UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

First steps towards solving the café problem

Mariel Evelyn Markussen Ellingsen

INF-3981 Master's Thesis in Computer Science - June 2022

Abstract

Hearing loss, and assistive technologies to compensate for the loss, are becoming more and more regular. Hearing aids have improved the quality of life for many suffering from hearing loss but are still insufficient in some social settings. *The café problem* rises when there are a group of people talking in a relatively noisy environment where one person has hearing aids. Even with modern advancements, such as speech recognition and noise cancellation, people using hearing aids have difficulties differentiating the group's conversation from other noises.

This thesis will provide the architecture, design, implementation and evaluation of a mobile application as a first step in creating a system that can counter this café problem. A critical factor in a system like this is to reduce the audio latency to a minimum. We investigate where latency is introduced in the system by creating an experimental setup and evaluating the system.

We implement a prototype system and use the experimental setup to identify latency-inducing components. We discuss how this latency can be reduced and bring forward future steps that must be made in completing the system.

Acknowledgements

This master thesis is not only the results of one year of work but from the total of 5 years dedicated to CS at UiT. So, therefore, all credit I have to give encompasses all of these 5 years. 5 years of friendship made, lessons learned, supporting words, and tears shed (happy tears, I promise).

I want to thank my class and all of the friends I've made here and will keep for the rest of my life. A special thank you must be made to my "work wife". Thank you, Tromsøstudentenes Dataforening, for giving me an extra reason to stay in school. And a huge thank you to my supervisor Edvard Pedersen, for the helpful pieces of advice, for giving grammatical corrections, and for wanting to continue exploring where this application can go.

I want to express my gratitude to HLF Tromsø for granting me some of their knowledge about hearing loss and how it is to live with it. I must also thank my family. Especially my siblings, who may not understand *exactly* what I do but support me regardless.

My last "thank you, very much" goes to my close friends. They have supported me for most of my life, and have always, *always*, motivated me to continue on. They are my biggest supporters, my loudest cheerleaders, and my first phone call whenever I need to talk (and shed some tears).

Contents

List of Figures

List of Tables

1 Introduction

Hearing impairment affects nearly 20% of the world's population, an estimated 1.5 billion people. 430 million of these have a disabling hearing loss, where the most affected population comes from low- and middle-income countries [1]. In Norway, a survey from National Hearing Impaired society (HLF) found that every fifth Norwegian has difficulties with hearing.

Having a hearing impairment has a physical and mental impact on the hearing impaired. In Norway, the societal cost of hearing impairment, ranging from health care services to reduction of life quality, is estimated to be 40 billion Norwegian kroner [2].

Assistive technologies like hearing aids, cochlear implants, and external microphones have improved the quality of life for many hearing impaired. External microphones and telecoils connected to the hearing aids assist in providing adequate sound to the hearing impaired in situations where the hearing aids alone are not sufficient. An in-depth interview, from our previous work [3], with representatives from HLF mentioned that using and showing the hearing aids are often socially stigmatizing and therefore a sizable percentage of people who would benefit from hearing aids choose not to wear ones in public. Carneiro et al. [4] support this claim with their research on assistive technologies (focused on wheelchair users) and their social effect on both the users and the society around them.

In [3] a theoretical mobile application is proposed to solve the café problem.

The application transmits sound from a smartphone microphone to the hearing aids. Multiple smartphones can be used collaboratively to provide this audio. By utilizing smartphones, that are readily available, the socially stigmatizing effect of assistive technologies is reduced.

Based on [3] we know that one of the essential requirements is low audio latency. This latency sensitivity encompasses the time from speech until the hearing impaired hear it. Lowering the latency is vital since many with disabling hearing loss depend on being able to read the lips of the speaker.

The proposed application from [3] has many latency-inducing components. The fundamental architecture must have as low latency as possible to enable the inclusion of more functionalities.

1.1 Thesis statement

To realize this application, we investigate if creating such a system is feasible. From [3] we can assume that there are no physical obstacles to overcoming the latency requirements in a modern smartphone hearing aids application. We need to determine if the latency requirements can be fulfilled in practice.

To investigate the practicality, we pose the following research questions that we aim to solve in this thesis:

- How can we measure the latency from the smartphone microphone to the hearing aids activation?
- How much latency can we expect from a naive implementation of such an application?
- Which part of the architecture introduces latency into the system?

1.2 Thesis contributions

This thesis has three primary contributions—first, an experimental setup to measure latency between two devices handling audio transference. The design will simulate a real-life experience applicable to hearing aids users. Secondly, an application that runs on smartphones to share real-time audio. Lastly, an overview of the latency barriers in such a system.

1.2.1 Methodology

In this thesis, we work toward answering the research questions. In Chapter 2: Background, we familiarize ourselves with some required elements that impact this investigation.

The first research question asks how we can measure the latency from the smartphone to the hearing aids activation. We simulate the process using two smartphones and a microphone and gather the end-to-end latency measurements by creating an experimental setup. Chapter 3: Related Work looks at how similar experimental setups have been used to measure user-activated end-to-end latency through a system. In Chapter 6: Experimental Design, we propose our take on the experimental setup to measure the end-to-end latency.

In the chapters 4: Application Architecture and Design, and 5: Implementation, we create a system that will be used as a prototype when exploring how much latency we can expect from a naive implementation proposed in [3]. The chapter Results will give us precise measurements of the end-to-end latency from the prototype.

We analyze, process, and discuss the experimental setup and the implementation in the chapters 8: Evaluation and 9: Discussion. Here we get an overview of the parts in the architecture where latency is introduced and why. Additionally, in Chapter 10: Future Work, we propose ways to improve this system and the experimental setup. Moreover, we bring forward the factors that must be considered in the future when continuing the development of this system.

We summarize and bring forth the concluding findings from this thesis in Chapter 11: Conclusion.

2

Background

2.1 Sound

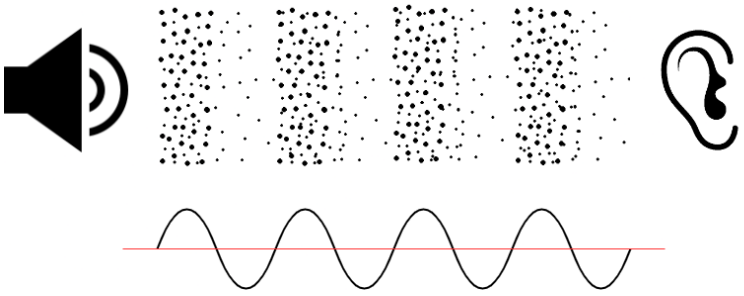

Figure 2.1: Vibration in air pressure with corresponding sinusoidal wave

Sound waves are longitudinal waves that travel in an expanding sphere from the source [5]. However, sound waves are often depicted in two-dimensional graphs with a fixed direction for visualization purposes. In figure 2.1 we see an example of how vibrations create air pressure and can be depicted as a wave. The compression of air particles can be seen as the amplitude of the wave, and the fluctuation of one amplitude length, or from one peek to the next, is a wavelength.

The frequency of sound is the number of waves that occur in a fixed unit of time. Frequency is expressed as Hz, where the literature generally states that

the human ear can hear sounds from 20 Hz to 20,000 Hz. The amplitude of a sound wave describes the sound's volume. The frequency of a sound wave describes the pitch of the sound.

2.1.1 Audio

Audio is the electrical representation of sound. Audio can be divided into analog audio and digital audio [6].

2.1.2 Analog audio

Analog audio represents sound with an AC voltage that coincides with the sound. When converting the sound to the AC voltage, the AC voltage should have the same frequencies and amplitude that the sound waves have. Analog audio technologies are used in microphone and speakers. The analog signal is continuous and has a constant change of amplitude and frequency.

2.1.3 Digital audio

Digital audio is introduced as a way to store audio. It is a computer-readable interpretation of sound [7]. The sample rate values describe the amplitude measurements from the analog audio. For creating accurate digital audio from sound, a concept arriving from digital sampling theory states that the signal of a specific frequency needs to have twice the number of samples. E.g. for creating a digital representation of a sound with a frequency of 20 Hz, the sample rate must be a minimum of 20 Hz.

The bit depth signifies the quality and audio resolution of digital audio. The more bits used for each sample (16, 24, or 32), the more accurate the amplitude measurements are.

2.2 Hearing impairment

A person *without* hearing impairment is said to have a hearing threshold of 20 dB or better in both ears. From this threshold of normal hearing at 20 dB, there is a gradient in the severity of the hearing loss, where a hearing threshold of 80 dB and higher is defined as being deaf [8].

The reason for impairment can be one or a combination of multiple reasons.

Some of the major causes of hearing loss as listed by [1] include congenital or early-onset childhood hearing loss, chronic middle ear infections, age-related hearing loss, and noise-induced hearing loss. Uses of ototoxic drugs can also damage the inner ear [1].

Some hearing loss is temporary with treatment, but a majority is permanent regardless of treatment. It is shown that untreated hearing impairment has a significant effect on the physical and mental well-being of the individual and those around them.

2.2.1 Hearing aids

Hearing aids are an assistive technology used by hearing impaired people. The hearing aids consist of three main components; the microphone, an amplifier, and the speaker. The microphone catches and translates the surrounding sounds to digital audio. The amplifier strengthens this digital signal before the speaker transforms the digital signal into detectable human sound and plays it. Additionally, hearing aids have a fitted earplug connecting the hearing device to the ear opening [9].

Analog hearing aids

The introductory capabilities of analog hearing aids are to amplify all incoming sounds equally without distinguishing between speech and noise [10]. Some analog hearing aids have progressed by having programmable microchips installed with settings to improve hearing in different listening environments, deducting speech from noise.

Analog hearing aids have a low hearing aid delay, meaning the amount of time taken from sound is received in the hearing aid microphone until it is processed and output. In [11], the analog hearing aids delay was tested by Frey as a contrast to digital hearing aids and showed a hearing aid delay of 0.5 milliseconds.

Digital hearing aids

Digital hearing aids convert the sound received to digital audio before amplifying it [10]. They have all the features of analog hearing aids but allow for more complex sound processing. This functionality improves the hearing quality, compared to analog hearing aids, in some environments. Digital hearing aids are more flexible and can be customized for specific patterns of hearing

loss.

In the hearing aid delay comparison from [11], the digital hearing aids have ten times the hearing aid delay as the analog hearing aids, with a delay of 5.5 milliseconds.

2.3 Swift

Swift is an open-sourced general-purpose programming language intended as a replacement for C-based languages (C, C++, Objective-C) [12]. The language is relatively new and modern (first appearance in 2014). It is built to match the performance and swiftness of C-based languages but with a more straightforward syntax that matches the preferred ways for developers to write code and maintain programs.

The language encompasses functionalities from low-level primitives such as type and flow control to object-oriented programming language design as classes, generics, and protocols [13]. Swift has automatic memory management and continuously checks arrays and integers for overflow. Swift can use C extensions through an Objective-C wrapper.

2.3.1 Mutlipeer Connectivity

Multipeer Connectivity is a framework provided by Apple to discover other iOS devices nearby, connecting and communicating between them [14]. Supported communication services between devices are message-based data, streaming data, and other resources, such as files. Multipeer Connectivity uses a variety of infrastructures for transporting data, such as Bluetooth personal area networks, Ethernet, peer-to-peer WiFi, and infrastructure WiFi.

The architecture of Multipeer Connectivity consists of multiple intractable objects. Integration and use of Multipeer Connectivity consist of two phases; the discovery phase and the session phase.

2.3.2 AVAudioEngine

AVAudioEngine is an object that manages a group of audio nodes and controls playback [15]. A node is a collections of available functionalities. These node instances must be attached to the engine before use and can be disconnected and removed during run time. The audio nodes have input and output busses

to serve as a connection point for the audio devices and are used for generating and processing audio [16].

A bus contains a format the framework expresses in terms of sample rate and channel count for managing audio input and output. For some nodes, the format must match strictly between the connected nodes, where a mixer node and an output node are the exceptions. The mixer node can correctly upmix or downmix any incoming channel count to the output channel and convert sample rates.

3

Related Work

3.1 Investigating the café problem for hearing impaired

The report *Investigating the café problem for hearing impaired* [3] was the result of a capstone project done at the University of Tromsø, the Arctic University of Norway, in 2021. In this report, the author proposes a theoretical application that provides audio from a collection of smartphones to a hearing aid user. It contains an in-depth interview with hearing aids users, an architectural proposal for the application based on the interview, and discussions mapping the difficulties of such an application.

The in-depth interview was with hard-of-hearing representatives from HLF Tromsø. The questions used in the interview were made to get the representatives to reflect on their daily problems as hearing aids users. The questions were phrased as "high-level" questions to let the representatives steer the discussion toward what is essential for them.

The purpose of the interview was to create the functionalities in the application based on the user's needs. The interview revealed the following wishes from the participants:

R.1 The application should be a cross-platform solution regardless of the smartphone and hearing aids vendor.

R.2 The application should provide noise cancellation without impacting speech.

R.3 The application should provide speech fidelity for the user without being face-to-face.

R.4 The application should provide support for personalizing the frequency of audio.

The architectural design from [3] tries to integrate the features requested by the participants and looks to related work on how to fulfill these in the proposed system. To address R.1 the architectural design consists of research, evaluation, and proposals on how to achieve cross-platform communication between smartphones with different operating systems. To address R.2 a noise cancellation feature is used on the recording devices to reduce the background noise and present a clear speech for the hearing aid user. By utilizing multiple smartphones and introducing speaker identification features, we fulfill request R.3 by giving the user the option to choose and hear any given speaker. To address R.4, a feature for personalizing the frequency of incoming audio is applied to the hearing aid user's smartphone. This enables the user to comprehend speech better.

Combining these functionalities will help fulfill the participants' wishes. However, the system's latency is heightened for each added functionality. It is then imperative that the fundamental structure of the system has such low latency that these functionalities can be applied while maintaining tolerable audio latency.

3.1.1 User story

A user story of the proposed system was conducted before the interview with the HLF representatives to show them its intent to solve the café problem. Later, this user story was modified to include the desired functionalities from the in-depth interview.

Scenario

Nora has hearing aids and meets three friends in a crowded café one Saturday afternoon. To allow Nora to participate in the conversations around the table, everyone in the group takes up their smartphones with the (h)ear application installed. Nora creates a united session and invites her friends to join it. This session will create a local network topology using the smartphone's WiFi chips

and Bluetooth. When the invitation is approved, everyone will put their smartphones on the table, with the microphones directed towards themselves.

Nora has a visual overview of all connected devices on her smartphone. She can choose whom to mute and whom to listen to. If they are unmuted and talk, Nora will be able to hear them. The microphone on the devices will catch the sound and send it to Nora's smartphone, where the audio will is assembled and sent to her hearing aids. Before the audio is sent to Nora's smartphone, a local machine learning algorithm will remove the background noise and unwanted speech from the recorded audio segment.

Nora has the option to adjust the incoming frequency of the audio. This functionality will enable Nora to hear the speech regardless of whom is talking. Some people with hearing loss may not hear some voices' pitch compared to others, so this functionality allows Nora to hear everyone.

3.2 Hearing Aid Delay and Current Drain in Modern Digital Devices

In [17], Alexander presents his research and investigation of throughput delay (hearing aid delay) and battery drain in digital hearing aids. Alexander provides comparative data and benchmarks from testing the hearing aids in his paper. The hearing aids used in these tests are from seven different vendors, with 62 hearing aids manufactured between 2009 and 2015.

Alexander uses the Frye Fonix 7000 Hearing Aid Test System to measure the delay *within* the hearing aids and the battery current drain with the appropriate battery substitution pill in the hearing aids. The 7000 Test System microphone collects information on the hearing aids for 20 milliseconds after an impulse is sent from sound chamber speakers to the hearing aids to measure the digital processing delay. The information gathered is a series of varying amplitudes, where the 7000 Test System finds the max amplitude peak and considers this the processing delay in the hearing aids.

In his paper, Alexander highlights the factors that influence delay and its acoustic and perceptual consequences. When talking about throughput delay, we refer to digital signal processing (DSP) hearing aids and not analog hearing aids. Analog hearing aids have a throughput delay but are inconsequentially small, unlike DSP hearing aids.

There are some factors that impact the delay when converting the audio signal

from the analog microphone to a digital representation. These factors are the resolution used, the sample rate, bit depth, the number of channels, and the algorithm's efficiency in converting the analog-to-digital representation. The higher the resolution is, and the more algorithms are needed, the processing time of the conversion increase.

Another determinant of delay for DSP hearing aids is the frequency specificity filters. These filters will analyze and adjust the incoming signal amplitude for each sample in time. It is not the filter bank that creates the delay but rather the amount of data arriving from input to output that needs to be processed and stored.

All hardware has its constraints and range from the processor clock and the number of transistors to the memory capabilities.

In some instances, the delay is not the problem, but it is the cause of the problem. These incidents are when the delayed signals interact with non-delayed or relatively less-delayed reference signals. We have audio-video dyssynchrony when audio is streamed to hearing aids but is notably slower than the visual from the tv. The same concept of audio-video dyssynchrony can be applied to delay when reading lips. Alexander points out that proficient lip readers express discomfort when the delay exceeds 40 ms, while other lip readers experience the same when the delay exceeds 80 ms.

Reference signals can be bone-conducted signals from the hearing aid user's voice through their skull or indirect sound coming through a large vent or open canal fittings. When the acoustic interaction between these signals is within +/- 10 dB they become most pronounced.

Humans are good at detecting audio delay, so engineers focus more on what delay is tolerable. They differentiate between delays caused by their voice and others' voices via direct acoustic pathways. There is also a difference in tolerance when the delay varies across frequency, which is often the case with filter banks in the hearing aids. Speech by others has been reported tolerable between 24 to 30 ms delay when the frequency is constant and 15 ms when the frequency varies. The tolerable delay of a person's own voice is 9 to 10 ms, so this has been the unofficial limit for engineers when designing hearing aids.

3.3 Measuring Latency in Virtual Reality Systems

Virtual reality (VR) systems have many complex, interacting components that affect the latency in the system. Raaen et al. have created an experimental setup that measures the end-to-end latency of a VR system, registering the time taken by a user triggers an action until the application processes it and the system outputs a response.

Raaen et al. have in [18] conducted and created an experimental setup for measuring the exact delay in VR applications. They define delay as the time a user takes to trigger an action until it is visible on a screen. They state that there is no linear relationship between frame rate and delay in traditional computer graphics setups and thereby assume that measurements through frame rate alone are not the best metrics.

Experimental setup component A VR device is attached to a camera tripod. The VR device alters between being smartphones with VR capabilities and VR headsets. If moved, the device will alter its virtual screen from white to black coloring.

One light sensor is connected to the VR device's screen and will capture the color switch. A laser pen is attached to the tripos and points towards a light sensor stationed one meter away from the tripod. When the tripod alters, the laser pen will signal this to the light sensor. Both light sensors connect to an oscilloscope, where the waveform and electrical signals' bandwidth will be captured and displayed.

The experimental setup flow is as follows: The laser pen signals the light sensor of its movement. The VR device's screen alters its color due to the movement, and the light sensor connected to it registered this shift. The oscilloscope registers the signals from both light sensors. The measurement between the first and second light sensors reflects the VR device's delay from a triggered action to the visual aspect rendered.

Raaen et al. conduct their experiments using Oculus Rift, iPhone, and Samsung as their VR device. The results provided are an average of 10 runs. For the Oculus devices, they test the experiment with and without VSync (vertical sync). VSync is a graphic technology that synchronizes the application's frame rate with the screen's refresh rate[1]. When not using VSync, the monitor has a chance of displaying portions of multiple frames in one go. This will create straight lines on the monitor and give the viewer a sense of cut and overlapping

1. Nvidia Adaptive VSync: `https://www.nvidia.com/en-gb/geforce/technologies/adaptive-vsync/`

images. From the tests performed by Raaen et al., the Oculus Rift devices not using VSync have the least delay. The results can be viewed in table 3.1.

VR Display	Average in ms
Oculus Rift dev kit 1, VSync ON	63
Oculus Rift dev kit 1, VSync OFF	14
Oculus Rift dev kit 2, VSync ON	41
Oculus Rift dev kit 2, VSync OFF	4
Samsung Galaxy S4(GT-I9505)	96
Samsung Galaxy S5	46
iPhone 5s	78
iPhone 6	78

Table 3.1: Results from delay measurements

Their paper mentions that the visible results of not using VSync might not be preferable to the user but that this is out of scope for the goal of the experimental setup. The goal is to provide a correct and trustworthy experimental setup that can measure the delay in VR devices and their applications.

4

Application Architecture and Design

This chapter will describe the architecture and design of the system. Within this and the following chapters, devices will be referred to as smartphones that have the applications installed.

4.1 Architectural overview

The system's primary purpose is to handle audio recording, audio transference between devices, and playing. A user activates a recording session through the device's user interface. The surrounding sound is recorded and sent to a connected device in its vicinity. Here it is played instantaneously from the device upon receiving the audio.

The system has three main architectural divisions: the part *shared* between all devices and the two *individual* parts for recording and receiving audio.

The shared part of the system handles the identification of nearby devices and audio transference between them. The individual part of the system consists of the recording and the receiving applications. The application consists of the module, the local storage, and the user interface with the shared components.

The recording and receiving applications must be installed on at least one device. As the naming implies, the recording application is responsible for recording the sound and the receiving application for playing it.

In figure 4.1 we can see an overview of the shared and the individual parts of the system.

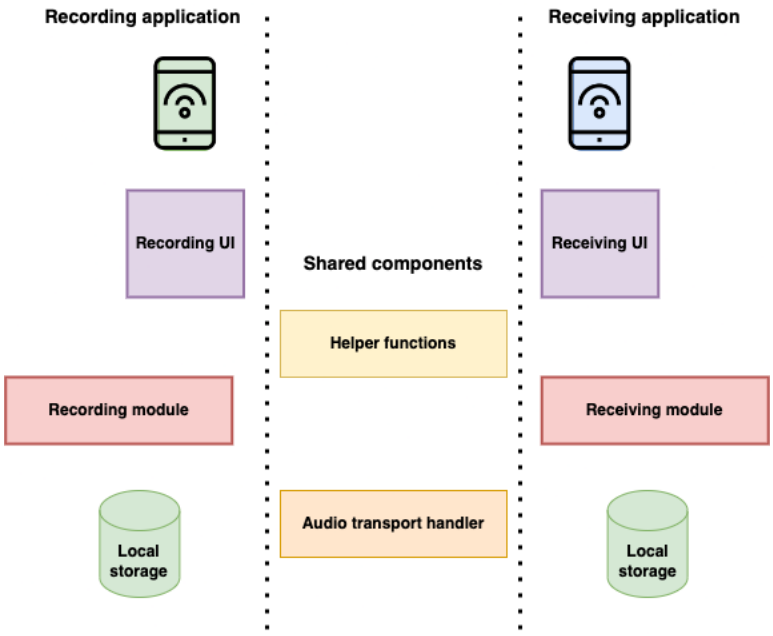

Figure 4.1: Architectural overview. Components between striped line is shared by devices.

The connection will be maintained regardless of the applications being in a recording session when the devices are connected. Only if an application terminates will the connection disband.

If there is a session with more than two devices, the devices construct a many-to-many relationship. For the experimental evaluation of the prototype, this thesis focuses solely on the connected pair of one recording application and one receiving application.

4.2 Design

4.2.1 Audio transport handler

Connecting peers

The audio transport handler is responsible for identifying and connecting the devices in a joint session. When first activated, it will attempt this and consistently maintain the connection if the applications are not terminated. The system will search for devices to connect to in its vicinity and continue this search after the first initial connections.

The connected devices, referred to as peers, are displayed through the application's user interface. The system does not need acceptance from another device to approve this connection.

Transference of audio

Sending and receiving packets is done by the audio transport handler. When sending a packet, it broadcasts this packet to all connected peers, if they exist. The handler does this immediately when the method is called.

The receiving method inside the handler is constantly listening for new packets. When a packet is received, it will restructure the raw data to an object that other modules can use further.

4.2.2 Recording application

The recording application consists of the recording user interface, a recording module, local storage, the audio transport handler, and shared helper functions.

The recording module is activated through the application's user interface. It is responsible for continuously gathering data from the microphone on its device. The application uses the audio transport handler to broadcast the gathered data to its connected peers. The helper functions assist in seamless converting data to correct types for processing.

4.2.3 Receiving application

Except for the receiving module, the receiving application consists of the same architectural components as the recording application. The receiving application utilizes these shared components differently than the recording application.

The application's user interface readies the receiving module through a button. The receiving module accesses its local storage to prepare itself to receive packets from a recording application.

The audio transport handler receives a packet consisting of an audio segment and additional information on how to play it. This packet is dispatched to the receiving module. The receiving module will use the helper functions to convert the audio segment before scheduling it to be played on the device. The receiving module then starts to play the audio accessible.

4.2.4 Experimental design

There are some specific functionalities for measuring the application's performance during run time. These measurements with time stamps are stored as logs. Both the recorder and the receiver applications have these.

Two buttons are available for interacting with the logs through the user interface. One is for deleting the locally stored logs, and the other is for gaining access and printing the logs when the device is connected to a computer.

During run-time, the logging activates and begins according to when the recorder and receiving mode is started. All logs for both devices are written down to a locally stored file when the recording and receiving session is over.

5

Implementation

This chapter will discuss the implementation-specific detail of the components in the system. We describe the method used to assess the system performance for the experimental evaluation and the development tools used with their versioning.

5.1 Development tools

The user interface uses SwiftUI[1] to create the user interface elements and attach them to correct actions. There are two primary user interfaces that the user can interact with, as seen in figures 5.1 and 5.2. Figure 5.1 shows the user interface for the recorder application, and figure 5.2 the receiving application.

1. SwiftUI Documentation: `https://developer.apple.com/documentation/swiftui/`

Figure 5.1: Screenshot of the user interface for the recorder device

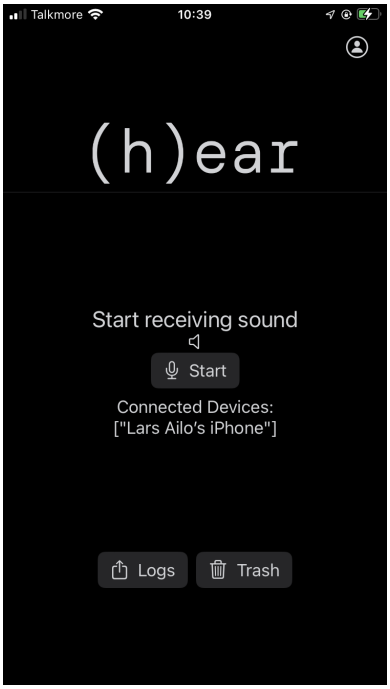

Figure 5.2: Screenshot of the user interface the for receiver device

Xcode[2] is the integrated development environment the system has used for compiling and building the applications. The specifics of versioning used are listed below:

- Xcode version 13.3.1
- Swift language version 5
- iOS Deployment Target: 15.2
- Simulator version 13.3.1 (977.2), using iPhone 13 iOS 15.4
- SwiftUI version follows the iOS target, where the lowest possible target is iOS 13.0

2. Xcode Documentation: `https://developer.apple.com/documentation/xcode`

5.2 Audio formatting

The recording module predefines a format to represent the incoming sound as digital audio. The format must consist of the sample rate, bit depth, and the number of channels. The module uses AVAudioFormat[3] to create the audio format object. The formatting object is treated as unique for each recording session. The format information from the recording module is needed in the receiving module to interpret and play the audio. The information is sent alongside the audio segment to the receiver application.

To initialize the receiver module's audio subsystem, it sets a formatting object. The audio subsystem uses an audio file from the local storage to get a formatting object baseline. When the receiver module receives the first packet from the recording application, it updates the current formatting object.

The values used for the audio formatting object are

- Bit depth: float 32-bit
- Sample rate: 44,1 kHz
- The number of channels: 1

5.3 Audio subsystem

The audio subsystem uses AVAudioEngine, referred to in Chapter 2.3.2, for handling all audio I/O in the system. The *engine* in the recording module consists of three nodes[4] and uses two nodes in the receiving module.

An input node is attached to a mixer node in the recording module, and the mixer node is attached to the main mixer node. The main mixer node is a property of the engine that is directly connected to an output node by default when first initialized.

The input node connects to a bus that receives the audio stream from the microphone on the device. The audio stream data is sent from the input node to the mixer node. The mixer node applies the proper formatting properties for interpreting the audio signal as digital audio. Lastly, the mixer node's output

3. AvAudioFormat documentation: `https://developer.apple.com/documentation/avfaudio/avaudioformat`
4. Functional elements for handling audio

is attached to the engine's main mixer node.

The receiving module has a player node that supports scheduling the playback of a PCM buffer[5]. The player node connects to the engine's main mixer node. Here the audio is delivered to the device's speaker.

A mixer node can convert the formatting object of an audio sample. This property is desirable if the devices used to record and play audio have different hardware restrictions and diverge from each other. E.g., the microphone used for recording comes from a Bluetooth headset, while the audio sample is played through a smartphone speaker. The prototype does not utilize this functionality since the devices have the same sample rate in the experimental setup.

5.3.1 Recording method

The recording module instantiates the mixer node to access and process the audio from the input node. The mixer node uses a function called installTap to buffer up the audio data from the microphone. The size of this buffer is predecided before the installTap function is activated. A code block is called to handle the audio data when the microphone buffer is full. The audio data is structured into a value type called Packet with its corresponding formatting information and sent to the audio transport handler. The audio data from the buffer is not compressed before transportation.

5.4 Audio transport handler

The audio transport handler utilized the Multipeer Connectivity framework, see Chapter 2.3.1, for establishing a connection and sharing audio between devices. The framework uses the Bonjour[6] protocol to discover, initiate and connect to nearby devices.

The handler provides a list of all connected peers to the user interface. The data recorded by the recording module is broadcast to said peers by the handler. When the handler receives the broadcast data, it dispatches the data to the receiving module for playing.

5. AVAudioPCMBuffer documentation: `https://developer.apple.com/documentation/avfaudio/avaudiopcmbuffer`
6. Bonjour documentation: `https://developer.apple.com/bonjour/`

Packet

The packet transmitted between devices consists of the audio data and its corresponding formatting object. These are converted and combined to a Data[7] object, which is a byte buffer object that can be manipulated as a Foundation[8] object. When reverted to a string form, the formatting object has a known byte count. When received by the receiver application, it can easily be parsed out from the Data object and remade into a formatting object.

Dispatch queue

DispatchQueue[9] is a DispatchObject that manages the execution of tasks. These tasks can be serial or concurrently done on the application through a pool of threads but are associated with the application's main thread. The queue used in this system is a serial FIFO (First in - first out) queue, running tasks in parallel with the rest of the system. The tasks submitted are in the form of a block object and behave similarly to the dispatch function used in other languages.

A dispatch queue is used in the recording module to dispatch the collected audio data in a Packet to the audio transport handler. The audio transport handler uses a queue to broadcast the Packet to the connected peers. The queues might be excessive, depending on the buffer size. It is a precaution to utilize multiple threads so that data intake from the microphone buffer is not halted.

In the receiving application, the audio transport handler uses a dispatch queue to assemble the incoming byte data to a Packet object and dispatch it to the receiving module

5.5 Logging the system performance

The system stores log with timestamps in an internal queue when activating a recording session from the user interface. When the session terminates, the log is written to a file in the local storage on the device.

7. Data documentation: `https://developer.apple.com/documentation/foundation/data`
8. Foundation documentation: `https://developer.apple.com/documentation/foundation`
9. DispatchQueue documentation : `https://developer.apple.com/documentation/dispatch/dispatchqueue`

These files can be accessed when the device is connected to a computer and runs the application. All stored files are viewed in the compiler through a button on the user interface.

We log the system where we assume latency is introduced. In the recording module, we log the start and end of the recording process. We get the time used for filling the buffer with audio data alongside the time used for dispatching this data to the audio transport handler. The time used for transferring the packets between devices is not available.

The logging starts after the packet is first received in the audio transfer handler in the receiving application. We log the time in the receiver module from receiving a packet until the audio segment is scheduled for playing.

6

Experimental Design

This chapter will describe the design of the physical experimental setup and the purpose and motivation of the different components.

The objective of the experimental setup is to measure end-to-end latency of a multi-device application through audio transmission. The experimental setup will simulate a real life use case for hearing impaired people using hearing aids and assistive technologies.

6.1 Design

The experimental setup consist of the following equipment:

- 2 x smartphones with recorder and receiver application
- Computer with spectrogram analysis
- Microphone
- Sound absorption barrier
- Sound emitting device (clicker)

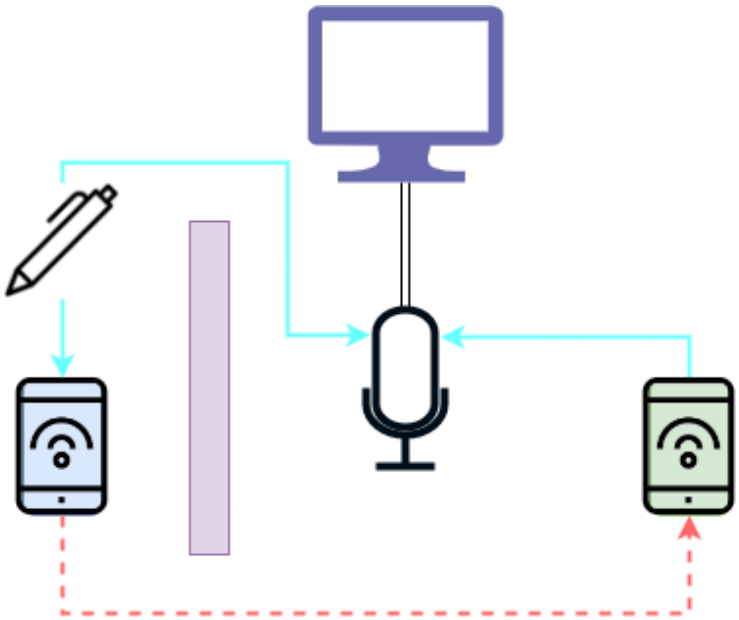

Figure 6.1: Experimental setup. Filled lines is sound transmitted through air. Dashed arrow represent audio transference between devices. Rectangular box is the sound absorption barrier. Pen is the sound emitting device.

6.1.1 Computer and microphone

The microphone is connected to the computer as the input. The microphone will pick up the sound from the clicker and the receiving device.

The computer runs a spectrogram program called Sonic Visualiser[1]. When recording from the microphone, all sound waves are pictured in real time. The program allows for precision in measuring the time span between registered sound waves.

Figures 6.2 and 6.3 shows examples of how the sound waves are displayed in the spectrogram program with and without marking the time differences.

6.1.2 Sound absorption barrier and sound emitting device

The sound emitting device, clicker, must emit the same type of sound for every time its used. The ideal sound gives a high amplitude when made and will thereby give as clear an image analysis as possible. After experimenting with

1. Sonic Visualiser: `https://www.sonicvisualiser.org/`

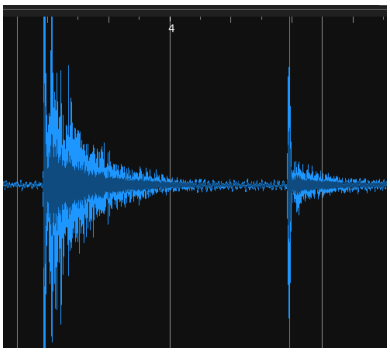

Figure 6.2: Screenshot of registered sound waves without markings

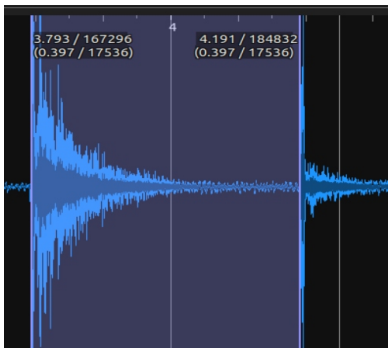

Figure 6.3: Screenshot of registered sound waves with markings

different types of sound sources, a large marker with a cap was found to be most proficient.

The purpose of the sound absorption barrier is to dampen the feedback between the software devices. It is built up with cardboard and tape to create a rectangular shape.

6.1.3 Device

The device have build-in microphones and speakers. In the experiment the devices must be connected and able to record, transfer and play the audio to be measured by the microphone and computer.

6.2 Experiment setup and execution

Figure 6.4: Picture of the physical experimental setup with all components

6.2.1 Setup

To give as consistent experiment through eventual iterations and runs, all components layout have been measured up and marked. The distances to notice of importance is between the recording device and the clicker, the clicker and the microphone and the receiving device and the microphone. The distances between the microphone and both devices is even, where the clicker is placed in proximity of the recording device. These distances is to to balance out the time sound travels through air, and the introduced delay this will take is even from both recorder and receiver side. With doing this, the latency will be equal on both sides.

The environment the experiment is set up in has been maintained the same. This is to reduce the human and environmental error factors. The microphone is connected to the computer and tested before the experiment begin. The baseline of the sound intake from the environment is checked and inspected before every experiment. This is viewed through the spectrogram and give an indication of the background noise in the room.

6.2.2 Execution

After the setup is done and both smartphones are prepared the experiment can begin. The spectrogram on the computer starts recording. The devices is readied to record, transmit and play sound. The clicker is then activated and makes a clear sound while the experimenter views the spectrogram to make sure it is registered. The smartphones is paused before much of the feedback is registered by the spectrogram after the clicker sounds are registered by both devices. The spectrogram program is paused and the sound wave spikes can be analysed and saved.

6.3 Limitations

Error factors that must be considered in this experiment is divided in human and environmental errors.

6.3.1 Human error

The experiments results depends on the experimenter being able to correctly mark and read the sound wave spikes in the spectrogram. Depending on the sound emitting device and its amplitude and frequency, the beginning of the original registered sound and the sound registered from the second device can vary and be hard to precisely know when it is starting if the amplitude is not pictured precisely enough.

The sound emitting device used gives a sharp click sound. The sound emanated from the cap pushing on a marker can vary based on how it is pressed. This will vary the sound made and must be taken into consideration when analysing the spectrogram.

The recording device is turned on right before the clicker is clicked, to reduce the time of feedback as much as possible. There is a correlation between when the device is starting to record, the sound is emanated and when the software in the device transfer the audio that can variate between each experiment run. This has to do with the microphone buffer on the device, but the human error here is where the length of time from starting the recording session on the device and creating the clicker sound can vary.

6.3.2 Environmental error

The environmental errors is tied to the room used and the physical setup of the experiment in relation to each other including outside sound sources. The errors in regard to the setup of the experimental components is reduced by marking each device positions.

Since the experiment is sound sensitive, the sound levels of the room must be known and consistent if there is some. The consistent sound levels, e.g. ventilation system, should be registered and seen on the spectrogram before the experiment run. With knowing the baseline of environmental sounds, the sound emitting device must give a sound that clearly breaks from the baseline to get as accurate measurements as possible. If there is other noises that occurs during the experiment, the experimental interval must be evaluated if it can continue.

7

Results

This chapter will cover the results from experiments conducted with the experimental setup from Chapter 6 and the application logs from Chapter 4.2.4. The results will indicate what the end-to-end latency of the system is, and where this latency internally arrives from. The analysis will give indication on where future work efforts should be focused.

There have been three experiments conducted in an iterative fashion. For each experiment alterations on the system was made. Every experiment is run six times, where the results provided is the average of these. At the end of this chapter an overall visualization of the end-to-end latency and system performance measurement will be provided. The evaluation of these results will be discussed further in Chapter 8.

7.1 First experiment creating the baseline

The first experiment was run after the initial version of the system was finished. This experiment was to evaluate if the experimental setup held true, and to get a baseline on the end-to-end latency from the system to compare for further testing.

The average results from the overall latency in this first iteration was **361.8 ms**. The logs from the applications output indicates that the time used for filling

the microphone buffer is large. With a microphone buffer of size 9216 (36 864 bytes) it was seen likely as a large amount of the latency was connected to this buffer.

In figure 7.1 an overview of the end-to-end latency from the physical experimental setup, the time of filling the microphone buffer and dispatching one packet, and at last, the time from receiving to processing is done for one packet on the device.

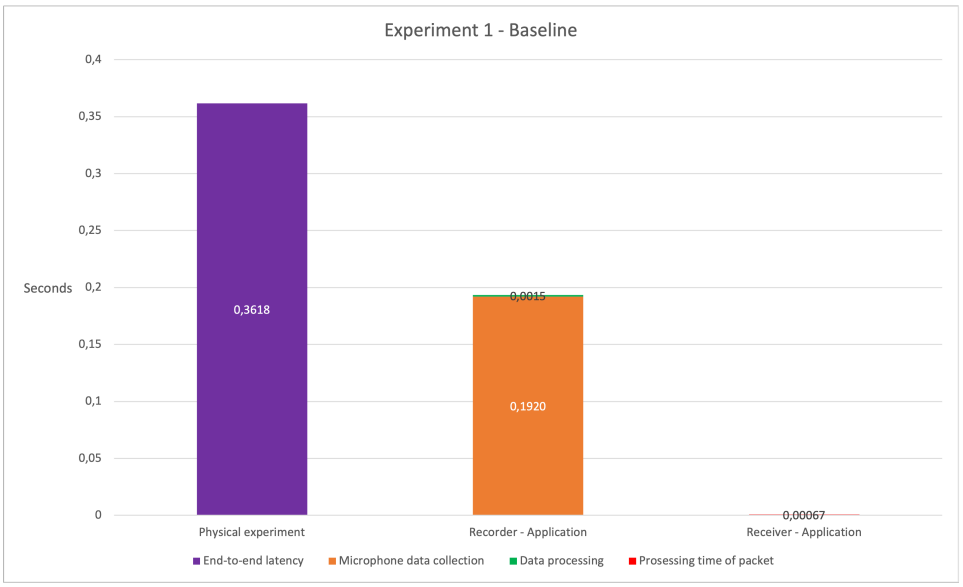

Figure 7.1: End-to-end latency and internal processing time for recording and receiving device

7.2 Reducing the microphone buffer

The microphone buffer was first reduced to a size of 2304. The audio transference was first checked by the developer to get a sense of the quality. The sound on the receiving device was perceived as stilted and unintelligible. The second test was then performed with a microphone buffer with a size of 4608, which is half the size of the microphone buffer from the first experiment.

The average result on the end-to-end latency from the experiment after this modification was **246.5 ms**. This is **115.3 ms** less than the first iteration.

7.3 Receiver buffer

To reduce the the microphone input buffer even more, and without getting the stilted experience in the audio, we introduced a queue on the receiving side for handling the incoming audio packages. The audio would start to play the queues content if the queue contained at least two packages. The microphone buffer size was reduced to a size of 512.

Before testing with the experimental setup, the quality was evaluated and there was no stuttering or unintelligible sound and the queue was checked to be working as expected. The experiment was carried out and the end-to-end latency resulted in **301.7 ms**, which is **55.2 ms** more than the previous experiment.

7.4 Graphs

7.4.1 Combining all results from experiments

Figure 7.2 shows the combined end-to-end latency and internal buffering time for the recording and receiving device. As with figure 7.1, we see that the average latency from receiving a packet is low and not significant in comparison.

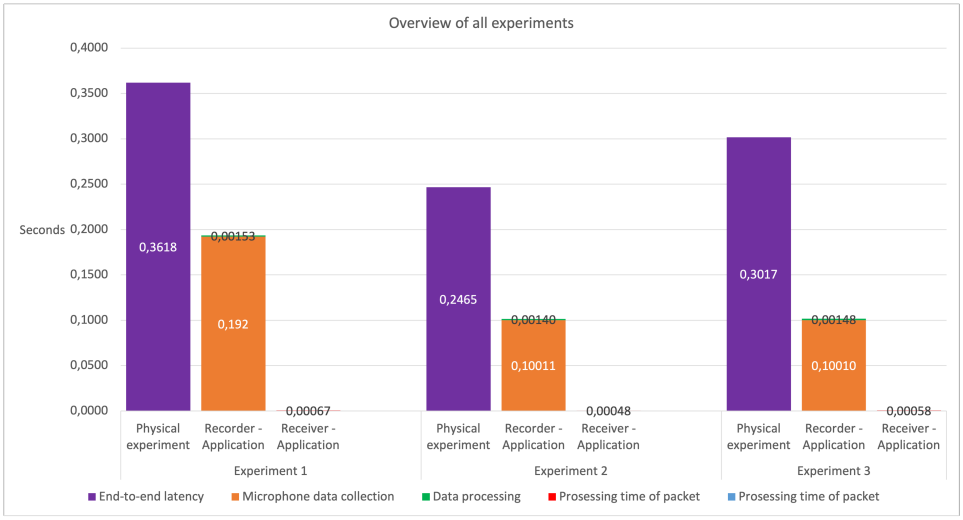

Figure 7.2: Latency and average of internal processing time for recording and receiving device for all experiments

Between the first and second experiment, reducing the microphone buffer

shows a drastic decrease in the end-to-end latency. However, the thirds experiment where the receiver queue is introduced and the microphone buffer is of a supposedly size of 512, has an increased end-to-end latency. The column depicting the microphone buffer time is exact the same as for experiment 2.

The processing time for receiving and scheduling the audio packet on the receiver device is only increased with **0.1 ms** indicating that the queue does not have much affect on the higher end-to-end latency seen in the experiment.

7.4.2 End-to-end latency

In figure 7.3 we see a box plot overview of the end-to-end latency for all 3 experiments from the physical experiment. Experiment 1 has the most spread results in the end-to-end latency of all three experiments, where the lowest measurement is the same at experiment 2's median.

We see that both experiment 2 and 3 is more consistent in all their measurements, compared with experiment 1.

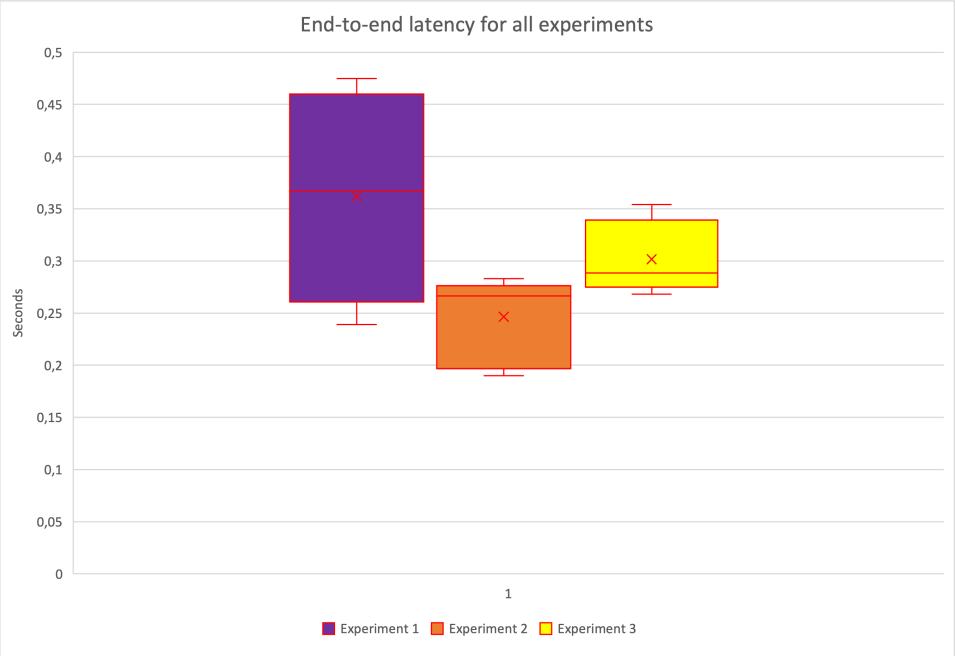

Figure 7.3: Experiments end-to-end latency

7.4.3 AVAudioEngine initialization

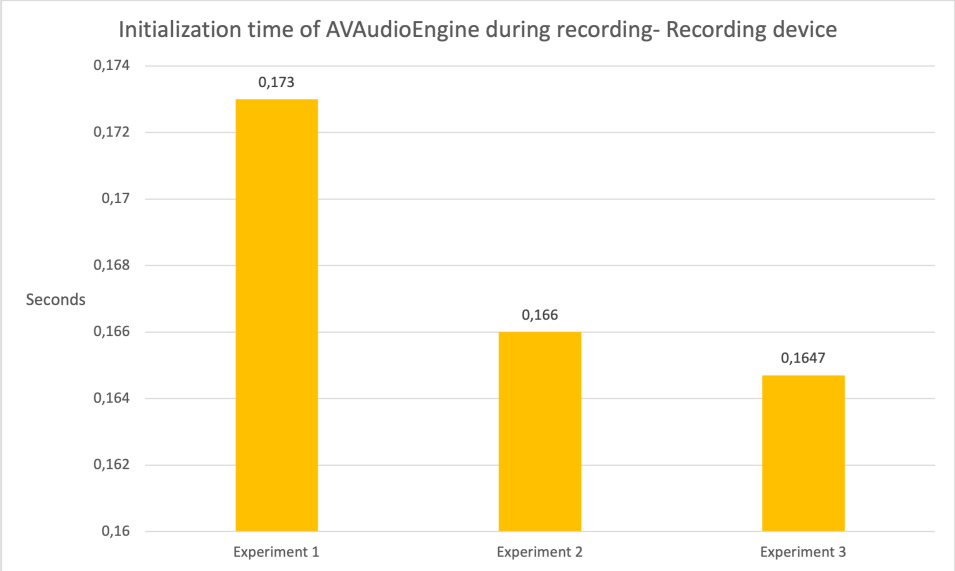

Figure 7.4: Initialization time of AVAudioEngine for recording

In figure 7.4 we see the time used from activating a recording session by pushing the button on the user interface until the recording module begins to gather the microphone data. The figures' measurements do not affect the end-to-end latency values or other measurements taken in the experiment that has been presented.

The time does not waver that much between the experiments, where the spread is **8.3 ms** between experiment 1 at **173 ms** and experiment 3 at **~165 ms**.

8 Evaluation

In this chapter, we will look at the results provided by the experimental setup from Chapter 7 and evaluate the registered measurements.

8.1 Audio length calculation

Before we start the evaluation, we need to understand how the buffer size, sample rate, and bit depth affect the audio file size and its quality, and from there, we can calculate the number of seconds with audio the packets consists of.

The formulas that will be used and explained in this section are based on the OMNI calculator[1]. First, the number of bits per second (dubbed bit rate) is calculated by multiplying the bit depth with the sample rate.

$$\text{bit rate} = \text{bit depth} * \text{sample rate}$$

1. OMNI calculator: `https://www.omnicalculator.com/other/audio-file-size#audio-file-size-calculation-formula-and-how-to-calculate-audio-file-sizes`

The bit rate is consistent throughout all the experiments and the value is $32 * 44100 = 1\ 411\ 200$, with gives 176 400 bytes per second when divided by 8.

The number of channels used when recording the audio affects the number of recorded waveforms in the audio samples. Each waveform has its designated speaker to give an immersing effect. One channel is used for all experiments in this thesis.

The formula for calculating audio file size is then as follows:

$$\text{audio file size} = \text{bit rate} * \text{audio length in seconds} * \text{number of channels}$$

Furthermore, since we know the audio file size from the microphone buffer size, we can restructure the formula to produce the audio length in seconds to this:

$$\text{audio length in seconds} = \frac{\text{audio file size}}{\text{bit rate} * \text{number of channels}}$$

8.2 Buffer size impact

From figure 7.1 we see our first overview of the end-to-end latency and processing delay in the system. The Recorder-Application and Receiver-Application column represents the processing time for *one* packet, while the Physical experiment column can be the result of more than one package.

The processing time for the receiving device is barely visible and represents the time the device uses to receive and process one packet. It does not include the actual time for playing this packet, but using the formula from Chapter 8.1 we can calculate the audio length it received in one packet. The audio length for one packet in experiment 1 is **209 ms**.

The processing time used by the receiver, regarding the audio length played, is of such a nominal length that this should be disregarded compared to the processing time used by the recording device for now.

It is clear that the recording device, and the time used to gather the microphone data, are more prominent regarding the high end-to-end latency. As with the time used in the receiving module, the processing time used to access the microphone buffer and dispatch the data has minimal impact.

We know the calculated audio length in a packet is 209 ms for experiment 1. While we do not know precisely how many packets are needed to transport

the sound emitted from the clicker device, we can assume that the maximum number of packets used is two, based on the end-to-end latency. Combining the time used for recording and playing the audio sample, one packet amounts to an audio length of **418 ms**.

8.2.1 Reducing the microphone buffer

In figure 7.2 we see a comparison between all experiments and their results. Reducing the microphone buffer size has affected the end-to-end latency between experiments 1 and 2. The graphs reinforce the first statement that the microphone buffer size affected the first experiments' end-to-end latency.

What is curious about this observation is that the supposed microphone buffer size of 512 (2048 bytes) in experiment 3 shows the same processing as for experiment 2. The end-to-end latency measurements for experiment 3 are higher than for experiment 2.

installTap

With this observation and results from figure 7.2, the installTap function needs to be looked at more. Some measurement confirmation tests are run by altering the microphone buffer size and comparing the size by validating the data length received from the buffer. It shows that regardless of the buffer size set, the function uses a minimum of **19200 bytes** to allocate the microphone buffer. This is a buffer size of 4800.

The documentation from Apple regarding installTap is not incredibly elaborate, but when reading it carefully, it states: .. *bufferSize: The size of the incoming buffers. The implementation may choose another size*. The installTap function will not be less than 19200 bytes, regardless of the preset buffer size.

We know that the correct microphone buffer size for experiments 2 and 3 is 19200 bytes with this additional information.

8.2.2 Theoretical impact

There are some inconsistencies within the theoretical number of audio length that is recorded and the physical measurements of the end-to-end latency when we look at figure 7.2.

While the end-to-end latency has been reduced from experiments 1 to 2, this

reduction is not linear with the reduction of latency from the microphone buffer. The end-to-end latency has declined with **32%** from experiments 1 to 2, and the latency from the microphone data buffering has declined with **48%**. This indicates that the microphone buffer size is not the only latency-inducing factor in the system. If it was, the latency should have downscaled linearly.

In experiment 1, we know that the audio length provided by one packet that is recorded *and* played is 418 ms. The end-to-end latency is **361.8 ms**. We know that the system can have other unidentified latency factors. However, if we calculate the numbers we have access to, the remaining **56.2 ms** of the theoretical audio length is not reflected in the end-to-end latency. It indicates that the microphone buffer size is more extensive than necessary.

In experiment 2, the audio length recorded and played assembles to **216 ms**. It is **30.5 ms** *less* than the end-to-end latency from experiment 2. It can either indicate that more than one packet is used, that there is unknown latency introduced of at least 30.5 ms, or that the intervals used for measuring these are too few to give exact representative measurements of latency.

In experiment 3, the end-to-end latency increases by **22%** from experiment 2, and the microphone buffer latency is the same. This increase can be tied to the additional queue on the receiver application or tied up to another unknown factor that introduces latency.

8.3 End-to-end latency

In figure 7.3 we see a box plot representing the end-to-end latency of the experiments. In experiment 1, there was a large microphone buffer. We can see in the plot that the minimum and maximum points of the numbers varied a lot. It can indicate that the sound throughput has varied, which is consistent with the first assumptions made in Chapter 8.2.

For experiments 2 and 3, we observe fewer variations and less dispersion with smaller microphone buffer sizes. Some size differences between the experiments can indicate that the size of the microphone buffer still exceeds the audio length that the sound emitting device makes and that this has influenced the varying data points. Alternatively, it can imply that there are unknown latency-inducing factors. In experiment 2, we see that the median is high in the interquartile range, indicating that over half of the latency measurement was closely related to each other. The same can be said for experiment 3, only with a reversed, positive skewness instead of negative skewness.

The skewness further indicates that the data points on the skew side (the smallest part of the box) are closer to each other in forms of value than the rest of the data points. Experiments 2 and 3 have a reversed skewness but are the closest relative. It tells us they have had many similar values in their end-to-end latency. This is understandable when they have operated with the same microphone buffer size, whereas experiment 3 has had some added delay with the receiving devices queue.

The box plot's overall dispersion for the experiments indicates that the end-to-end latency has varied a lot. More intervals to create an intermediate representation could have given a more accurate indication of the system's end-to-end latency.

8.4 One-time induced latency

Figure 7.4 shows the measured time used when the recording session is first initialized. This period consists of the recording button being pushed, the time allocation of the microphone buffer, and the startup of the engines.

The time used is considerable enough that it should be noted and should be handled, but it only happens once for every recording session. This does not affect the end-to-end latency measurements but can affect the overall user experience of the system. Especially if the user often switches between whom to listen to. The added latency can be subtracted from the session. The results of this profoundly depend on how the end-user uses the system and how multiple connected devices will interact with each other.

9 Discussion

The discussion in this chapter will reflect on the research questions from Chapter 1.1 by looking at the experimental setup, the systems design and implementation choices, and the results gathered from the evaluation.

9.1 Experimental setup

The experimental setup disclosed in Chapter 6 is one of the thesis contributions. This section will evaluate the experimental setup's design and equipment and provide ideas on how the setup can evolve in the future. Before we start to evaluate the different components, the experimental setup goal should be clarified again. The goal of the experimental setup is to measure the end-to-end latency in an audio sharing system, simulating how a user would interact and perceive the transferred sound in as realistic circumstances as possible.

The reading of the end-to-end latency strictly depends on the exactness of the spectrogram. The exactness entails the visualization of the recorded sound and the reading of this recording. How reliable and precise the spectrogram in and of itself is can not be known for sure. The spectrogram software has not been cross-referenced for accurateness with other spectrogram software. If we assume that the visualization of the sound is accurate in the software, we still have the error factor of *reading* the output of the spectrogram accurately. While it is somewhat dependent on the sound registered, it can be hard to know the

precise start of the sound wave. In the best case, this could add milliseconds of uncertainty to the presented results.

The sound-emitting device used is a marker with a cap. The sound registered was deemed good enough when testing, but it should preferably be a more reliable sound that does not alternate based on how it is clicked. The sound used should follow some specific properties to ensure a good reading. With this in mind, a device specifically designed to produce consistent clicks can be applied to these experiments and perhaps give a more accurate reading.

Even though each experiment execution involves a few steps, it contains enough human-made steps that can give the experiments uncertainty and lack repetitiveness. More of the experiment should be automatized. This ensures that if there are external error factors, they will be constant and more transparent when analyzing the data.

Isolating the experiments in a more contained environment, as done in [17] with the 7000 Test System, will reduce some external error factors. However, the experimental setup will lose its quality as it focuses more on the end-to-end latency of the system from a user's use-case perspective.

The experimental setup is made distinctively for having one recorder and one receiving device. In the future, it should be changed to accommodate more devices. The experimental setup can evolve, opening up for testing and evaluating additional functionalities, such as the audio quality and the noise-canceling capabilities. These features need to be tested to measure their effect and correctness in a group setting. This thesis contribution can contribute to more extensive development that measures and evaluates these things and the end-to-end latency.

In the results of the experiments, we have some unknown latency factors and dispersion of the end-to-end latency measurements. We think that running more iterations to acquire the average will give more insight into these unknown factors and the dispersion of measurements.

9.2 System improvement

9.2.1 Restructure and make alternations of code to remove latency

We have known and unknown factors that add excessive latency in the system. The system logs have documented the ones we know of, which can be seen in

the experiment results. The unknown factor can only be discovered by adding additional logs throughout the system and reducing the known factor to the extent that makes this more transparent.

Not all of these known factors have impacted the end-to-end latency but affected the system's overall latency. Some of these are the time used for allocating buffers for both the recording and receiving applications and the redundant computing time used for sending the formatting object of the audio with each audio sample. These things can be executed before the recording session initiates or only once per recording session by restructuring the source code. In figure 7.4 we see an example of how much time is used for initiating the recording engine.

We know from [17] that the higher resolution of the audio samples is, the processing time increases. To decrease the end-to-end latency of the system, we can downgrade the audio quality. The system uses a bit dept of float 32-bit and a sample rate of 44.1 kHz, which gives 176,4 kbps (kilobytes per second). The music streaming service Spotify has options for using between low (23 kbps), normal (96 kbps), high (160 kbps), and very high (320 kbps)[1]. Since their normal audio quality is 96 kbps, it is worth testing this out in the future and seeing its effect on the end-to-end latency.

9.2.2 Alternatives for accessing microphone buffer

From Chapter 8.2.1 we discover that the installTap function has a minimum buffer size of 19200 bytes. We still believe that reducing the microphone buffer size and having shorter audio segments will significantly affect the end-to-end latency.

We have not discovered alternatives to installTap when using AvAudioEngine as of now. An alternative to installTap is to create our own module for accessing the microphone bus. Swift can have C functions run in its application if one uses an Objective-C wrapper.

Using C, there will be more ways to utilize the microphone intake buffer and give the developers more control in regulating the audio length in the audio segments. One tradeoff is that this new module must also handle audio formatting. Nevertheless, we believe that having this module in C makes it possible to be used by the system when it runs on Android devices. We also think that we can more easily identify the other unknown latency factors in

1. Spotify audio quality: `https://support.spotify.com/us/article/audio-quality/`

the system by controlling the microphone buffer size.

9.2.3 Alternatives to Multipeer Connectivity

Multipeer Connectivity is only available for iOS devices. The latency of Multipeer Connectivity is not measured in this project. Other latency-inducing processes, such as the microphone buffer, heavily influence the end-to-end latency results from the experimental setup to the extent that the amount of Multipeer Connectivity's influence can not be given an accurate guess. To evaluate this, we need a more detailed and extensive experimental setup where the device's internal processing clocks are synchronized to give a precise reading. Alternating between TCP and UDP standards would also be interesting to see. The system uses TCP now.

From Chapter 3.1 we know that one essential attribute of this system is cross-platform capabilities. Multipeer Connectivity as the communication handler will not meet this criterion when combining Android and iOS device participation in a session.

In [3] Bridgefy[2] is a messaging application that does not use cellular or internet connections to work but instead creates a local mesh network between devices operating their app. Bridgefy is available for both Android and iOS devices. For Android devices, Bridgefy utilized Bluetooth Low Energy (BLE) for communication between devices. BLE is faster and more efficient than other local communication methods and protocols, such as Classic Bluetooth and WiFi Direct.

While WiFi Direct has shown promise, it is not suitable since it is not compatible with iOS devices. BLE has been integrated into Android[3] and iOS[4] devices as the new standard. We believe, like with Bridgefy, a module utilizing BLE can be used as an audio transport handler in this system.

2. Bridgefy: `https://bridgefy.me/`
3. Bluetooth Low Energy Android: `https://developer.android.com/guide/topics/connectivity/bluetooth/ble-overview`
4. Bluetooth Low Energy iOS: `https://developer.apple.com/library/archive/releasenotes/General/WhatsNewIniOS/Articles/iOS5.html#//apple_ref/doc/uid/TP30915195-SW1`

9.3 Delay in general

When this project started, the total amount of tolerable latency was perceived to be between 24-30 ms, based on [3]. We know from [17] that there is a plateau of sorts with a 10 ms delay that the hearing aid engineers try to stay under when designing hearing aids. This 10 ms barrier is used with consideration to the reference audio signal that comes from a hearing aid user's own voice.

We can assume that majority of people with hearing loss have developed a decent ability to read lips, and based on [17] we can assume that the tolerable delay for audio-video dyssynchrony of 40 ms can be applied. With this knowledge, the latency criterion we first perceived as the tolerable latency can most likely be adjusted to be higher than 24-30 ms. The system still needs to decrease its latency, but the threshold to reach can be somewhat adjusted for future versions.

9.3.1 Handling audio delay in the system from a user perspective

We have two options for how the hearing aids interact with the system. The first is if the hearing aids only audio input is from the system and not from their own microphones. The second is a combination, where the hearing aids get input from the system and their internal microphones.

For the first option, we need to consider these things. For the system to be experienced as good, everyone in the group must participate in the session. The hearing aids user's smartphone must transfer the user's voice to the hearing aids. This would demand that the latency in the system between the hearing aids user's smartphone and the hearing aids is equal to that of the hearing aids as a standalone assistive technology. Other audio latency from the connected devices can be somewhat less.

If the second option is used, there is a much higher demand for the system latency to be as low as possible. There will be a delay reference signal from the speakers and the input from the system. Furthermore, if there are to be applied noise cancellation features on the speech received by the system, this can be negated by the sounds the hearing aids will pick up from their microphones. This solution sounds the most taxing on the hearing aids user.

In the first solution, we need to consider the latency of the hearing aid user's voice and how it affects their comfort when using the system. The delay from the other users in the system has more room to be tolerable but still will need to be adjusted to be minimized.

9.4 Human-centered design

Deciding and validating the system is best done when real users evaluate it. One can follow different methodologies to ensure the best-developing practices and give the best system and application results. One highly favored in software development is the Agile software development method [19]. The Agile methods have shown to be very effective with small to medium-sized products and with other software development where the customer is committed to being a regular part of the development chain.

Another popular software development methodology connects the end-user directly with the developers, following the human-centered design (HCD) principles. As stated by ISO 9241-210 [20], HCD aims to enhance effectiveness and efficiency while improving human well-being by introducing human factors throughout the developing process. Throughout the entire developing phase, the human factors will contribute by enforcing the focus on the users, their needs, and the system's requirements in the making. Continuous evaluation of the system given by the users will also ensure cost-effectiveness to the developers. HCD is essential when creating systems for minorities, where the developers might not have the needed insight.

10

Future Work

To realize this system as a usable application much remains to be done. We want to include in the system all wishes presented in Chapter 3.1 by the representatives from HLF. This chapter will cover some of the future remarks that need to be handled, but not all.

10.1 Reducing microphone buffering

When this system is developed further, the audio segments recorded must be reduced for the system to be feasible. For using Swift and AVAudioEngine on iOS devices, the installTap function must be replaced since Apple has a minimum enforced limit on the buffer size.

Another framework or library must be used, or more realistically, this needs to be written on the author's own accord in Swift, C or Objective-C for best performance. Writing this functionality in a more general-purpose language will make room for the convertibility of this code to Android devices later. However, doing this will be time-consuming, but we believe that the knowledge gathered throughout this project will make it attainable. It should be mentioned that there has been no effort done to research this system's applicability to Android devices, except for the network communication between devices.

10.2 User Interface - Human-centered design

Creating a universal design is an art form, and it must be prioritized to the same extent as that of the rest of the system. It will be beneficial for creating the backend system and the user interface to follow the HCD principles by regularly getting input from the end-users. However, to use HCD as a practice, we need to locate devoted end-users that will participate in the developing process.

The majority of hearing-impaired in Norway are of the older generation, which will impact how the user interface looks and is operated. The user interface in the created system is basic and only made to access the specific functionalities that allow for the conducted experiments.

The user interface diverges today from what the future application should have. The receiving device should decide when the recording device starts and stops a recording session. Today, the recording device activates this, where the receiver device receives the audio samples without choice. It is not enough to only get the hearing aids users' perspective on the user interface. To ensure a good user experience for everyone using the application, we need input from the other people who will operate the recording devices. Therefore what all participants think of the user interface and the features utilizing their devices is equally important.

10.3 Security

As with all systems, many risk factors must be considered. Maintaining the users' privacy policies is one of the most important things. There are some facets of the system that must gain a throughout risk analysis to accomplish. We need to set questions on who can join the session and how that procedure is secured. The system was initially designed to be used in close-quarter groups. A physical nearness, such as scanning a code from one device, can be used to access the session. This, alongside a handshake protocol, will enforce that only trusted parties are accepted to join.

Audio segments should not be stored longer than necessary on either of the devices, except during processing and transporting. Even though we hope that the local network is secure when it is initiated, added security can be applied by encrypting the audio segment during transportation. There is a tradeoff between increasing security by encrypting the segments and the end-to-end latency, so this process must be researched and evaluated for its benefits.

As with most applications today, some user information must be stored to create user accounts. This information will be stored in a central database, where the user's preferences and other information can be accessed. These practices are widely researched, and there is a community standard to follow, so it is not believed to cause problems.

10.4 Hearing aids compatibility

This thesis assumes that every hearing aid with Bluetooth compatibility can be connected to any smartphone. We know from [3] that this is not always the case. Time to ensure that, or discover which, if not, must be invested to provide the requested cross-platform solution.

There can be a range of hearing aids models and vendors with hardware limitations in their products, enabling a smartphone application to connect to the hearing aids. Such limitations would halter the purpose of the system, and other solutions for the original problem must be discussed.

11

Conclusion

We conclude this thesis by comparing the summarized outcome of the project with the original thesis statement. When we do this, the contributions from the thesis will also become apparent and reinforced.

How can we measure the latency from the smartphone microphone to the hearing aids (HA) activation?
We did this by creating an experimental setup. We did not simulate the transference time from a smartphone to hearing aids and instead recorded the sound output from the receiving smartphone. The experiment did not cover the transportation time of sound through Bluetooth the hearing aids.

How much latency can we expect from a naive implementation of such an application?
Through the experimental setup, we pushed the end-to-end latency from **361.8 ms** to **246.5 ms**. This decrease was apparent when reducing the microphone buffer. The reduction could not be pushed down even more with the implementation of specific choices made. We believe that the microphone buffer can be decreased by circumventing the functions as a minimum microphone buffer and creating this functionality ourselves.

Which part of the architecture introduces latency into the system?
The microphone buffer size is the most significant factor regarding the end-to-end latency in the system. This factor has dominated the measurements and results to such an extent that it is uncertain what other undiscovered factors

have played into the end-to-end latency.

We know that the latency decline between experiments 1 and 2's end-to-end latency and size reduction of the microphone buffer is not linear. This enforced our belief that there are unknown latency-inducing factors in the system.

We have, through this thesis, answered the research questions we stated in Chapter 1: Introduction. The end-to-end latency in the system will need to be reduced to be desirable to users. We have identified where latency is introduced in the system and have determined how to identify other latency-introducing facets. We will encompass HCD practices in developing the system in the future, as we believe this will help establish a feasible system and solve *the café problem*.

References

[1] Worlds Health Organization, *Deafness and hearing loss*, 2022. [Online]. Available: https://www.who.int/health-topics/hearing-loss#tab=tab_1 (visited on 05/04/2022).

[2] Oslo Economics, *Nedsatt hørsel i arbeidsfør alder*, 2020. [Online]. Available: https://www.hlf.no/globalassets/dokumenter/dette-jobber-vi-med/nedsatt-horsel-i-arbeidsfor-alder.pdf (visited on 05/04/2022).

[3] M. E. M. Ellingsen, *Investigating the café problem for hearing impairde*, Unpublished report, Dec. 2021.

[4] L. Carneiro, T. Oliveira, P. Noriega, and F. Rebelo, "Can the context stigmatize the assistive technology? a preliminary study using virtual environments," in *Advances in Ergonomics in Design*, F. Rebelo and M. Soares, Eds., Cham: Springer International Publishing, 2016, pp. 289–297, ISBN: 978-3-319-41983-1.

[5] T. Gjestland, *Lyd i store norske leksikon*. [Online]. Available: https://snl.no/lyd (visited on 05/13/2022).

[6] A. Fox, *What is the difference between sound and audio?* [Online]. Available: https://mynewmicrophone.com/what-is-the-difference-between-sound-and-audio/ (visited on 05/13/2022).

[7] P. Mantione, *Digital audio 101: The basics*. [Online]. Available: https://theproaudiofiles.com/digital-audio-101-the-basics/ (visited on 05/13/2022).

[8] S. Birkeland, *Forekomst på hørselsområdet*, 2021. [Online]. Available: https://www.hlf.no/globalassets/prosjekter/prosjektdokumenter/forekomst-pa-horselsomradet-hlf-18.-mars-2021.pdf (visited on 11/25/2021).

[9] F. Ø. Winther, *Høreapparat i store medisinske leksikon*. [Online]. Available: https://sml.snl.no/h%C3%B8reapparat (visited on 12/16/2021).

[10] U.S. Food and Drugs - Administration, *Types of hearing aids*. [Online]. Available: https://www.fda.gov/medical-devices/hearing-aids/types-hearing-aids (visited on 06/04/2022).

[11] G. J. Frye, "Testing digital and analog hearing instruments: Processing time delays and phase measurements," *The Hearing Review*, pp. 34–40, Oct. 2001.

[12] Apple Inc., *Swift*. [Online]. Available: https://www.swift.org/ (visited on 05/15/2022).

[13] ——, *Swift*. [Online]. Available: https://developer.apple.com/swift/ (visited on 05/15/2022).

[14] ——, *Multipeer connectivity*. [Online]. Available: https://developer.apple.com/documentation/multipeerconnectivity (visited on 05/16/2022).

[15] ——, *Avaudioengine*. [Online]. Available: https://developer.apple.com/documentation/avfaudio/avaudioengine (visited on 05/16/2022).

[16] ——, *Avaudionode*. [Online]. Available: https://developer.apple.com/documentation/avfaudio/avaudionode (visited on 05/16/2022).

[17] J. Alexander, "Hearing aid delay and current drain in modern digital devices," *Canadian Audiologist*, vol. 3, Jul. 2016.

[18] K. Raaen and I. Kjellmo, "Measuring latency in virtual reality systems," in *Entertainment Computing - ICEC 2015*, K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, and R. Malaka, Eds., Cham: Springer International Publishing, 2015, pp. 457–462, ISBN: 978-3-319-24589-8.

[19] I. Sommerville, *Engineering Software Products: An Introduction to Modern Software Engineering, eBook, Global Edition*, 1st ed. Pearson (Intl), 2020, ISBN: 9781292376356.

[20] *Ergonomics of human-system interaction – part 210: Human-centred design for interactive systems*, ISO 9241-210:2019(E), International Organization for Standardization, Geneva, CH, Jun. 2019.