest achieved average latency (see above). The worst average result gave the buffers of 1024×6 samples with ~36% correctly aligned audio which had the average latency of ~87.3ms, which was also the shortest average latency (see above).

The standard deviation for average amount of correctly aligned audio was quite high, and ranged ~20-23% (lower sized buffers), ~33.5-34.4% (mid-sized buffers) and ~24.3-24.8% (higher sized buffers) which shows, that the variation was quite large.

All buffer types managed to achieve at least one audio stream that had 100% correctly aligned audio (22s or 17s long). The lowest amount of correctly aligned audio was as low as 0% up to ~1.5% (22s or 17s long).
6.1.2 Tests with Redundancy of 2

This sub-subchapter presents the analysis of the tests with redundancy of 2, network by network, and finally comparing all thirteen buffer settings over all networks.

The charts for average and median results for the baseline LAN network with redundancy of 2 are displayed in APPENDIX AC.

Chart 17, which represents latency for the LAN network with redundancy of 2, shows that there were clear differences between low buffer and high buffer in latency, although higher periods did not seem to affect the total latency in greater amounts. Lowest average latency was at ~51.8ms, which had the buffers of 1024×6 samples, which achieved an average of ~39.6% amount of correctly aligned audio. The highest average latency was ~251.8ms with the buffer of 4096×4 samples, which achieved an average of ~87% amount of correctly aligned audio.

The standard deviation of the average latency was quite high and ranged ~18.8-21ms (lower sized buffers), ~39.4-41.3ms (mid-sized buffers) to ~77-84.9ms (higher sized buffers), which shows that the variation was large.

The lowest latency achieved was around ~21ms for buffers of 1024 samples (all periods), ~43.8ms for buffers 2048 samples (all periods) and ~91ms for buffers of 4096 samples (all periods). The highest latency was ~90-139ms for buffers of 1024 samples, ~185-232ms for buffers of 2048 samples and ~371.4ms for buffers of 4096 samples.

Chart 18 which represents the average and median amount of correctly aligned audio for the LAN network with redundancy of 2, shows that there were clear correlations between high buffers and amount of correctly aligned audio, and vice versa. As with the latency test, the number of periods did only affect the result slightly. The best average result gave the buffers of 4096×2 samples with ~89.5%
correctly aligned audio, which had the average latency of ~251.4ms, which was also the one with longest average latency. The worst average result gave the buffers of 1024×4 samples with ~42.9% correctly aligned audio which had the average latency of ~52.9ms, about the same result as the best achieved average latency result (~1ms more) (see above).

The standard deviation for average correctly aligned amount of audio was quite high, and ranged from ~23.7-27.2% (lower and higher sized buffers) to ~34.5% (mid-sized buffers), which shows that the variation was large.

All buffer types managed to achieve at least one audio stream that had 100% correctly aligned audio (22s or 17s long). The lowest amount of correctly aligned audio was as low as 0.2% up to ~3.9% (22s or 17s long).

The charts for average and median results for the “Netherlands” network with redundancy of 2 are displayed in APPENDIX AD.

Chart 19 which represents latency for the “Netherlands” network using redundancy of 2 shows that there were clear differences between low buffer and high buffer in latency, although higher periods did not seem to affect the total latency in greater amounts. Lowest average latency was at ~125.24ms, which had the buffers of 1024×4 samples, which achieved an average of ~33.9% amount of correctly aligned audio. The highest average latency was ~312.6ms with the buffer of 4096×2 samples, which achieved an average of ~87.3% amount of correctly aligned audio.

The standard deviation of the average latency was quite high and ranged ~16.2-17.5ms (lower sized buffers), ~35-38ms (mid-sized buffers) to ~70.9-87.9ms (higher sized buffers), which shows that the variation was large.

The lowest latency achieved ranged ~66.5-89.4ms for buffers of 1024 samples (all periods), ~89.6-92ms for buffers of 2084 and ~183.5 for 4096 samples (all periods). The highest latency was ~162.5ms for buffers of 1024 samples, ~278.5ms for buffers of 2048 samples and ~464ms for buffers of 4096 samples.

Chart 20, which represents the average and median amount of correctly aligned audio for the “Netherlands” network using redundancy of 2, shows that there were clear correlations between high buffers and amount of correctly aligned audio, and vice versa. As with the latency test, the number of periods did only affect the result slightly. The best average result gave the buffers of 4096×3 samples with ~88.5% correctly aligned audio, which had the average latency of ~308.5ms, which was about 4ms less than the longest achieved average latency (see above). The worst average result gave the buffers of 1024×6 samples with ~33.79% correctly aligned audio which had the average latency of ~125.28ms, about the same result as the best achieved average latency result (see above).

The standard deviation for average amount of correctly aligned audio was quite high, and ranged ~13.5-14.7% (lower sized buffers), ~32.7-33.8% (mid-sized buffers) and ~25-26% (higher sized buffers) which shows that the variation was large.
The buffer types of 1024 samples managed to maximally achieve 66.4-100% correctly aligned audio, the mid-sized buffer types of 2048 samples managed to maximally achieve 100% correctly aligned audio, and all buffers of 4096 samples managed to maximally achieve 100% correctly aligned audio, all of which were either 17s or 22s long. The lowest amount of correctly aligned audio was as low as ~0.1% up to ~3.1% (17s or 22s long).

The charts for average and median results for the “USA” network with redundancy of 2 are displayed in **APPENDIX A**.

**Chart 23**, which represents latency for the “USA” network with redundancy of 2, shows that there were clear differences between low buffer and high buffer in latency, although higher periods did not seem to affect the total latency in greater amounts. Lowest average latency was at ~89.7ms, which had the buffers of 1024×3 samples, which achieved an average of ~39.8% amount of correctly aligned audio. The highest average latency was ~271.8ms with the buffer of 4096×2 samples, which achieved an average of ~84.9% amount of correctly aligned audio.

The standard deviation of the average latency was quite high and ranged ~18.5-20ms (lower sized buffers), ~36.7-39.7ms (mid-sized buffers) to ~80.5-83.3ms (higher sized buffers), which shows, as with the other networks, that the variation was large.

The lowest latency achieved was around ~43.6-44.3ms for buffers of 1024, ~45-90.2ms for buffers of 2048 samples (all periods), ~90-91ms for buffers of 4096 samples (all periods). The highest latency was ~116-139ms for buffers of 1024 samples, ~232ms for buffers of 2048 samples and ~371.4-460ms for buffers of 4096 samples.

**Chart 24**, which represents the average and the median amount of correctly aligned audio for the “USA” network with redundancy of 2, shows that there were clear correlations between high buffers and amount of correctly aligned audio, and vice versa. As with the latency test, the number of periods did only affect the result slightly. The best average result gave the buffers of 4096×2 samples with ~84.9% correctly aligned audio, which had the average latency of ~271.7ms, which was also the longest achieved average latency (see above). The worst average result gave the buffers of 1024×4 samples with ~37.8% correctly aligned audio which has the average latency of ~92.2ms, about 2.5ms more than the best achieved average latency result (see above).

The standard deviation for average amount of correctly aligned audio was quite high, and ranged ~19.8-21.5% (lower sized buffers), ~32.3-34.8% (mid-sized buffers) and ~27.6-28.8% (higher sized buffers) which shows, that the variation was large.

The buffer types of 1024 samples managed to maximally achieve ~94.2-98% correctly aligned audio, the mid-sized buffer types of 2048 samples managed to maximally achieve 100% correctly aligned audio, and all buffers of 4096 samples managed to maximally achieve 100% correctly aligned audio, all of which were either 17s or 22s long. The lowest amount of correctly aligned audio was as low as ~0% up to ~2% (17s or 22s long).
The charts for average and median results for the **all networks** with redundancy of 2 are displayed below.

**Chart 3** which represents overall latency for the networks with redundancy of 2, sums up as with the individual networks, that there were clear differences between low buffer and high buffer in latency, although higher periods only affected the latency very slightly. Lowest average latency was at ~89.6ms, which is had the buffers of 1024×6 samples, which achieved an average of ~39.6% amount of correctly aligned audio. The highest average latency was ~278.6ms with the buffer of 4096×2 samples, which achieved an average of ~87.2% amount of correctly aligned audio.

The standard deviation of the average latency was quite high and ranged ~35-36.3ms (lower sized buffers), ~52.9-54.3ms (mid-sized buffers) and ~80.7-87.7ms (higher sized buffers) which summed up what could be seen with the individual networks, that the variation was large.

The lowest latency achieved was around ~21.1-21.8ms for buffers of 1024 samples (all periods), ~43.8ms for buffers of 2084 samples (all periods) and ~91ms for 4096 samples (all periods). The highest latency was ~162ms for buffers of 1024 samples, ~278.5ms for buffers of 2048 samples and ~464.2ms for buffers of 4096 samples.

**Chart 4** which represents the overall average and median amount of correctly aligned audio, shows that there were clear correlations between high buffers and amount of correctly aligned audio, and vice versa. As with the latency test, the number of periods did only affect the result slightly. The best average result gave the buffers of 4096×2 samples with ~87.2% correctly aligned audio, which had the average latency of ~278.6ms, which was also the longest achieved average latency (see above). The worst average result gave the buffers of 1024×6 samples with ~38.2% correctly aligned audio which had the average latency of ~89.3ms, which was also the shortest average latency (see above).

The standard deviation for average amount of correctly aligned audio was quite high, and ranged ~20.8-21.9% (lower sized buffers), ~33.9-34.4% (mid-sized buffers) and ~25.9-27.1% (higher sized buffers) which shows that the variation was large.
All buffer types managed to achieve at least one audio stream that had 100% correctly aligned audio (22s or 17s long). The lowest amount of correctly aligned audio was as low as 0% up to ~1% (22s or 17s long).

### 6.1.3 Comparison of the Redundancies

This chapter will analyse the comparison of no redundancy and redundancy of 2. The tests results are divided into four parts:

1. All buffers
2. Buffer of 1024 samples (all periods)
3. Buffer of 2048 samples (all periods)
4. Buffer of 4096 samples (all periods)

**Chart 5** and **Chart 6** below displays the test data for all buffers. The test with no redundancy gave an average latency result of ~150.8ms (standard deviation: ~85.6ms) with an average of ~57.9% correctly aligned audio (standard deviation: ~33.6%), while the test with redundancy of 2 gave an average latency result of ~157.4ms (standard deviation: ~90.9ms) with an average of ~58.5% (standard deviation: ~33.4%) correctly aligned audio.
Chart 5: Latency when comparing no redundancy with redundancy of 2. Error bars is standard deviation.

![Latency comparison chart](image)

<table>
<thead>
<tr>
<th></th>
<th>No redundancy</th>
<th>Redundancy of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average latency (ms)</td>
<td>150,808</td>
<td>157,490</td>
</tr>
<tr>
<td>Median latency (ms)</td>
<td>136,077</td>
<td>136,100</td>
</tr>
</tbody>
</table>

Chart 6: Amount of correctly aligned audio, no redundancy versus redundancy of 2. Error bars is standard deviation.

![Correctly aligned audio chart](image)

<table>
<thead>
<tr>
<th></th>
<th>No redundancy</th>
<th>Redundancy of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average %</td>
<td>57,931%</td>
<td>58,556%</td>
</tr>
<tr>
<td>Median %</td>
<td>50,806%</td>
<td>52,444%</td>
</tr>
</tbody>
</table>

Chart 7 and Chart 8 below displays the test data for buffers of 1024 samples. The test with no redundancy gave an average latency result of ~87.9ms (standard deviation: ~34.3ms) with an average of ~38.2% correctly aligned audio (standard deviation: ~22%), while the test with redundancy of 2 gave an average latency result of ~90.2ms (standard deviation: ~39.1ms) with an average of ~39.1% (standard deviation: ~21.4%) correctly aligned audio.
Chart 7: Latency when comparing no redundancy with redundancy of 2 using buffer of 1024 samples. Error bars is standard deviation.

Chart 8: Amount of correctly aligned audio comparing redundancies when using buffer of 1024. Error bars is standard deviation.
In **APPENDIX AG** the charts for the 2048 and 4096 buffers are displayed.

**Chart 25** and **Chart 26** shows the test data for buffers of 2048 samples. The test with no redundancy gave an average latency result of ~147.8ms (standard deviation: ~51.3ms) with an average of ~59.2% correctly aligned audio (standard deviation: ~33.8%), while the test with redundancy of 2 gave an average latency result of ~152.8ms (standard deviation: ~39.1ms) with an average of ~60.7% (standard deviation: ~34%) correctly aligned audio.

**Chart 27** and **Chart 28** shows the test data for buffers of 4096 samples. The test with no redundancy gave an average latency result of ~260.5ms (standard deviation: ~82.7ms) with an average of ~88.1% correctly aligned audio (standard deviation: ~24.6%), while the test with redundancy of 2 gave an average latency result of ~277.2ms (standard deviation: ~83.9ms) with an average of ~87.1% (standard deviation: ~34%) correctly aligned audio.

6.1.4 **Erroneous “Sweden” Network**

The tests with the network called “Sweden” did not produce any reliable results, since its latencies were lower than the LAN network which should have given more optimal results, and no clear difference between the buffers of 2048 sample and buffers of 4096 samples could be found. The probable cause of this was that the since there were pre-programmed dropouts in the WAN simulation, there were still sounds from the audio file played earlier in the series transmitted when the recording started. This caused the latency analysis not to function since it did not check if the sound was correct, only measuring the number of samples from the start of the recording until the first sample that were not “[ -inf ]” or under the threshold.

For this reason, all results of the “Sweden” networks, both with no redundancy and with redundancy of 2 were excluded from the analysis. For the sake of completeness, the charts for the “Sweden” network results are still available in **APPENDIX AA** (no redundancy) and **APPENDIX AE** (redundancy of 2).
7 Conclusions

This chapter draws conclusions based on the results and discusses if it was possible to answer the research questions.

7.1 Conclusion and Discussions Drawn from the Analysis and Result

This study showed support for clear correlations between latency and number of buffers, and that the larger buffer used the more correctly aligned audio was achieved. However, the result also showed that the longest average latency did not necessarily produce the best average amount of correctly aligned audio, neither did the shortest average latency necessarily produce the least amount of correctly aligned audio. Likewise, the average most correctly aligned audio did not necessarily produce the longest average latency, and the average least correctly aligned audio did not necessarily produce the shortest average latency.

The study shows a support that a distributed client-server solution produced a quite low amount of correctly aligned audio, even though high buffers were used, which as a side effect causes a rather long latency. The average result had a high standard deviation, which means that the results had a large variation. However, by looking at the average and median values, the result did not show any significant surprise, since naturally using higher buffers should also increase the latency and amount of correctly aligned audio.

The study excluded that greater number of periods with the buffers affected the results to any large extent, which is probably caused by how Jack and JackTrip functions, rather than a universal phenomenon with NMP and streaming of audio.

Using a redundancy of 2 with Jacktrip did only increase the amount correctly aligned audio with a few percentage points in comparison with no redundancy, and the study also showed that it will add extra latency with about 4-6%.

7.2 Network Comparisons

Since the latency was quite high, and as described in the problem description under section 3, with latencies between 50-150ms, musicians will have problems keeping pace which will compromise the music, and if the delay is over 150ms it’s basically impossible to play together (Chew, et al. 2005). This will obviously exclude all higher buffers out of this discussion since they will produce too high latencies.

In the LAN test with no redundancy, the lowest buffers (1024-2048 samples) achieved an average latency of ~50-108ms and managed to produce an average of ~41-65% correctly aligned audio, and with redundancy of 2, ~51-108ms average latency produced an average of ~42-64% correctly aligned audio, both with buffer settings of 1024-2048.

In the WAN with the “USA” network with the lowest buffers (1024 samples) achieved an average latency of ~88-91ms for no redundancy and an average of ~88-92ms latency for redundancy of 2, which produced an average of ~38-44% correctly aligned audio for no redundancy and ~37-39% correctly aligned audio for redundancy of 2.
In the WAN test with the “Netherlands” network, the smallest buffers (1024 samples) had an average latency of ~120ms for no redundancy and average latency of ~125ms for redundancy of 2, and they produced an average of ~30–31% correctly aligned audio for no redundancy and ~33–35% correctly aligned audio for redundancy of 2.

The WAN simulation clearly showed support for that using WAN will produce higher latencies and less amount of correct audio in comparison of using a LAN, and that using a client-server model might not be optimal for this kind of service. For musicians, using a client/server solution might not produce enough low latency without losing too much audio quality when communicating over WAN. This do not seem to be the case with using LAN, where a client/server solution should function well.

The method used to analyse the results from a WAN simulation with pre-programmed dropouts did not function very well, which excluded the “Sweden” network out of the results.

7.3 Answers of Research Questions
The research questions are answered below.

1. Are there correlations between latency, dropouts and jitter regarding the performance of NMP?
Yes. The study shows that there is a clear correlation between the choice of having a low latency and having a higher quality audio stream without dropouts and jitter.

2. What impact do redundancy have on the performance?
   a. Will it achieve significant better results regarding the amount of correctly aligned data?
No. The results from using redundancy of 2 did not achieve much of an impact of the result at all. Using lower number of buffers showed a slight better performance regarding the amount of correctly aligned audio, but in opposite, surprisingly higher number of buffers caused it to perform worse with a few percentage points less correctly aligned audio.

   b. Will it cause any other drawbacks such as increased latency?
Yes. This experiment did not include any measurement of bandwidth requirements, but according to Cáceres & Chafe (2010b) the requirements for the bandwidth will increase. The experiment did however show that the latency increased with a 4-6% in comparison with using no redundancy.

3. Based on acceptable latency for musicians declared by Chew, et al. (2005), how jitter free/correct streaming is it possible to achieve without passing that limit in a LAN or WAN environment?
Depending on type of the network (LAN or WANs), the amount correctly aligned audio ranged from ~30% to 65%. The most optimal results were achieved within LAN environment (~41-65%, ~50-108ms), the second best was the WAN simulation (~37-44%, ~88-92ms) with following settings: latency of ~10.2ms and jitter of 10ms, with a bandwidth of ~27mbit (“USA”) and the worst result were achieved with a WAN simulation (~30-35%, ~120-125ms) with following settings: latency of ~18.6ms and 17.6ms jitter, with a bandwidth of ~16mbit (“Netherlands”).
8 Discussion

The aim of this study was to benchmark NMP with different settings in different scenarios, however the method and system that was developed to achieve this is perhaps of more importance since it is very possible to reuse it for more research on this topic. Much time has been put down on creating this system to automate it to be able to handle a big set of data.

The results and analysis from this study could be used to set up guidelines for system administrators, network technicians, creators and companies who wants to start services for low latency audio communications for musicians and other interested. Its method can be used to make more tests for those who are developing NMP services and/or engines.

The method used for this study worked fine for ¾ of tests. The “Sweden” network did not work well with the processing of the data especially not with the latency test. A probable cause of this was that since the WAN simulation used dropouts, the built-in wave iterator of JackTrip caused the last received packet to play for a while because of the dropped data, and that looped waveform did still play when next recording started. Processing of the latency data only checks whether there is something recorded or not, not what is recorded.

A lot of tests had to be redone quite late in the project, because of a bug where the buffer settings did not automatically get applied to JackTrip. This was eased up a bit by the fact most of the surrounding scripts that made the tests work were already developed, it was merely just to redo the tests. A big mistake was that there were no pilot tests done, which would have caught this bug before starting the real testing.

It is important to remember that all the results achieved in this study was made in a completely virtualized setup, therefore they might not completely reflect the results from a scenario using real LAN or WAN.

Another aspect that must be thought of is that this method used phase cancelling method to extract the difference between the original file and the audio sent through the network. A waveform that has been iterated with the wave table mechanism in JackTrip might sound fine for a human listening, but since it would not be sample exact as the original file, it would, in this system, be counted as an error anyway.

An internet QoS which suffers from the latencies and weak audio quality presented in this study, might not produce results that would work for a client/server audio service. A solution for this would be to use a method were latency was not that important to be low, for example using a synchronised metronome, either a visual one or one with audio clicks. Although, this could lead to results where musical improvisation and ‘jamming’ would be harder and suffer from individual expressions, and moreover unplanned tempo variation would be impossible. On the other hand, this could work fine for practicing music over distances for artists and bands that plays music that are arranged and structured. Using a great number of buffers to get high audio quality with this solution would theoretically even work for concerts.

The probable reason why there are a lot of dropouts is because the audio stream is uncompressed to be able to minimize the time for the computer to process the audio data before transmitting and
playing data audio, and that increases the demand of bandwidth. A solution for this would be to develop a compression algorithm that is very fast to both compress and decompress, to be able to decrease the bandwidth demand without adding too much latency. For remote audio streaming services, special user clients could be developed, where metronomes could be an option if the dropouts are too extensive that the buffers need to be increased. The clients could use a simple button to test how much latency is added and how much of the audio data that is dropped out, using for example a method reminding of the one developed in this study.
9 Future Work

This study focused on technical aspects of benchmarking of how much latency and how much of the audio that managed to go through flawlessly with an UDP stream using JackTrip. There was no, or very little, focus on how the audio that were streamed subjectively would be perceived. It’s easy to look at an audio file and draw certain technical conclusions out of it, but that might not paint the whole picture, since the experience of how it sounds might be better or worse depending of situations and subjects. A suggestion would be that a study would be made with real human subjects, blind listening to the audio, either after listening to the original, or just pre-emptively listening to the streamed audio, and filling out a review of how they perceived it.

The experiment in this study used a distributed client-server model. A suggestion would be doing an experiment of a technical benchmark using the peer to peer model. That would most likely result in shorter latencies and more flawless audio streams.

It was impossible to know exactly where the latency and dropouts/jitter were created in this experiment. Also, the way the data was analysed it did not explain why jitter/dropouts were created, nor if it was in fact jitter or dropouts that were the cause of not flawlessly sent audio data. This suggests that a more detailed study of smaller segments of the experiments would be good, to determine where it would be suitable to work on better solutions for this type of communication.

Since this experiment was solely conducted with JackTrip it would be interesting to see benchmarks with similar packages, such as Soundjack, jack-tools, Beatme, StreamBD and Jamberry in terms of latency and correctly aligned audio. The method developed to conduct and analyse the data could quite easily be adapted to be used with any other setup.

This experiment is only made up with virtualized machines with virtual switches and with WAN-simulations, which suggests that an experiment with real LAN-environment and Internet could be conducted with the method developed for this experiment. Experiments with WAN would probably also introduce clock synchronization drift (see section 2.1.5) as an extra layer of audio problems. If using WAN-simulation again, a suggestion is to use bigger dataset of WAN-simulation settings.

Since the analysis of the data, from the test with the WAN-simulation which had dropouts pre-programmed, did not work out very well, an experiment where dropouts are extensively tested with JackTrip and/or similar engines would be interesting. It would evaluate how they act under more extreme circumstances. There should be significant difference between a stream without redundancy and one with redundancy of at least 2, since in the latter case, it will be possible to replace the lost and late packets.
References


37
Appendix A – Graphical overview of scripts

Figure 7: A graphical overview of the scripts acting in the testing process
#!/bin/bash
#
# This script loads sample data export files from Audacity, and counts
# the [ inf ] (null values) lines and lines under the threshold. Every line represents one
# sample, and therefore represents how much latency there is from start to first sample.
# Outputs to a CSV-file with semicolons as separator.
#
# Parent directory of where CSV-files will be placed
outputbase="/home/daniel/Desktop/CSV-files"

# Lists all available subdirectories in the parent directory
outputpath=( `ls -l $outputbase` )

# The script should be executed from the path where all sample-data files are located
echo "Please execute script from parent input folder"
parentpath=`pwd`

countmenu=0
echo "Which test is this?"

# The loop below will loop through all subdirectories in the output parent directory
# giving the one executing the script a choice where to put the CSV-files (depending of test)
for z in ${outputpath[@]}
    do
        echo "[$countmenu] -> $z"
        ((countmenu++))
    done

read -p 'Specify test (number): ' choice

read -p 'Do the files contain *.txt? (y/n): ' txtchoice

if [[ $txtchoice == *"y"* ]]
    then
     txt=".txt"
    elif [[ $txtchoice == *"n"* ]]
    then
     txt=""
    else
     echo "Something went wrong. Did you choose y or n in the .txt-question?"
    fi

# Counts the number of tests (directories with the sample-data files)
numberoftests=`ls -l | wc -l`
((numberoftests--))

# Loops through all tests in the test-array
for y in ${tests[@]}
    do
        # Counts how many files (sample-data) there are in the subdirectory
        # referred to in the array
        numberoffiles=`ls -1 $y | wc -l`

        # Puts all sample-data files in an array
        file=`ls -1 $y`

        # Since there are two additional meta files included with data about
        # latency compensation, the total numberoffiles must be divided by three
        numberoffiles=$(($numberoffiles/3))

        echo numberoffiles

        # Declares some arrays used to count the sample-data lines.
        declare -a inf
        declare -a threshold
        declare -a latency

        # If the first file does not have the count number of 0
        # (just "sample-data" or "sample-data.txt")
        # it's renamed to sample-data0 (or sample-data0.txt),
        # to be able to be used in the
        # counting loop below
        if [ -f "$y/sample-data.txt" ]
            then
             mv "$y/sample-data.txt" "$y/sample-data0.txt"
        fi
if [ -f $y/sample-data.txt ]
    then
echo "Filename *.txt. Aborting..."
    exit 127
fi

count=0
while [ count -ne $numberoffiles ]
do
    echo $count " - " $y " - " tests[count]
# Counts all "[ inf ]" lines
inf[count]=`cat $y/sample-data$count.txt | grep inf | wc -l`

# Counts all lines below the threshold of -90dB
threshold[count]=`grep "\(-9[0-9]\)|\(-10[0-9]\)|\(-11[0-9]\)|\(-12[0-9]\)|\(-13[0-9]\)|\(-14[0-9]\)|\(-15[0-9]\)|\(-16[0-9]\)|\(-17[0-9]\)|\(-18[0-9]\)|\(-19[0-9]\)|\(-20[0-9]\)" $y/sample-data$count.txt | wc -l`

# Sums the number of [ inf] lines with the number of lines below the threshold
latency[count]=`($inf[count] + $threshold[count])" samples"
buffer=""
buffertype=""

# Adds the buffer-type to a variable (buffer-type is created with Powershell and is encoded with unicode, "iconv" decodes and outputs it to the chosen ascii encoding (utf8)
buffer=`iconv -f unicode -t utf8 $y/sample-data$count.buffer`
buffertype=`iconv -f unicode -t utf8 $y/sample-data$count.buffertype`

# If-statement to choose correct buffertype
if [ [ buffer == "1024" ]]
    then
        if [ [ buffertype == "hythm" ]]
            then
                latencycompensation[count]=1447
            else
                echo "Something went wrong, latencycompensation 1024"
                exit 127
        fi
        else
            if [ [ buffertype == "2048" ]]
                then
                    if [ [ buffertype == "hythm" ]]
                        then
                            latencycompensation[count]=2441
                        else
                            echo "Something went wrong, latencycompensation 2048"
                            exit 127
                    fi
                else
                    if [ [ buffertype == "4096" ]]
                        then
                            if [ [ buffertype == "hythm" ]]
                                then
                                    latencycompensation[count]=4426
                                else
                                    latencycompensation[count]=4112
                            else
                                echo "Something went wrong, latencycompensation 4096"
                                exit 127
                        fi
                    else
                        echo "Something went wrong in latencycompensation. Aborting.."
                        exit 127
                fi
            fi

else
    echo "Something went wrong in latencycompensation. Aborting.."
    exit 127
fi
fi

echo "latencycompensation: " ${latencycompensation[$count]} " samples"

# Adds a number to the count so the loop will be broken when all files is counted
((count++))

done

# Removes the output file if it already exists. If it's not deleted the new values will be appended to it. The output filename is based on what the test directory is called
outputfile=/outputpath/$outputbase"".csv
if [ -f $outputfile ]
then
  rm $outputfile
fi

# Adds the headers to the CSV file
printf "%snumber;latency (samples 44.1khz);latencycompensation\n" >> $outputfile

count=0
for i in ${latency[@]}
do
  printf "%s\n" $count"";"";${latencycompensation[$count]} >> $outputfile
  ((count++))
done

# Outputs the array to a csv file

done

2>&1 | tee ~/logs/inf.log

date="date '+%Y-%m-%d %H:%M:%S'"

echo "-- End of log -- $date" >> ~/logs/inf.log
Appendix C – daemon_cli.sh (Linux Bash-Script)

This script is used as a loop which starts and kills QjackCTL and Jackdbus.
They must be closed before new buffer settings are applied.
The script is listening to the files called “jack.status”, “daemon.status”,
“buffer.status” and “period.status” which are edited via SSH by the main-loop script
running on the management machine.
The reason why this script is needed is because QjackCTL and Jackdbus must
be started from the Desktop. This script is running on both clients.

Initializing the status files when starting up. 
runstatus="n"
loop=0

```
echo "y" > ~/Desktop/status/jack.status
echo "n" > ~/Desktop/status/jack.killed
echo "y" > ~/Desktop/status/daemon.status
```

The daemon loop begins here, and will loop until "daemon.status" gets a "n" sent from
management machine

```
date=`date '+%Y-%m-%d %H:%M:%S'`
echo ""
echo "---- New iteration at $date ----"
echo ""
dateloop=`date '+%Y-%m-%d %H:%M:%S'`
echo "Loop $loop at $dateloop"

# Reads the current status in jack.status
jackstatus=`cat ~/Desktop/status/jack.status`

# Checks whether the status is "y" in the jack.status (via the variable $jackstatus)
# If it’s true, then it will execute these if-statements, which will try to
# start QjackCTL and Jackdbus.
if [[ $jackstatus == "y" ]]
then
  echo "Trying to start Qjackctl"
  sleep 2
  if ! pgrep -x "qjackctl" > /dev/null
  then
    buffer=`cat ~/Desktop/status/buffer.status`
    period=`cat ~/Desktop/status/period.status`
    echo "Starting Jackdbus"
    jackdbus -P99 -dalsa -r44100 -p$buffer -n$period -S -D -Chw:AudioPCI -Phw:AudioPCI &>/dev/null &
    disown
    echo "Starting QjackCTL"
  fi
fi
```

This while loop will iterate while jack.status = "y"

```
jackstatus=`cat ~/Desktop/status/jack.status` # Checks status
```

This if statement checks whether qjackctl runs
if pgrep -x "qjackctl" > /dev/null
then
  # runstatus is only used to avoid putting out new lines in
  # every iteration in the CLI output with the echo command below.
  if [[ runstatus == "n" ]]
  then
    echo "QJackCTL started"
  fi
  runstatus="y" # By changing this, the echo command above will only
  print out once
# Sends "n" to jack.killed, which the main loop of the managemachine reads.
echo "n" > ~/Desktop/status/jack.killed

# Greps the current buffer/period status
currentframe=`cat ~/.config/rncbc.org/QjackCtl.conf | grep Frames= | head -1 | sed 's/[^0-9]//g'`

currentperiods=`cat ~/.config/rncbc.org/QjackCtl.conf | grep Periods= | head -1 | sed 's/[^0-9]//g'`

datestatus=`date '+%Y-%m-%d %H:%M:%S'`

# Changes status in jack.kill which main loop at management computer reads
runstatus="y"

# If qjackctl isn't started, this else statement will be executed
else
if ! pgrep -x "qjackctl" > /dev/null
then
if [[ "runstatus" == "*n*" ]] then
    echo "Not started yet..."
    echo "Trying to kill first..."
    pgrep "jackd" | xargs kill -9
    pgrep "qjackctl" | xargs kill -9
    kill $pid
fi
oldqjackctl=($(pgrep -x "qjackctl"))
for i in [oldqjackctl[@]]
    do
    # Tries to kill qjackctl and jackdbus in several ways
    kill $i
    pgrep "jackd" | xargs kill -9
    pgrep "qjackctl" | xargs kill -9
    kill $pid
    wait
done

if pgrep -x "qjackctl" > /dev/null
then
    buffer=`cat ~/Desktop/status/buffer.status`
    period=`cat ~/Desktop/status/period.status`
    echo "Starting jackdbus"
    /usr/bin/jackd -P99 -dalsa -r44100 -pbuffer -n$period -S -D -Chw:AudioPCI - Phw:AudioPCI &>/dev/null &
    disown
default & export pid=$!
    sleep 2

    # If it's started, this will be executed
    if pgrep -x "qjackctl" > /dev/null
then
    echo "n" > ~/Desktop/status/jack.killed
    # Changing status in jack.kill which main loop at management computer reads
    runstatus="y"
    # By changing this, the echo command above will only print out once.

    currentframe=`cat ~/.config/rncbc.org/QjackCtl.conf | grep Frames= | head -1 | sed 's/[^0-9]//g'`
    currentperiods=`cat ~/.config/rncbc.org/QjackCtl.conf | grep Periods= | head -1 | sed 's/[^0-9]//g'`
    datestatus=`date '+%Y-%m-%d %H:%M:%S'`
    echo "Frame: $currentframe Periods: $currentperiods Status: $datestatus" >> ~/logs/jack_bufferstatus.log
    echo "QJackCTL started"
fi
else
    echo "Can't start QJackCtl. Something is wrong, killing script."
if exit 127
fi

if
  # If jack.status is "n" this inner loop will be killed
  if [[ jackstatus == "n" ]]
    then
      break
  fi
done
# End of if statement if jack.status = "y".
fi

if [[ jackstatus == "n" ]]
  # If jack.status = "n" then this if statement will be executed
  then
    echo "Checks if Qjackctl is killed"
    sleep 2
    if pgrep -x "qjackctl" > /dev/null
      # If qjackctl is running, then this will be executed
      then
        # Kills QjackCTL and jackdbus
        echo "Killing QjackCTL"
        pgrep "jackd" | xargs kill -9
        pgrep "qjackctl" | xargs kill -9
        kill $pid
        wait
      fi

    fi
  fi

while:
  # This while loop will iterate while jack.status = "n"
  do
    # Checks the current status of jack.status
    jackstatus=`cat ~/Desktop/status/jack.status`
    if ! pgrep -x "qjackctl" > /dev/null
      then
        # Kills QjackCTL and jackdbus
        echo "Killing QjackCTL"
        pgrep "jackd" | xargs kill -9
        pgrep "qjackctl" | xargs kill -9
        kill $pid
      wait
  fi
  else
    # $runstatus is only used to avoid putting out new lines in
    # at every iteration in the CLI output with the echo command below.
    then
      echo "y" > ~/Desktop/status/jack.killed
      echo "QjackCTL is killed"
      runstatus="n"  # By changing this, the echo command above will only
      # print out once.
    fi
  fi

else
  # $runstatus is only used to avoid putting out new lines in
  # at every iteration in the CLI output with the echo command below.
  then
    # If qjackctl and jackdbus is not killed, it will try again
    echo "Not killed yet.."
    echo "Trying again"
    pgrep "jackd" | xargs kill -9
    pgrep "qjackctl" | xargs kill -9
    kill $pid

    # Checking and killing once more
    oldqjackctl="$(pgrep -x "qjackctl")"
    for i in $oldqjackctl[
      do
        kill $i
        pgrep "jackd" | xargs kill -9
        pgrep "qjackctl" | xargs kill -9
        kill $pid
        wait
      done
    if ! pgrep -x "qjackctl" > /dev/null
      then
        if [[ $runstatus == "y" ]]
          then
            echo "QjackCTL now killed"
            echo "y" > ~/Desktop/status/jack.killed
        fi
      fi
    fi

  fi
runstatus="n"  # By Changing this, the echo command above will only print out once.

fi

fi

if ! pgrep -x "qjackctl" /dev/null
  # If qjackctl is not killed, it will break the kill the script
  then
    echo "Can’t kill QjackCtl. Something is wrong. Killing script."
    exit 127
  fi

fi

# If jack.status changes to "y" then the inner loop will break
if [[ $jackstatus == "y" ]]
  then
    break
fi

done

((loop++))

daemonrun=`cat ~/Desktop/status/daemon.status`

if [[ $daemonrun == "n" ]]
  # If mainloop at management computer tells everything is done, this loop will be broken
  then
    break
fi

done
Appendix D – jacktrip_client1_clientstart.sh (Linux Bash-Script)

```bash
#!/bin/bash
{
  date=`date '+%Y-%m-%d %H:%M:%S'
  echo ""
  echo "---- New log at $date ----"
  echo ""
  echo "Starting client.."
  jacktrip -c 192.168.8.11 -n 1 -o 10 --clientname o10
  # -c declares it's a client, and targets the ip-adress of Client 2 server instance.
  # -n declares number of audio channels
  # -o puts an offset from default port of JackTrip (4464)
  # --clientname declares an alias for the audio ports (in and out).
  } 2>&1 | tee -a ~/logs/JackTrip_Client1.log
  # Sends logfile to management computer
  rsync -a --rsync-path="rsync" -v -e ssh ~/logs/JackTrip_Client1.log daniel@10.208.15.184:/logs/
  
```
Appendix E – jacktrip_client1_serverstart.sh (Linux Bash-Script)

```bash
#!/bin/bash

date=`date '+%Y-%m-%d %H:%M:%S'

--- New log at $date ----

This start script is for Client 1 server, make sure QJackCTL is started.

# Redundancy choice is parsed from the main loop in the management computer.
if [[ $@ ]] then
    parse=($@)
    redundancy=${parse[0]}
else
    redundancy=1
fi

# The innerbuffer of Jacktrip was always set to 4.
queue=4

Redundancy = $redundancy
Queue = $queue

# There's an option which wasn't used in this test where the buffer under/over-run can use either zero or wavetable to fill up the gaps. In the test wavetable was always used. These choices are here printed out.
if [[ $zerochoice == "" || $zerochoice == "n" || $zerochoice == "N" ]] then
    echo "Using Wavetable."
    type="wave"
else if [[ $zerochoice == "y" || $zerochoice == "Y" ]] then
    echo "Using Zerounderrun"
    type="zero"
else
    echo "Please choose y/N at the zerounderrun option"
    exit 127
fi

Red_%$redundancy_-_Queue_"$queue_Underrun_"$type >> /home/daniel/serverstatus.txt

# Uploads the newly created files to Client 2.
```

1
rsync -v -e ssh /home/daniel/serverstatus* daniel@10.208.15.101:~/
wait

# There's an option which wasn't used in this test where the buffer under/over-run
# can use either zerounderrun or wavetable to fill up the gaps. In the test wavetable was
# always used, which is also the option that will be used if no -z is entered.
if [[ $zerochoice == '' || $zerochoice == "n" || $zerochoice == "N" ]] then
  echo "Starting server..."
  jacktrip -s -n 1 --redundancy $redundancy -q $queue --clientname o0
elif [[ $zerochoice == "y" || $zerochoice == "Y" ]] then
  echo "Starting server..."
  jacktrip -s -n 1 --redundancy $redundancy -q $queue --clientname o0 -z
else
  echo "Something went wrong, this shouldn't be printed.. exiting."
  exit 127
fi

# The if statement choose it's destination depending of what type of buffer underrun action that should
# be taken by JackTrip. "Zerounderrun" (-z) puts zeros if the buffer is empty, contrary to
# "wavetable" (no -z declared) which will loop the last sample until the buffer catches up again.
# -s declares that it's a server instance
# -n declares number of audio channels
# -o puts an offset from default port of JackTrip (4464)
# --redundancy declares if there should be redundancy in JackTrip for lost packets (at the cost of bandwidth)
# --queue declares internal buffer size of JackTrip.
# --clientname declares an alias for the audio ports (in and out).

} 2>&1 | tee -a ~/logs/JackTrip_Server1.log

# sends log to managementcomputer
rsync -a --rsync-path="rsync" -v -e ssh ~/logs/JackTrip_Server1.log daniel@10.208.15.184:~/logs/
Appendix F – jacktrip_client2_clientstart.sh (Linux Bash-Script)

```
#!/bin/bash
{

date=`date '+%Y-%m-%d %H:%M:%S'`

echo ""

echo "" ---- New log at $date ----"" 

echo ""

echo "This start script is for Client 2, make sure QJackCTL is started."

read -p "Press return to start client 2 client connection, ctrl-c to break...

echo "Starting client..."

jacktrip -c 192.168.0.10 -n 1 --clientname o0

# Starts Jacktrip on Client 2 (the machine that acts sending and receiving client)
# -c declares it's a client, and targets the ip-address of Client 1 server instance.
# -n declares number of audio channels
# -o puts an offset from default port of JackTrip (4444)
# --clientname declares an alias for the audio ports (in and out).
}
} 2>&1 | tee ~/logs/JackTrip_Client2.log

rsync -a --rsync-path="rsync" -v -e ssh ~/logs/JackTrip_Client2.log daniel@10.208.15.184:/logs/
```
Appendix G – jacktrip_client2_serverstart.sh (Linux Bash-Script)

```bash
#!/bin/bash
{
  date='date '+%Y-%m-%d %H:%M:%S''
  echo ''
  echo "---- New log at $date ----"
  echo ''
  echo "This start script is for Client 2, make sure QJackCTL is started, and that server on the Client 1 is started prior to this."

  readarray -t serverstatus < ~/.serverstatus.xml
  # Fetches the server status from the Client 1 server.
  # The XML file was uploaded here when Client 1 server was started was started.
  # It removes newlines.
  echo "Redundancy = " [serverstatus[0]]
  echo "Queue buffer = " [serverstatus[1]]
  echo "Zero or wave?: " [serverstatus[2]]

  if [[ [serverstatus[2]] == "wave" ]]
    then
      echo "Starting server.."
      jacktrip -s -o 10 -n 1 --redundancy [serverstatus[0]] --queue [serverstatus[1]] --clientname o10
    elif [[ [serverstatus[2]] == "zero" ]]
      then
        echo "Starting server.."
        jacktrip -s -o 10 -n 1 --redundancy [serverstatus[0]] --queue [serverstatus[1]] --clientname o10 -z
    else
      echo "Something went wrong, this shouldn't be printed.. exiting."
      exit 127
  fi

  # The if statement choose it's destination depending of what type of buffer underrun action that should
  # be taken by JackTrip. "Zerounderrun" (-z) puts zeros if the buffer is empty, contrary to
  # "wavetable" (no -z declared) which will loop the last sample until the buffer catches up again.
  # -s declares that it's a server instance
  # -n declares number of audio channels
  # -o puts an offset from default port of JackTrip (4464)
  # --redundancy declares if there should be redundancy in JackTrip for lost packets (at the cost of bandwidth)
  # --queue declares internal buffer size of JackTrip
  # --clientname declares an alias for the audio ports (in and out).
} 2>&1 | tee -a ~/logs/JackTrip_Server2.log

rsync -a --rsync-path="rsync" -v -e ssh ~/logs/JackTrip_Server2.log daniel@10.208.15.184:~/logs/
```
Appendix H – Latencyscript_Ecasound. (Linux Bash-Script)

```bash
#!/bin/bash
{

date="`date '+%Y-%m-%d %H:%M:%S'`"

# Starts a script that connects the correct audioports together. If the server is restarted it needs to be redone.
ssh daniel@10.208.15.177 '~/Desktop/Scripts/snapshot_client1.sh'
sleep 2

# If there are no parsed options, this if statement will be executed.
if ! [[ @ ]]
then
    read -p 'Session name: ' sessionvar
    read -p 'How many times do you want to loop?: ' count
    read -p 'Do you want to record via network (Y/enter) or via local audio interface? (n): ' record
    read -p 'Which soundsource do you want to use? (1) Music / (2) Rhythm: ' sourcechoice
else
    parse=($@)

    echo "Parse = $@

    # Name of the test (in this case the networks or local)
    sessionvar=${parse[0]}
    echo "Sessionvar = $sessionvar"

    # How many times should these options be recorded?
    count=${parse[1]}
    echo "count = $count"

    # If a "Y" is sent to this option, network recording will be done. A "n" will cause a local recording
    record=${parse[2]}
    echo "record = $record"

    # Which source audio file should be used? (1 or 2)
    sourcechoice=${parse[3]}
    echo "sourcechoice = $sourcechoice"
else
    echo "Something went wrong with parsing.."
    exit 127
fi

# This if statement will prepare the two variables $interfacein and $interfaceout with what connections that should be used, network or local.
if [[ sourcechoice == "1" || record == "Y" || record == "y" ]]
then
    # Network connection
    interfaceout="jack,o10"
    interfacein="jack,o8"
    echo "Interface out: $interfaceout"
    echo "Interface in: $interfacein"
elif [[ record == "n" || record == "N" ]]
then
    # Local connection
    interfaceout="jack,ecasound"
    interfacein="jack,ecasound"
    echo "Interface out: $interfaceout"
    echo "Interface in: $interfacein"
else
    echo "Please choose Y/enter/n at interface choice"
    exit 127
fi

# This if statement prepares the variable $sourcechoice with chosen sound source.
```
if [[ $sourcechoice == "1" ]]
then
testsound="Music.wav"
testfilename="Music"
seconds=30
echo "Choosing $testsound, for $seconds seconds"

elif [[ $sourcechoice == "2" ]]
then
testsound="Rhythm_8-160bpm.wav"
testfilename="Rhythm"
seconds=17
echo "Choosing $testsound, for $seconds seconds"
else
echo "You must choose a soundsource!"
exit 127
fi

# Cleaning the connections in QJackCtl..
aj -snapshot -x

# Paths for saving the wav files.
testsoundpath="/home/daniel/Desktop/"
ramdisk="/mnt/ramdisk/"

# The files will be saved in the directory where the script is executed in
# savepath="pwd" /
# Collecting what type of server preferences there are, to be able to mirror it in the filename.
serverstatus=`cat ~/serverstatus.txt`
# Collecting buffer preferences of jack, to be able to mirror it in the filename.
jackperiod=`cat ~/.config/rncbc.org/QjackCtl.conf | grep Frames= | head -1 | sed 's/\[^0-9\]//g'`
jacknperiod=`cat ~/.config/rncbc.org/QjackCtl.conf | grep Periods= | head -1 | sed 's/\[^0-9\]//g'`

# Collecting buffer preferences of jack, to be able to mirror it in the filename.

# This while loop creates a variable for the filename for the recorded files, and starts the playing and
# recording.
# the $loop puts the loop count in the filename
while [ $loop -lt $count ];
loop=$((loop + 1))
# Loop stops at $count
# Copies the testsound to ramdisk to achieve as latency free loading time as possible
wait

done

# Uses Ecasound to play and record sound.
echo "Playing $ramdisk/testsound to $interfacein and saving it to $ramdisk$name via $interfaceout"
ecasound -t:$seconds -a:1 -i:$testsoundpath -f:s16_1e,1,44100 -a:2 -i:$interfacein -o:$ramdisk$name -f:s16_1e,1,44100 -a:1 -o:$interfaceout -f:s16_1e,1,44100
wait

# Increases the loop variable
((loop++))

sleep 0.5

# Moves the audio file to the non volatile save path.
echo "Moving $ramdisk$name to $savepath"
mv $ramdisk$name $savepath
wait
```bash
rm $ramdisk$testsound
# Removes the saved test soundfile in the ramdisk

echo "Iteration: " $loop
done

# Copies all files to the Management computer
if [[ $interfaceout == "jack,o10" ]]; then
  syncvar=$sessionvar"_"$testfilename"_"$serverstatus"_"$jackperiod"_"$jacknperiod
  echo "Syncvar: $syncvar"
  # Rsyncs all files to the management computer
  rsync -a --rsync-path="mkdir -p ~/Desktop/Wav/$syncvar && rsync" -v -e ssh $syncvar
daniel@10.208.15.184:/~Desktop/Wav/
# This will start the trim_invert_merge-script on the management computer, which will process the newly recorded files
  ssh daniel@10.208.15.184="/bin/bash ~/Desktop/Scripts/trim_invert_merge-script.sh $syncvar"
elif [[ $interfaceout == "jack,ecasound" ]]; then
  syncvar=$sessionvar"_"$testfilename"_"$serverstatus"_"$jackperiod"
  echo "Syncvar: $syncvar"
  # Rsyncs all files to the management computer
  rsync -a --rsync-path="mkdir -p ~/Desktop/Wav/$syncvar && rsync" -v -e ssh $syncvar
daniel@10.208.15.184:/~Desktop/Wav/
# This will start the trim_invert_merge-script on the management computer, which will process the newly recorded files
  ssh daniel@10.208.15.184="/bin/bash ~/Desktop/Scripts/trim_invert_merge-script.sh $syncvar"
else
  echo "Something went wrong. This shouldn’t be written out.. Rsync."
  exit 127
fi

wait
}
$1 & 1 | tee -a ~/logs/Latencyscript.log

# Copies the log file to the management computer
rsync -a --rsync-path="rsync" -v -e ssh ~/logs/Latencyscript.log daniel@10.208.15.184:/logs/
```
#!/bin/bash
# This is a simple script which will sync files to OneDrive.
# It is running from Crontab and will execute every 30m.
{
    echo "" "" New logfile ----"
    wavfolders=`ls -1 ~/Desktop/Nav`
    scriptfiles=`ls -1 ~/Desktop/Scripts`
    logfiles=`ls -1 ~/logs`
    csvfiles=`ls -l ~/Desktop/CSV-files`
    for i in "${wavfolders[@]}"
    do
        rclone copyto -v ~/Desktop/Nav/ onedrive:Exjobb-Data/
    done
    for i in "${scriptfiles[@]}"
    do
        rclone copyto -v ~/Desktop/Scripts/ onedrive:Exjobb-Scripts/
    done
    for i in "${logfiles[@]}"
    do
        if ! [[ $i == "Onedrive"* ]] then
            rclone copyto -v ~/logs/ onedrive:Exjobb-Logs/
        fi
    done
    for i in "${csvfiles[@]}"
    do
        if ! [[ $i == "Onedrive"* ]] then
            rclone copyto -v ~/Desktop/CSV-files/ onedrive:Exjobb-CSV-files/
        fi
    done
    } 2>&1 | tee ~logs/Onedrive.log
    rclone copyto -v ~/logs/Onedrive.log onedrive:Exjobb-Logs/Onedrive.log
#!/bin/bash

# This simple script is executed from the main loop in the management computer.
# It will change the buffer and periods of QJackCTL, after checking if QJackCTL is killed.
# This script is available and executed on both clients.

date=$`date '+%Y-%m-%d %H:%M:%S'

# New log in qjackctl-buffer.sh on Client 1 at $date ----

input=($@)
killasswer="n"

while do
    # Check the status of jack.killed
    killed=`cat ~/Desktop/status/jack.killed`
    if [[ $killed == "n" ]]
        if [[ $killasswer == "n" ]]
            echo "Waiting for QJackCTL to be killed"
            killasswer="y"
            # This will change so that
            # so that the while loop won't
            # print out the echo line above
            # every iteration
        fi
    elif [[ $killed == "y" ]]
        echo "Something went wrong, this shouldn't happen. While waiting for QjackCTL to be killed. Killing script.."
        exit 127
    else
        echo "Waiting for QJackCTL to be killed"
        killasswer="y"
        # This will change so that
        # so that the while loop won't
        # print out the echo line above
        # every iteration
    fi
    break
    echo "If it reads that it's killed, it will break the while loop"
else
    echo "Waiting for QJackCTL to be killed"
    killasswer="y"
    # This will change so that
    # so that the while loop won't
    # print out the echo line above
    # every iteration
fi

# Kills jackdbus
pgrep "jackd" | xargs kill -9

# Reads the current buffer of the qjackctl config file.
currentframe=`cat ~/.config/rncbc.org/QjackCtl.conf | grep Frames= | head -1 | sed 's/[^0-9]//g'

wait

# Replaces the current buffer with the new one that was parsed from main loop in the management computer.
rpl Frames="$currentframe" Frames="${input[0]}" ~/.config/rncbc.org/QjackCtl.conf

wait

# Reads the current period of the qjackctl config file.

currentperiods=`cat ~/.config/rncbc.org/QjackCtl.conf | grep Periods= | head -1 | sed 's/[^0-9]//g'

wait

# Replaces the current period with the new one that was parsed from main loop in the management computer.
rpl Periods="$currentperiods" Periods="${input[1]}" ~/.config/rncbc.org/QjackCtl.conf
Appendix K – Scriptloop.sh (Linux Bash-Script)

#!/bin/bash
# This simple script sends the two sound sourcechoices
# choices to Latencyscript_Ecasound.sh, with the options
# parsed from main loop script in the management computer.
# After it’s done it will kill the tmux session that is
# currently running via the main loop script in the management
# computer.
{

date=`date '+%Y-%m-%d %H:%M:%S'`

echo "---- New log in Scriptloop at $date ----"

echo ""

echo "" 

input=($@)
# Arguments passed from tmux script with session (network) and number of iterations/test, third variable
# is whether the test should be via network ("Y") or locally ("N")

session=${input[0]}
iterations=${input[1]}
record=${input[2]}
sourcechoice1=1
sourcechoice2=2


echo "Sessionname: $session"

echo "Number of iterations/test: $count"

echo $session

# Starts the source choice 1.
~/Desktop/Scripts/./Latencyscript_Ecasound.sh $session $iterations $record $sourcechoice1

echo "~/Desktop/Scripts/./Latencyscript_Ecasound.sh $session $iterations $record $sourcechoice1"

# Starts the source choice 2.
~/Desktop/Scripts/./Latencyscript_Ecasound.sh $session $iterations $record $sourcechoice2

echo "~/Desktop/Scripts/./Latencyscript_Ecasound.sh $session $iterations $record $sourcechoice2"

date=`date '+%Y-%m-%d %H:%M:%S'`

} 2>&1 | tee -a ~/logs/Scriptloop.log

# Kills the TMUX-session in the main loop script in the management computer.
ssh daniel@10.208.15.184 'tmux kill-session -t daniel'
Appendix L – snapshot_client1.sh (Linux Bash-Script)

```
#!/bin/bash
{
    date="date '+%Y-%m-%d %H:%M:%S'"
    echo """" New log at $date ----"
    echo """
    # Simple script to disconnect and connect audio ports on Client 2 based on
    # an xml file using "aj-snapshot".

    aj-snapshot -x
    wait
    aj-snapshot -r ~/Desktop/Scripts/Client1.xml
}
2>&1 | tee -a ~/logs/snapshot.log
rsync -a --rsync-path="rsync" -v -e ssh ~/logs/snapshot.log daniel@10.208.15.184:~/logs/
```
Appendix M – threshold.sh (Linux Bash-Script)

```bash
#!/bin/bash

# This file counts how many samples over the threshold of -90dB and below and inf (none) that 
# exists in the data export files from Audacity. It is used with the files that are phase 
# inverted and compared to the original file. In that way it is possible to count how many 
# errors there are per sample basis. All samples that are over the threshold are counted 
# as an error. The result is output to a CSV-file.

outputbase="/home/daniel/Desktop/CSV-files"
outputpath="ls -1 "$outputbase"

echo "Please execute script from parent input folder" parentpath="pwd"
countmenu=0
echo "Which test is this?"
echo ""
for z in $outputpath[@]
do
echo "$countmenu -> $z" ((countmenu++))
done
read -p 'Specify test (number): ' choice
read -p 'In the base directory, are the both type of tests included (y/n)?' merged

# If both types of tests (latency and amount of correctly aligned lines) are present 
# in the same parent directory, search only for *erged.
if [[ $merged == -*y*- ]]
then
  # Counts number of tests
  numberoftests="ls -1 -d *erged | wc -l"
  # Add tests into an array
  tests="ls -1 -d *erged"
  read -p "Make sure that the directories are called "erged"
elif [[ $merged == -*n*- ]]
then
  # Counts number of tests
  numberoftests="ls -1 -d | wc -l"
  # Add tests into an array
  tests="ls -1 -d"
  read -p "Make sure that there are only merged tests in the base directory"
else
  echo "Did you choose y or n?"
  exit 127
fi

read -p 'Do the files contain *.txt? (y/n): ' txtchoice
# If the sample-data filenames have the suffix .txt, choose Y below.
if [[ $txtchoice == -*y*- ]]
then
txt=".txt"
echo "$tests"
numberoffiles="ls -1 $y | wc -l"
# Add files into an array
files="ls -1 $y"
# Takes away one, since they start with 0.
((numberoffiles--))
declare -a total
declare -a inf
declare -a threshold
declare -a zero
# Although, they start with zero, first file might lack a number,
```
```
# therefore a 0 is added to its filename, to be able to be used in the # count loop.
if [ -f $y/sample-data$count$txt ]
then mv $y/sampledataInicial $y/sample-data$count0$txt
fi

# Counts the number of total lines, and the number of lines that are under # the threshold.
count=0
while [ count -le numberoffiles ]
do
  echo $count " " "$y "{tests[$count]}
  # Counts the total lines. Needed for percent calculation.
total[$count]=`wc -l $y/sample-data$count$txt`
  echo "Total: "$total[$count]
  # Counts all "[ inf ]" lines.
  inf[$count]=`grep inf $y/sample-data$count$txt | wc -l`
  echo "Inf: "${inf[$count]}
  # Counts all lines below the threshold of -90dB
  echo "Threshold: "$threshold[$count]
  # Sums the number if [ inf ] lines with the number of lines below the threshold
  zero[$count]=${inf[$count]} + ${threshold[$count]}
  echo "Zero: "${zero[$count]}
  ((count++))
done

# Outputfiles path based on choice in the beginning of the script
outputfile="$outputbase"/"${outputpath[$choice]}"/"$y.csv"

echo $outputfile

# Checks to see if the output file do not exist, otherwise it’s deleted.
if [ -f $outputfile ]
then
  rm $outputfile
fi

# Outputs headers to the csv-file
printf "%snumber;total;zeros
" >> $outputfile

# Outputs the numbers to the csv-file.
count=0
for i in $total[@]
do
  printf "%s" $count "$i" "$zero[$count]" >> $outputfile
  ((count++))
done

# Output files path based on choice in the beginning of the script
outputfile="$outputbase"/"${outputpath[$choice]}"/"$y.csv"

echo $outputfile

# Checks to see if the output file do not exist, otherwise it’s deleted.
if [ -f $outputfile ]
then
  rm $outputfile
fi

# Outputs headers to the csv-file
printf "%snumber;total;zeros
" >> $outputfile

# Outputs the numbers to the csv-file.
count=0
for i in $total[@]
do
  printf "%s" $count "$i" "$zero[$count]" >> $outputfile
  ((count++))
done

done
} 2>&1 | tee -a ~/logs/Threshold.log
date=`date '+%Y-%m-%d %H:%M:%S'`

echo "-- End of log -- $date" >> ~/logs/Threshold.log
Appendix N – tmux.sh (Linux Bash-Script)

```bash
#!/bin/bash

# This is the main loop of the management computer, and which is controlling all the tests
# by sending information to the test computers, mainly from a CSV-file created.

# A if statement for the parsing of information from CLI.
if [[ $# ]] # If there are any options parsed from CLI, then execute this.
  then
    # If the option is an ?, display the following line
    if [[ $@ == -*?* ]] then
      echo "Usage in correct order: tmux.sh [number of tests] [csv-file]. (Default: 25
      /home/daniel/Desktop/testiterations_2018-05-12_anewhope_red2.csv"
      exit 127
    fi
    # Parse 0 = Number of tests
    # Parse 1 = csv-file to use into the test
    parse="($@)
    tests=$(parse[0])
    csv=$(parse[1])
  else
    # If no options are added from the CLI, these options will be used.
    tests=25
    csv="/home/daniel/Desktop/testiterations_2018-05-12_anewhope_red2.csv"
  fi

read -p "Are the QjackCTL daemons started on both clients? Press return to continue."

echo "Using following CSV-file: $csv"

# Adding the csv-file containing series of tests that should be executed to an array
data_array="(cat "$csv")"

for i in ${data_array[@]}
do
  echo "$i"
done

read -p "Okay? Press enter.. Or ctrl-c to break."

date=`date '+%Y-%m-%d %H:%M:%S'

echo "" # ---- New TMUX log at $date ----"

```

**wan**

- Netherlands Jitter
- Netherlands Latency
- Netherlands Loss
- Netherlands Rate

- Sweden Jitter
- Sweden Latency
- Sweden Loss
- Sweden Rate

- USA Jitter
- USA Latency
- USA Loss
- USA Rate
# Counting number tests that will be done, by counting number of cells
# in the array containing tests
countdataarray=0
for i in `${data_array[@]}`
do
  ((countdataarray++))
done
echo "Countdataarray= $countdataarray"
# Dividing test number by 6, to get number of tests (6 values per line).
numberoftests=$((countdataarray/6))
echo "Number of tests= $numberoftests"
y=0 # Counting variable for testloop
z=0 # Counting variable for testloop (times 5 every iteration)
while [ $y -lt $numberoftests ];
do
  loopdate=`date '+%Y-%m-%d %H:%M:%S'`
echo "----- TMUX-loop $y at $loopdate ----- "
echo "Count variable at beginning of loop: $y"
echo ""
  WANoffset=0
  # Adding the count number times 6 to get next tests from the CSV-file
  z=$((y*6))
  network=${data_array[$z]}
    # Which network
  echo "Network: " $network
  period=${data_array[$z+1]}
    # Low value (period)
  echo "Period: " $period
  buffer=${data_array[$z+2]}
    # High value (period)
  echo "Buffer: " $buffer
  sound=${data_array[$z+3]}
    # Which sound should be used. Not used in this script.
  echo "Sound: " $sound
  redundancy=${data_array[$z+4]}
  echo "Redundancy: " $redundancy
  finished=${data_array[$z+5]}
    # Whether the test is finished or not
  echo "Finished: " $finished
  if [[ $finished == "*"* ]];
    # If the test is finished (checking the CSV-file) it will be skipped.
    then
      # Adding offset to the network variable to use with the WAN-simulation
      if [[ $network == "Netherlands"* ]];
        then
          WANoffset=0
          echo "network"
        elif [[ $network == "Sweden"* ]];
          then
            WANoffset=4
            echo "network"
        elif [[ $network == "USA"* ]];
          then
            WANoffset=8
            echo "network"
        elif [[ $network == "LAN"* ]];
          then
            WANoffset=100
            echo "network"
        elif [[ $network == "Local"* ]];
          then
            WANoffset=101
            echo "network"
        fi
  fi
  y=$(($y+1))
done

then
WANoffset=200
else
echo "Oops, something went wrong. This shouldn’t be written out. Network-selection."
fi

if [[ $WANoffset -eq 200 ]] # No WAN-simulation
then
  echo "WAN offset = $WANoffset"
fi

if [[ $WANoffset -eq 100 ]] # No WAN-simulation
then
  echo "LAN-simulation"
fi

if [[ $networktest -eq "n" ]] # No WAN-simulation
then
  echo "Will start Ecasound with local audio interface."
  networktest="n"
fi

sleep 0.5

# Killing QjackCTL on Client 2
ssh daniel@10.208.15.101 'echo "n" > ~/Desktop/status/jack.status'
sleep 2

ekilled=`ssh daniel@10.208.15.101 'cat ~/Desktop/status/jack.killed'`
if [[ killed == "$n"* ]]
  then
date=`date '+%Y-%m-%d %H:%M:%S'`
  echo "Couldn’t kill QjackCTL at client 2, breaking script at $date"
  exit 127
fi

# Adjusting buffers on Client 2
ssh daniel@10.208.15.101 "echo $buffer > ~/Desktop/status/buffer.status"
ssh daniel@10.208.15.101 "echo $period > ~/Desktop/status/period.status"
ssh daniel@10.208.15.101 "~/Desktop/Scripts/qjackctl-buffer_cli2.sh $buffer $period"
sleep 0.5

# Starting QjackCTL via the daemon on Client 2
ssh daniel@10.208.15.101 'echo "y" > ~/Desktop/status/jack.status'
sleep 5

# Checking to see if it’s really started
killed=`ssh daniel@10.208.15.101 'cat ~/Desktop/status/jack.killed'`
if [[ killed == "$y"* ]]
  then
date=`date '+%Y-%m-%d %H:%M:%S'`
  echo "Couldn’t start QjackCTL at client 2, breaking script at $date"
  exit 127
fi

if [[ $networktest -eq "n" ]] # No WAN-simulation
then
  "LAN-simulation"
fi

if [[ $networktest -eq "y" ]] # Network test
then
  "Will start Ecasound with network test."
fi

# Killing QjackCTL on Client 2
ssh daniel@10.208.15.101 'echo "n" > ~/Desktop/status/jack.status'
sleep 5

# Checking to see if it’s really killed
killed=`ssh daniel@10.208.15.101 'cat ~/Desktop/status/jack.killed'`
if [[ killed == "$n"* ]]
  then
date=`date '+%Y-%m-%d %H:%M:%S'`
  echo "Couldn’t kill QjackCTL at client 2, breaking script at $date"
  exit 127
fi

sleep 0.5

# Adjusting buffers on Client 2
ssh daniel@10.208.15.101 "echo $buffer > ~/Desktop/status/buffer.status"
ssh daniel@10.208.15.101 "echo $period > ~/Desktop/status/period.status"
ssh daniel@10.208.15.101 "~/Desktop/Scripts/qjackctl-buffer_cli2.sh $buffer $period"
sleep 0.5

3
# Starting QjackCTL via the running daemon on Client 2

```bash
ssh daniel@10.208.15.101 'echo "y" > ~/Desktop/status/jack.status'
sleep 5
```

# Checking to see if it's really started

```bash
killed=ssh daniel@10.208.15.101 'cat ~/Desktop/status/jack.killed''
if [[ $killed = "*y*" ]]
    then
date=`date '+%Y-%m-%d %H:%M:%S'`
    echo "Couldn't start QjackCTL at client 2, breaking script at $date"
    exit 127
else
fi
```

# Deleting all WAN-simulation rules just in case.

```bash
ssh daniel@10.208.15.177 'sudo tc qdisc del dev ens192 root'
ssh daniel@10.208.15.177 'sudo tc qdisc del dev ifb0 root'
```

# Listing the rules

```bash
ssh daniel@10.208.15.177 'sudo tc qdisc ls dev ens192'
ssh daniel@10.208.15.177 'sudo tc class ls dev ifb0'
```

```bash
sleep 0.5
```

# Killing QjackCTL on Client 2

```bash
killed=ssh daniel@10.208.15.177 'cat ~/Desktop/status/jack.killed'
if [[ $killed = "*n*" ]]
    then
date=`date '+%Y-%m-%d %H:%M:%S'`
    echo "Couldn't kill QjackCTL at client 1, breaking script at $date"
    exit 127
else
fi
```

# Adjusting buffers on client 1.

```bash
ssh daniel@10.208.15.177 "echo $buffer > ~/Desktop/status/buffer.status"
ssh daniel@10.208.15.177 "echo $period > ~/Desktop/status/period.status"
```

```bash
```

# Starting QjackCTL via the daemon on Client 2

```bash
ssh daniel@10.208.15.101 'echo "y" > ~/Desktop/status/jack.status'
sleep 5
```

# Checking to see if it's really started

```bash
killed=ssh daniel@10.208.15.101 'cat ~/Desktop/status/jack.killed'
if [[ $killed = "*y*" ]]
    then
date=`date '+%Y-%m-%d %H:%M:%S'`
    echo "Couldn't start QjackCTL at client 2, breaking script at $date"
    exit 127
else
fi
```

```
```

# Deleting all WAN-simulation rules just in case.

```bash
ssh daniel@10.208.15.177 'sudo tc qdisc del dev ens192 root'
ssh daniel@10.208.15.177 'sudo tc qdisc del dev ifb0 root'
```

```
```

# Listing the rules

```bash
ssh daniel@10.208.15.177 'sudo tc qdisc ls dev ens192'
ssh daniel@10.208.15.177 'sudo tc class ls dev ifb0'
```

```
```

# Killing QjackCTL on Client 2

```bash
killed=ssh daniel@10.208.15.177 'cat ~/Desktop/status/jack.killed'
if [[ $killed = "*n*" ]]
    then
date=`date '+%Y-%m-%d %H:%M:%S'`
    echo "Couldn't kill QjackCTL at client 2, breaking script at $date"
    exit 127
else
fi
```

```
```

# Adjusting buffers on client 2

```bash
```

if [[ $WANoffset -lt 10 ]]; then
    echo "WAN-simulation"
    echo "$WAN-simulation"
    fi
```

```
```

```
```

```
```

```
```

# Starting Ecasound with network test

```bash
networktest="Y"
```

```
```

# Adding WAN-simulation on client 2, and adjusting buffers on both

```bash
ssh daniel@10.208.15.101 'echo "$buffer" > ~/Desktop/status/buffer.status'
ssh daniel@10.208.15.101 'echo "$period" > ~/Desktop/status/period.status'
sleep 0.5
```

```
```

# Starting QjackCTL via the daemon on Client 2

```bash
ssh daniel@10.208.15.101 'echo "y" > ~/Desktop/status/jack.status'
sleep 5
```

# Checking to see if it's really started

```bash
killed=ssh daniel@10.208.15.101 'cat ~/Desktop/status/jack.killed'
if [[ $killed = "*y*" ]]
    then
date=`date '+%Y-%m-%d %H:%M:%S'`
    echo "Couldn't start QjackCTL at client 2, breaking script at $date"
    exit 127
else
fi
```bash
ssh daniel@10.208.15.101 ~/Desktop/Scripts/qjackctl-buffer_cl1.sh $buffer $period
sleep 0.5

# Starting QJackCTL via the running daemon on Client 1
ssh daniel@10.208.15.101 'echo "y" > ~/Desktop/status/jack.status'
sleep 5

# Checking to see if QJackCTL is killed
if [ $(/usr/bin/grep "killed" ~/Desktop/status/jack.killed) ]; then
date=`date '+%Y-%m-%d %H:%M:%S'`
echo "Couldnt kill QJackCTL at client 1, breaking script at $date"
exit 127
fi

# Deleting all rules just in case
ssh daniel@10.208.15.177 'sudo tc qdisc del dev ens192 root'
ssh daniel@10.208.15.177 'sudo tc qdisc del dev ifb0 root'

# Adding rules for incoming traffic on Client 1
ssh daniel@10.208.15.177 'sudo tc class add dev ens192 parent 1: classid 0:1 htb rate $(wan[$WANoffset+1])'
echo "(Ingoing) Jitter & delay: $(wan[$WANoffset]) $(wan[$WANoffset+1]). Loss: $(wan[$WANoffset+2])"
ssh daniel@10.208.15.177 'sudo tc filter add dev ens192 parent 1:0 handle 10: protocol ip u32 match u32 0 0 flowid 1:1 action mirred egress redirect dev ifb0'

# Starting virtual interface for incoming traffic on client 1.
ssh daniel@10.208.15.177 'sudo ip link set dev ifb0 up'
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ens192 root handle 1: htb default 1'
ssh daniel@10.208.15.177 'sudo tc class add dev ens192 parent 1: classid 0:1 htb rate $(wan[$WANoffset+3])'
echo "(Outgoing) Rate: $(wan[$WANoffset+3])"
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ens192 parent 1:1 handle 10: netem delay $(wan[$WANoffset+1]) $(wan[$WANoffset+1]) distribution normal loss $(wan[$WANoffset+2])"
echo "(Outgoing) Jitter & delay: $(wan[$WANoffset]) $(wan[$WANoffset+1]). Loss: $(wan[$WANoffset+2])"
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ifb0 parent ffff: protocol ip u32 match u32 0 0 '

# Starting virtual interface for outgoing traffic on Client 2
ssh daniel@10.208.15.177 'sudo tc qdisc del dev ifb0 root'

# Adding rules for outgoing traffic on Client 2
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ifb0 root handle 1: htb default 1'
ssh daniel@10.208.15.177 'sudo tc class add dev ifb0 parent 1: classid 0:1 htb rate $(wan[$WANoffset+3])'
echo "(Ingoing) Rate: $(wan[$WANoffset+3])"
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ifb0 parent 1:1 handle 10: netem delay $(wan[$WANoffset+1]) $(wan[$WANoffset+1]) distribution normal loss $(wan[$WANoffset+2])"
echo "(Ingoing) Jitter & delay: $(wan[$WANoffset]) $(wan[$WANoffset+1]). Loss: $(wan[$WANoffset+2])"
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ens192 root handle 1: htb default 1'

# Starting virtual interface for outgoing traffic on client 2.
ssh daniel@10.208.15.177 'sudo ip link set dev ifb0 up'
ssh daniel@10.208.15.177 'sudo tc qdisc add dev ens192 ingress'
ssh daniel@10.208.15.177 'sudo tc filter add dev ens192 parent ffff: protocol ip u32 match u32 0 0 '

# Adjusting buffers on Client 1
ssh daniel@10.208.15.177 'echo "buffer" > ~/Desktop/status/buffer.status'
sleep 0.5

# Killing QJackCTL on Client 1
if [ $(/usr/bin/grep "killed" ~/Desktop/status/jack.killed) ]; then
date=`date '+%Y-%m-%d %H:%M:%S'`
echo "Couldn't kill QJackCTL at client 1, breaking script at $date"
exit 127
fi

# Adjusting buffers on Client 2
ssh daniel@10.208.15.177 'echo "$buffer" > ~/Desktop/status/buffer.status'
sleep 0.5

# Starting QJackCTL via the daemon on Client 2
ssh daniel@10.208.15.177 'echo "y" > ~/Desktop/status/jack.status'
sleep 5
killed=ssh daniel@10.208.15.177 'cat ~/Desktop/status/jack.killed'
```
```bash
if [[ $killed == "y" ]]
    then
        date=`date '+%Y-%m-%d %H:%M:%S'`
        echo "Couldn't start @jackctl at client 1, breaking script at $date"
        exit 127
fi

# Starting new Tmux-session with scripts executed inside. (C-m means enter)

tmux new-session -d -s daniel
    # Starting new Tmux-session
    # Creates window

tmux new-window -t daniel:1 -n 'Scripting'

    # Splits the window vertically
    # Splits the window vertically
    # Will even out the vertical to same size

tmux select-pane -t 0
    # Selects pane 0
    # Splits it horizontally

tmux select-pane -t 2
    # Selects pane 2
    # Splits it horizontally

tmux select-pane -t 4
    # Selects pane 4
    # Splits it horizontally

tmux select-pane -t 0
    # Selects pane 0

tmux send-keys "ssh 10.208.15.101" C-m
    # Connect with SSH to Client 2

tmux send-keys "cd ~/Desktop/Scripts/" C-m
    # Goes to scripts directory

    # Selects pane 2
    # Goes to ~/Desktop. This pane is for the local computer
    # and is not used by the script.

    # Selects pane 4
    # Goes to scripts directory.

    # Connect with SSH to Client 2

    # Connect with SSH to Client 1

    # Connect with SSH to Client 1

    # Creates directories depending of redundancy

    # Connect with SSH to Client 1, with the redundancy
    # linked from the CSV-file

    # Starts server instance of Client 1

    # Starts server instance of client 2

    # Starts client instance on client 1
```

---

```bash
# Ending the script

echo "# Ending the script"
if [[ $killed == *"y"* ]]
    then
        date	=""date '5Y.5m.5d %H:%M:%S'"
        echo ""Couldn't start @jackctl at client 1, breaking script at $date"
        exit 127
fi
```

---

```bash
# Ending the script

echo "# Ending the script"
if [[ $killed == *"y"* ]]
    then
        date	=""date '5Y.5m.5d %H:%M:%S'"
        echo ""Couldn't start @jackctl at client 1, breaking script at $date"
        exit 127
fi
```

---

```bash
# Ending the script

echo "# Ending the script"
if [[ $killed == *"y"* ]]
    then
        date	=""date '5Y.5m.5d %H:%M:%S'"
        echo ""Couldn't start @jackctl at client 1, breaking script at $date"
        exit 127
fi
```

---

```bash
# Ending the script

echo "# Ending the script"
if [[ $killed == *"y"* ]]
    then
        date	=""date '5Y.5m.5d %H:%M:%S'"
        echo ""Couldn't start @jackctl at client 1, breaking script at $date"
        exit 127
fi
```
tmux select-pane -t 1

# Selects pane 1

tmux send-keys "./jacktrip-client2_clientstart.sh" C-m

# Starts client instance of client 2

sleep 2
echo "Sending following information to Scriptloop: $network $tests"

tmux send-keys "./Scripts/./Scriptloop.sh $network $tests $networktest" C-m

# Starts playing and recording (Ecasound)

tmux select-pane -t 3

# Selects pane 3 (client 2)

sleep 2

# Waiting for TMUX to break (Scriptloop breaks it..)

while kill -0 PIDS 2> /dev/null
done

# Getting all processnumbers of "jacktrip" (script and program) from both clients and killing them

ssh daniel@10.208.15.101 "ps cax | grep jacktrip | grep -o '^\s*\S\s*\S' | xargs kill"

ssh daniel@10.208.15.177 "ps cax | grep jacktrip | grep -o '^\s*\S\s*\S' | xargs kill"

# Adding y to the test iteration file to be able to know what is done or not.

string1="$network $buffer $period $sound $finished.*"

string2="$network $buffer $period $sound y"

echo $string1

echo $string2

loopdate=`date '+%Y-%m-%d %H:%M:%S'`

sed -i "s/${string1}/${string2}/" ~Desktop/testiterations_2018-04-27.csv

echo $string2 $loopdate >>~/logs/testiterations_done.csv

# Adding 1 internally to know it's finished.

# Turning off all WAN-simulation to be able to start a new one in next iteration

ssh daniel@10.208.15.101 'sudo tc qdisc del dev ens192 root'

ssh daniel@10.208.15.177 'sudo tc qdisc del dev ifb0 root'

sleep 0.5

ssh daniel@10.208.15.177 'tc -d qdisc show dev ens192'

sleep 1

fi

echo "Count variable at end of loop $y"

((y++))

# Adding one to the count variable

done

echo "All is well."

2>&1 | tee -a ~/logs/tmux.log

# Kills QjackCtl-Daemons on Clients

ssh daniel@10.208.15.101 "echo "n" > ~/Desktop/status/daemon.status"

ssh daniel@10.208.15.177 "echo "n" > ~/Desktop/status/daemon.status"
APPENDIX O - trim_invert_merged-script.sh (Linux Bash-Script)

#!/bin/bash
{

date=`date '+%Y-%m-%d %H:%M:%S'`

echo ""'

""'---- New log at $date ----"

echo ""

if [ $@ ]
# If there are option parsed to the script
# via cli or another script, then run this:
then
input=(@)

cd ~/Desktop/Wav/Input/

path="Helpers"

echo "Runs from $path"
#else
# Otherwise, run this:
path="Helpers"

echo "Run this from the input folder"
fi

# Creates filename from the folder name
filename="${PWD##*/}_""'

echo "Filename used: $filename"

# Checks what type of file that are going to be processed
# Music or rhythm
if [[ $filename == *"Music"* ]] && [[ $filename == *"Rhythm"* ]] then

echo "Both Music and Rhythm exists in filename. Exits."
exit 127
elif [[ $filename == *"Music"* ]] then
testfile="Music.wav"

elif [[ $filename == *"Rhythm"* ]] then
testfile="Rhythm_85-160bpm.wav"
else

echo "No Music or Rhythm i filename. Exits.""'
exit 127
fi

files=`ls -l | wc -l`

(files=+)

outputtriminv="../TrimInv"

outputmerged="../Merged"

if [ ! -d $outputtriminv ]
# Checks to see if the output folder for the trimmed and inverted files exists

mkdir $outputtriminv

# otherwise it is created
fi

if [ ! -d $outputmerged ]
# Checks to see if the output folder for the merged files exists

mkdir $outputmerged

# otherwise it is created
fi

count=0

while [ $count -le $files ]
# Loops as many times as there are files in the folder

do
# Runs the SoX application which inverts the audio (-v) and trims its silence in the beginning
# If it's less than 1% audio and shorter than 0.000001 second.
sox -v -1 $filename$count.wav $outputtriminv/$filename$count_triminv.wav silence 1 0.000001 0

echo "sox -v -1 $filename$count.wav $outputtriminv/$filename$count_triminv.wav silence 1 0.000001 0"

((count++))
done
count=0
while [ $count -le $files ]

do
  # loops as many times as there are files in the folder.
  # runs the SoX application which merges the reference file with the recorded file, which has been inverted.
  # if its output is only silence, the wave files are exactly the same.
  # trims 0.2 seconds in the end.
  sox -n $outputtriminv/$filename$count(triminv).wav /home/daniel/Desktop/Wav/$testfile
  $outputmerged/$filename$count_merged.wav trim 0 -0.2
  echo "sox -n $outputtriminv/$filename$count(triminv).wav /home/daniel/Desktop/Wav/$testfile
  $outputmerged/$filename$count_merged.wav trim 0 -0.2"
  ((count++))
done

# saves a logfile
} 2>&1 | tee -a ~/logs/TrimInvertMerge.log
Appendix P – Client1.xml (XML-file for QjackCtl-Settings)

<?xml version="1.0" encoding="utf-8"?>
<!-- This XML file is used with script "Snapshot_client1.sh" which is executed from the script "Latency_Ecasound.sh" and is used to connect the internal audio connections in QjackCTL on Client1 -->

<jack>
  <client name="system">
    <port name="capture_1" />
    <port name="capture_2" />
  </client>
  <client name="o0">
    <port name="receive_1" />
  </client>
  <client name="o10">
    <port name="receive_1">
      <connection port="o0:send_1" />
    </port>
  </client>
</jack>
APPENDIX Q – Audiosort.ps1 (Windows Powershell-Script)

# This sorts the files the "windows way" by and inserts a number to it in the beginning with
# the pattern "00000". This way it's easy to keep track of which file is which when
# using Audacity convert it to sample-data file, which will load it in the correct order
# when using that pattern. Audacity will then output a sample-data file with the "normal"
# way of numbering it, with first single digit, then two digits when coming to 10 and so on.
# Audacity can only process one folder at the time.
# When Audacity converted it, the script "Sortback" will be used and sort back the files
# to correct folder, based on the index-files this script outputs.

# Fill in this data before starting!

# Input directory, relative to base input directory, will form a outputdirectory based on that name
$jobname = "MTU1500_red1_batch3"

# A check of how many items there should be in each folder before starting
$numberofitems = 25

# Stops at any error
$ErrorActionPreference = "Stop"

# Basedirectory for the output and input
$basedirectory = "C:\Users\Student\OneDrive\Exjobb-Data\" + $jobname
$outputbasedirectory = "C:\Users\Student\OneDrive"

# Timestamp
$timestamp = Get-Date -Format o | foreach {$_ -replace ":", "."}

# Outdirectory
$todirectorylatency = $outputbasedirectory + $jobname + "_latency_list"
$todirectorymerged = $outputbasedirectory + $jobname + "_merged_list"

# Index files to be saved
$indexfilelatency = "C:\Users\Student\OneDrive\Exjobb-misc\Index\" + $jobname + "_latency_" + $timestamp + ".csv"
$indexfilemerged = "C:\Users\Student\OneDrive\Exjobb-misc\Index\" + $jobname + "_merged_" + $timestamp + ".csv"

# Creates a new output directory for latency-test if it doesn't exist.
if (!test path $todirectorylatency){
    new-item -ItemType directory -path $todirectorylatency
}

# Creates a new output directory for "amount of correctly aligned audio"-test (merged) if it doesn't exist.
if (!test path $todirectorymerged){
    new-item -ItemType directory -path $todirectorymerged
}

# Creates a timestamp for the index-file
$timestamp = Get-Date -Format o | foreach {$_ -replace ":", "."}

# Declares arrays
$directories = @()
$child = @()

# Get all directories from the parent directory
$directories = Get-ChildItem -path $basedirectory -directory -name
echo $directories

$countermusic = 0
$countermerged = 0
$counter = 0

# Creates the index-files' headers
out-file $indexfilelatency -inputobject "File,Begin,End"
out-file $indexfilemerged -inputobject "File,Begin,End"
# This loop counts through all directories to see if all files that are entered # in $numberofitems exists. Otherwise script will exit.

**ForEach** ($d in $directories){

  $childmusic = get-childitem -path $basedirectory"\$d" -file -name
  $childmerged = get-childitem -path $basedirectory"\$d\merged" -file -name

  $countermusicbegin=0
  $countermergedbegin=0

  foreach ($music in $childmusic){
    $countermusicbegin++
    echo $countermusicbegin
    if ($countermusicbegin -ne $numberofitems){
      echo $music
      exit
    }
  }

  foreach ($merged in $childmerged){
    $countermergedbegin++
    echo $countermergedbegin
    if ($countermergedbegin -ne $numberofitems){
      echo $merged
      exit
    }
  }

  $countermusic=0
  $countermerged=0
  $counter=0

  # This loop goes through all directories with test files
  **ForEach** ($d in $directories){

    # Gets the name of the audio file that should be copied
    $childmusic = get-childitem -path $basedirectory"\$d\" -file -name

    # Resets the counter for the series
    $countermusicbegin=0

    # The innerloop below goes through all directories with latency tests (music) # and gives them numbers according to how many files there are represented # in the directory.
    foreach ($music in $childmusic){

      # Converts the total counter in $countermusic variable to the pattern "00000"
      $countermusicstring=$countermusic.tostring("00000")

      # Puts the file to be copied in the variable $from
      $from=$basedirectory+"\"+$d+"\"+$d+"\"+$music

      # Puts the directory and name of file in the variable $to
      $to=$todirectorylatency+"\"+$countermusicstring+"_"+$music

      echo $to

      # Copies only if the file doesn’t already exist
      if (!((test-path -path $to))){
        copy-item $from $to | out-null
        echo "..copied!"
      } else{
        echo "..exists!"
      }

    }
    $countermusic++
    $countermergedbegin++
  }
}
$countermusicbegin = $countermusic - $countermusicbegin
$countermusicbegin = $countermusicbegin.tostring('00000')

$countermergedbegin = $countermerged - $countermergedbegin
$countermergedbegin = $countermergedbegin.tostring('00000')

foreach ($merged in $childmerged) {
    $countermergedstring = $countermerged.tostring('00000')
    $from = $basedirectory + "\"$d\"merged\"$merged"
    $to = $todirectorymerged + "\"$countermergedstring\"\"$merged"
    echo $to
    if (!($test -path $to)) {
        copy-item $from to $to | out-null
        echo '..copied!'
    } else {
        echo '..exists!'
    }
    $countermerged++
    $countermergedbegin++
}

$countermergedfinal = $countermerged - $countermergedbegin
$countermergedfinal = $countermergedfinal.tostring('00000')

foreach ($merged in $childmerged) {
    $countermergedstring = $countermerged.tostring('00000')
    $from = $basedirectory + "\"$d\"merged\"$merged"
    $to = $todirectorymerged + "\"$countermergedstring\"\"$merged"
    echo $to
    if (!($test -path $to)) {
        copy-item $from to $to | out-null
        echo '..copied!'
    } else {
        echo '..exists!'
    }
    $countermerged++
    $countermergedbegin++
}

}
APPENDIX R – Chain-creation.ps1 (Windows Powershell-Script)

# This script creates Audacity chain setting files, which is used with Audacity to convert wave files to sample data files. Since there’s a bug in Audacity making it to only be able to convert around 750 files at the time, four different files are created for each test; latency (music) and amount of correct aligned sample data (merged).

$job = "red1_batch3"

$destinationmerged = "C:\Users\Student\OneDrive\$job\merged"
$destinationmusic = "C:\Users\Student\OneDrive\$job\music"

# This is what the chain files will be called
$destinationchainmusic = "C:\Users\Student\AppData\Roaming\audacity\Chains\"+_+$job+"_"" + "Latency_merged_original.txt"
$destinationchainmerged = "C:\Users\Student\AppData\Roaming\audacity\Chains\"+_+$job+"_"" + "Merged_merged_original.txt"

# These are the directories that will be created. If there are more than 2600 files, more categories must be created.
$mergedsubdirs = ("Merged_0-749", "Merged_750-1499", "Merged_1500-2249", "Merged_2250-2999")

# Where merged sampledata should be placed
$destinationmergedrepl = "C:\Users\Student\OneDrive\$job\merged"
$destinationmusicrepl = "C:\Users\Student\OneDrive\$job\music"

# Where latency ("music") sampledata should be placed
$destinationmusicrepl = "C:\Users\Student\OneDrive\$job\music"

# This needs to be same as the lines above, but needs to be with two backslashes, because that’s what Delimiter specifies
$destinationmergedrepl = "C:\Users\Student\OneDrive\$job\merged\"
$destinationmusicrepl = "C:\Users\Student\OneDrive\$job\music\"

# This is what the chain files will be called
$destinationchainmusic = "C:\Users\Student\AppData\Roaming\audacity\Chains\"+_+$job+"_" + "Latency_merged_original.txt"
$destinationchainmerged = "C:\Users\Student\AppData\Roaming\audacity\Chains\"+_+$job+"_" + "Merged_merged_original.txt"

# Creates directories for the test categories
if ( !(test-path -path $destinationmerged)){
  new-item -ItemType directory -path $destinationmerged
}
if ( !(test-path -path $destinationmusic)){
  new-item -ItemType directory -path $destinationmusic
}

# Creates directories inside the categories
foreach ($i in $musicsubdirs){
  if ( !(test-path -path $destinationmusic\$i)){
    new-item -Itemtype directory -path $destinationmusic\$i
  }
}
foreach ($i in $mergedsubdirs){
  if ( !(test-path -path $destinationmerged\$i)){
    new-item -Itemtype directory -path $destinationmerged\$i
  }
}

# Gets the directories existing in the destination folders
$directorymusic = Get-ChildItem -path $destinationmusic -directory -name
$directorymerged = Get-ChildItem -path $destinationmerged -directory -name

# Gets content from the preexisting chain files
$original = get-content -path "C:\Users\Student\AppData\Roaming\audacity\Chains\Latency_original.txt"
$originalmerged = get-content -path "C:\Users\Student\AppData\Roaming\audacity\Chains\Latency_merged_original.txt"
ForEach ($i in $directorymusic)

# Loop which goes through the chain creation for the latency (music) test
{
$orig1 = "optext=""test""
$modi1 = "optext=""$i"
$orig2 = "path=""C:\\Users\\$Student\\"
$modi2 = "path=""$destinationmusicrepl$i"

$orig1 = $orig1 -replace ($orig1, $modi1)
$orig2 = $orig2 -replace ($orig2, $modi2)

# Checks if there exists such file, otherwise it will be created
if(!((test-path "$destinationchainmusic$i.txt")){
    out-file -filepath "$destinationchainmusic$i.txt" -inputobject $modified2
}}

ForEach ($i in $directorymerged)

# Loop which goes through the chain creation for the amount of correct aligned sample data (merged)
{
$orig1 = "optext=""test""
$modi1 = "optext=""$i"
$orig2 = "path=""C:\\Users\\$Student\\"
$modi2 = "path=""$destinationmergedrepl$i"

$orig1 = $orig1 -replace ($orig1, $modi1)
$orig2 = $orig2 -replace ($orig2, $modi2)

# Checks if there exists such file, otherwise it will be created
if(!((test-path "$destinationchainmerged$i.txt")){
    out-file -filepath "$destinationchainmerged$i.txt" -inputobject $modified2
}}
}
APPENDIX S - CreateTests.ps1 ((Windows Powershell-Script)

# Source originally from https://stackoverflow.com/questions/34186488/how-to-create-a-permutation-array-in-powershell
# Forum post by "TheMadTechnician" 9th of December 2015.
# Modified for this case.
# Used to create CSV files for the main loop of (tmux.sh) the tests on the management machine.
# All tests are combined in all ways possible.

Remove-Variable array, alliterations -ea 4

$columnheaders = "Buffer", "Period", "Network", "Soundfile", "Finished?"

# Puts the tests in columns
[array]array += [PSCustomObject]@{
    'Column 1' = 'LAN'
    'Column 2' = '2'
    'Column 3' = '1024'
    'Column 4' = 'Music'
    'Column 5' = 'n'
}
[array]array += [PSCustomObject]@{
    'Column 1' = Sweden
    'Column 2' = '3'
    'Column 3' = '2048'
    'Column 4' = '
    'Column 5' = '
}
[array]array += [PSCustomObject]@{
    'Column 1' = Netherlands
    'Column 2' = '4'
    'Column 3' = '4096'
    'Column 4' = '
    'Column 5' = '
}
[array]array += [PSCustomObject]@{
    'Column 1' = USA
    'Column 2' = '5'
    'Column 3' = '
    'Column 4' = '
    'Column 5' = '
}
[array]array += [PSCustomObject]@{
    'Column 1' = '
    'Column 2' = '6'
    'Column 3' = '
    'Column 4' = '
    'Column 5' = '
}

# Combines the tests with an nested loop, so all possible variants will be
# combined. Data put in an array.
ForEach($a in (array. 'Column 1'| Where{$_})){
    ForEach($b in (array. 'Column 2'| Where{$_})){
        ForEach($c in (array. 'Column 3'| Where{$_})){
            ForEach($d in (array. 'Column 4'| Where{$_})){
                ForEach($e in (array. 'Column 5'| Where{$_})){
                    [array]AllIterations += [PSCustomObject]@{
                        'Column 1' = $a
                        'Column 2' = $b
                        'Column 3' = $c
                        'Column 4' = $d
                        'Column 5' = $e
                    }
                }
            }
        }
    }
}
```powershell
# Timestamp
$timestamp = Get-Date -Format o | foreach {$_ -replace ":", ","}

# Puts all the tests into a CSV file
$AllIterations | ConvertTo-Csv -Delimiter '' -NoTypeInfo | % {$_.Replace("","")}'" | Select -Skip 1 | out-file -filepath "C:\Users\Student\testiterations_$timestamp.csv"

# Puts all headers in a text file
$columnheaders | out-file -filepath "C:\Users\Student\headersofiterations_$timestamp.txt"
APPENDIX T – GatherSampledata.ps1 (Windows Powershell-Script)

```powershell
# This script will gather all sample-data that is created in Audacity from 0-749 to one directory where it will get its correct series number so it can be sorted back as with the sort back script.

$job=Read-Host -prompt 'Which job is this? >'
$inputdirectory=read-host -prompt 'Input directory >'
$startnumber=read-host -prompt 'Serie start number >'
$endnumber=read-host -prompt 'Serie end number >'
$merged=read-host -prompt 'Merged? >'
$outputdirectory=$inputdirectory"\sample-data0" + "$timestamp"

# If the merge choice is "y" output directory an output directory called "Exjobb_Merged" plus the jobname chosen will be created in the parent directory.
# If the merge choice is "n" output directory an output directory called "Exjobb_music" plus the jobname chosen will be created in the parent directory.
# This is prepared in the $outputdirectory variable.

if ($merged -eq "y"){
  echo "merged"
  $outputdirectory = "C:\Users\Student\OneDrive\Exjobb_Merged\"+$job+"_Gathered"
} elseif ($merged -eq "n") {
  echo "Music"
  $outputdirectory = "C:\Users\Student\OneDrive\Exjobb_"+$job+"_Gathered"
}
else {
  echo "Please choose if it's merged or not."
  exit
}

# If first file is called "sample-data.txt", without a, it will be renamed to that, so it can be used in the counting.
if (((test-path $startfile))){
  rename-item -path $inputdirectory\"sample-data.txt" -newname $startfile
  echo "Renaming $inputdirectory\sample-data.txt to $startfile"
} else {
  echo "$startfile is fine"
}

# If the output directory do not exists, it will be created here.
if (((test-path $outputdirectory))){
  new-item -itemtype directory -path $outputdirectory
  echo "Creating $outputdirectory"
} else {
  echo "$outputdirectory exists."
}

# Timestamp for use in the log file
$timestamp = Get-Date -Format o | foreach {$_ -replace ":", ","}

# Variable for the log file with a timestamp
$logfile="C:\Users\Student\OneDrive\Exjobb-Misc\GatherSampledata_"+$startnumber+"-"+$endnumber+"-"+$timestamp+.txt"

$count=0
# Coops through from startnumber to endnumber
for ($i = [int]$startnumber; $i -lt [int]$endnumber+1; $i++){"
# This script sorts back all files to their respective directory based on the # CSV-file created in the AudioSort.ps1
#
# Enter which CSV-file to import (the one corresponding to the job)
$csv = import-csv "C:\Users\Student\OneDrive\Exjobb-Misc\Index_MTU1500_Red2_batch1_2018-05-31T22.33.35.7971565+02.00.txt"
#
# Enter which input directory that should be used (directories will be created)
$inputdirectory = "C:\Users\Student\OneDrive\Exjobb\Red1_Batch2_Gathered"
#
# Enter which output directory that should be used (directories will be created)
$outputdirectory = "C:\Users\Student\OneDrive\Exjobb-Misc\Sortback\_sorted"
#
$merged = read-host "Merged? "
if ($merged -eq "y"){
# Enter which parent output directory that should be used (directories will be created)
    echo "Merged"
} elseif ($merged -eq "n") {
    echo "Music"
} else{
    echo "Please choose if it's merged or not."
    exit
}

# Timestamp
$timestamp = Get-Date -Format o | foreach {$_ -replace ":", "."}
#
# Variable for the logfile with a timestamp
$logfile="C:\Users\Student\OneDrive\Exjobb-Misc\Sortback_$timestamp"
#
# Renames first file if its name is sample-data.txt to sample-data0.txt so it will work
# with the move loop below
if ((test-path -path $inputdirectory\"sample-data.txt"){
    rename-item -path $inputdirectory\"sample-data.txt" -newname $inputdirectory\"sample-data0.txt"
}
#
# Loops through each line of the csv-file
foreach ($csv in $csv){
    # Loop $csv.begin
    $count=0
    # Loops through the serie number that that is referenced in the line of the csv-file
    for ($i = [int]$csv.begin; $i -lt [int]$csv.end; $i++){
        # Creates a variable from the 'file' header of the csv-file
        $file=$csv.file
        echo "File: $file"
        # Extracts the buffer information from filename
        $buffer=$file.Substring($file.Length-6)
        $buffertype=$file.Substring($file.Length-48)
        $buffer=new-object string($buffer,6)
        $buffertype=new-object string($buffertype,6)
        # Creates directories based on the 'file' header of the csv-file if it doesn't already exist
        if ((test-path -path $outputdirectory\$file)){
            new-item -ItemType directory -path $outputdirectory\$file
        } else{
            copy-item $inputdirectory\"sample-data\$i.txt" $outputdirectory\"\$file\"$sample-data\$count"
        }
        # Outputs metafiles with the audio buffer and buffertype
        if ($merged -eq "n"){
            out-file $outputdirectory\"\$file\"$sample-data\$count\".buffer" -inputobject $buffer
            out-file $outputdirectory\"\$file\"$sample-data\$count\".buffertype" -inputobject $buffertype
        }
        # Outputs sortback
        out-file $logfile -inputobject "Copying $inputdirectory\sample-data\$i.txt to $outputdirectory\$file\" -append
        echo ":.. already exists.." }
$count++
}
}
}
**APPENDIX V – Testiterations.csv (Example File)**

Header: Network, Period, Buffer, Soundfile (rotated in Scriptloop.sh instead), Redundancy, Finished

<table>
<thead>
<tr>
<th>Network</th>
<th>Period</th>
<th>Buffer</th>
<th>Soundfile</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAN 2</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 2</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 2</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 3</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 3</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 3</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 4</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 4</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 4</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 5</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 5</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 6</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>LAN 6</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 2</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 2</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 2</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 3</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 3</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 3</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 4</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 4</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 4</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 5</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 5</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 5</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 6</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Sweden 6</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 2</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 2</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 2</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 3</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 3</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 3</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 4</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 4</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 4</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 5</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 5</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 5</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 6</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 6</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>Netherlands 6</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 2</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 2</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 2</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 3</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 3</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 3</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 4</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 4</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 4</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 5</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 5</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 5</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 6</td>
<td>1024</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 6</td>
<td>2048</td>
<td>Music 1</td>
<td>n</td>
</tr>
<tr>
<td>USA 6</td>
<td>4096</td>
<td>Music 1</td>
<td>n</td>
</tr>
</tbody>
</table>
APPENDIX W – Sample-data (Short Snippet of Example File)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>90.30900</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>90.30900</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>90.30900</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>90.30900</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>-inf</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>-inf</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>-inf</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>-inf</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>-inf</td>
</tr>
<tr>
<td>-inf</td>
<td>-inf</td>
<td>-inf</td>
</tr>
<tr>
<td>22.32938</td>
<td>21.53815</td>
<td>23.02924</td>
</tr>
<tr>
<td>25.24224</td>
<td>21.33178</td>
<td>23.90607</td>
</tr>
<tr>
<td>22.27754</td>
<td>21.87363</td>
<td>22.08671</td>
</tr>
<tr>
<td>25.18427</td>
<td>26.54731</td>
<td>26.92139</td>
</tr>
<tr>
<td>26.92139</td>
<td>24.80277</td>
<td>26.45208</td>
</tr>
<tr>
<td>21.33178</td>
<td>23.90607</td>
<td>22.27754</td>
</tr>
<tr>
<td>21.87363</td>
<td>22.08671</td>
<td>25.18427</td>
</tr>
<tr>
<td>26.54731</td>
<td>26.92139</td>
<td>26.45208</td>
</tr>
<tr>
<td>26.92139</td>
<td>24.80277</td>
<td>26.45208</td>
</tr>
<tr>
<td>22.32938</td>
<td>21.53815</td>
<td>23.02924</td>
</tr>
<tr>
<td>25.24224</td>
<td>21.33178</td>
<td>23.90607</td>
</tr>
<tr>
<td>22.27754</td>
<td>21.87363</td>
<td>22.08671</td>
</tr>
<tr>
<td>25.18427</td>
<td>26.54731</td>
<td>26.92139</td>
</tr>
<tr>
<td>26.92139</td>
<td>24.80277</td>
<td>26.45208</td>
</tr>
<tr>
<td>22.32938</td>
<td>21.53815</td>
<td>23.02924</td>
</tr>
<tr>
<td>25.24224</td>
<td>21.33178</td>
<td>23.90607</td>
</tr>
<tr>
<td>22.27754</td>
<td>21.87363</td>
<td>22.08671</td>
</tr>
<tr>
<td>25.18427</td>
<td>26.54731</td>
<td>26.92139</td>
</tr>
<tr>
<td>26.92139</td>
<td>24.80277</td>
<td>26.45208</td>
</tr>
<tr>
<td>22.32938</td>
<td>21.53815</td>
<td>23.02924</td>
</tr>
<tr>
<td>25.24224</td>
<td>21.33178</td>
<td>23.90607</td>
</tr>
<tr>
<td>22.27754</td>
<td>21.87363</td>
<td>22.08671</td>
</tr>
<tr>
<td>25.18427</td>
<td>26.54731</td>
<td>26.92139</td>
</tr>
<tr>
<td>26.92139</td>
<td>24.80277</td>
<td>26.45208</td>
</tr>
<tr>
<td>22.32938</td>
<td>21.53815</td>
<td>23.02924</td>
</tr>
<tr>
<td>25.24224</td>
<td>21.33178</td>
<td>23.90607</td>
</tr>
<tr>
<td>22.27754</td>
<td>21.87363</td>
<td>22.08671</td>
</tr>
<tr>
<td>25.18427</td>
<td>26.54731</td>
<td>26.92139</td>
</tr>
<tr>
<td>26.92139</td>
<td>24.80277</td>
<td>26.45208</td>
</tr>
<tr>
<td>22.32938</td>
<td>21.53815</td>
<td>23.02924</td>
</tr>
<tr>
<td>25.24224</td>
<td>21.33178</td>
<td>23.90607</td>
</tr>
<tr>
<td>22.27754</td>
<td>21.87363</td>
<td>22.08671</td>
</tr>
<tr>
<td>25.18427</td>
<td>26.54731</td>
<td>26.92139</td>
</tr>
<tr>
<td>26.92139</td>
<td>24.80277</td>
<td>26.45208</td>
</tr>
<tr>
<td>22.32938</td>
<td>21.53815</td>
<td>23.02924</td>
</tr>
<tr>
<td>25.24224</td>
<td>21.33178</td>
<td>23.90607</td>
</tr>
<tr>
<td>22.27754</td>
<td>21.87363</td>
<td>22.08671</td>
</tr>
<tr>
<td>25.18427</td>
<td>26.54731</td>
<td>26.92139</td>
</tr>
<tr>
<td>26.92139</td>
<td>24.80277</td>
<td>26.45208</td>
</tr>
</tbody>
</table>
APPENDIX X – Index File for Sorting Back Sample-Data Files (Example File)

File,Begin,End
LAN_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_2_merged,00800,00824
LAN_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_3_merged,00825,00849
LAN_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_4_merged,00850,00874
LAN_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_5_merged,00840,00844
LAN_Rhythm_17s_Red_1_-_Queue_4_-_Underrun_wave_2048_2_merged,00455,00474
LAN_Rhythm_17s_Red_1_-_Queue_4_-_Underrun_wave_2048_3_merged,00845,00849
LAN_Rhythm_17s_Red_1_-_Queue_4_-_Underrun_wave_2048_4_merged,00505,00524
LAN_Rhythm_17s_Red_1_-_Queue_4_-_Underrun_wave_2048_5_merged,00555,00574
LAN_Rhythm_17s_Red_1_-_Queue_4_-_Underrun_wave_4096_2_merged,00950,00954
LAN_Rhythm_17s_Red_1_-_Queue_4_-_Underrun_wave_4096_3_merged,00955,00959
LAN_Rhythm_17s_Red_1_-_Queue_4_-_Underrun_wave_4096_4_merged,00960,00964
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_2_merged,00650,00674
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_3_merged,00675,00699
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_4_merged,00700,00724
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_5_merged,00725,00749
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_2_merged,00750,00774
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_3_merged,00845,00849
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_4_merged,00850,00854
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_5_merged,00855,00874
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_2_merged,00900,00924
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_3_merged,00905,00909
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_4_merged,00910,00914
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_5_merged,00915,00919
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_2_merged,00125,00129
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_3_merged,00120,00124
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_4_merged,00125,00129
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_5_merged,00130,00134
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_2_merged,00120,00124
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_3_merged,00125,00129
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_4_merged,00130,00134
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_5_merged,00135,00139
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_2_merged,00125,00129
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_3_merged,00130,00134
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_4_merged,00135,00139
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_5_merged,00140,00144
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_2_merged,00150,00154
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_3_merged,00155,00159
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_4_merged,00150,00154
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_5_merged,00155,00159
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_2_merged,00150,00154
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_3_merged,00155,00159
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_4_merged,00160,00164
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_5_merged,00165,00169
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_2_merged,00150,00154
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_3_merged,00155,00159
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_4_merged,00160,00164
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_5_merged,00165,00169
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_2_merged,00725,00729
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_3_merged,00720,00724
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_4_merged,00725,00729
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_1024_5_merged,00730,00734
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_2_merged,00725,00729
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_3_merged,00730,00734
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_4_merged,00735,00739
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_2048_5_merged,00740,00744
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_2_merged,00725,00729
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_3_merged,00730,00734
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_4_merged,00735,00739
Netherlands_Music_30s_Red_1_-_Queue_4_-_Underrun_wave_4096_5_merged,00740,00744
APPENDIX Y - Latency and Amount of Correctly aligned audio – LAN (No Redundancy)

Chart 9: Average and median latency of LAN without redundancy. Error bars is standard deviation.

Chart 10: Average and median percent of correctly aligned audio with LAN without redundancy. Error bars is standard deviation.
APPENDIX Z - Latency and Amount of Correctly aligned audio – Netherlands (No Redundancy)

**Chart 11:** Average and median latency of Netherlands network without redundancy. Error bars is standard deviation.

**Chart 12:** Average and median percent of correctly aligned audio with Netherlands network without redundancy. Error bars is standard deviation.
APPENDIX AA - Latency and Amount of Correctly aligned audio – Sweden (No Redundancy)

Chart 13: Average and median latency of Sweden network without redundancy. Error bars is standard deviation. Erroneous bars because of flaws in the design of processing of this type of data.

Chart 14: Average and median percent of correctly aligned audio with Sweden network without redundancy. Error bars is standard deviation. Erroneous bars because of flaws in the design of processing of this type of data.
APPENDIX AB - Latency and Amount of Correctly aligned audio – USA (No Redundancy)

Chart 15: Average and median latency of USA network without redundancy. Error bars is standard deviation.

Chart 16: Average and median percent of correctly aligned audio with USA network without redundancy. Error bars is standard deviation.
APPENDIX AC - Latency and Amount of Correctly aligned audio - LAN (Redundancy of 2)

Chart 17: Average and median latency of LAN with redundancy of 2. Error bars is standard deviation.

Chart 18: Average and median percent of correctly aligned audio with LAN with redundancy of 2. Error bars is standard deviation.
APPENDIX AD Latency and Amount of Correctly aligned audio - Netherlands (Redundancy of 2)

Chart 19: Average and median latency of Netherlands network with redundancy of 2. Error bars is standard deviation.

Chart 20: Average and median percent of correctly aligned audio with Netherlands network with redundancy of 2. Error bars is standard deviation.
APPENDIX AE - Latency and Amount of Correctly aligned audio - Sweden (Redundancy of 2)

Chart 21: Average and median latency of Sweden network with redundancy of 2. Error bars is standard deviation. Erroneous bars because of flaws in the design of processing of this type of data.

Chart 22: Average and median percent of correctly aligned audio with Sweden network with redundancy of 2. Error bars is standard deviation. Erroneous bars because of flaws in the design of processing this type of data.
APPENDIX AF - Latency and Amount of Correctly aligned audio - USA (Redundancy of 2)

Chart 23: Average and median latency of USA network with redundancy of 2. Error bars is standard deviation.

Chart 24: Average and median percent of correctly aligned audio with USA network with redundancy of 2. Error bars is standard deviation.
APPENDIX AG - Latency and Amount of Correctly aligned audio – Redundancy Comparison with Buffers of 2048 Samples and 4096 Samples

Chart 25: Latency when comparing no redundancy with redundancy of 2 using buffer of 2048 samples. Error bar is standard deviation.

Chart 26: Amount of correctly aligned audio when comparing no redundancy and redundancy of 2 using 2048. Error bar is standard deviation.
Chart 27: Latency when comparing no redundancy with redundancy of 2 using buffer of 4096 samples. Error bar is standard deviation.

Chart 28: Amount of correctly aligned audio when comparing no redundancy and redundancy of 2 using buffers of 4096. Error bar is standard deviation.
## APPENDIX A – Traceroute

### Table 7: Traceroute to Sweden

<table>
<thead>
<tr>
<th>Hop</th>
<th>Address</th>
<th>Latency</th>
<th>min</th>
<th>max</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.0.1 (192.168.0.1)</td>
<td>0.512 ms</td>
<td>0.335 ms</td>
<td>0.400 ms</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>sde-bbr-1-be10-10.net.comhem.se (213.200.166.123)</td>
<td>7.727 ms</td>
<td>7.637 ms</td>
<td>7.545 ms</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>vrr-core-1-be113.net.comhem.se (213.200.163.115)</td>
<td>12.234 ms</td>
<td>12.301 ms</td>
<td>12.411 ms</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>vrr-cgn-1-et2.net.comhem.se (213.200.168.28)</td>
<td>12.122 ms</td>
<td>12.236 ms</td>
<td>12.548 ms</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>kt-core-1-be19.net.comhem.se (213.200.168.23)</td>
<td>13.874 ms</td>
<td>12.651 ms</td>
<td>14.544 ms</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>vrr-peer-1-be1.net.comhem.se (213.200.162.38)</td>
<td>14.277 ms</td>
<td>16.750 ms</td>
<td>13.687 ms</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>bck-core-1.gigabiteth3-0-0.swip.net (194.68.130.21)</td>
<td>11.514 ms</td>
<td>11.430 ms</td>
<td>11.344 ms</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>bck-core-1.tengige0-7-0-14.tele2.net (130.244.71.54)</td>
<td>13.330 ms</td>
<td>13.616 ms</td>
<td>13.533 ms</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>lb2-core-2.bundle-ether5.tele2.net (130.244.71.217)</td>
<td>20.534 ms</td>
<td>24.299 ms</td>
<td>17.409 ms</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>vst-pe-2.bundle-ether7.tele2.net (130.244.71.177)</td>
<td>17.308 ms</td>
<td>18.985 ms</td>
<td>18.814 ms</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>130.244.12.250 (130.244.12.250)</td>
<td>16.700 ms</td>
<td>18.574 ms</td>
<td>18.491 ms</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>fw1-1.fw.core.loopia.se (194.42.55.176)</td>
<td>18.027 ms</td>
<td>17.935 ms</td>
<td>17.842 ms</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>controlcluster.loopia.se (194.9.94.162)</td>
<td>15.549 ms</td>
<td>13.169 ms</td>
<td>17.300 ms</td>
<td></td>
</tr>
</tbody>
</table>

### Table 8: Traceroute to USA

<table>
<thead>
<tr>
<th>Hop</th>
<th>Address</th>
<th>Latency</th>
<th>min</th>
<th>max</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.0.1 (192.168.0.1)</td>
<td>0.511 ms</td>
<td>0.368 ms</td>
<td>0.461 ms</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>sde-bbr-1-be10-10.net.comhem.se (213.200.166.123)</td>
<td>8.671 ms</td>
<td>8.582 ms</td>
<td>8.497 ms</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>vrr-core-1-be113.net.comhem.se (213.200.163.115)</td>
<td>11.242 ms</td>
<td>15.874 ms</td>
<td>15.791 ms</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>vrr-cgn-1-et1.net.comhem.se (213.200.168.20)</td>
<td>15.707 ms</td>
<td>15.625 ms</td>
<td>15.541 ms</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>vrr-core-1-be19.net.comhem.se (213.200.168.23)</td>
<td>20.983 ms</td>
<td>16.171 ms</td>
<td>18.269 ms</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>lib-core-1-be4.net.comhem.se (213.200.162.25)</td>
<td>15.901 ms</td>
<td>17.691 ms</td>
<td>17.549 ms</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>lib-peer-1-be1.net.comhem.se (213.200.162.42)</td>
<td>19.681 ms</td>
<td>17.416 ms</td>
<td>17.842 ms</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>212.237.192.246 (212.237.192.246)</td>
<td>26.306 ms</td>
<td>26.545 ms</td>
<td>24.129 ms</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>104.20.54.70 (104.20.54.70)</td>
<td>24.044 ms</td>
<td>23.958 ms</td>
<td>18.355 ms</td>
<td></td>
</tr>
</tbody>
</table>

### Table 9: Traceroute to Netherlands

<table>
<thead>
<tr>
<th>Hop</th>
<th>Address</th>
<th>Latency</th>
<th>min</th>
<th>max</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192.168.0.1 (192.168.0.1)</td>
<td>0.598 ms</td>
<td>0.342 ms</td>
<td>0.428 ms</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td>sde-bbr-1-be10-10.net.comhem.se (213.200.166.123)</td>
<td>8.243 ms</td>
<td>8.159 ms</td>
<td>8.079 ms</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>vrr-core-1-be113.net.comhem.se (213.200.163.115)</td>
<td>11.242 ms</td>
<td>15.874 ms</td>
<td>15.791 ms</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>vrr-cgn-1-et1.net.comhem.se (213.200.168.20)</td>
<td>15.707 ms</td>
<td>15.625 ms</td>
<td>15.541 ms</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>kt-core-1-be19.net.comhem.se (213.200.168.23)</td>
<td>21.463 ms</td>
<td>17.894 ms</td>
<td>17.725 ms</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>lib-core-1-be4.net.comhem.se (213.200.162.25)</td>
<td>17.929 ms</td>
<td>17.929 ms</td>
<td>17.929 ms</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>lib-peer-1-be1.net.comhem.se (213.200.162.42)</td>
<td>17.150 ms</td>
<td>17.150 ms</td>
<td>17.150 ms</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>80.249.213.143 (80.249.213.143)</td>
<td>40.165 ms</td>
<td>41.478 ms</td>
<td>38.876 ms</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>138.197.251.233 (138.197.251.233)</td>
<td>36.795 ms</td>
<td>36.795 ms</td>
<td>36.795 ms</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>178.128.246.209 (178.128.246.209)</td>
<td>38.822 ms</td>
<td>36.725 ms</td>
<td>36.487 ms</td>
<td></td>
</tr>
</tbody>
</table>