Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Speaker Synchronization

Semester Thesis

Marc Urech

maurech@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Pascal Bissig, Laura Peer
Prof. Dr. Roger Wattenhofer

June 13, 2016

Acknowledgements

I would like to express my gratitude especially to my supervisors Pascal Bissig and Laura Peer for supporting me during the whole project with their constant input of new ideas and improvement proposals and also for providing me their assistance whenever I was stuck and needed a helping hand. I would also like to thank Prof. Dr. Roger Wattenhofer who gave me the opportunity to write this thesis.

Abstract

In this thesis, an Android application was developed which is able to synchronize the audio playback of multiple smart phones to increase the volume of the integrated speakers. To synchronize, the phones use a random sequence which contain 2400 bits, modulated with binary phase-shift keying with a carrier frequency of 480 Hz. The phones are connected with a web server which distributes the songs and synchronization sequences among the smart phones. When all devices are ready to play, the initial synchronization is performed. All devices synchronize to a master phone with the help of the synchronization sequence. During the playback of the song, the phones periodically mute and detect errors in the synchronization and correct them if possible.

The measurements show that the synchronization error is smaller than 14.40 ms. It was discovered that the slave phones synchronized sometimes to sound waves reflected from nearby surfaces and not to the sound waves arriving on the direct path. Most of the errors seem to originate from reflections. The number of supported devices is theoretically not limited, but the more devices try to synchronize, the more data has to be distributed by the server which leads to an increased network usage.

Contents

Chapter 1

Introduction

1.1 Motivation

A recently published survey from *Comparis.ch*[1] reveals that around 78% of Swiss people[1] own a smart phone. These devices aggregate a wide range of functionalities. Among others, these devices serve as mobile music players for many people. However, if one wants to listen to music without headphones, these devices are limited. Although there is substantial progress in the quality of the integrated speakers, there are physical limitations because of the compact size of smart phones. If one wants to reach a reasonably loud volume and clear sound, external hardware like for example Bluetooth speakers are necessary. However, these speakers are not as compact and light as a smart phone.

In this thesis, an approach is examined which does not require additional hardware: If a bunch of people meet, there are usually several smart phones available for playing music. A promising approach would be to unite these phones into one speaker group able to play music synchronously. The phones could be spread across a room to boost the volume.

1.2 Related Work

There are several apps available in the *Google Play Store*. One of them is *SoundSeeder Music Player*[2]. This application only broadcasts the played music of a master phone over WiFi, it is not guaranteed that the different speakers are actually synchronized. The fine tuning has to be made by hand, which is quite inconvenient. There are several reviews which complain about this point. In personal tests, the broadcasting failed several times and was not reliable.

Another app is *AmpMe*[3]. This application is capable of connecting different phones within groups and play music synchronously. However, the mechanism seems not to be fully matured. Around 11.4% of the reviews have 2 or less

[1] Age of asked people between 15 and 74, 1202 participants.

stars[2][3]. Many of these users are complaining that they could not connect to a group or that the application does not work reliable.

There were already two Android applications developed as part of two semester theses in the Distributed Computing Group at ETH Zurich, namely *SynBa*[4] from Kevin Luchsinger and *SyncedPlayer*[5] from Samuel Willi.

SynBa connects several devices over Wi-Fi Direct and is independent from an Internet connection. The devices try to get a common time by exchanging time stamps and measuring the round trip times within the network. In a next step, the audio latency of the different devices is examined with audio measurements. Finally, the master phone distributes an audio file which is played by all devices simultaneously. The drawbacks of this application are that Wi-Fi Direct is quite undependable and it was discovered that the sound was not always sufficiently synchronized[5, p. 2].

SyncedPlayer which was developed after *SynBa* pursued a totally different strategy. The application is able to synchronize to an arbitrary source, for example a radio. To achieve this, the application first records a sample and tries to find the played title with the help of an online music recognition service. After that, it downloads the needed music file and synchronizes to the original source. This approach is very inventive but has serious drawbacks. From 50 tested songs, the application could find the correct music file for only 28 of them[5, p. 18]. Another major drawback was the time from starting the application until it was playing synchronously to the original source. This took an average time of around 23 s[5, p. 17] which is quite long compared to the usual duration of a song.

The application developed in this thesis is based on the source code of *SyncedPlayer*.

[2] 911 reviews with 1 star, 357 with 2 stars, in total 11133 reviews. Page visited 06.06.2016, 11:45.

Chapter 2

Background

2.1 Android Music Playback

Android is an operating system used mostly on mobile devices like smart phones. It is developed by Google and is based on the Linux kernel[6].

Applications for Android are typically written in Java. The source code is subsequently translated to byte-code for the *Java Virtual Machine*, this code is then further converted to *Dalvik byte-code*. This code is then interpreted by a *process virtual machine* (*Dalvik Virtual Machine* until Android 4.4; *Android Runtime* since Android 5.0). The virtual machine uses the *Linux Kernel* to access the underlying hardware like for example the speakers and the microphone[7, 8, 9].

Android faces the problem, that especially older versions have a big round trip audio latency, which is defined as "*the time it takes for an audio signal to enter the input of a mobile device, be processed by an app running on the application processor, and exit the output*"[10]. Google has measured these times for several of their devices and provides a list which can be seen in Figure 2.1. It is important to notice, that these values are averaged and meant to give an overview about the expected magnitude of the latency for a certain device. However, the audio round trip latency can be different for each application execution.

These delays are not important, as long as no real time operations have to be performed. But for a time sensitive task like synchronizing two phones such that they play with an offset which is not hearable, dozens or even hundreds of milliseconds can not be neglected.

On Android, sound is played by writing audio samples to the audio sink for playback (see e.g. API *AudioTrack*). The time between writing the first sample to the buffer of this audio sink until the phone speaker starts to emit sound is depending on the smart phone model, but varies also for the same phone, for example if the initialization was delayed due to a higher priority thread. But as long as the buffer is never completely emptied and a buffer underrun occurs, this delay stays constant. The same property holds for the recording of audio signals:

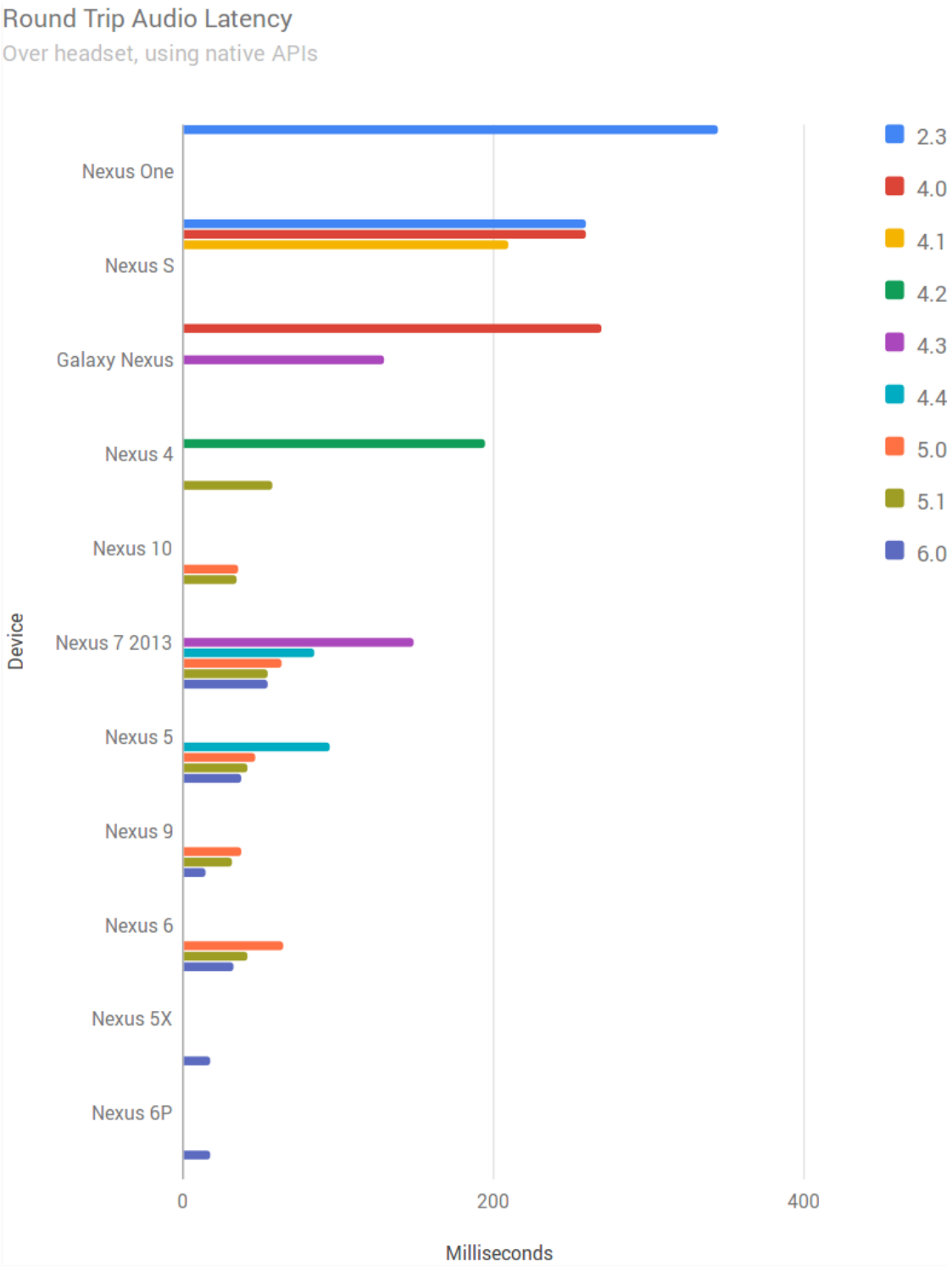

Figure 2.1: Audio round trip latencies for different Google devices under various versions of Android (taken from [10]).

The timespan between giving the phone the command to record its environment until the first audio sample can be taken from the output buffer is unknown and may vary for different executions. But if the output buffer is constantly emptied and no buffer overrun occurs, the delay also stays constant. These properties can be used to calculate the audio round trip latency for one specific application execution.

2.2 Graph Terminology

For defining the goals and explaining the model, it is useful to have a graph terminology to describe the relationships between the single phones.

All phones to synchronize are modeled as vertices. If phone A can clearly record the sound of phone B, a directed edge is drawn from vertex A to vertex B. If both phones can hear each other clearly, two edges with opposite directions are drawn between the vertices. An example of such a graph can be seen in Figure 2.2.

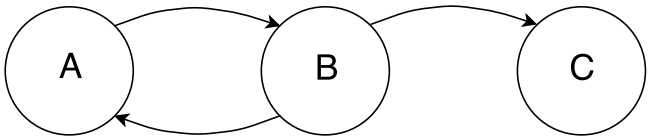

Figure 2.2: Example of a hearing graph: Phones A and B hear each other mutually, C hears no other phone but is heard by B.

Chapter 3

Problem Description

In this chapter, the goals of this thesis are presented. Additionally the model is introduced, under which circumstances the intended solution should work.

In this thesis, an Application for Android smart phones is developed which is able to let a cluster of phones play music synchronously without an audible playing offset. The goal was to extend, adapt and improve an application developed in a previous semester thesis by *Samuel Willi*[5] such that the following requirements are accomplished:

- The application should be easy to handle for the user. This requires that the user interface is simple and no difficult manual configurations are needed. The number of required interactions with the phone should be as small as possible.
- The single phones should converge fast such that a song is played synchronously without an audible playing offset. The phase when only a small part of the group is playing a song should be as short as possible.
- The software should be able to work with a large number of devices. This includes taking into account that the phones are distributed over a big area and not all phones are able to record each others played back signals.
- The software should be capable of correcting errors, for example if the output of one single phone got delayed because a higher priority thread in the background has interrupted the application.

The software should work under the following circumstances:

- The participating phones are separated in two groups, one master phone and several slave phones. Each slave phone has to be able to record the master directly or indirectly over other slaves. Speaking in terms of the definition made in Section 2.2:
  If the master phone is modeled as vertex M and the n slave phones are

modeled as vertices S_1 to S_n, for each slave phone S_m ($m \in \{1, 2, \ldots, n\}$) there has to exist a directed path to the vertex M.

- The background noise should be moderate such that all phones can hear what their neighbors are playing. Temporary isolation where a phone cannot hear other phones (e.g. because of temporarily and locally increased background noise from speaking people etc.) may occur.

Chapter 4

Methods

This chapter explains the techniques used to synchronize multiple phones, such that they are able to play music without a hearable offset between the different devices.

4.1 Sample Accurate Player

The audio output from the speakers of two phones must be synchronized in the order of milliseconds if the human ear should not notice a playing offset between the speakers. A simple way to achieve this would be to pause the playback for a certain time and resume it afterwards to achieve a delay, or use a fast forwarding function to speed up a delayed device. However, in the previous thesis it was examined that this approach was not reliable, standard audio players like *AudioPlayer* are not accurate enough for this purpose, because the content of the audio buffer is cleared and filled again when performing these operations, which leads to unpredictable timing behavior[5, p. 13].

To reach the goal of accurate synchronization, an audio player was needed which is able to shift the samples forwards and backwards by an arbitrary amount without emptying the playing buffer, because the playback start is randomly delayed as already mentioned in section 2.1. This requirement is quite unusual for standard audio players and since no standard library was found which supports this feature, a custom solution had to be implemented. For this solution, the APIs *MediaExtractor* and *MediaCodec* were used for decoding audio samples; *AudioTrack* was used to implement a player which is able to write the decoded samples to the audio sink for playback. When the output of the speakers needs to be delayed, a certain amount of zeros can be inserted between the decoded audio samples; to advance the output, a certain amount of decoded samples are skipped.

The audio player described in this section was already implemented and described in the previous thesis[5, p. 13]. During this project, it was improved and adapted to new requirements.

4.2 Determining Offset between Devices

In the previous section it was explained, how two phones can be synchronized when their playing offset in samples is known. This section explains, how this offset is determined.

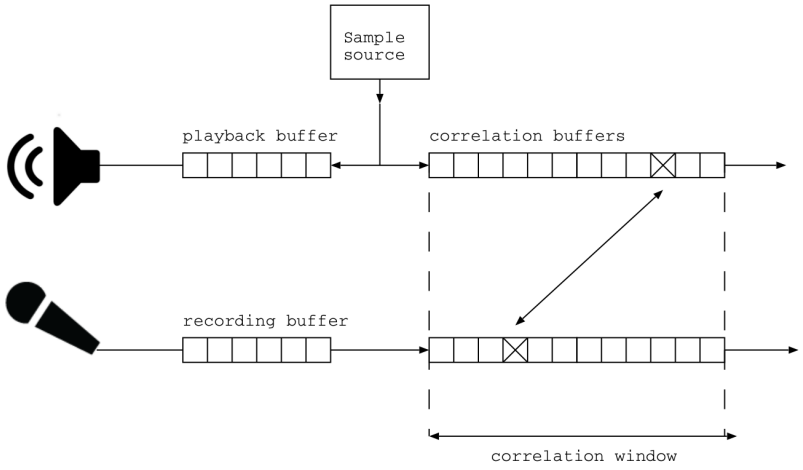

Figure 4.1: Mechanism to determine playing offset between recorded samples of other phone and decoded samples of own phone, taken from [5, p. 11].

When a phone requires to know how its own speaker output is shifted to the one of another phone, it requires information about what the other device is currently playing. To achieve this, a simple approach was to record the environment of the phone and make a comparison between the recorded samples and the decoded samples for the audio sink for playback and evaluate, at which shift-index the signals match the best. This approach is visualized in Figure 4.1.

This comparison was made by performing a cross-correlation between recorded samples of the other phone and decoded audio samples for the own speakers. A time discrete cross-correlation is defined as in Equation 4.1.

$$(f \star g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m] \cdot g[m+n] \qquad (n, f[\cdot], g[\cdot] \in \mathbb{R}) \tag{4.1}$$

For finding the playing offset, the maximal value and its corresponding index was searched in the result of the cross-correlation as described in Equation 4.2.

$$offset = \arg\max_{n}((f \star g)[n]) \tag{4.2}$$

The synchronizing phone is temporarily muted during the recording of the samples needed for the cross-correlation in order that the microphone can clearly

record its environment. This is necessary, because the intensity of sound waves decreases quadratically in distance[11, p. 597]; if the recording phone was not muted, it would only record its own speakers which are very close to the microphone; the sound waves of the other phone would be drowned.

Cross-correlating with the samples from the decoder leads to a problem mentioned in Chapter 2: The round-trip audio latency of a device depends on the phone model and is even varying for the same device when measured multiple times. Therefore one cannot know exactly how long it will take from writing a decoded sample to the audio sink for playback until it is actually audible from the speaker. Because of this, each phone determines its round-trip time at the very beginning by playing a certain sound pattern, recording it and finding the own round-trip time with a cross-correlation. More information about this step will be provided in Section 4.3.

If the recording- and playing-buffer are never flushed, the round-trip time will stay the same during one application execution. This property can be used in the synchronization process: Performing the cross-correlation with the recorded samples of the other phone and the samples currently written to the output buffer, one can determine the sample shift if the samples in the output buffer were played immediately. By taking into account the previously calculated round-trip time, the actual delay can be found. When the needed shift was computed and performed by the player mentioned in Section 4.1, the volume is restored to the original intensity.

4.3 Synchronization Algorithm

In the previous sections it was described, how a playing offset between two phones can be determined with the help of a cross-correlation and how it can be corrected with a sample accurate player. This section will explain which algorithm was chosen to actually synchronize the phones.

In the previous thesis[5] about speaker synchronization, a fully distributed algorithm was chosen. The phones which had to synchronize recorded their environment, determined the song currently played by an arbitrary source with the help of a recognition service (ACRCloud[5, p. 7]), searched and downloaded the song from an audio-file database (mp3Skull[5, p. 9]); finally, they joined the original source and synchronized such that there was no audible offset between the original source and the joined device anymore.

This algorithm worked well, but it had major drawbacks. From the time the initial source started playing until all joining devices were playing synchronously, an average time of 23.0 seconds[5, p. 17] passed, provided that the phones found the correct sound-file in the online collection. If the joining phones did not find the correct song file, they stayed muted or played with a wrong offset, which

happened quite often: 22 out of 50 downloaded songs were faulty, which is a failure rate of 44%[5, p. 18].

In the current thesis, the goal was to speed up the synchronization phase between the different phones and to decrease the failure rate, where joining phones are not able to synchronize to the original source. To achieve this, a combination of centralized and distributed mechanisms was applied:

Centralized hierarchical algorithm: The group of smart phones are separated into one master and several slaves. The master phone determines the songs to play and distributes the song files together with individual synchronization sequences among the slaves with the help of a web-server. When all devices have downloaded the required files, the master arranges that all devices start playing approximately at the same time. When the phones start playing, they first determine their own audio round-trip latency with an individual synchronization sequence. After that, all phones mute except the master which will play another synchronization sequence. With the help of this sequence, the slaves can synchronize to the master phone. More information about this first phase will be provided in the following sections. The file distribution process will be explained in Section 4.3.1, the initial synchronization algorithm in Section 4.3.2.

Distributed algorithm: For correcting errors which may occur during playing of the song, the slaves mute in a circular way to check if they are still synchronized to the rest of the group. More details to this algorithm will be provided in Section 4.3.3.

This revised algorithm has big advantages compared to the algorithm of the previous thesis. Since the smart phones are now connected with a web-server, they can exchange data directly and the problem of acquiring incorrect song files disappears. However, due to the distributed part of the algorithm, the phones can stay synchronized even tough single phones may make errors or leave the group, e.g. when running out of battery.

4.3.1 File Distribution

This section explains, how the song files are distributed and how an approximate synchronous start time for all devices is found.

The smart phone group is separated into one master and several slaves. The files and the start time are exchanged with the help of a web server. The whole procedure can be seen in Figure 4.2 as a timeline.

- In a first step, all slaves register themselves on the server and get assigned a unique number. With the help of this ID, the server can later distinguish

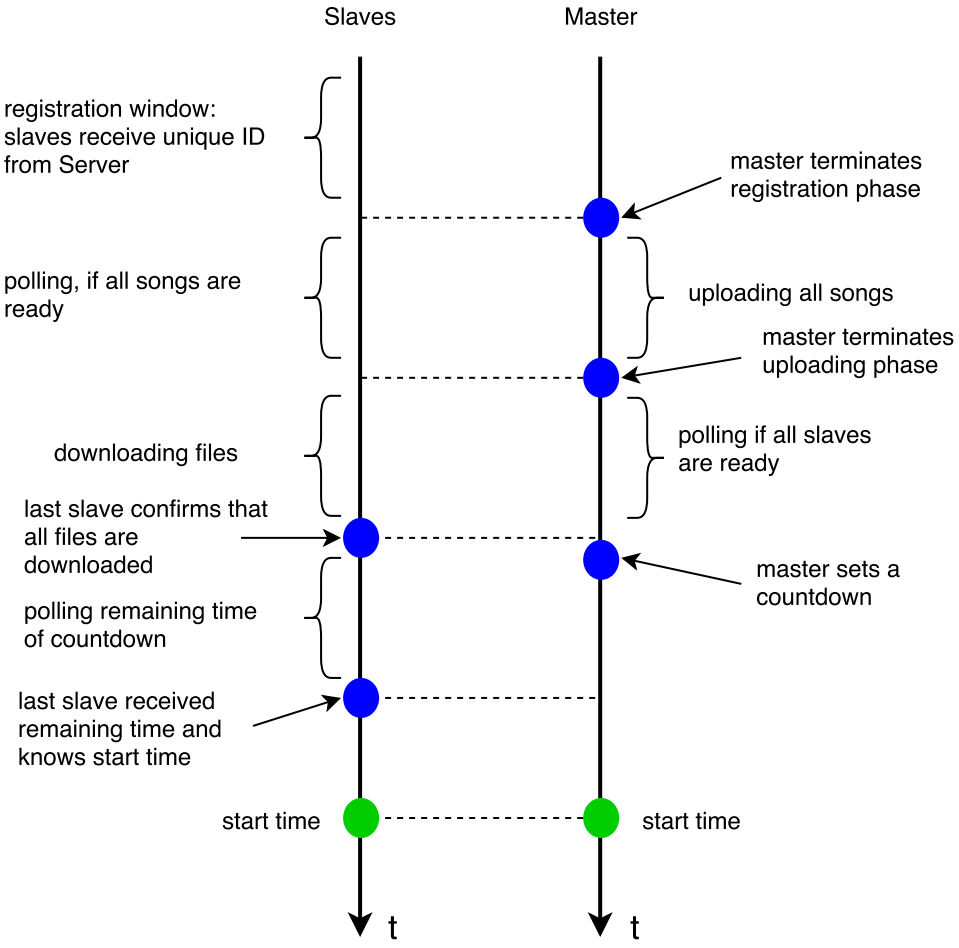

Figure 4.2: Timeline of file exchange process on server.

which slave is currently asking for files or updating its state. When all slaves are registered and received their ID, the master phone terminates the registration phase. After this step, no additional slaves can be registered.

- The master now uploads all song files which the user has previously copied to an application specific folder. When all available songs were uploaded, the master terminates the uploading phase. In the meantime, the slaves poll regularly to see if all files are ready. The server will give them a negative answer, as long as the master has not yet terminated the uploading phase.
- In the downloading phase, each device downloads a personal synchronization sequence, a public synchronization sequence used by the master and finally all song files. When a slave has downloaded all necessary files, it informs the server. Based on the ID of the slave, the server can detect which slaves are ready to start. During this time, the master regularly polls the server if all slaves are ready.
- When all devices are ready, the master initiates a countdown of 3 seconds on the server. The devices which are now again in a polling phase, receive the remaining time of the countdown, calculate a local start time (the clocks of the server and the individual phones are not synchronized) and wait until this time stamp is reached. After that, they start playing approximately simultaneously.

The slave phones are able to synchronize to the master, if they start playing at most one second earlier or later than the master phone. For this requirement to be met, the round-trip time (RTT) must at most be 2 seconds, such that $RTT/2$ is at most 1 second.

4.3.2 Initial Synchronization

In Section 4.3.1 it was explained, how the song files are distributed and an approximately synchronous start time can be found. In this section, information about the synchronization algorithm will be provided.

For the synchronization, random noise sequences were used. These sequences are distributed together with the song files described in Section 4.3.1. A synthetic random sequence has two main advantages against a short snippet of an actual song. Firstly, different random sequences have a small cross-correlation if they are long enough. This is an asset, if multiple phones are playing such a synchronization sequence simultaneously. Secondly, most songs contain repetitive patterns (e.g. sampled beats, where a short sound sequence is repeated periodically over the whole song), this would be a disadvantage when computing

the cross-correlation of a recorded version with the decoded samples, multiple maximas could occur.

The random sequences are a series of *1* and *-1* modulated with binary phase-shift keying. As carrier signal, a sinus wave with a frequency of $480Hz$ was chosen. The symbols were encoded as shown in Equation 4.3 and 4.4, where T_b is the duration of one bit ($1/(480Hz) \approx 2.08ms$) and f_c is the carrier frequency ($480Hz$). The symbol *1* is mapped to the signal $s_0(t)$ and the symbol *-1* is mapped to the signal $s_1(t)$. One synchronization sequence has a length of 2400 symbols, which leads to a duration of 5 seconds.

$$s_0(t) = \begin{cases} sin(2\pi \cdot f_c \cdot t), & \text{if } 0 < t \leq T_b \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

$$s_1(t) = \begin{cases} -sin(2\pi \cdot f_c \cdot t), & \text{if } 0 < t \leq T_b \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

The whole initial synchronization algorithm can be seen in Figure 4.3 as a timeline.

For determining the own audio round trip latency, each phone in a first step plays a random noise sequence which is different for every phone. Like this, all phones can simultaneously play with only small interferences since two different random sequences have a small correlation.

In a second step, all devices play the same master random sequence but only the master phone plays it loudly, the slave devices play it muted and try to find the cross-correlation offset as described in Section 4.2.

The correlation window of decoded samples has a size of 179072 samples, which corresponds to 3.73 s with a sampling rate of 48 000 Hz. These samples are correlated with 83072 recorded samples which corresponds to 1.73 s. This means that the recorded and decoded samples can be shifted 1 second at maximum without that the recorded samples leave the correlation window. The number of samples is chosen such that the sum of recorded and decoded samples corresponds to a power of 2, ($2^{18} = 262144$). The sum of both lengths was taken to be able to apply zero padding to avoid a cyclic convolution. The sum has to be a power of 2, because the chosen implementation of the cross-correlation works the most efficient when the arrays have a length which is a power of 2.

If a slave phone can record the sequence of the master clearly and therefore compute the offset unequivocally, it shifts the needed amount of samples and can start playing the song synchronous with the master. If the slave was not able to hear the master clearly and could therefore not calculate a shifting offset, it starts to play the song muted and tries to synchronize later with a correction algorithm described in Section 4.3.3.

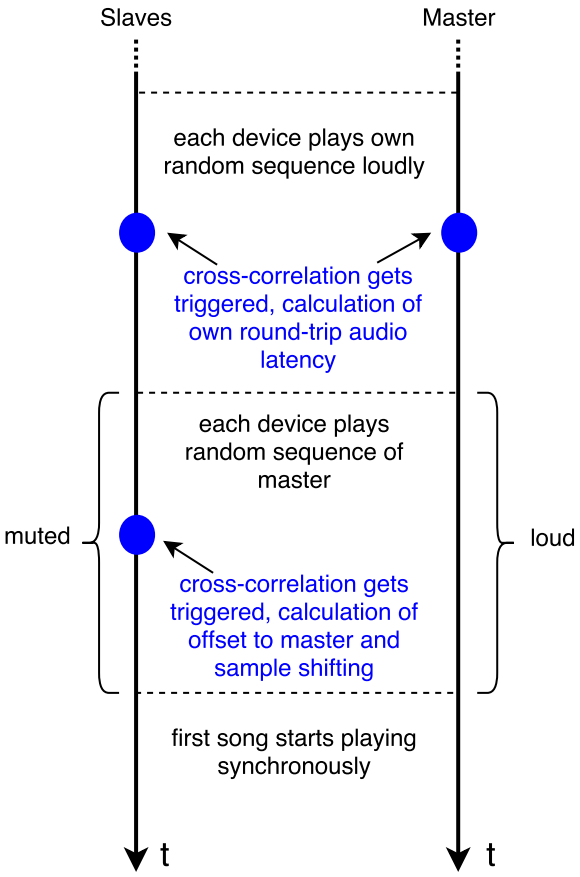

Figure 4.3: Timeline of initial synchronization algorithm. This timeline is the continuation of the timeline in Figure 4.2.

4.3.3 Error Correction during Music Playback

Although the initial synchronization algorithm described in Section 4.3.2 works well, there could be need for further corrections. Here, two main scenarios could lead to the requirement for further synchronizations:

- A phone was not able to hear the synchronization sequence of the master: In this case, the phone plays the song silently and tries to find its offset to the other phones. When it could clearly identify the number of needed samples to shift, it makes the needed correction and sets the volume to high.
- A phone had a short interruption in its playing continuity, caused for example due to a higher priority thread of the OS in the background. This can lead to a delay of a few hundred milliseconds which would be clearly hearable. To correct such mistakes, the slave phones mute cyclically to detect, whether they are still synchronized with the rest of the group. These synchronizing windows last 5 seconds for each phone. Only one phone is allowed to mute at a time, this means that the more phones are present in a group, the longer the individual phone has to wait. If a slave phone cannot confirm if it is still synchronized, it stays muted until it knows for sure if it is still synchronous or has computed a clear shift such that it can correct its error. The master phone is always playing loud such that for few phones always one is playing loud. However, if the master phone would make an error due to a short pause (caused e.g. from higher priority thread), all slave phones would have to correct this error.

This part of the algorithm works completely distributed, each phone choses their action whether to mute or not independently from the other phones. In this phase the devices no longer communicate with the server; if they lose the connection to the network there are no influences to the precision of the synchronicity.

Chapter 5

Experiments

In order to measure how accurate the synchronization between different devices works, two different experiments were performed. In a first experiment it was examined, how well the initial synchronization with the random sequence works. In a second experiment it was tested, how accurate the offset correction during the playback of a song is.

5.1 Measurement Setup

Both experiments were performed under the same conditions in an ordinary closed room. The room was not particularly prepared for measurements to have a location where the application also would be executed in a real life usage.

A slave phone was placed 1.20 m away from the master phone. Near each device an external microphone was placed, its distance to the phone speaker was around 10 cm. Both microphones were simultaneously connected to a computer with an adapter, which allowed to record both microphones at the same time on different channels. To measure the synchronization accuracy, the application was started on both phones approximately synchronously. When the synchronization was finished, a sequence of a song of around 10 s was recorded. This sequence was then analyzed with a MATLAB script, which calculated the offset between the audio-channels using a cross-correlation. Due to the distance of 1.20 m, the slave phone records the master phone only after the time of flight[1] of the sound waves. So even if the synchronization would work perfectly, there would still be an offset. This offset was subtracted from the result in the MATLAB script.

The external microphones were theoretically recording both phones which would influence the result of the cross-correlation. But because the distant phone is more than 10 times further afar than the close phone and the intensity of sound waves decreases quadratically with distance[11, p. 597], this influence could be neglected.

[1] time of flight $= 1s/343.2m \cdot 1.20m \approx 3.5 \cdot 10^{-3}s$

For the measurements, three different smart phones were used, an *HTC One M8*, a *Samsung Galaxy S4 mini* and a *Samsung Galaxy S3*. The phones will be labeled hereafter as *HTC*, *S4 mini* and *S3* respectively.

5.2 Results

5.2.1 Initial Synchronization with Random Sequence

For this first experiment, in total 90 measurements were performed in 3 different master-slave relationships which can be seen in Table 5.1. Each configuration was measured 30 times. The chosen song was always *Can't Hold Us* from *Macklemore & Ryan Lewis* and was never changed, because the synchronization was performed over the random sequences and did not depend on the chosen song. The song was only needed to measure the offset afterwards with the MATLAB script.

The measurement results for the synchronization with the random sequence can be seen in Figure 5.1.

Config. Nr.	Master	Slave
1	S4 mini	HTC
2	S3	S4 mini
3	HTC	S3

Table 5.1: Measurement configuration for initial synchronization with a random sequence.

5.2.2 Corrective Synchronization with Audio-Samples

For the second experiment, also 90 measurements were performed, 30 for each of 3 different configurations. The configuration of the single measurements can be seen in Table 5.2. In this experiment, the synchronization depends on the currently played song. To see if this has an influence on the accuracy, the measurements were performed with 3 different songs.

Figure 5.2 shows the measurement results for the synchronization with the help of audio-samples, after the initial synchronization failed (on purpose) by 500 ms.

5.3 Evaluation

For both experiments it is remarkable that the offset is not distributed in a Gaussian way. Small variations in the order of fractions of milliseconds (as seen for

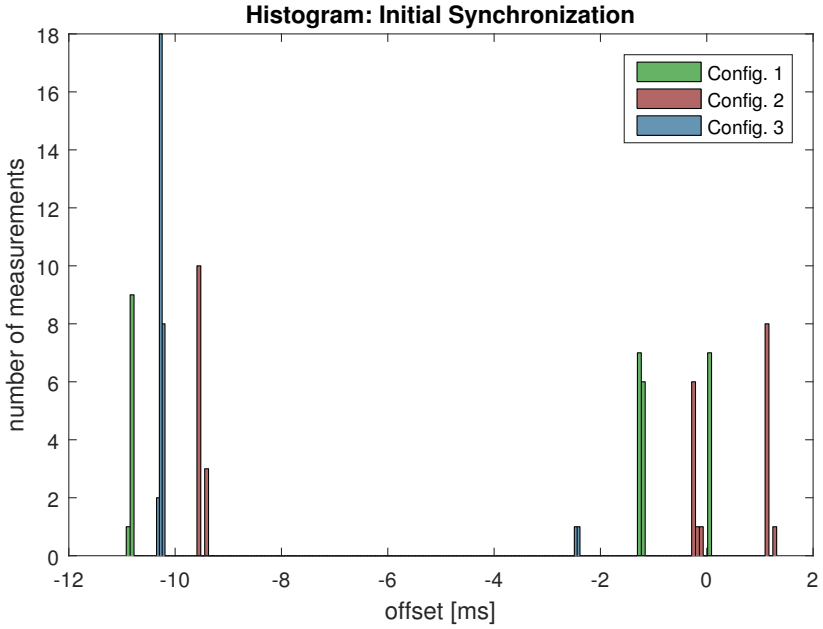

Figure 5.1: Measured offsets between two phones after the initial synchronization with a random sequence. The 3 configurations are specified in Table 5.1.

Config. Nr.	Master	Slave	Song
1	S4 mini	HTC	*Can't Hold Us*
2	S3	S4 mini	*City of Gold*
3	HTC	S3	*No Woman, No Cry*

Table 5.2: Measurement configuration for corrective synchronization with audio samples.

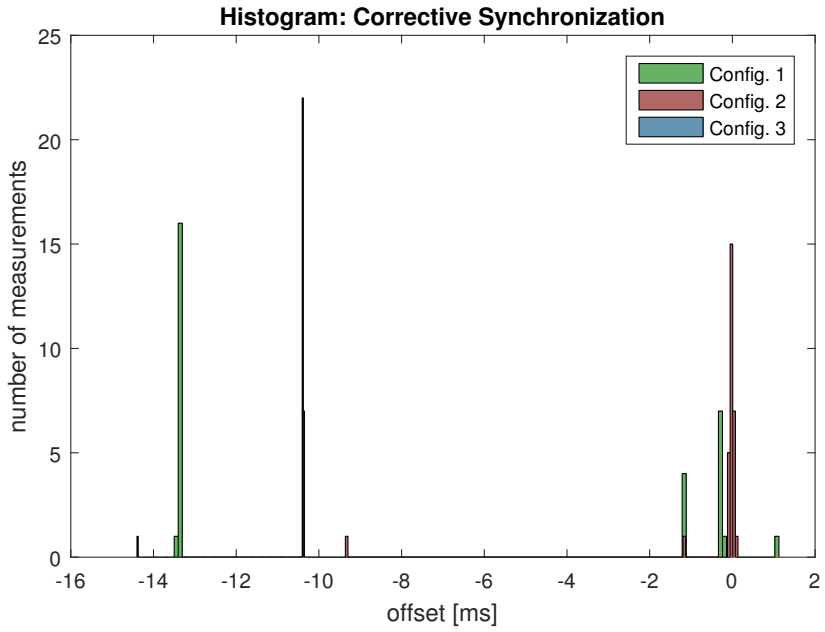

Figure 5.2: Measured offsets between two phones after the initial synchronization with a random sequence failed by 500 ms and has been corrected with a second synchronization with audio-samples. The 3 configurations are specified in Table 5.2.

example around zero in both experiments) between the different configurations are to be expected, because the different devices have their speakers and microphones at different places and therefore the distances the sound waves have to travel are not exactly the same for each configuration. However, this could not be the cause for offsets of 10 and more milliseconds as they can be seen in the measurements.

In these results, two different random processes can be discovered. The first one has a small variance in the order of fractions of milliseconds. The second one has a bigger variance in the order of several milliseconds.

A possible explanation for the first random process with a small variance are small synchronization errors of the phones in the order of a few samples because of microphone and speaker noise. A second possibility are inaccuracies in the measurement setup. To reset the phones between two consecutive measurements, they were touched and in this process slightly moved.

For the second random process, it cannot be ruled out that multipath interference occurred. In the room where the measurements were performed, multiple reflective surfaces, like furniture or the ceiling, were present. If enough reflected waves interfered constructively, it is possible that the phones did not synchronize to the sound waves arriving on the direct path but with a reflection from a nearby object. In Figure 5.1 for example, one can clearly see that for all three configurations the slave phone was often delayed roughly 10 ms. During this time span, sound waves can travel around 3.43 m [2]. This distance corresponds approximately to the distance from the master phone to the slave phone via the ceiling. Another indication for this supposition is the fact that the microphone of the slave phone was not necessarily aligned directly to the speaker of the master phone. The *HTC One M8* for example has its microphone on the front inside the speaker grill. The phones were placed flat on the ground, it could be possible that sound waves entering the microphone vertically from above were recorded better than sound waves arriving from the side which could have been shielded by the phones body.

To examine this supposition for the occurrence of the second random process, an additional measurement was performed outside on a lawn, where no nearby objects were present which could reflect sound waves. The synchronization using the random sequence was performed 30 times. The results can be seen in Figure 5.3. The biggest measured absolute offset is smaller than 0.15 ms. The mean is -0.072 ms and the standard deviation is 0.044 ms. A possible explanation for the negative offset mean is an error in the experiment setup. In 0.072 ms sound waves can travel approximately 2.5 cm [3]. When placing the phones, the distance was measured center-to-center and not speaker-to-microphone. By taking also into account that the external microphones for the measurements could have

[2] $(343.2m/s) \cdot 0.010s \approx 3.43m$
[3] $(343.2m/s) \cdot 72 \cdot 10^{-6}s \approx 24.71 \cdot 10^{-3}m$

been placed slightly incorrect, a distance shift of 2.5 cm is plausible.

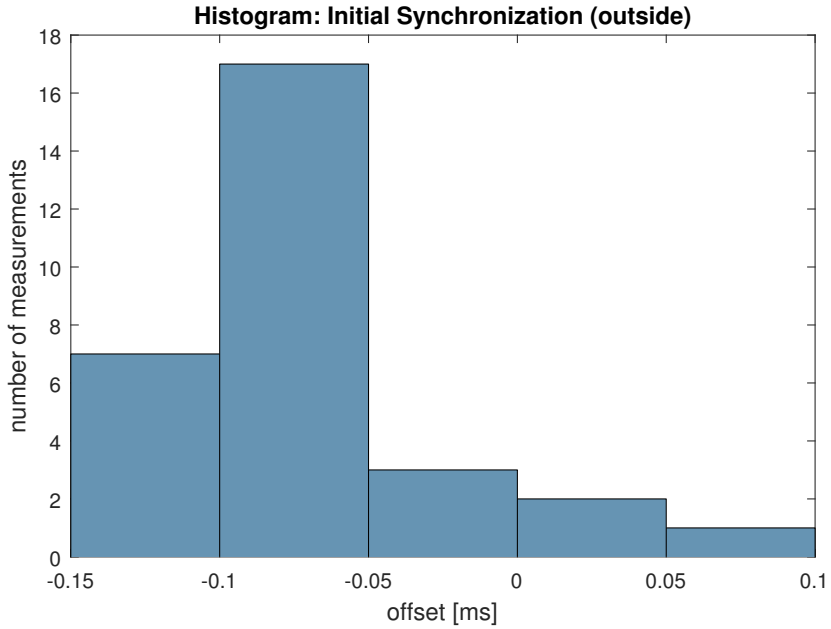

Figure 5.3: Measured offsets between two phones after synchronizing, phones placed on a lawn where no near objects were present which could reflect sound waves.

5.4 Summary

In summary the errors are below 14.40 ms. Such offsets were not noticeable by the tester during the execution of the experiments.

The measured offsets were distributed irregularly. A possible explanation for this phenomenon would be that the slave phones were synchronizing to a sound reflection of the master and not to the sound waves traveling on the direct path. This assumption is substantiated by an additional measurement in an environment where no reflective surfaces were present in close proximity. However, to fully prove this hypothesis, the experiment would have to be performed under conditions where the reflections were utilized in a controlled manner.

Chapter 6

Future Work

As stated in Chapter 5, both synchronization techniques worked reliably. However, the limitations of performance are not yet reached. If the start time for all devices would have a higher accuracy, for example with the aid of clock synchronization, the window for the initial correlation could be decreased. The properties of the synchronization sequences could also be improved; in this thesis, no further examination was performed on how well the random sequences perform. If better sequences with more efficient modulations were found, the sequences could be shortened. This, together with the previous mentioned point, would lead to a shorter initial synchronization sequence.

An interesting topic would also be to synchronize the single phones in such a way, that at a specific place in a room no significant offset between the devices was measurable. This would require distant phones to emit their sound earlier than close phones due to the limited propagation velocity of sound waves. A possible approach would be to designate a master phone; at its place, no significant offset between the phones should be measurable even if the master phone gets moved around in the room.

A first possibility to achieve this is that all slave phones must know their distance to the master. The disposal of the slave phones in the room has to be evaluated dynamically such that the slaves always know their current distance to the master. To achieve this, an efficient and reliable procedure has to be found.

A second approach could be to let the master record the slave phones and give them a feedback, how they should adapt their playback such that the master does not detect an offset anymore. However, this would require that the master phone can distinguish the different slave phones by their audio output which could be difficult to achieve since all phones play the same song. A possible approach to distinguish the slave phones based on their audio output would be to assign each device an individual sequence. The devices could use this sequence to modify the song slightly, for example hopping back and forth a few samples during playback or changing the output volume according to the sequence and therefore create an individual pattern.

Bibliography

[1] comparis.ch: Drei von vier Schweizern sind smart unterwegs. (2016) URL: https://www.comparis.ch/comparis/press/medienmitteilungen/artikel/2016/telecom/smartphone-studie-2016/smartphone-verbreitungsstudie-2016.aspx, visited: 06.06.2016.

[2] JekApps: SoundSeeder Music Player. (2016) URL: https://play.google.com/store/apps/details?id=com.kattwinkel.android.soundseeder.player&hl=en, visited: 06.06.2016.

[3] Amp Me inc: AmpMe. (2016) URL: https://play.google.com/store/apps/details?id=com.amp.android&hl=en, visited: 06.06.2016.

[4] Luchsinger, K.: Distributed Speaker Synchronization. (May 2015) Distributed Computing Group, Computer Engineering and Networks Laboratory, ETH Zurich, URL: ftp://ftp.tik.ee.ethz.ch/pub/students/2015-FS/SA-2015-02.pdf.

[5] Willi, S.: Distributed Speaker Synchronization. (December 2015) unpublished thesis, Distributed Computing Group, Computer Engineering and Networks Laboratory, ETH Zurich.

[6] android.com: Android Interfaces and Architecture. (June 2016) https://source.android.com/devices/, visited: 09.06.2016.

[7] tutorial4android.com: What is Android? (June 2016) http://tutorial4android.com/, visited: 09.06.2016.

[8] android.com: Application fundamentals. (June 2016) http://developer.android.com/guide/components/fundamentals.html, visited: 09.06.2016.

[9] Dcirovic: Dalvik (software). (June 2016) https://en.wikipedia.org/wiki/Dalvik_(software), visited: 09.06.2016.

[10] android.com: Audio Latency Measurements. (June 2016) https://source.android.com/devices/audio/latency_measurements.html, visited: 09.06.2016.

[11] Tipler, P.A., Mosca, G.: Physik für Wissenschaftler und Ingenieure. Spektrum Akademischer Verlag (2009)