Android Audio Synchronization

Semester Thesis

Dimitra Azariadi
adimitra@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Simon Tanner, Gino Brunner
Prof. Dr. Roger Wattenhofer

June 19, 2017
We address the problem of audio playback synchronization of multiple Android devices scattered across a room. As a first step, the clocks of the different devices are synchronized using the Network Time Protocol (NTP), and a common start time of the playback is set. Even if we assume high precision of the time synchronization protocol and absence of other factors that can introduce offset to the starting point of the playback, a precise initial synchronization is not enough. Periodic resynchronization of the playback is needed, due to the effect of clock drift. In this direction, we propose a solution of having each device periodically record the audio output of the rest of the devices, calculate the offset of its own playback to this by performing cross correlation, and correct the offset so that it gets synchronized to what the rest of the devices are playing at that moment. We use up to five devices to evaluate the maintenance of synchronized playback over a period of time, as well as the ability to correct initial offsets in the starting point of the playback.
Contents

Abstract i

1 Introduction 1
  1.1 Motivation ......................................... 1
  1.2 Related work .................................... 2

2 Design and Implementation 3
  2.1 Initial Synchronization ........................... 3
    2.1.1 Network Time Protocol ........................ 3
    2.1.2 Problems .................................... 5
  2.2 Synchronization ................................ 5
    2.2.1 Cross Correlation Algorithm .................. 5
    2.2.2 Refinements .................................. 7
    2.2.3 Offset correction .............................. 9
  2.3 Android Application .............................. 10

3 Evaluation 13
  3.1 Initial Offset Correction ......................... 13
    3.1.1 Two Devices .................................. 13
    3.1.2 Multiple devices ............................... 14
  3.2 Clock Drift Correction ........................... 16
    3.2.1 Two Devices .................................. 16
    3.2.2 Multiple devices ............................... 18
  3.3 Real-life Scenario ................................. 21

4 Conclusion and Future Development 23
  4.1 Conclusion ....................................... 23
  4.2 Future Development ............................... 23
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>iii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td>25</td>
</tr>
</tbody>
</table>
1.1 Motivation

Playing music in a room can be a costly endeavor. It usually requires people to spend a few tens, or perhaps a few hundreds, on extra products, such as Bluetooth speakers. In order to avoid such expenses, some people even go to the extent of crafting their own do-it-yourself amplifiers to boost the sound of their smartphones or other audio sources. Such tricks include placing a smartphone into a ceramic bowl, or placing a rolled up sheet of paper on the speaker of the audio source.

Multiple smartphones are usually available in a group of people, as most people carry their smartphone in their pockets wherever they go. Smartphone speakers are designed for personal use and therefore typically are not loud enough to satisfy a group of people listening to music in a room. Nevertheless, if all available smartphones play music simultaneously, the result will be an amplified and surround sound. This set up will boost the emitted sound without the users having to buy, craft or carry any extra hardware, other than their readily available smartphones.

Of course, such a system cannot be as trivial as having all users try to press the \textit{START} button at the same time, hoping that all smartphones will start playing music simultaneously. The devices will have different delays when initiating the playback, which will cause an echo effect that reduces the enjoyment of the sound by the listener. Furthermore, additional delays will be introduced during the playback, over a period of time, because the smartphones have slightly different playback rates.

That being said, we need a robust method to synchronize the emitted sound of all participating smartphones. In this direction, we aim to develop a smartphone application that will synchronize the playback between multiple devices scattered across a room.
1. Introduction

1.2 Related work

In the past years, the task of synchronous playback between multiple devices has been addressed repeatedly.

Sullivan et al. [1] developed a method of synchronizing the playback of a digital audio broadcast by inserting a control track pulse. A transmitting device inserts control track pulses into the digital audio stream at known intervals. Each receiving device obtains the time of the received track pulse and then delays the playback of the audio stream until the received pulse is aligned with the next pulse at the known interval.

Moore et al. [2] calculate the rate discrepancies between the clocks of the devices based on timestamp information. They then perform resampling on the audio data for each device, in order to correct these discrepancies and have ‘slower’ devices catching up to the playback of ‘faster’ ones.

Bowman [3] proposes a synchronization method when a device serves media streams to multiple output devices. Data is requested from the media server by one master output device to maintain a nominal buffer fill level over time. The slave output devices receive streamed data from the media server at the rate determined by the master device’s data requests. Slave devices make playback rate corrections to maintain respective buffer fill levels within upper and lower threshold levels. Thus, all output player devices play the streamed data at the same average rate.

In the work of Goddard [4], a device transmits audio stream data with embedded control code. Each of the receiving devices produces an audio pattern based on the control code. The receiving devices are synchronized by comparing time differences from the time the control codes are sent to them, until the reception of the audio patterns from the transmitting device.
Design and Implementation

We consider the case of multiple users in a room, having in their possession a number of smartphone devices.

The initial stage towards addressing the problem of audio synchronization of multiple devices, is to set a common starting point of the playback. This is achieved by using a time synchronization protocol to estimate the time differences of the clocks of the different devices. Once these differences are known, a start time is set for the playback. This stage is described in detail in Section 2.1.

The next stage towards achieving synchronized playback is to correct the initial offsets in the starting point of the playback, and maintain the synchronization of audio over time. For this stage, which is the main focus of this thesis, we propose a cross correlation algorithm to measure and correct the time delay between the audio output of each one of the smartphones and the audio output of the rest, as described in detail in Section 2.2.

We implement the above mentioned stages in an Android application, presented in Section 2.3.

2.1 Initial Synchronization

In order to facilitate event synchronization between multiple devices, we need to synchronize their clocks so that they share a common time base. In our case, the event to be synchronized is the start point of the playback of different smartphones.

2.1.1 Network Time Protocol

To estimate the time differences of the individual clocks of the participating smartphones, we choose to implement an algorithm based on the Network Time Protocol (NTP) [5]. All smartphones are required to be connected to the same
Wi-Fi network, and the smartphone which obtains the smallest IP address becomes the *master*, while the rest become *slaves*. The clock of the *master* is used as the common time base for all smartphones.

Figure 2.1: Clock offset calculation and common playback starting point.

Figure 2.1 illustrates an example of the communication between the *master* and each *slave* and the calculation of the latter’s offset to the *master* clock. The *master* initiates the communication with the *slave* by sending a *sync* packet containing the timestamp of the departure time of the packet. The *slave* receives the *sync* packet and timestamps the packet’s arrival time using its own clock. The difference between the *sync* packet’s departure timestamp and the *sync* packet’s arrival timestamp is the sum of the *slave* clock offset to the *master* clock and the network propagation delay. This is calculated by the *slave* as *offset_1*. The *slave* then sends a *delay request* packet to the *master* and timestamps its departure time corrected by *offset_1*. The *master* receives the packet, timestamps its arrival time, and sends the timestamp back to the *slave* in a *delay response* packet. The difference in the two latter timestamps is now double the propagation delay, *offset_2*. This way, the *slave* can now calculate the true offset to the *master* clock, which is (*offset_1 + offset_2*). At the end of the NTP procedure, the *master* sends one last packet to the *slave* containing a time in the future in which to begin the playback. The *slave* corrects this time by (*offset_1 + offset_2*) and waits until the calculated time in the future to start the playback. Once this procedure has been applied to all *slave* smartphones, playback should start at
the same moment from all smartphones.

2.1.2 Problems

Several factors affect the synchronization levels achievable by the above procedure. A clock’s resolution can affect the accuracy of the timestamps transmitted in the NTP packets in a range of tens of milliseconds [6]. Inherent tendencies of the network can cause jitter. But most importantly, time differences can occur until the audio is actually output by the speakers of the different smartphones. Such time differences are due to differing latencies in the audio stream processing, resulting in the audio outputs being slightly out of sync with each other, in a range of tens or hundreds of milliseconds. This can cause an echo effect to the listener. The thresholds of delays for humans to perceive sounds as echoes are discussed in chapter 3.

2.2 Synchronization

The main focus of this thesis is to not only correct any initial error in the playback starting time synchronization, but to also maintain a smooth and synchronized playback between multiple devices over a long period of time.

An inevitable problem when working with synchronization is that of clock drift, which just as the name implies is the phenomenon of a clock drifting apart from another reference clock, over time. This means that, even if perfectly synchronized in the beginning of the playback, over a period of time the smartphones will eventually be playing the audio stream at slightly different rates, resulting in them getting out of sync from each other [7].

2.2.1 Cross Correlation Algorithm

As a solution to the problem of maintaining synchronization, we propose that each participating smartphone periodically records what the other smartphones are playing at that moment. It then performs cross correlation with its own playback in order to estimate its offset to what the other devices are playing. Finally it corrects this offset so as to become synchronized with the others.

To describe this procedure in detail, let’s assume that a number of smartphone devices are playing an audio stream and one of them, which is possibly out of sync, wants to synchronize its playback with what is being played by the rest. At time $t_i$ the smartphone starts recording. By that time, the smartphone has played $x_i$ samples of the audio track, so its playback position is $x_i$ at $t_i$. We assume this position to only be approximate and not accurate, in order to allow for expected errors in the estimation of the number of samples actually played.
and those still in the buffer waiting to be consumed. The smartphone goes on to record for ten seconds. The first five seconds of the recording, the smartphone does not alter the volume of its sound, resulting in actually recording its own sound. For the remaining five seconds of the recording, the smartphone mutes its own sound, resulting in recording just what the rest of the devices are playing. By the end of the recording, the smartphone’s playback position will be \((x_i + samplingRate \times 10\text{seconds})\). These intervals are depicted in Figure 2.2.

![Figure 2.2: Recording and playback fragments.](image)

The next step is to calculate the time offset between the recorded signal and the smartphone’s estimated corresponding playback interval. We achieve this by performing cross correlation between the signals. This is the method of sliding one signal relatively to the other and calculating the integral of their product at each position. When the signals match, the value of the function is maximized, indicating the time delay between them. In order to perform cross correlation, we choose a fragment of the playback interval with a window size of \(window_{\text{big}} = 5\text{ seconds}\), and a fragment of the recording with a window size of \(window_{\text{small}} = 2\text{ seconds}\). We do this in two different positions on the intervals, first where the smartphone was not muted, and second where the smartphone was muted, as shown in Figure 2.2. The cross correlation between the first set of signals will give the error of the playback position estimation, as \(offset_1\). The cross correlation between the second set of signals will give the time offset of the estimated playback and the recorded sound of the others, as \(offset_2\). The difference of these two offsets \((offset_2 - offset_1)\) will give the true time offset of the smartphone’s playback to the audio output of the rest of the devices.

The cross correlation between each pair of signals returns an array of the calculated values of the sliding inner product. The length of this array equals the number of sliding positions, which is the sum of the lengths of the two signals minus one. From this result, we only take into account the part of the cross correlation where the smaller signal is fully included into the bigger signal in terms of sliding position, so as to avoid the influence of the borders. This
leaves \((\text{window}_\text{big} - \text{window}_\text{small})\) sliding positions included. From the way the fragments are chosen, we expect that if synchronized, the cross correlation will be maximized when the two signals are approximately aligned in the centre. This allows for an approximate worst case of \(\pm(\text{window}_\text{big} - \text{window}_\text{small})/2\) detectable offset of the signals, as shown in Figure 2.3. In our case, this is \(\pm1.5s\).

![Figure 2.3: Maximum correctable offset.](image)

The choice of the two window sizes was made having in mind that this procedure should only improve the synchronization of devices, leaving almost no space for erroneous offset corrections that would deteriorate the synchronization. Window sizes should stay relatively small so as to not include the frequency error caused by the clock drift phenomenon. This would have a negative influence on the cross correlation results. A \(\text{window}_\text{small} = 2\) seconds was chosen as a desirably small window size. Smaller windows were also tested, for example 1 second long, but resulted in some erroneous cross correlation results in the sense of strength of the correlation, as explained in the following section. The choice of \(\text{window}_\text{big} = 5\) seconds mainly depended on the maximum detectable offset we want to allow. As mentioned above, this combination allows for a \(\pm1.5s\) offset between the signals, which we consider a reasonable one. An even bigger window could lead to a repeated sound clip being matched in multiple positions in the bigger audio clip.

### 2.2.2 Refinements

#### Strength of correlation

The maximum value returned by the cross correlation function, which implies matching of the signals, is difficult to interpret. Its amplitude depends on the energy of the signals. For this reason, we normalize this value, once calculated, in such a way that it can have an absolute value between 0 and +1. The factor by which we normalize is

\[
\text{norm}_\text{xcorr-factor} = \frac{1}{\sqrt{\sum_{i=0}^{n-1}|x[i]|^2 + \sum_{i=0}^{n-1}|y[i]|^2}} \quad [8],
\]
where $x$, $y$ are the two signals, and $n$ is the length of the shorter one. For the longer signal, we compute this sum over the fragment of it, of length $n$, which contributed in the integral of the inner product of the signals for the sliding position where the maximum cross correlation value was computed. In this way, if the two signals are identical, the maximum value of the cross correlation will have the absolute value of $+1$ after the normalization. That means that they correlate 100%.

We do not normalize the whole cross correlation result, because that will require computing the corresponding normalization factor for each sliding position of the signals. When the two signals have the same length, this factor is constant for all sliding positions. Thus, we only 'locally' normalize the maximum value of the cross correlation.

We choose to take into consideration only the calculated cross correlation peaks with absolute value greater than 0.1. Any absolute value less than 0.1 is rejected as erroneous and no valid offset between the signals is determined. In this way, we avoid calculating wrong offsets that can result from very noisy recordings, or uncorrelated signals due to their offset falling outside of the detectable range.

### Multiple peaks

In the case of recording the sound coming from one source, the cross correlation will result in a single peak value. However, if the recording includes sounds coming from more sources, meaning that multiple other devices are playing the audio stream, possibly not synchronized with one another, then the cross correlation function will detect as many peaks as the devices playing the audio. These peaks are not always clearly distinguishable. Sounds coming from different sources that are synchronized with each other, will appear as one peak in the cross correlation. The values of the different peaks may differ a lot in amplitude. Echoes in the recording created by nearby walls and objects, will also appear as peaks.

One approach would be to choose the highest peak calculated by cross correlation and have the smartphone correct its offset to that. That generally works, since each device will be synchronizing itself to another, probably the one it 'hears' most clearly, and after some rounds of this procedure all smartphones will end up synchronized. But this approach could also end up in a partitioning of different, and possibly out of sync, groups of synchronized devices. For example, device A synchronizing to device B and device B synchronizing to device A, while device C is synchronizing to device D and device D is synchronizing to device C.

To achieve better overall synchronization of the participating devices, we need each smartphone to synchronize to the mean position of the peaks detected. We use the following procedure. We detect the highest peak returned by the cross correlation. This is probably coming from the device that the smartphone
‘hears’ most clearly. We also calculate the average absolute value of all local maxima in the cross correlation. By local maxima we mean every sample in the waveform that is greater in value than its two neighboring samples. We then set a threshold as \( \text{threshold} = \text{local\_maxima\_average\_value} + (\text{highest\_peak\_value} - \text{local\_maxima\_average\_value})/2 \). We consider all peaks with value larger than this threshold as positions of correlation to sound sources. Figure 2.4 illustrates an example cross correlation output, for a recording of three audio sources. The black circle indicates the highest peak. The red line indicates the average value of all local maxima, and the yellow line indicates the threshold. The purple circles indicate all peaks taken into account, together with the highest peak, for the calculation of the mean position to get synchronized to. This position is indicated by the green line.

![Figure 2.4: Correlation to multiple audio sources.](image)

### 2.2.3 Offset correction

As soon as we calculate the time offset of the smartphone’s playback to the sound that the rest of the devices are playing, the offset needs to be corrected.

One way to do this, is by resampling part of the playback to either speed
up or slow down, until the offset is corrected. We tried both upsampling and downsampling a five second fragment of the playback to correct an offset of 200ms. The quality of the resulting sound we got had deteriorated. This happens because, when changing the duration of an audio clip by resampling, the new samples are played at the original sampling frequency, which causes a shift in the pitch of the audio [9]. Slowing down the recording lowers the pitch, and speeding it up raises the pitch. For this effect to not be noticeable by the listener, an even longer period than five seconds should be dedicated to the correction of a 200ms offset. But this would make the correcting period unacceptably long, even more for bigger offsets.

Instead, we tried an instant correction of the offset, by skipping the required amount of audio samples, or replaying the required amount of audio samples, depending on the offset. In this case, no distortion of the sound is noticed, and no correction period is needed.

2.3 Android Application

The synchronization procedure described above, was implemented in an Android application. In the application, we specify the number of devices that will participate in the synchronized playback of a specific audio track. All participating smartphones should be in the same room and connected to the same WiFi network. A screenshot of the application’s interface is shown in Figure 2.5.

Once the application is launched by a smartphone, it opens up a socket to listen to packets sent by other participating devices. It lets the other participants know about its own existence, by sending out a broadcast. As soon as it learns the IP addresses of all devices, it goes on to decide its role in the NTP implementation. The device with the smallest IP address becomes the master and all other devices become slaves. The master device initiates the NTP, as described in Section 2.1, with all slave devices, one after the other. The devices communicate with TCP packets. In the end of this procedure all devices have determined a common starting point for the playback.

When the PLAY button is clicked, the application waits until the determined start time, and then starts the playback of the track. The MediaCodec class [10] is used to decode the mp3 track file into raw PCM audio data. We feed the chunks of data we get from the decoder’s output buffer, to the input buffer of an AudioTrack object [11]. Each chunk of data we feed to the audio player, we also store it in an array where we keep track of the last 12 seconds of what has been played. We use this array to retrieve the playback interval that we cross correlate with the recording signal. We also keep track of the total number of samples that have been fed to the buffer, which constitutes the current playback position.
Knowing the IP addresses of all participating smartphones, the application sorts them. Depending on its own position in the IP addresses sorting, it determines when to execute the resynchronization procedure for the first time. The device with the first position in the sorting starts recording 5 seconds after the playback has started, and the recording lasts for 10 seconds. The AudioRecorder class [12] is used in order to get uncompressed recording. Immediately after that, the offset is calculated, as described in Section 2.2, and corrected. Each other device starts this procedure 10 seconds later than the previous device. After all devices have recorded once, the same procedure is repeated.

The offset correction part is implemented in the following way. If we need to speed up the playback, we skip feeding the required number of samples to the audio player’s input buffer. If we need to slow down the playback, we retrieve the required number of last played samples from the previously mentioned array. We then feed them again to the buffer, before continuing with the new samples to be played.

To serve the purpose of this thesis, we have added buttons in the application’s
interface, specifying an extra delay to be added to the starting point of a device’s playback. These are used in the evaluation of the resulted synchronization of the participating devices, presented in Chapter 3.
We use five smartphones to evaluate the developed Android application. Three Samsung Galaxy Nexus smartphones, a Samsung Galaxy S7 and a Motorola Nexus 6. The sampling rate being used for the playback and the recording is 44100 Hz.

First, we need to answer the question of what is the interval of time difference below which two sounds can be perceived as one by the listener. In [13] Wallach et al. examined sound localization by the human ear, investigating when sounds appear as two distinct sources and when they seem fused into a single sound. They concluded that the interval over which fusion of sounds takes place is not the same for all kinds of sounds. They found the upper limit of the interval to be about 5 milliseconds for single clicks. For sounds of a complex character, like orchestral music, they found the upper limit to be much longer, as much as 40 milliseconds. When these limits are exceeded, the listener notices echo effects in the sound, or can clearly distinguish that the sounds are coming from different sources.

We can consider these numbers as approximations, as there are many factors that can affect what the listener perceives as synchronized. For example, the difference in the quality of sound when coming out of a pair of expensive headphones, or a cheap speaker, affect what the listener hears. The acoustics of the room where the sound is being played can also cause different echo effects, affecting what the listener hears and perceives as synchronized.

3.1 Initial Offset Correction

3.1.1 Two Devices

As already mentioned, the lengths of the cross correlated signals allow for a worst case of detectable offset between them, which in our case is ±1.5 seconds. In the following figures we present the detection of such an offset. Two smartphones, recorded by a third muted one, start off the playback with an initial offset of
3. Evaluation

1.5 seconds. In the cross correlation of the recording with the corresponding playback fragment, we can see the two peaks corresponding to each one of the two sound sources. The two peaks have a distance of around 66000 samples (1.5 seconds), as shown in Figure 3.1. After one of the two smartphones corrects its offset to the other one, we can no longer distinguish them as two different peaks, as depicted in Figure 3.2. Their distance can be estimated from looking closely at the plot in Figure 3.3, to be less than 1000 samples, which means less than 22 ms.

3.1.2 Multiple devices

Using the five smartphones we have available, we examine how many rounds of synchronization it takes for all of them to reach synchronization. One round of synchronization involves all devices executing the synchronization procedure once. The number of rounds it takes for synchronization depends on the following factors: How many smartphones start off the playback with an initial offset, what that offset is for each smartphone, the order in which the smartphones with initial offset execute the synchronization procedure.

As described in Section 2.3, the first smartphone starts recording 5 seconds after playback starts, and each next smartphone starts recording 10 seconds later than the previous one. The recordings last for 10 seconds. This means that, for five devices, the first round of synchronization by all devices is concluded in the first 55 seconds of the playback. Each next round of resynchronization takes the same time.
3. Evaluation

Figure 3.3: Precision of the synchronization achieved between two smartphones.

Table 3.1 presents the results of some indicative test cases. The smartphones are listed in the order that they record. That means that A records first, B records second, and so on. The initial delay is added to each smartphone’s playback starting point using the corresponding buttons of the Android application’s interface. The devices reach synchronization at some point in the stated rounds of resynchronization. Each test case is executed 10 times and the result stated is the average result of those executions. The delay added does not include the initial offset that the smartphone has anyway. For this reason some results vary around ± 1 round between executions.

Table 3.1: Test cases for initial offset correction.

<table>
<thead>
<tr>
<th>smartphone</th>
<th>initial delay (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1300 0 0 0 0 0 200 0</td>
</tr>
<tr>
<td>B</td>
<td>0 0 0 0 500 0 200 0</td>
</tr>
<tr>
<td>C</td>
<td>0 1300 0 200 200 0 300 500</td>
</tr>
<tr>
<td>D</td>
<td>0 0 0 200 1000 0 1000 0</td>
</tr>
<tr>
<td>E</td>
<td>0 0 1300 0 0 0 500 1300</td>
</tr>
</tbody>
</table>

number of required rounds of synchronization: 1 2 2 1 2 2 2 3

When the smartphone with initial delay records first, it only takes one round for the devices to get synchronized. This happens because the offset gets cor-
3. Evaluation

Figure 3.4: One scenario of smartphones’ positions.

Figure 3.5: Second scenario of smartphones’ positions.

rected before it influences other smartphones when calculating their own offset. This can be seen in the first test case. The contrary situation is seen in the next two test cases, where two round of synchronization are required. Test cases 2 and 4 show that a smaller delay results in faster synchronization. However, when more smartphones have the latter delay, it takes again more rounds for the devices to get synchronized, as shown in test case 5. The last test case shows a situation where all devices have different initial delays, requiring three rounds to reach synchronization.

Figures 3.4 and 3.5 show the different positions in which the smartphones are placed for the execution of the test cases. No difference is observed between the two scenarios. Furthermore, two songs are used for the test cases, a classical music track and a pop song. In these scenarios also, no difference is observed.

3.2 Clock Drift Correction

3.2.1 Two Devices

We evaluate the magnitude of the clock drift effect, by recording two smartphones playing an audio track with approximate duration of 1 hour and 30 minutes. The two smartphones are a Samsung Galaxy Nexus and a Motorola Nexus 6. They start the playback after executing the NTP for initial synchronization. Figure 3.6 shows the two smartphones synchronized in the beginning of the playback, as only one peak appears in the cross correlation. After 30 minutes of playback, two peaks appear in the cross correlation, with around 8000 samples distance, as shown in Figure 3.7. Finally, after 70 minutes of playback, we see in Figure
3. Evaluation

3.6 Cross correlation of recording showing Motorola Nexus 6 and Samsung Galaxy Nexus synchronized after NTP.

3.7 Cross correlation of recording showing the clock drift between Motorola Nexus 6 and Samsung Galaxy Nexus after 30 minutes of playback.

3.8 A distance of around 21000 samples between the two peaks. For a sampling rate of 44100Hz, this translates to 476 ms offset between the two audio outputs after 70 minutes of playback. Thus, the relative rate error for the two devices is 5 samples/second.

3.8 Cross correlation of recording showing the clock drift between Motorola Nexus 6 and Samsung Galaxy Nexus after 70 minutes of playback.
3. Evaluation

Figure 3.9: Cross correlation of recording showing Motorola Nexus 6 and Samsung Galaxy Nexus synchronized after NTP.

Figure 3.10: Cross correlation of recording showing Motorola Nexus 6 and Samsung Galaxy Nexus still in sync after 70 minutes of playback.

We then evaluate the resynchronization procedure developed, by recording the two smartphones playing the same track again, but this time having them resynchronize to each other. Figure 3.10 clearly shows that after 70 minutes of playback, the interval between the two peaks is approximately the same as in the begging of the playback, and does not exceed 2000 samples. This translates to an offset of less than 45 ms between the two sounds.

Furthermore, we measure the clock drift effect between the rest of the smartphone models, in the same way described above. For the Samsung Galaxy s7 and the Samsung Galaxy Nexus, we get a distance of around 12000 samples (270 milliseconds) after 70 minutes of playback. The relative rate error is 2.85 samples/second. For two Samsung Galaxy Nexus, the offset we get after 70 minutes is around 4000 samples (90 milliseconds). The relative rate error here is 0.95 samples/second. These results are depicted in Figure 3.11 and 3.12, respectively.

3.2.2 Multiple devices

We perform the same tests, having all five smartphones playing sound. Figure 3.13 shows the five devices starting off the playback synchronized. Figure 3.14 shows the cross correlation, 70 minutes later. We can distinguish a number of different peaks, within an interval of around 22000 samples.

The effect of clock drift is corrected, as demonstrated in the next two figures.
3. Evaluation

Figure 3.11: Cross correlation of recording showing the clock drift between Samsung Galaxy s7 and Samsung Galaxy Nexus after 70 minutes of playback.

Figure 3.12: Cross correlation of recording showing the clock drift between two Samsung Galaxy Nexus after 70 minutes of playback.

Figure 3.13: Cross correlation of recording of five smartphones at time $t_i$.

Figure 3.14: Cross correlation of recording of five smartphones out of sync at time $t_i + 70$sec.
3. Evaluation

Figure 3.15: Cross correlation of recording of five smartphones at time $t_j$.

Figure 3.16: Cross correlation of recording of five smartphones still in sync at time $t_j + 70\text{sec}$.

Figure 3.15 shows the smartphones starting off synchronized, and 70 minutes later, in Figure 3.16, they still are in sync. A close look at the latter plot, shows a maximum interval of peaks that is no more than 2000 samples wide, which translates to 45 ms.
3.3 Real-life Scenario

Finally, we evaluate the application with all five smartphones in a real-life scenario. We place the devices in a room, as demonstrated in Figure 3.17. The room’s dimensions are 5m by 3.5m.

![Figure 3.17: Evaluation of application in real-life scenario.](image)

The five smartphones are recorded by a sixth one placed in the middle of the room, for a duration of 70 minutes. Figure 3.18 shows the initial interval of peaks, which is around 8000 samples (181 milliseconds). The smartphones execute rounds of synchronization continuously during the 70 minutes. The accuracy of synchronization during this period fluctuates between 6000 samples (136 milliseconds) and 2000 samples (45 milliseconds) of peaks interval. Such cross correlation results are shown in Figures 3.19 and 3.20, respectively. These numbers are upper bounds of the achieved accuracy. The cross correlation results presented also include peaks that correspond to reflections. Sound travels at 343 m/s, so it takes 30ms to cross twice the room of 5m length. These reflections also influence the offset calculation, not always in the same way, but depending on the sound being played. This can explain the fluctuation in the synchronization accuracy.
3. Evaluation

Figure 3.18: Initial peaks interval in real-life scenario.

Figure 3.19: Worst case peaks interval in real-life scenario.

Figure 3.20: Best case peaks interval in real-life scenario.
Chapter 4

Conclusion and Future Development

4.1 Conclusion

In this work, we designed a procedure to synchronize the audio playback of multiple Android devices scattered across a room. The devices first synchronize their clocks using NTP and set a common starting point for the playback. Multiple factors affect the precision of this initial synchronization, and the devices can end up with different delays when starting the playback. Furthermore, even if initially synchronized, the clock drift effect causes the smartphone to get out of sync with each other over time. To correct these delays, each smartphone periodically records the rest of the devices, and cross correlates the recording with its own playback. This way it calculates its time offset to what the rest of the devices are playing, and corrects it.

We implemented this procedure in an Android application, and evaluated its performance using five smartphones of three different models. As demonstrated, we managed to correct initial offsets of up to 1500ms. We also managed to synchronize the five smartphones in situation where each of them had a different initial offset. Moreover, we investigated the clock drift effect between the smartphones models, and found the largest relative rate error to be around 5 samples/second. We then demonstrated that the developed application manages to maintain the smartphones synchronized over such a period of time. Finally, we tested the application with five smartphones in a real-life scenario. The results showed an achieved synchronization accuracy between 45ms and 136ms.

4.2 Future Development

This work can be extended by implementing a method to distribute the audio files to be played between the smartphones. One solution can be to connect the Android application to Spotify, using the Spotify Android SDK [14] and Spotify
Web API [15]. Users of the application will have to own a Spotify account. After completing an authentication flow with their credentials, the application on the multiple smartphones can share a retrieved URI for a particular audio track or playlist. The requirement would be that the songs are either downloaded locally to each device, or that some amount of buffering of the audio stream is occurring, to allow the devices to perform the cross correlations and offset correction.
Bibliography

digital audio broadcast by inserting a control track pulse (April 24 2007)

audio for synchronized playback by multiple devices (December 8 2009) US

streaming media in multiple output devices (September 6 2011) US Patent
8,015,306.


developer.android.com/reference/java/lang/System.html#
currentTimeMillis()


com/help/signal/ref/xcorr.html

279–309

reference/android/media/MediaCodec.html

reference/android/media/AudioTrack.html

reference/android/media/AudioRecord.html
