Master Thesis

Playout delay of TV broadcasting

Wouter Kooij

06/03/2014

University of Twente
Faculty of Electrical Engineering, Mathematics and
Computer Science
Nederlandse Organisatie voor
toegepast-natuurwetenschappelijk onderzoek, TNO

Supervisors UT
Prof. Dr. Ir. Boudewijn R. Haverkort
Dr.ir. Pieter-Tjerk de Boer

Supervisors TNO
Ir. Hans Stokking
Ray van Brandenburg, M.Sc.

Date of the graduation
13/03/2014

Contents

Abstract

The distribution of content in a television broadcast chain is not a simple linear process. Content is broadcasted to different locations is received at different moments. This is causing synchronization problems. In traditional television settings where the TV was merely a device to present content, this might not be a very large issue. However, with the introduction of new media and increasingly digital interaction between people over the internet, the TV landscape is changing. The TV is becoming more than just a device to watch content. Interaction with content on the television or other TV users is becoming part of the TV experience. Examples of these directions are smart TVs, second screen apps, social TV and undiscovered new directions of telemedia. To accommodate for these changes, synchronizing the content on different TVs is important.

In this study, the TV content distribution chain is examined to discover where delays are introduced in the distribution process. Next, an examination of how the displaying of TV content can be synchronized. To do this, a technique called audio fingerprinting is used. Moreover, a mobile, easy-to-use measurement system was created to measure so called delay differences between different TV setups. This measurement system is tested for accuracy and precision. And last but not least, it is used to measure playout differences (relative to a reference) between different TV setups.

Acknowledgments

This master thesis concludes the end of my study at the University of Twente. During the writing of this thesis, I have received assistance of a number of other persons which helped me made this thesis to what is now.

First, I would like to express my gratitude to my supervisors from TNO, Ir. Hans Stokking and Ray van Brandenburg, M.Sc. Their supervision and weekly meetings have been a great help through the duration of this research. Besides my supervisors from TNO, I would like to thank my supervisors from the University of Twente, Dr.ir. Pieter-Tjerk de Boer and Prof. Dr. Ir. Boudewijn Haverkort. Their ideas, comments and support have been a great help.

Next, I would like to thank everyone from the Media & Network Services department at TNO, especially my roommates Maarten, Koen, Saskia, Yorick, Eric and Harrie. Also I would like to thank Oskar van Deventer for his help in general and in promoting my App to his international colleagues.

Also, I would like to thank Roel ter Horst from KPN for his help and information regarding the TV broadcasting process of KPN.

Furthermore, I would like to thank everyone who has used my App to perform playout difference measurements.

Last but not least, I would like to thank my friends and family for their support throughout the entire duration of my study at the University of Twente.

Nomenclature

Audio fingerprint	A digital summary of characteristics of audio that can be used to identify an audio sample against reference fingerprints.
B-frame	Bi-predictive picture. A frame that may reference preceeding and succeeding I- or P-frames.
CDN	Content Distribution Network
DVB	Digital Video Broadcasting. Variants include DVB-C (Cable), DVB-T (Terrestrial) and DVB-S (Satellite)
EPG	Electronic Programming Guide
GPS	Global Positioning System
H.264	One of the most commonly used codecs for video
HDTV	High-Definition Television
I-frame	Intra-codec picture. A frame that can be decoded independent of other frames.
IPTV	Internet Protocol TeleVision
MPEG	Moving Picture Experts Group
NTP	Network Time Protocol
P-frame	Predicted picture. A frame that may reference preceeding I- or P-frames.
Playout difference	The difference in delay between the displaying of content on different TV systems. When is referred to the playout difference in this text, unless stated otherwise, this term indicates the playout difference of a local TV compared with the playout moment of the reference server.
SDK	Software Development Kit
SDTV	Single-Definition Television

1. Background

1.1. Introduction

The following scenario might sound familiar: imagine you are watching an exciting soccer match, such as a world cup game. The team you are supporting is in ball possession and is setting up an attack when you suddenly hear loud cheering noises coming from your neighbours. A little later you see where this cheering came from: a goal was scored. This is an example often given to illustrate a playout difference. A playout difference is the difference in delay between the displaying of a certain piece of content on different TV systems, possibly using different techniques or obtaining content from different content providers. These playout differences have been shown to be noticeable or annoying, even for small differences as 1 second [1].

This thesis aims to research how relative playout differences can be measured in automated fashion. It indirectly contributes to the EU funded project STEER[2] in which TNO is involved. In this project new directions of social Telemedia are researched, such as for example new applications designed for second screen companion devices. Many of these second screen applications require a strict synchronization with television content. One way to achieve this is by synchronizing afterwards using a playout difference. Additionally, having such a measurement tool allows for easily obtaining an overview of playout differences on lots of different TV setups, allowing one to choose the TV distributor which is the faster. Alternatively this could even be used for synchronizing a whole broadcast chain from different distribution channels.

1.2. Research questions

To be able to research this, research questions were composed to research this in a structured manner. The main research question tries to answer the question if synchronizing a broadcast chain is a reasonable solution for the problems mentioned in sec. 1.1. The main research question is formulated as follows:

Main research question

- Is it feasible to synchronize different types of broadcasted television signals of different television distributors by modifying broadcasting delays in a television broadcast chain?

This question cannot be answered without first asking a few sub research questions. Knowing what is causing these playout differences might provide valuable background information for answering this question. Furthermore it is necessary to know how to measure these playout differences and how accurate this can be done. Having this knowledge allows for actually using this tool and obtaining playout difference values of different TV setups. This leads to the following sub research questions:

Sub research questions

1. Which factors in TV content delivery networks are introducing delay in the content delivery process and what is their influence?
2. How can delay in the content distribution chain of TV content be measured and how accurate can this be done?
3. What is the difference of playout times of television content between different content distribution techniques, channels, distributors and geographical locations in the Netherlands?

To answer the first question, the TV content delivery chain will be researched to discover and assess elements introducing delay. Since this is expected to be largely system and configuration dependent, an expert in this field is interviewed.

Furthermore, to answer the other questions, a prototype than can measure playout differences of TV signals (cable, terrestrial, satellite, IPTV) will be developed. To do this, a prototype measurement system on the Android platform will be designed, implemented and tested. To be able to measure playout differences, the measurement system will make use of an audio fingerprinting SDK provided by the company Gracenote. The system will record audio content from a TV system and perform digital fingerprinting on this content to determine the position of the recorded sample in the content. Several tests will also be performed to test the accuracy of the system.

When this system is deployed, measurements will be performed for different TV system setups and distributors. Data obtained by these measurements combined with manually provided parameters of these tests, such as moment of recording, type of TV signal used, channel, distributor, location will be stored in a database. With this information, playout differences between different measurements are then calculated and this information will be analyzed.

1.3. Outline

This report is structured as follows. First, in chapter 3, a look into the television broadcast chain is given. This will provide an overview of the elements in the distributions chain and will give insight in where delays in the television broadcast chain arise. In chapter 4, some of these delay introducing elements are examined

more closely and an estimation of the amount of the delay introduced is given. After this theoretical part, a more practical part follows in chapter 5, explaining the development of a playout difference measurement system that can measure playout difference between a local TV setup and a fingerprinting reference. When this is done, the performance of the created system is looked at in chapter 6. Several performance tests are performed and described testing the precision and accuracy of the system. After this, the system will be used to measure playout differences between several TV broadcasters (chapter 7) and these results will be reviewed briefly.

2. Related Work

This study tries to measure delay difference in live TV broadcasts in an automated fashion with an easy to use measurement device. Being able to measure this is an automated fashion is useful for different kind of advanced television services such as social TV, shared video watching and second screen applications. Another reason for obtaining delay differences between different TV providers is scientific curiosity.

A study that identifies the need for synchronization for advanced television services is for example [3]. These advanced television services that require synchronization include concepts such as social TV, shared video watching and second screen applications. Several sources report or expect an increase in the use of these applications, especially second screen applications [4][5]. A discussion of shared video watching and the required means of synchronization can be found in [6] and [7]. The effect of a delay difference on the Quality of Experience (QoE) of viewers is examined in [8].

There are other studies that have measured delay differences of TV broadcasts [9, 6]. These studies showed that there can be delay differences of up to 5 seconds between different TV broadcasts in a single area. These measurements were done manually however and not in an automated fashion as will be done here. No related studies that measure differences in playout delay of TV broadcasts in an automated manner were found by the author.

Closely related to measuring playout differences in an automated fashion, is the direct synchronization of second screens with broadcasted TV content. There are other companies that make use of technologies such as audio fingerprinting, video fingerprinting, audio watermarking for the similar purpose of so called second screen synchronization. This allows for second screen applications such as smartphones or tablet to be synchronized with content on a television screen. For example the company Civolution [10] provides second screen synchronization techniques by using audio *watermarking*. TvTak [11] uses a different approach than most second screen synchronization techniques and uses *video fingerprinting* as a means for second screen synchronization. Other companies such as Gracenote [12] use audio fingerprinting for second screen synchronization purposes.

3. TV content delivery networks

3.1. Introduction

Different TV broadcasting techniques have different distribution channels and techniques. Because of this, different broadcasting techniques impose a difference in delays between the broadcasting moment and the delivery of content. What often can be seen in practice is that analog television is faster than digital television. This is because digital television has more delay introducing steps such as encoding or transcoding.

Furthermore, broadcasting techniques such as television from satellite usually introduce more delay than cable television broadcasting for example because the television signal has to travel from the broadcaster to the satellite and back from the satellite to a customer. This can introduce hundreds of milliseconds of delay.

To be able to get an idea where these delays in a television network arise, the content delivery networks and techniques for several of the available television broadcasting techniques are examined in this chapter. Part of this information describes the structure of the television broadcast architecture as how it is actually used by the Dutch television distributor KPN.

3.2. Overview

For different kind of TV broadcasting or distribution techniques, different amounts of delays are introduced in the TV content delivery chain. Globally, the structure of a TV broadcast chain might look as depicted in Fig. 3.1. This figure describes the complete chain the content travels, such as for example for a live registration of an event. More static content such as a movie for example is not something that is live recorded and directly fed into the content distribution chain, which means that the camera, base and editing steps are not part of the broadcast chain for all scenarios. Furthermore, the delivery from the studio to the broadcaster might be different than the delivery from a live registration to a broadcaster. For example when a live event such as a sports event is recorded, content recorded on the scene will probably transmitted via a satellite to the broadcaster whereas this may not be the case for content recorded in a static environment such as a studio. The measurement tool that will be developed will measure the delay introduced in the

path on the right hand side of Fig. 3.1. In other words, the part of the distributor, the user and the path in between these two blocks. The following subsections will describe the structure of the techniques used between the distributor and the user, i.e. in the cloud in figure Fig. 3.1. Part of this is done by using the architecture as used bij KPN as an example.

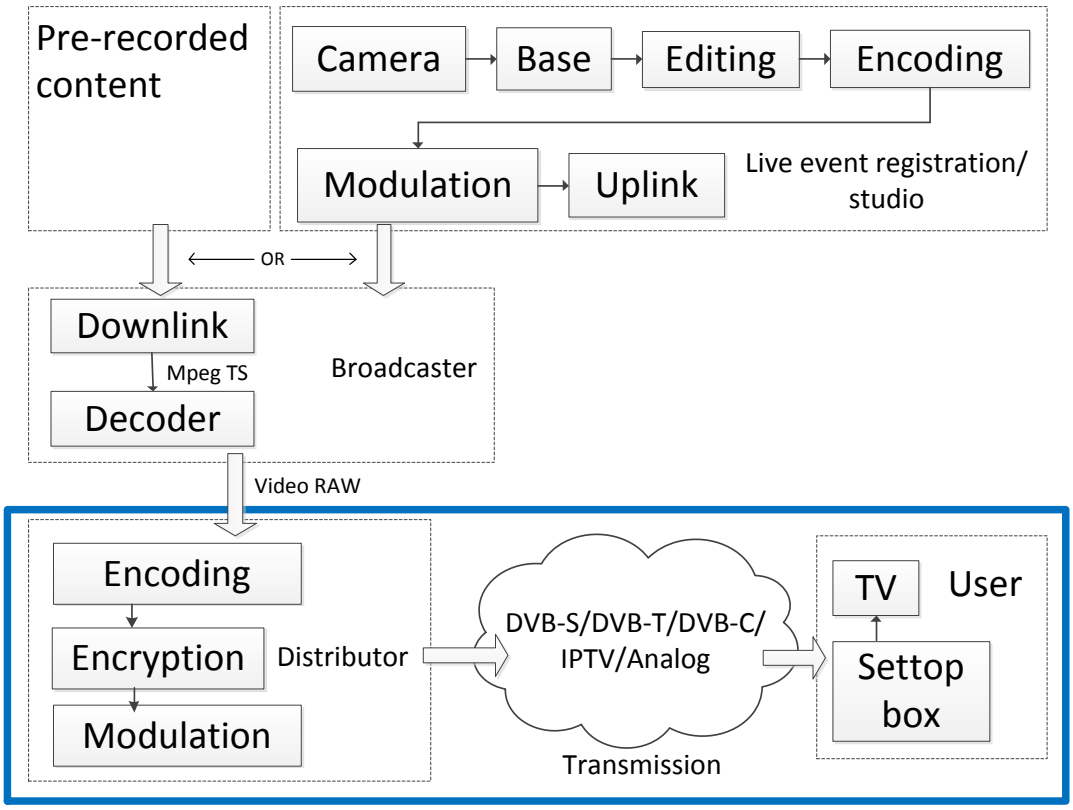


Figure 3.1.: Chain structure of a TV content delivery chain

3.2.1. Analog TV

Analog television is should be due to its technological nature faster in delivering TV content than its counterpart, digital television. This is because analog television originally doesn't use digital techniques such as decoding or encoding video. The analog television broadcasting chain is presented inFig. 3.2. Something remarkable that this picture shows is that there are actually digital distribution techniques used in the analog distribution chain, such as transport over an IP network. After the digital transport over the ETN (Enterprise Telecommunications Network) IP network, the content is decoded and modulated as an UHF (Ultra High Frequency) signal and distributed over a cable network to the homes of customers.

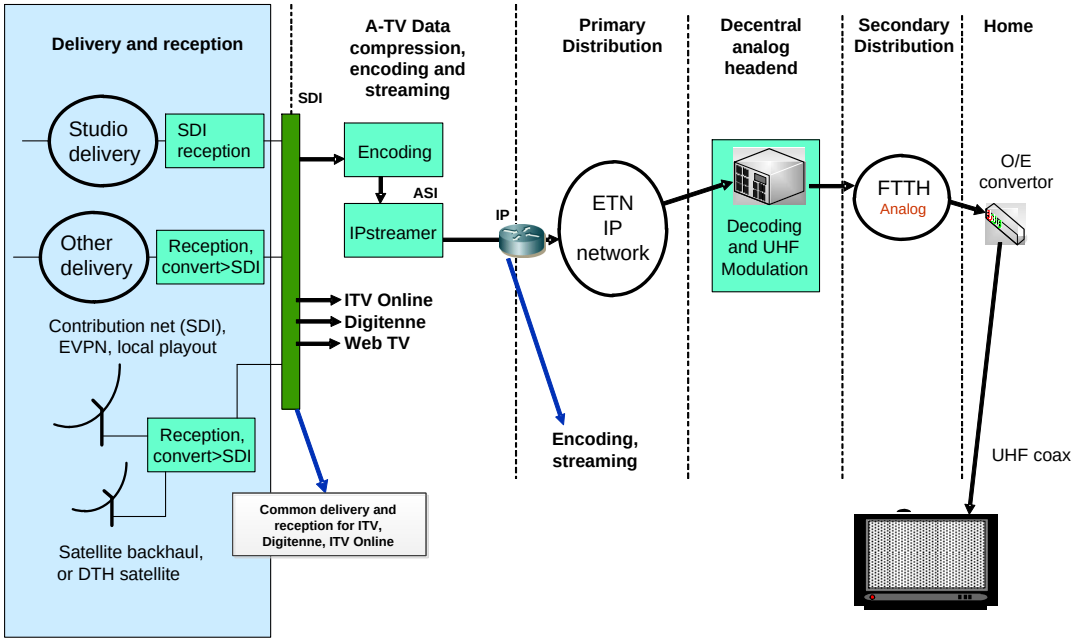


Figure 3.2.: KPN Analog delivery chain[13]

3.2.2. Terrestrial, Satellite and Cable TV (DVB)

Television that is terrestrially distributed does not make use of cables or satellites for distribution. Instead it makes use of radio towers for distributing the content. In Fig. 3.3 the terrestrial distribution chain (called Digitenne) of KPN is given. Comparing this with the analog chain (Fig. 3.2) shows that the beginning of the chain is the same. There is a common delivery for all of the KPN products. After this common delivery the content is encoded, encrypted and modulated before it is distributed. Furthermore, EPG (Electronic Programming Guide) information is multiplexed to the content to provide information what is being broadcasted. Next, the content is distributed to radio towers which in turn distribute the content to end receivers.

Part of Fig. 3.3 depicts the process of distribution of television content via satellite. A large part is the same as for Digitenne, however the last part of the chain uses satellites instead of radio towers for the end distribution. The modulation signal is transmitted to a communications satellite which then transmit this signal to a large area on earth.

3.2.3. IPTV

The structure of an IPTV network is different than one of the previously described ways of delivering TV content. IPTV makes use of an IP (Internet Protocol) net-

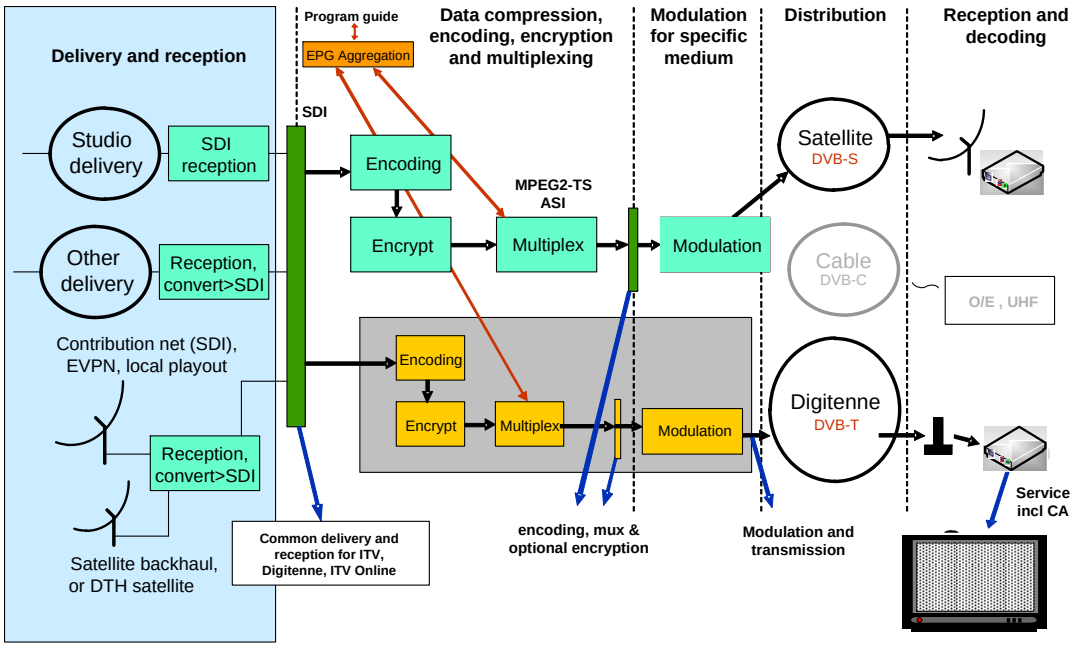


Figure 3.3.: KPN Digitenne delivery chain[13]

work for distributing video content. Unlike the association that the name IP might give, IPTV is not distributed over the public internet (unlike so called "Internet Television", which actually does use the public internet to distribute content). Instead it is distributed over a private IP network owned by a distributor. The most important difference between IPTV and the previously mentioned types of TV is that IPTV is bidirectional whereas the other types are unidirectional. IPTV requires communication in both directions between the broadcaster and the receiver whereas the other types of TV broadcast the content independent of whether the receiver uses this or not.

In Fig. 3.4 a diagram displaying all the elements included in the delivery chain of KPN, which is called "Interactieve TV", abbreviated as ITV. This figure also shows parts of a TV streaming service called ITV Online. This actually displays television content over the public internet, however it is only available if the IP address is it streamed to is part of the KPN network. ITV Online is a streaming service and not part of IPTV standards.

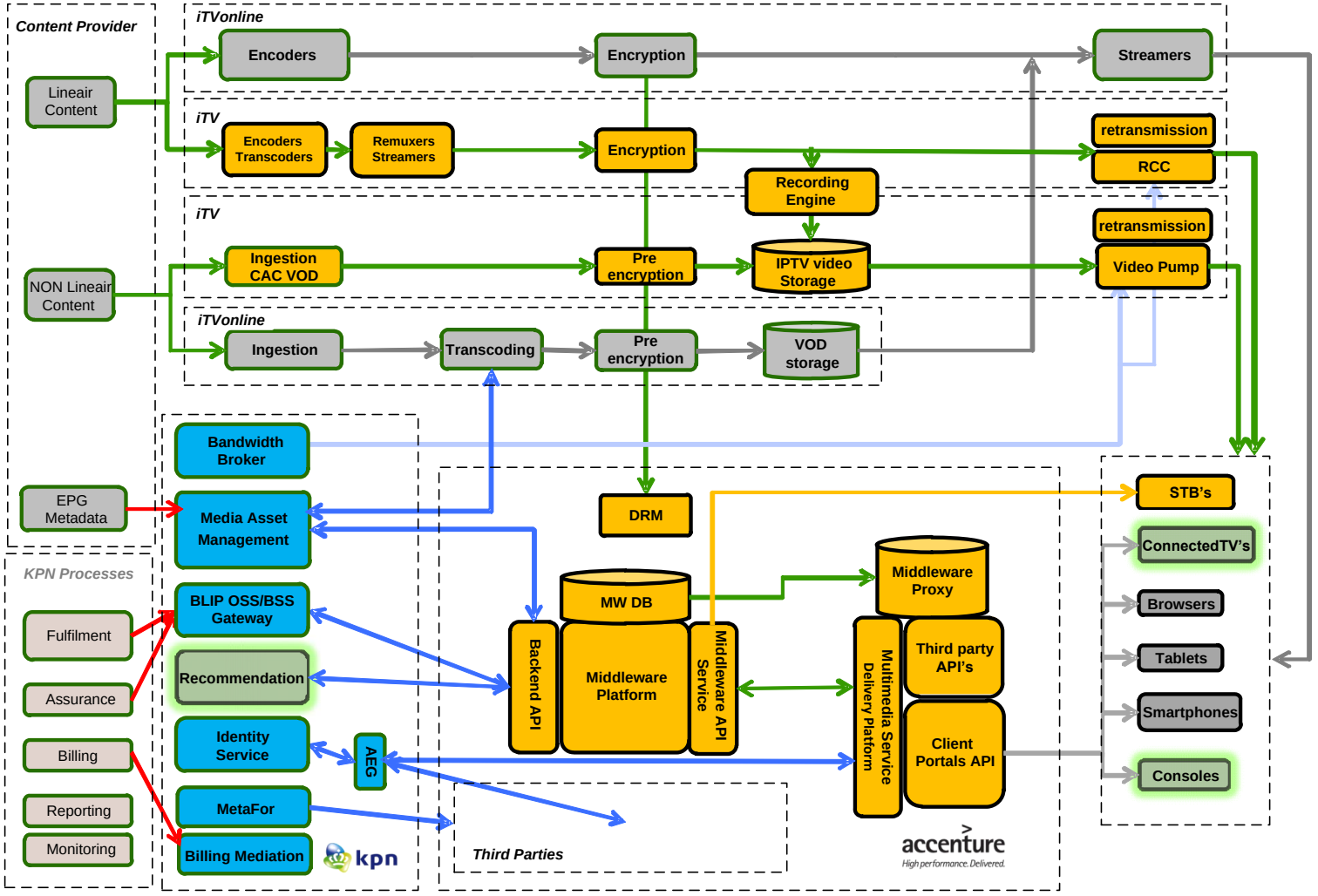


Figure 3.4.: KPN ITV delivery chain[14]

3.3. TV Content delivery chain elements

In sec. 3.2, several content distributions were described, in this section these elements are briefly examined and also is looked at the amount of delay that is introduced by these individual elements. In Fig. 3.5, some common elements in a TV content delivery chain and their order are given. Each of these elements are briefly discussed next.

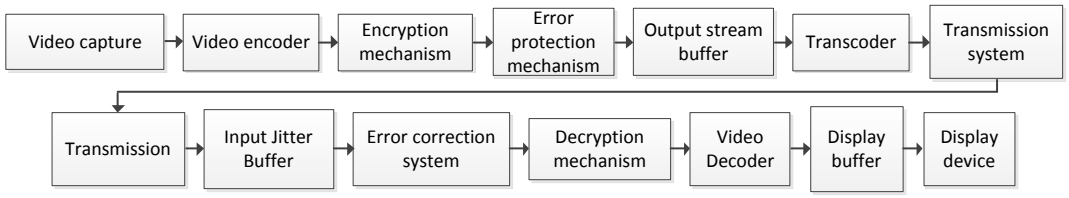


Figure 3.5.: Elements in a content delivery chain.

- **Video capture**. This is the process of recording the actual video content using cameras. Depending on the hardware, this will introduce a small amount of delay to the content delivery chain structure.
- **Video encoder**. The video encoder is the device that processes the captured video from a possibly large recording format to a format suitable for output. It does this by compressing the video into a smaller format. One part of the video encoding part is to find a good balance between video quality and the amount of data needed to represent this. Different digital broadcasting techniques use different ways to encode video, however an often used encoding format is the H.264/MPEG-4 AVC standard. This can for example be used in IPTV (Internet Protocol Television) or in DVB (Digital Video Broadcasting). The process of video encoding can introduce a relatively large amount delay. Efficient coding processes use future frames to predict current frames. This is explained in more detail in sec. 4.2.
- **Encryption**. The encryption mechanism is responsible for adding encryption to the content, to achieve access control over the content. In this way, access can be restricted to paying subscribers. This should not pose too much load on the system, so the added delay will be small.
- **Error protection**. This is a technique that allows for controlling errors in data transmission over an unreliable channel. It allows for correction errors that occurred as a consequence of the transmission of the data. This is usually done by adding error correcting bits to a data packet.
- **Output stream buffer**. The output stream buffer is a buffer that buffers the video content such that it can be transported in chunks over the transmission system without much delay.

- **Transmission**. The transmission process is the process that provides for the transport of the content. It transfers the content from content distributor to end systems.
- **Transcoder**. This is the process of converting video from one format to another. The variety of devices on which content can be displayed imposes the usage of different video formats. Not each devices can display a certain encoding format, so the transformation from one codec to another is necessary.
- **Input Jitter Buffer**. The input jitter buffer is a buffer that allows for a consistent availability of data at the receiving side.
- **Error correction.** is the process of correcting errors that were possibly introduced in the transmission process. The bits that were added in the error protection phase are now used to correct possible transmission errors.
- **Decryption**. The decryption phase decrypts the encrypted content such that it can be further processed.
- **Video decoder**. The video decoder does the reverse of the video encoder, it decodes the video to a suitable format for displaying.
- **Display buffer**. The display buffer is used to buffer the video content such that it can be displayed on the screen frame by frame without interruptions.

4. Delays in TV content delivery networks

4.1. Introduction

In chapter 3, the chain structure of TV delivery content was examined and delay introducing elements were identified. In this chapter, these identified delay introducing elements are further examined. This is done by looking at related work that inspects some of these elements individually.

In [3], a manual measurement was performed to measure these delays. This gives an idea of the order of magnitude of the delays in a TV chain. Those results are provided in Fig. 4.1. According to these measurements, the largest amounts of delay are introduced by the encoding and transcoding process. Also, the buffering and decoding process can introduce a considerable amount of delay. The estimations in this measurement only include delays introduced in a CDN. This is not something that is completely representative for IPTV. In IPTV, there are other processes and techniques such as retransmission and Rapid Channel Change (RCC) that cause delays in a broadcasting chain.

Delays		**Typical delay range (ms)**
Source	Video capture	17 - 40
	Video encoder	50 - 2000
	Encryption	0 – 50
	Error protection	0 – 100
	Output stream buffer	50 - 500
Uplink	Transmission	10 - 300
Processing	Transcoder	0 - 2000
Downlink	Transmission	10 - 300
Home	Input jitter buffer	50 - 500
	Error correction	0 - 100
	Decryption	0 - 50
	Decoder	50 - 500
	Display buffer	0 - 50
Total		**250-6500**

Figure 4.1.: Delays in a CDN [3]

In Fig. 4.2, delay is classified in two types, variable and constant delay. The encoding and decoding of a video and the buffering part is classified as a variable amount of

delay. The transmission is classified as a constant amount of delay. According to this figure, which is based on the DVB standard, the total amount of broadcasting delay can be considered constant. The digital processing among which the encoding and decoding of video can accumulate to a significant difference compared to a signal that is transported using analogue techniques [7]. Standards such as the DVB-standard are not designed to provide for delay differences between different end points. Studies show that there are delay differences of up to 5 seconds for TV broadcasts in a single geographical area [9, 6]

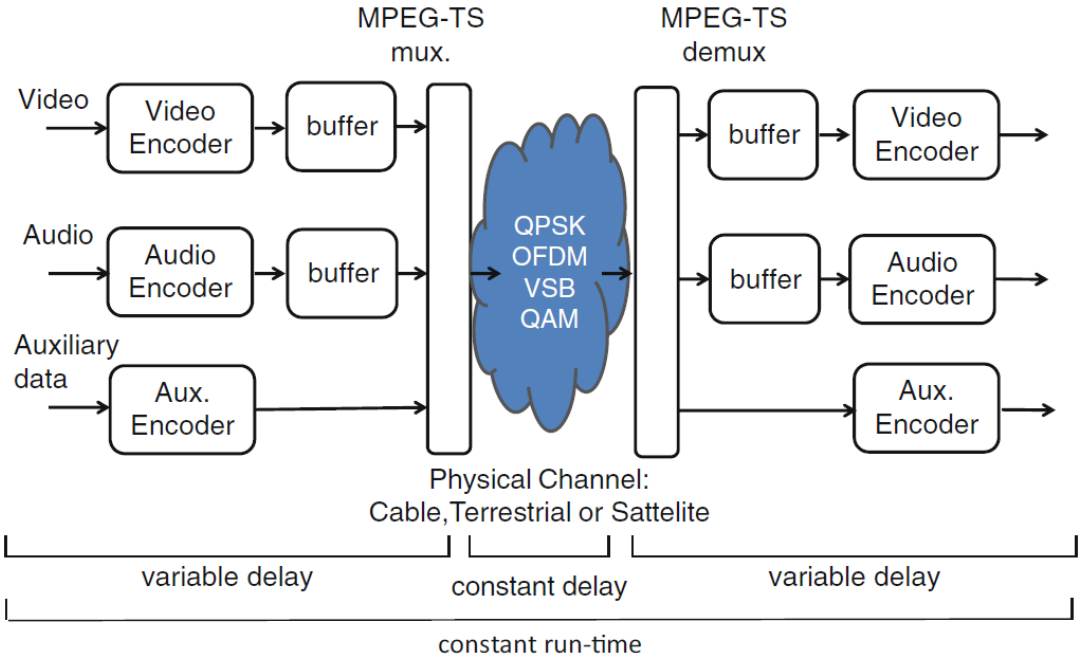


Figure 4.2.: End-to-end delay in digital TV [7][15].

4.2. Encoding and decoding

As was mentioned earlier and can also be seen from Fig. 4.1, encoding (and decoding to a lesser degree) are among the largest factors that introduce delay. The amount of delay introduced in these steps depends on the quality of the TV content. High quality video such as High-Definition (HD) requires more processing power to encode than lower quality video such as Standard-Definition (SD). In various SDTV systems, MPEG-2 standard is used for encoding or decoding whereas MPEG-4 is used in many HDTV systems. As part of the MPEG-4 standard, the H.264 codec is responsible for encoding and decoding video. This codec can be used in different settings, which have different performance and efficiency characteristics. So the actual amount of delay introduced by this codec is dependent on the coding profile settings that are being used.

In [16], a delay analysis is given of delays introduced by the buffering process of H.264, which is displayed in Fig. 4.3. In this analysis, delays are expressed as functions of the duration of a single frame period, T_{frame}, which is the duration of the processing of one frame (4.3). In this theoretical estimation, factors such as the basic processing block size and the pipelining strategy used by the encoder or decoder are not included. So the actual delay might well be larger. Furthermore this analysis is based on using a constant bitrate (CBR), which is something that will produce less peaks in bitrate and therefore allow for a more constant content delivery. This makes it suitable for providing constant quality of service which would be desirable property for TV content delivery systems.

The amount of delay introduced by the complete system depicted in Fig. 4.3 is defined in (4.1). This is the sum of all the elements. Note that D_{net} is actually part of the distribution and not of the actual encoding or decoding. Two of these variables, D_{cap} and D_{proc} can adopt different values, depending on the macroblock size used in the encoding process. In 4.1, these values are both assigned a value equals to the period of a single slice T_{slice}, which roughly equals $0.05\,T_{\text{frame}}$ (4.2). This is based on a macroblock size that is representative for MPEG-2. This means that for higher quality encoding (such as MPEG-4), this value will be larger.

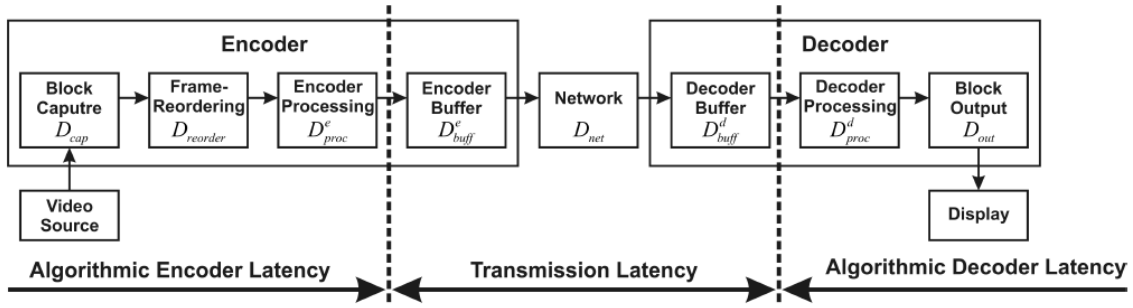


Figure 4.3.: Delay added in video transmission [16]

$$D_{\text{sys}} = D_{\text{cap}} + D_{\text{reorder}} + D^e_{\text{proc}} + D^e_{\text{buff}} + D_{\text{net}} + D^d_{\text{buff}} + D^d_{\text{proc}} + D_{\text{out}} \tag{4.1}$$

$$D_{\text{cap}} = D_{\text{proc}} = T_{\text{slice}} \approx 0.05\,T_{\text{frame}} \tag{4.2}$$

$$T_{\text{frame}} = frame\,period \tag{4.3}$$

4.2.1. Coding types

As mentioned, a large part of the processing power and thus the delay of encoding depends on the efficiency of the codec. This is something that is defined by different frame types. There are 3 basic frametypes: I-, P- and B-frames. Together, these

frametypes are called a Group of Pictures (GOP). Not all types of codes make use of all three of these frametypes. Some codecs do not use P- and/or B-frames for example. An I-frame can be decoded independent of other frames and is basically a single image. A P-frame is frame that may reference other *preceeding* I- or P-frames. A B-frame may reference both preceeding and succeeding I- or P-frames, see Fig. 4.4. Depending on the coding types, the amount of B- or P-frames differs. The more efficient the codec is, the larger ratio B- and/or P-frames to I-frames usually is. Logically, this has the implication that the longer the distance between succeeding I-frames is (and thus the amount of B- and/or P-frames in between), the more frames needs to be buffered, the longer the encoding delay will be. In other words, more efficient encoding introduces more delay.

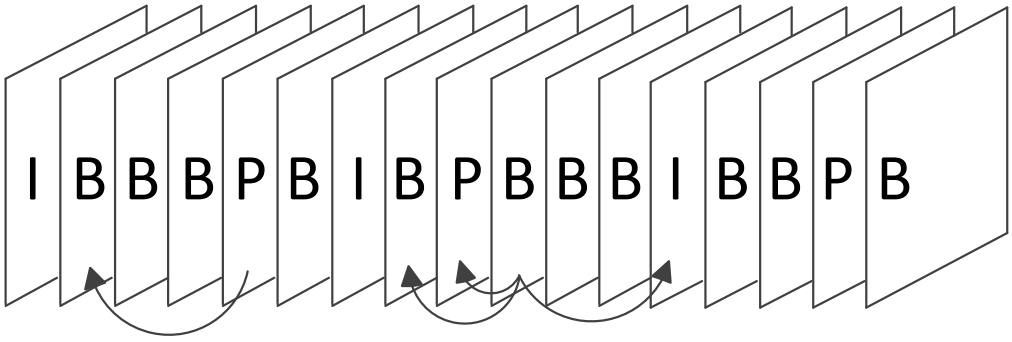


Figure 4.4.: Video encoding frame types

Intra frame coding (I coding)

This coding mode uses only information contained in its own frame. It does not use any information from other frames. The delay introduced in this inefficient coding mode is little, because this coding mode is simpler than the other coding methods and no temporal processing is performed outside of the current picture or frame. The buffering delay for an unbalanced video texture (e.g. a "worse" case scenario) is estimated in [16] to have a value as displayed in (4.4). Substituting this formula (and the other known values this far) in the equation for the total delay (4.1) yields equation (4.5).

$$D^{e}_{\text{buff,I}} = D^{d}_{\text{buff}} \approx 0.4\, T_{\text{frame}} \tag{4.4}$$

$$D_{\text{sys,I}} = 0.9\, T_{\text{frame}} + D_{\text{net}} + D_{\text{out}} \tag{4.5}$$

Inter-frame coding (Predicted, IP coding)

In this coding mode, not only information from the current frame is used, but also from other frames. Frames are predicted based on content of previous frames using a technique called block-based motion compensation. Here, the buffer delay is defined in (4.6) [16]. p_{ratio} is the average size ratio of predicted (P) frames versus intracoded (I) frames. N_{GOP} is the number of frames in a GOP. Substituting this into the equation for the total delay (4.1) and using realistic values of $p_{\text{ratio}} = 0.25$ and $N_{\text{GOP}} = 12$ yields equation (4.7).

$$D_{\text{buff,IP}} = \frac{1}{1 + (N_{\text{GOP}} - 1)p_{\text{ratio}}} N_{\text{GOP}} T_{\text{frame}} \tag{4.6}$$

$$D_{\text{sys,IP}} = 4.1 T_{\text{frame}} + D_{\text{net}} + D_{\text{out}} \tag{4.7}$$

Inter-frame coding (Bi-directional predicted, IPB coding)

This mode is similar to predicted inter-frame coding, except that content of future frames are also taken into account when predicting the content of a frame. This automatically means that a next frame has to be available before it can be used, thus introducing more delay. Because of this, the buffer delay is the same as in inter-frame coding with an added frame reordering delay of 2 T_{frame} (IBP-mode) or 3 T_{frame} (IBBP-mode), this yields (4.8) and (4.9) as equations for the total system delay respectively.

$$D_{\text{sys IPB}} = 6.1 T_{\text{frame}} + D_{\text{net}} + D_{\text{out}} \tag{4.8}$$

$$D_{\text{sys IPBB}} = 7.1 T_{\text{frame}} + D_{\text{net}} + D_{\text{out}} \tag{4.9}$$

4.2.2. Conclusion

As mentioned at the start of sec. 4.2.1, more efficient codecs use more P- and/or B-frames. In other words, IPB coding is more efficient than IP coding, whereas IP coding in turn is more efficient than I coding. The previous equations show that the more efficient the codec is, the larger the amount of frames to be buffered is. So the more efficient the codec profiles are, the more delay is introduced.

4.3. Transmission delays

Another source of delays is the transmission of the TV signal. Depending on the technique used this can introduce a different amount of delay. The TV broadcasting technique where the transmission delay is large is broadcasting TV over a satellite.

TV satellites are located in a geostationary earth orbit with a distance of 35786 km from the earth's equator. Radiowaves between satellites and earth travels at the speed of light, $c = 3.0. \cdot 10^8 m/s$, so this means that the time it takes at least $2 \cdot \frac{35786000}{c} \cdot 1000 = 239ms$ for a signal to travel from a transmitter on earth to the satellite and back to a receiver on earth. Since most senders and receivers will probably not be located at the equator and the satellite is probably not orthogonally situated on the the sender or receivers location on earth, this will be a bit more. This is only the actual travel time for the signal and other steps for receiving and processing the signal are also required.

4.4. IPTV Techniques

In addition to delays that are common for multiple types of TV distribution techniques such as encoding delays, IPTV has other techniques that influence the delay in the TV broadcast. These techniques are retransmission and RCC (Rapid Channel Change). These mechanisms try to increase the Quality of Service (QoS) and the Quality of Experience (QoE) of the IPTV service. For the retransmission technique, this is done by offering packets in an UDP (User Datagram Protocol) stream to the receiver in case a packet over the TCP (Transport Control Protocol) stream was lost. This is because the UDP protocol is a connectionless protocol and does not require a connection to be set-up, unlike TCP. Therefore this is much faster for quickly resuming an interrupted stream or preventing a stream from being interrupted.

RCC is a technique which sends a burst of TCP video packets to allow for quickly initiating the TV stream for a specific channel. This allows for quickly displaying a stream and a little later changing this unnoticed into a UDP stream arriving later.

4.5. Delays in the KPN Chain

In sec. 3.2 several figures were presented which displayed how the architecture of TV systems in general and specifically those of KPN look like. In the previous parts of this chapter and specifically in Fig. 4.1, delay durations in the TV content delivery chain were estimated. Based on an interview with Roel ter Horst from KPN, estimations of delays in the KPN TV content delivery chain were obtained and are presented in Tab. 4.1. Most delay in ITV ("Interactieve TV", IPTV) is coming from the encoding process. Furthermore, in Digitenne, the distribution to and over the DVB-T Single Frequency Network is large source of delay, although no exact numbers are known. Comparing the numbers of the encoding process for KPN ITV with the numbers sec. 4.2, there is quite a difference between the different estimations. Depending of the coding profile, the estimated delay according to [16] is at most $7,1T_{\text{frame}} + D_{\text{net}} + D_{\text{out}}$. Assuming that $T_{\text{frame}} = 40ms$, this would translate into a delay of $284ms + D_{\text{net}} + D_{\text{out}}$. So either $D_{\text{net}} + D_{\text{out}}$ are introducing 3250-3750

ms delay, or KPN is using a more advanced coding scheme. The latter seems to be the case. During the interview was also mentioned that KPN uses multi-pass encoding for at least the ITV Online service. This allows for achieving a better encoding performance, at the cost of extra time needed for encoding.

Element	Delay estimation (ms)
Encoding KPN ITV (IPTV, MPEG 4 high profile level 4)	3500-4000
Encoding KPN ITV (IPTV, MPEG 2, low profile level 3)	1000-1500
KPN ITV Settopbox (STB) buffer	400
KPN ITV STB audio/video sync	20-40
Encoding in chunks for KPN ITV Online	60000

Table 4.1.: Estimation of delays in the KPN content delivery chain

5. Design and development of a playout difference measurement system

5.1. Introduction

Now that delaying introducing factors in the TV broadcasting chain have been identified and analyzed, the next step is to measure these actual (relative) delays. There are several to do this. The most simple one would be to go to the start of a television content distribution chain, accurately register what can be seen or heard there and in the meanwhile do the same at a TV at the end of the TV broadcasting chain. This could be done for example with a direct telephone connection where the sound on the start of the TV chain gets correlated with the sound of a TV at the other end of the chain. This approach would not be very practical however, especially if this has to be done on a larger scale. Not to mention the fact that acces to the place where the distribution of the TV content starts is often restricted.

There are other, automated techniques that can be used to measure this delay. Techniques such as digital (video/audio) watermarking, digital (video/audio) fingerprinting allow for detecting and comparing certain audio or video content and correlating this to a moment in time. Applying these techniques is much more practical than the previous example. These techniques would make it possible to automate the process of measuring a delay difference between the start and end of a television content distribution chain. In this chapter, sec. 5.2 first provides a brief introduction on content recognition techniques. After this, sec. 5.3 provides and overview of the design of the measurement system, including an explanation of how one of these content recognition techniques is included in the measurement system. Part of this system is obtaining accurate timing information on the mobile measurement device, an Android smartphone, which is also something that is examined in this section.

5.2. Content recognition techniques

Content recognition techniques are techniques that make it possible to recognize content based on audiovisual characteristics of audio and/or video content. It includes

techniques such as audio- or video fingerprinting and audio- or video watermarking. Fingerprinting techniques make use of so-called fingerprints that digitally summarize the content of a (part of) audio or video. These fingerprints are then matched against an earlier calculated (or in case of live TV fingerprinting a real-time calculated) set of reference fingerprints in a fingerprint database.

These fingerprints are usually created from short pieces of audio (audio fingerprinting) or video (video fingerprinting) and are created from different and a varying combination of techniques that digitally summarize the audio or video content based on features of this audio or video. These fingerprint pieces usually represent media content, which can be as short as a few seconds for audio fingerprinting.

For audio fingerprinting, these features can consist of amplitudes of audio combined with timing information (a spectrogram), selecting several anchor points and creating hashes from this information. This is what the underlying technology for the popular smartphone app "Shazam" uses [17]. Or it can make use of a mathematical cross correlation computation [18]. Video fingerprinting on the other hand, makes use of visual features of video, such as spatial, temporal, color or transform domain features.

An overview of the process of fingerprinting can be found in Fig. 5.1. First, features are extracted from a larger digital audio fragment. These extracted features are then combined and form together a digital summary of the audio piece. Combined with the starting moment (offset) of the piece of audio inside the larger stream, the audio fingerprint is then stored in a database. By querying this database, newly generated audio fingerprints can then be compared for similarity with audio fingerprints already inside the database. This matching process compares the fingerprint for perceived similarity between the audio that is represented by these fingerprints. This is not a comparison that checks for exact digital similarity (i.e. bitwise) but for the degree of similarity. Based on the similarity score between the two fingerprints and a confidence threshold, a decision is made whether both fingerprints represent the same content or not.

As opposed to fingerprinting, digital watermarking is a content recognition technique that embeds additional information in the audio or video content itself. It embeds a watermark that can be detectable in an audio or video signal for only a fraction or the whole duration of the audio or video stream. This watermark can be an added piece of audio that is in the human hearing spectrum or not. For the case of video watermarking, it can be a semi-transparent image that is added as overlay to the video content. Next to this, content can also be watermarked at packet level, inserting a digital sequence in data packets carrying the audio or video signal.

For the measurement system, audio fingerprinting has been selected to be the basis for the system. A important reason why fingerprinting was chosen over watermarking is that audio fingerprinting has the advantage that it does not require content to be modified. It can be used on content without inserting a special watermark in the audio or video to make it recognizable. It would not be possible to use audio

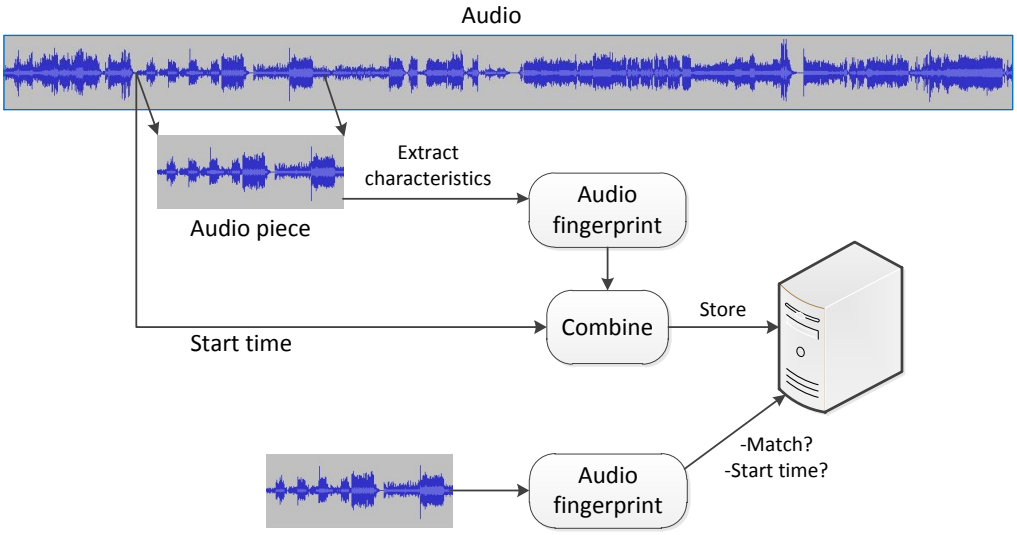


Figure 5.1.: Fingerprinting overview

watermarking, since the content of the TV cannot be changed. Since audio fingerprinting is used as technology in the app, the next subsection will explain audio fingerprinting in a little more detail.

Furthermore, for practicality and availability reasons, it was decided to create an audio fingerprinting system on a mobile device, on an Android smartphone. This platform has a large, global group of potential users of the system, making it possible to collect much data regarding playout differences. Remaining sections of this chapter explain how this measurement system is designed and created.

5.2.1. Audio fingerprinting

As mentioned in the previous chapter, the app makes use of audio fingerprinting. There are several approaches to audio fingerprinting. These approaches have a different performance for different type of audio recognition. Some approaches will work only with audio with no added environmental noise, while other approaches might be designed to recognize audio with a changed pitch or a changed playback speed. Since there are multiple approaches to audio fingerprinting, only one approach will be discussed here as an example of how audio fingerprinting might work. This is not the approach used in the app, which is closed source. The approach described here is an approach that consist of the underlying technology of the smartphone app "Shazam" and is described in [17].

This approach is able to quickly match audio fingerprint fragments against a database of earlier created fingerprints. Both files in the database and in the sample audio file undergo the same fingerprinting process and analysis. Creating a fingerprint is

done based on a time frequency graph (a spectrogram) as can be seen in Fig. 5.2. This is done by selecting several points (anchors) in this spectrogram and hashing them. Furthermore, the structure of the audio fingerprint is designed to make it easy to index in a database.

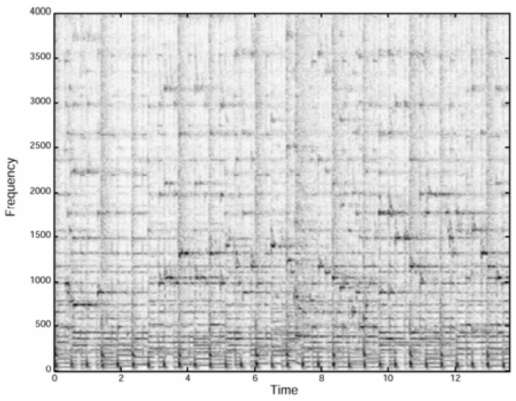


Figure 5.2.: Spectrogram of an audio fragment [17]

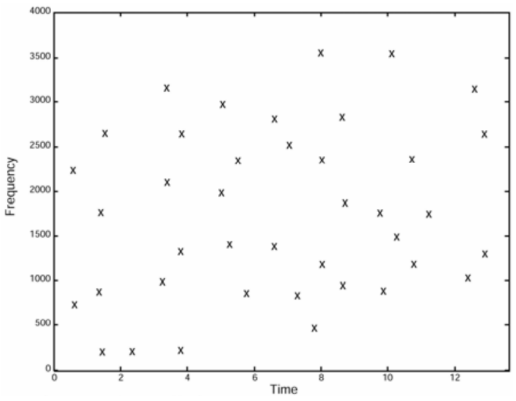


Figure 5.3.: Constellation map [17]

A fingerprint has to be reproducible from anywhere within an audio file. Furthermore, a fingerprint should have sufficient entropy (the amount information contained in the fingerprint). If this number is too low, there will be too many false positives. On the other hand, if the entropy is too high, the fingerprints will be too fragile and fingerprints will not be reproducible, especially not with noise and distortion. The audio fingerprinting technique presented by Shazam consists of three main concepts, namely robust constellations, fast combinatorial hashing and searching and scoring.

Constellations

This first concept, robust constellations is based on spectrogram peaks. These peaks are robust in the presence of noise. Spectrogram candidate peaks (time-frequency points) are chosen such that they have a higher energy content than all its neighbours in a region around it. Also they are chosen such that they have a uniform coverage in an audio file. Furthermore, the amplitude of the peak is also a criterion for peak selection. This basically means that a spectrogram is reduced to a set of coordinates as shown in Fig. 5.3, also called a constellation. These constellations form the basis for comparison of audio fragments.

Hashing

The second main concept in the Shazam approach is the use of fast combinatorial hashing. Comparing fragments based on their raw constellation points might be too

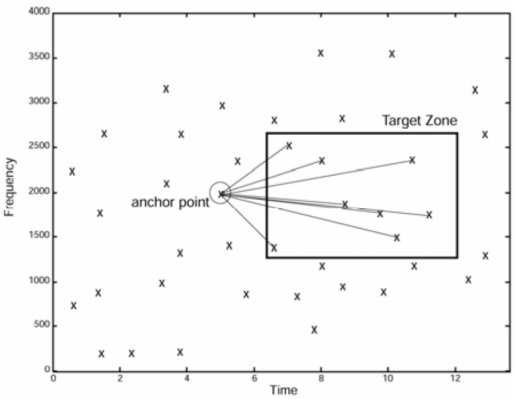


Figure 5.4.: Combinatorial hash generation [17]

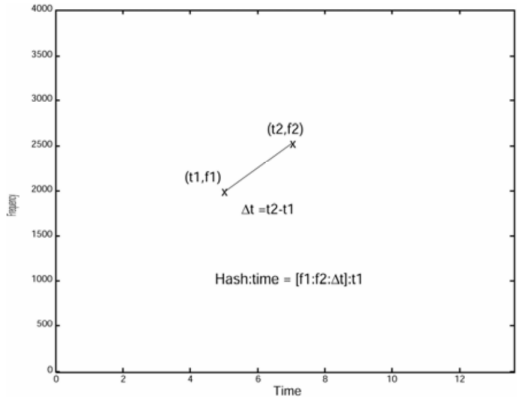


Figure 5.5.: Hash details [17]

slow, because constellation points have a low entropy. Therefore, fingerprints are created based on pairs of time-frequency points. This is done by selecting anchor points and associating them with points in its target zone. These pairs consist of two frequency points and the time difference between them. Also, each hash is associated with the time offset from the beginning of the respective file to its anchor point. This is illustrated in Fig. 5.4 and Fig. 5.5. This means that the number of hashes is the number of constellation points multiplied by the number of points in the target zone. Therefore the number of points is the target zone should be limited.

By using these pairs, searching is a lot faster than in the case of raw constellation points. However by introducing these pairs, the number of pairs to search for is also much greater. The trade-off made here introduces a speed improvement of 10000 times at the cost of an increase in storage space of about 10 times and a small loss in probability of signal detection. Depending on the application, the density of anchor points and the number of points in the target zone can be adjusted. For clean audio, this density can be low and the number of target points small.

Searching & scoring

Next is the searching and scoring. For an audio sample, multiple hashes are calculated in the manner described above. For each of these hashes, the database is searched. For each matching hash, the stored offset of this hash found in the database is then stored in a time pair together with the offset of the hash from the beginning of the sample. Next, these pairs are put into data bins according to the track ID associated with the matching database hash. These bins are then scanned for a matching audio song. The contents of a bin can be visualized as a scatterplot and a matching hit could be depicted as seen in Fig. 5.6. As can be seen here, a match results in a diagonal line, so the matching process is now reduced to detecting a diagonal line in this scatterplot. This can be done with several techniques, such as for example a Hough transform. For simplicity, assuming that the diagonal has

a straight slope (such as 1.0 for example), a histogram can be plot subtracting the values of the y-axis from the values on the x-axis. In the case of a hit, the histogram will have a clear peak in the case of a match, as can be seen in Fig. 5.7. The height of the peak is the score of match.

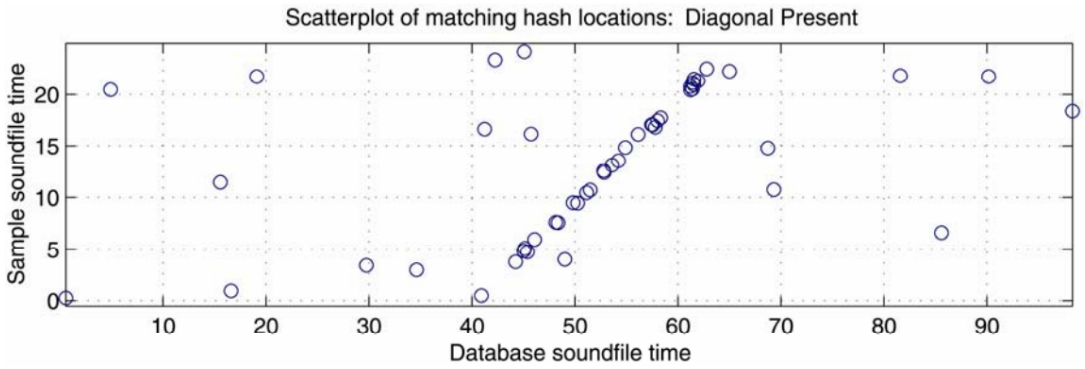


Figure 5.6.: Scatterplot of a match (time offset file vs. time offset track database) [17]

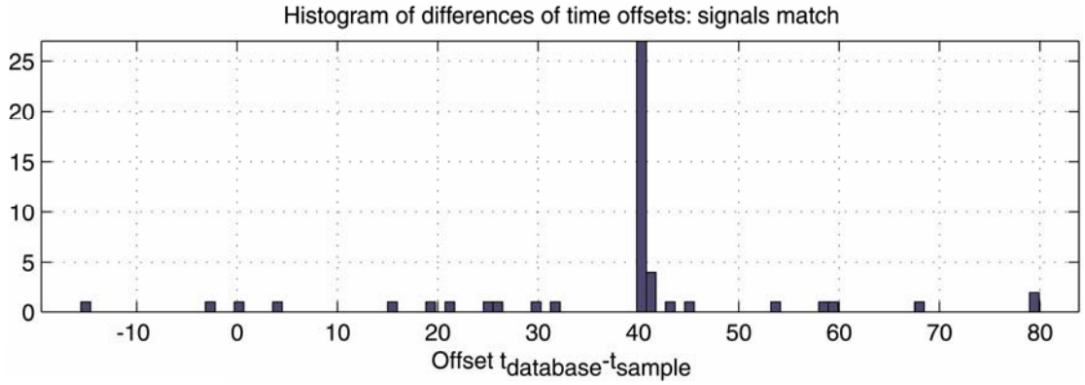


Figure 5.7.: Histogram of differences of time offsets of a match [17]

Notice that it does not matter where a sample starts and ends, since the database hashes are taken from samples uniformly in the audio fragment. Furthermore it can be remarked that the longer the sample is, the more matching pairs are found and the more dots in the scatterplot will be in a straight line. This in turn leads to a higher peak in the histogram and thus a better score. Also notice that introducing noise will reduce the number of hits and results in a lower histogram peak. It does not matter where the noise is located in the song, as long as there are enough remaining hash matches, the song can still be identified correctly.

5.3. Overview

The app makes use of a audio fingerprinting mechanism for recognizing content on a local television and submits this to a server for matching against a real-time fingerprint reference database of live fingerprinted TV content. Based on the position of this match and an accurate notion of elapsed time, a playout difference between the local television and the reference is created. The obtained measurements are then stored in a database. This process is illustrated in Fig. 5.8. The smartphone application acts as the center of communication between the components. The smartphone application records sound from a TV, queries the NTP server for a timestamp and starts creating a fingerprint right at this moment. Next, the created fingerprint is submitted to the reference server which compares the fingerprint with its database of live, continuously created fingerprints of its own TV source. If a match is found, the time offset (relative to the start of matching TV program) is returned to the smartphone. Next, the smartphone queries the EPG server to determine the absolute moment in time the fingerprint was recorded on the reference. The moment of local fingerprint recording and the moment of reference fingerprint recording are compared against each other to obtain the playout delay difference. Finally, combined with manually entered metadata, the playout delay difference measurement is submitted to a back-end database, storing the results.

There are several frameworks available for implementing audio fingerprinting technology which can be used as a robust basis for the system. Several fingerprinting technologies have been considered among which the open-source project Echoprint [19] and Audible Magic [20] but the Entourage [21] Software Development Kit from the company Gracenote [12] has been chosen to be used in the app. Test results of the Echoprint fingerprinting technologies can be found in Appendix B.

Furthermore, in figure Fig. 5.9, a time-sequence diagram illustrates the whole process of measuring a playout difference. This is the same process as in Fig. 5.8 but now with an added notion of time. This process consists of the following steps:

- The mobile device on which the app is running initiates the measurement.
- The first thing that the app does is obtaining accurate timing information from an external time-source such as NTP or GPS.
- Right after the moment an accurate time-stamp is obtained, the app starts with fingerprinting. This is done by calling the fingerprint engine which records the audio from the TV. Based on classification of this audio (such as silence, speech, music or noise), the fingerprint engine decides when exactly the fingerprint is made. Because of this, the moment the fingerprint engine is called and the actual moment the fingerprint is created is not exactly the same for consecutive attempts. However, this is automatically compensated for by the fingerprint engine.
- The created fingerprint is then sent by the fingerprint engine to the fingerprinting server. This fingerprinting server records live TV content from multiple

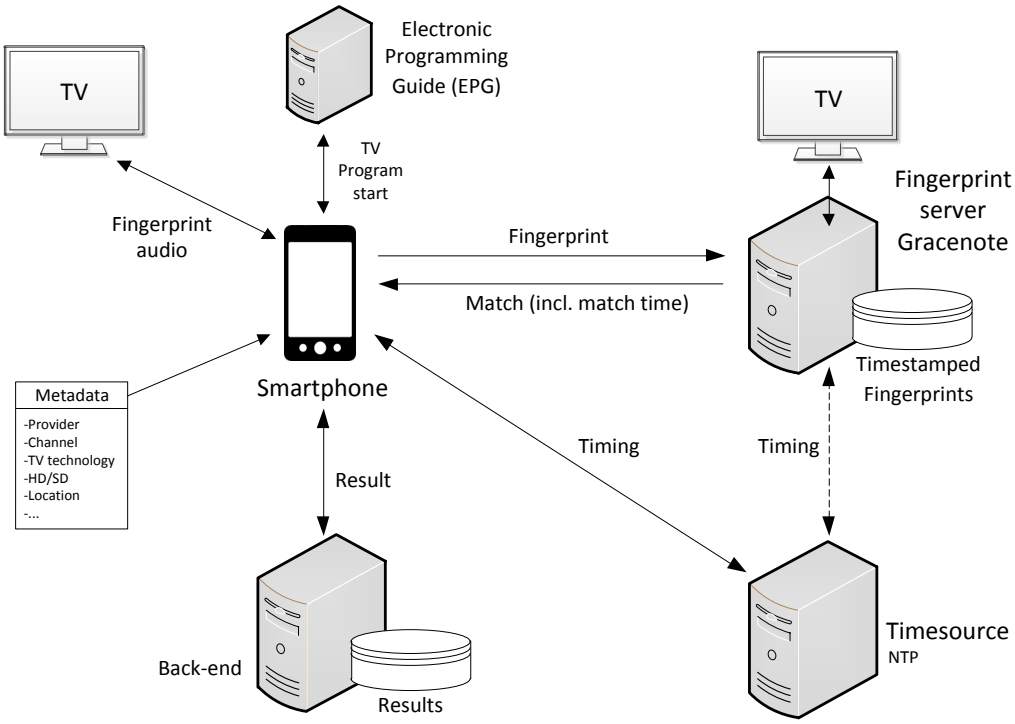


Figure 5.8.: Overview of the playout measurement system

channels and real-time fingerprints these. Also the *moment* this content was fingerprinted is saved here. To answer the fingerprint identification request, a response is sent if there is a match. If there was a match, additional information about the match is also sent. This is includes the channel, TV program and the moment the fingerprint (offset to the start of the matched program) was matched in the reference.

- In case the match was a valid match, the match details are saved. Since the offset of the match in the reference is only returned as a value relative to the start of the program aired at the moment of the match, the moment the current program was started also needs to be obtained to be able to compute the absolute time-stamp. To do this, an EPG server is queried for the time-stamp the currently aired program was started.
- On receiving these results, the playout difference is calculated between the local TV and the reference on the server by adding the measured offset in the program to the program start according to the EPG and subtracting the local reference time-stamp of the moment the fingerprinting process started from this value.

Apart from the actual playout value, additional information such as TV subscription

type, quality and possibly location are manually entered in the app. Next, for successful playout measurements the results are submitted to a database and can be used for analysis or comparison.

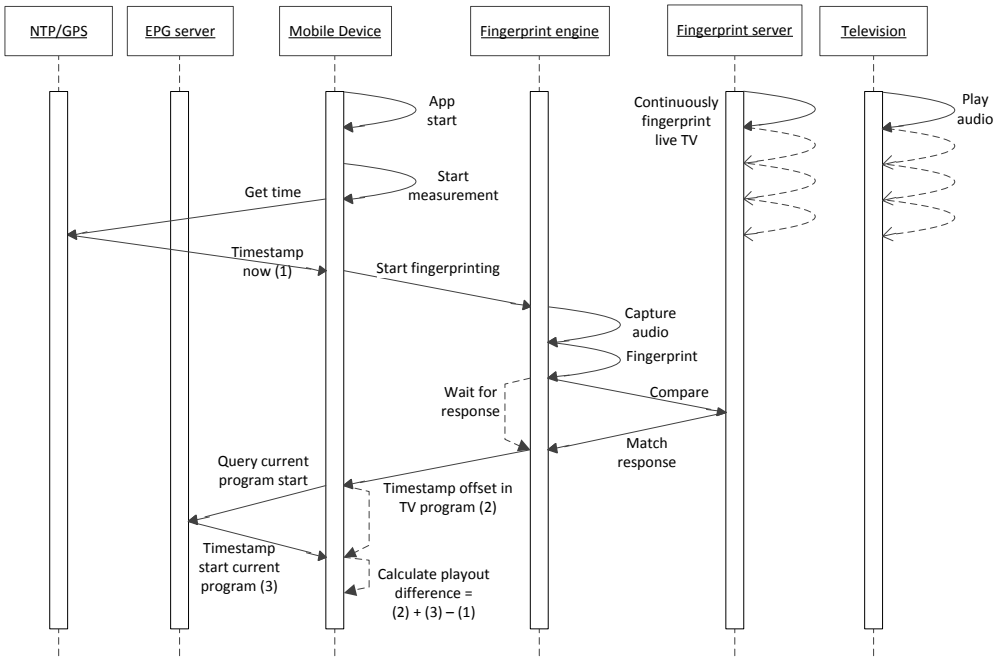


Figure 5.9.: Measuring playout difference.

Fig. 5.10 displays the most important actions that can be performed by an app user. Activities such as viewing a help or settings menu or showing more detailed error feedback are left out for simplification. As can be seen from this activity diagram, the process of obtaining a time-stamp is performed immediately before the process of fingerprinting. Furthermore it is possible to fingerprint and input metadata apart from each other. The activity to submit the measurement can be activated when the fingerprint and meta data input activities are completed. The remainder of this section elaborates further on several design choices made during the process of creating this measurement system. Apart from developing the app itself, a substantial part of creating the measurement system was spent by examining ways to obtain accurate time information on Android.

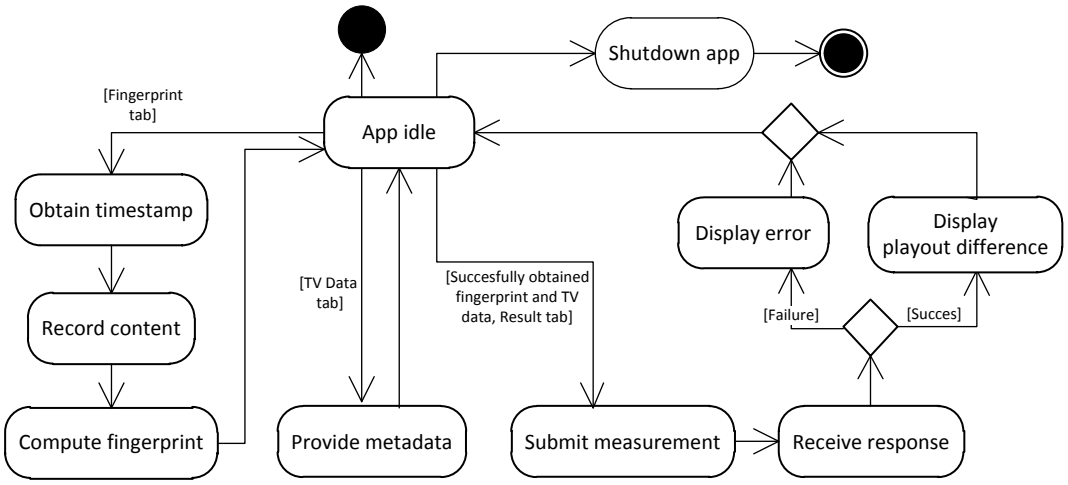


Figure 5.10.: Activity diagram of the client application.

5.4. Reference time-source

As mentioned in sec. 5.3, there is a need for an absolute time-source reference that can be used to record the exact moment a fingerprint is created. The clock inside the Android system cannot be assumed to be accurate enough and is also adjustable by users. Also, there could be timezone differences between different users. So this cannot be used as a reliable source of time. It would be possible to use GPS or NTP as a time-source. Most Android devices have a GPS chip built-in and they also have an internet connection. Other ways to obtain timing information are from the telephone network, using protocols such as Secure User Plane Location (SUPL) or Network Identity and Time Zone (NITZ) protocol), but these do not seem to be very accurate. So GPS or NTP are the most prominent options and are examined in the remainder of this chapter.

5.4.1. GPS as time-source

The GPS (Global Positioning System) is a system used for navigation on earth. It makes use of several satellites orbiting around the earth which send signals to GPS receivers on earth. Each satellite has an atomic clock which is set by the North American Aerospace Defense Command (NORAD) [22] several times a day to ensure a very high level of accuracy. The radio signals that these satellite are constantly sending contain accurate time and satellite position information. Combining the current time on a GPS receiver, the time the signal was sent, and the value of the speed of light, the distance between a satellite and the GPS receiver is calculated. Doing this for multiple GPS satellites, it is possible to calculate the position of the

GPS receiver by using trigonometry. There are other systems which also have the same purpose as GPS. This includes the Russian GLONASS system as the largest counterpart and the unfinished European Galilei system. Since the clock the GPS satellites have on-board is so accurate and so often adjusted to maintain an accurate time, GPS technology can provide a very accurate time-stamp. It is so accurate that an accuracy in the order of tens of nanoseconds can be achieved[23]. Since it can provide such an accurate time-stamp and it is widely available on Android systems, this would be an ideal option to use as time-source for the measurement system.

GPS fix required for accurate timing

One disadvantage of obtaining time from GPS is that a so-called fix is needed before it is possible to receive a time in this order of accuracy. A GPS fix is a status obtained by the GPS device after the moment the position is calculated. To obtain a fix, information from multiple satellites has to be obtained by the receiver or data has to be obtained from other sources, such as internal memory.

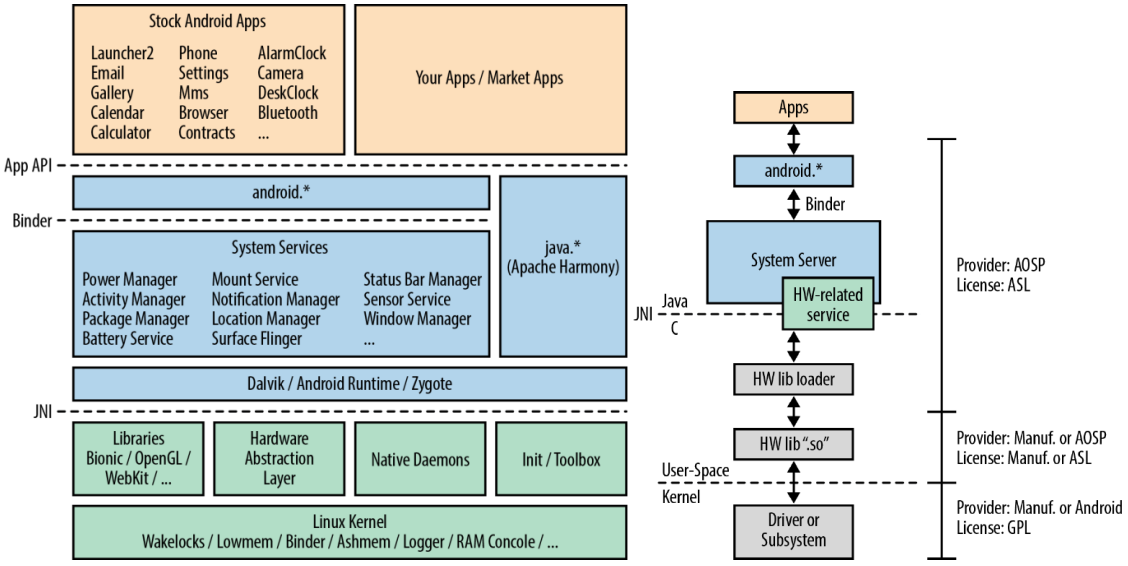
Obtaining a GPS fix is indoors not always possible due to a lack of sight to the GPS satellites. It would be possible to extract the time from the GPS signal without obtaining a fix first, but this would be less accurate. One way, to accomplish this on an Android smartphone would be by extracting the GPS timing information at the NMEA data communication specification[24] outputted by GPS receivers (more specifically, the $GPRMC or $GPZDA string). In this message, the time-stamp representing the moment of sending by the satellite can be extracted. This message does not compensate for the distance between the GPS receiver and the GPS satellite however, which is not known until a fix is obtained. Since GPS satellites are located in a medium earth orbit at 20200 km above the earths equator and a GPS message is transmitted at the speed of light ($c = 3.0. \cdot 10^8 m/s$), this would mean that it would take $\frac{20200000}{c} \cdot 1000 = 67$ms seconds to travel from satellite to the receiver in the most positive case. This can also be a larger amount, in case the receiver is located further away from the equator and the GPS satellite is not exactly orthogonal above the receiver, which is more likely. So without a fix, the accuracy of the timing signal could be more than 67 ms off when only using at the time-stamp message sent from the satellite without any additional information. So the problem of having to wait for a GPS fix could be avoided at the cost of an accuracy loss.

5.4.2. GPS architecture in Android

To understand how GPS timestamps in Android are obtained, the underlying architecture of GPS on Android is briefly examined here. The GPS receiver hardware on an Android device is the device that obtains the GPS signals from a GPS satellite. It

is a piece of hardware that is not the same for every device. Different Android devices have different hardware equipment. To support these different types of hardware, Android uses an approach that is significantly different from the classic approach in Linux kernels. There are abstractions built on the hardware support in the form of a Hardware Abstraction Layer (HAL). The Android stack typically relies on shared libraries provided by manufacturers to interact with hardware. Because of this, the interface, behavior and function of abstracted hardware components differ greatly between Android devices [25].

The architectural overview used by Android is displayed in 5.11a. Here, a System Service called Location Manager which includes GPS functionality can be seen. To access hardware such as the GPS receiver, Android still ultimately relies on the kernel. This is done either through shared libraries that are either implemented by the device manufacturer or provided as part of the AOSP (Android Open-Source Project). The HAL layer can be considered as the hardware library loader. Android does not specify how a shared library and the driver or kernel subsystem should interact. Only the API provided by the shared library to the upper layers is specified by the HAL [25]. The interactions involving the HAL are displayed in 5.11b. What clearly can be seen here is that there are parts that are manufacturer specific, which can include the GPS hardware drivers.



(a) Architecture **(b)** Hardware Abstraction Layer (HAL)

Figure 5.11.: Android and HAL Architecture [25]

App developing for Android is usually done with Java language. To use the GPS receiver on Android, java Location and LocationManager objects are needed. This corresponds with the LocationManager system service displayed in 5.11a. This is done as follows:

```
1 Context context;
2 ... //assign context
3 LocationManager locationManager = (LocationManager) context.
    getSystemService(LOCATION_SERVICE);
4 Location location = locationManager.getLastKnownLocation(
    LocationManager.GPS_PROVIDER)
```

By calling the method getSystemService(String name), a handle to the LocationManager system-level service is obtained, which is uniquely identified by the String constant LOCATION_SERVICE. This location manager provides acces to the GPS functionality of the Android device. This LocationManager automatically communicates with the HAL. The HAL automatically starts the implemented shared library (either implemented by the manufacturer or from the AOSP). This shared library allows the drivers in the Linux kernel to acces the GPS receiver chip. How the GPS receiver can be used at a software level is examined in the next section, sec. 5.4.3.

5.4.3. Obtaining GPS time in Android

To use GPS timing in Android, three main API functions are used. The android.os.SystemClock.elapsedRealtimeNanos(), Location.getTime() and Location. getElapsedRealtimeNanos(). Note that this last method requires Android API level 17 (Android 4.2) or higher. There is no alternative for lower Android versions that provides the same functionality as this method. Here, each of these functions is described:

5.4.3.1. Internal Android clock (elapsedRealtimeNanos)

To obtain the GPS time, an internal clock must be polled to compare the moment the time-stamp was received with the moment the time-stamp is needed. This is what the android.os.SystemClock. elapsedRealtimeNanos() method does.

This method queries the internal clock of Android and returns its value in nanoseconds. This clock does not return an absolute time-stamp value but a value relative to the boot of the device. This method is described in the API as *"Returns nanoseconds since boot, including time spent in sleep"* [26]. According to the API, this method is guaranteed to be monotonic [26]:

"elapsedRealtime() and elapsedRealtimeNanos() return the time since the system was booted, and include deep sleep. This clock is guaranteed to be monotonic, and continues to tick even when the CPU is in power saving modes, so is the recommend basis for general purpose interval timing."

In sec. 6.3, this clock is tested for monotonicity. Furthermore, Android offers two more clocks, namely System.currentTimeMillis() and uptimeMillis(). These clocks are not suitable for usage in the app. They are not accurate and they can be set by a user or can be stopped when the device is in sleep mode[26].

5.4.3.2. GPS fix time-stamp (getTime)

To obtain the value of the GPS time-stamp that has been received by the GPS receiver, the Location.getTime() method is used. This method obtains the time from a Location object. It returns its value as a UNIX time-stamp. This method is described in the API as *"Return the UTC time of this fix, in milliseconds since January 1, 1970".* [26]

Furthermore, the API also guarantees that this value returns a valid UTC time: *"All locations generated by the LocationManager are guaranteed to have a valid UTC time, however remember that the system time may have changed since the location was generated."*

This sentence could be read in an ambiguous way, with "valid UTC time" could indicate a valid UTC time *format* or a valid UTC time *representation*. Also, there are some reports on the internet of this value returning the time from the device itself on some smartphones. To rule out that it does the latter (on the specific test device), the following was done. First the system time was changed by a large amount (hours) to a an incorrect time. Next, airplane was enabled on the telephone. This mode mode disables Wifi and the GSM network on the telephone. After this, the telephone was rebooted to make sure there is no GSM network or Wifi time information available that that could possibly provide timing information. Next, an app was started that calls this method. Once a fix was obtained, this fix was compared with the Android device clock as well as System.currentTimeMillis(). Doing this showed that even if the Android device (and the currentTimeMillis) clock is way off, this method actually provides a correct representation of the current time. So the value obtained from this method is unrelated to the local time on the device on which this method is called, i.e. it must be from an external source.

Further investigation using the source code of Android (API 17) shows the following code snippet in Location.java:

```
1  private long mTime = 0;
2
3  public void getTime() {
4    return mTime;
5  }
6
7  public void setTime(long time) {
8    mTime = time;
9  }
10
11 public static final Parcelable.Creator<Location> CREATOR =
12   new Parcelable.Creator<Location>() {
13
14   @Override
15   public Location createFromParcel(Parcel in) {
```

```
16      String provider = in.readString();
17      Location l = new Location(provider);
18      l.mTime = in.readLong();
19      ...
20      return l;
21    }
22
23      ...
24  };
```

So the Location.getTime() method returns the mTime variable. Apart from a helper method (for preventing mTime to have a value of 0), a reset method and a set method based on a Location parameter, the private variable mTime does not get set anywhere else in this class. Neither can it be assigned outside this class, as it is a private variable. This means that setTime or the createFromParcel method must be the way how the GPS module of the smartphone is setting this value. The first method, setTime does not have calls anywhere else in the Android source code. This means that the way the Location class is obtaining its value of mTime (and thus getTime()) is through the createFromParcel(Parcel in) method. A Parcel is a *"container for a message (data and object references) that can be sent through an IBinder"* [27]. The IBinder is in turn part of the remote procedure (IPC) that connects the API with the system services, as can be seen in 5.11a.

5.4.3.3. Time since GPS fix (getElapsedRealtimeNanos)

The getTime method provides a time-stamp from a GPS satellite. However, this is not the GPS time at the moment getTime is called. It is the time on the exact moment the time-stamp has been sent by the satellite. To compensate for this, and obtain the actual time at an arbitrary moment, the Location.getElapsedRealtimeNanos() method can be called. This returns the difference between the moment the method is called and the moment the time-stamp has been sent by the satellite (getTime). Adding this to the value obtained from getTime represents the GPS time at the exact moment getElapsedRealtime (or getElapsedRealtimeNanos) is called.

This method is described in the API as *"Return the time of this fix, in elapsed real-time since system boot"* [26]. So it does not actually return nanoseconds that have passed since the fix, but it expresses this as nanoseconds since system boot. So the actual GPS time at an arbitrary moment is obtained as follows:

```
1 Location loc;
2 ... //initalize loc and wait for a GPS fix
3
4 long gpsFix = loc.getTime();
5 long elapsedNow = Android.os.SystemClock.elapsedRealtimeNanos
    ();
```

```
6 long gpsFixAge = (elapsedNow - loc.getElapsedRealtimeNanos())
    /1000000;  //1 ms = 1000000 ns
7 long gpsTime = gpsFix + gpsFixAge;  //current GPS based UTC
    time in ms since January 1, 1970
```

Furthermore, according to the API description, this value is also guaranteed to have a valid elapsed real-time and is therefore comparable with elapsedRealTimeNanos():

"This value can be reliably compared to elapsedRealtimeNanos(), to calculate the age of a fix and to compare Location fixes. This is reliable because elapsed real-time is guaranteed monotonic for each system boot and continues to increment even when the system is in deep sleep (unlike getTime().

All locations generated by the LocationManager are guaranteed to have a valid elapsed real-time."

5.4.4. NTP as time-source

Instead of using time information from GPS, using the Network Time Protocol (NTP)[28] is another option. This is a protocol that provides timekeeping functionalities over computer networks. The current version is NTPv4 [29]. It makes use of a hierarchy of servers synchronizing their time based on other servers located higher in the hierarchy. The top of the hierarchy obtains its time from very accurate clocks such as atomic clocks. To synchronize their timing, NTP servers use a mechanism to continuously adjust their internal clock rate. In ideal situations, NTP servers can have an accuracy of tens of microseconds [29] but in worse conditions, the error can become more than 100ms [30].

The hierarchical architecture of NTP is defined in so called "stratums". A stratum is an indication for the accuracy of an NTP server. A stratum 1 is the most accurate stratum. Stratum 1 servers obtain their time directly (within a few microseconds) from stratum 0 time-sources, which are very accurate time sources such an atomic clock or a GPS receiver. Stratum 2 NTP servers obtain their time from stratum 1 NTP and can also peer with other stratum 2 servers for more robust timekeeping. In turn, stratum 3 servers obtain their time from stratum 2 servers. Lower level stratum obtain their time from 1 stratum level above theirs, until the maximum defined stratum level 15. In practice however, stratum 2 servers seem to be the most common. This architecture is displayed in Fig. 5.12.

Something that makes NTP servers very accurate is that they not only obtain a time-stamp from another NTP server once in a while to set their local clock, but that also their internal clock *rate* is adjusted based on received timestamps over time. Logically, to adjust their internal clock rate over time, a stabilization period is needed. This stabilization period can span multiple hours before it becomes accurate.

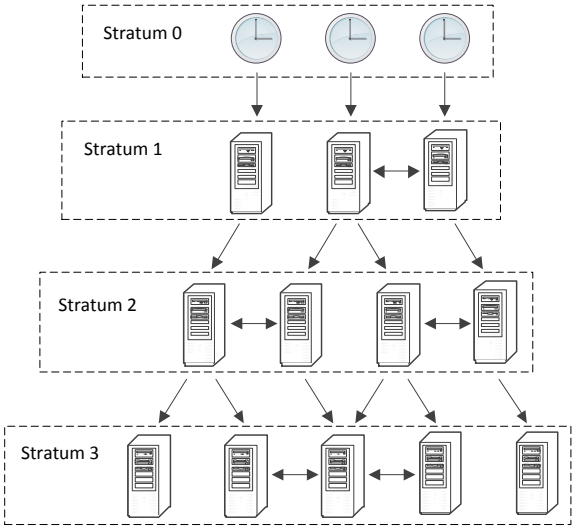


Figure 5.12.: NTP Stratums

5.4.4.1. SNTP

On an Android device however, it would be cumbersome to run a complete NTP server to obtain time from multiple sources and stabilize it over multiple hours. Instead, it would be possible to use the SNTP protocol. This is a less complex implementation of NTP without requiring these statekeeping information. Because of this, it is less accurate than normal NTP. The accuracy depends on the distance and path the time-stamp has to travel between the server and client. Of course, this is upper-bounded by the Round-Trip-Time (RTT) between the requester of the time-stamp and the server providing it. This means that when the RTT between the client and server is low, the inaccuracy introduced by the network is bounded by this amount. Possible inaccuracy of the server is still a factor of course. So the inaccuracy of an obtained SNTP time-stamp is the combined accuracy of the NTP server providing the time-stamp combined with a value that is at most the RTT between the requester and the server.

5.4.5. NTP implementation in Android

The measurement app can use the Simple Network Time Protocol (SNTP) to obtain an accurate absolute time-stamp. As opposed to NTP, SNTP does not have the possibility to evaluate the accuracy of multiple time-sources. For the measurement system, a controlled NTP server is used. This means that the accuracy of the NTP server is known and the SNTP accuracy is then almost exclusively dependent on the round-trip-time (RTT) of the SNTP time-stamp request. So the accuracy of an

SNTP request in the app with a very low rtt will be close (enough) to the accuracy of the NTP server.

To obtain an NTP time-stamp using SNTP with a low RTT, the app measures the RTT of the request. If this request has an RTT that is above a certain threshold, the app waits a moment and sends another request until a time-stamp is obtained with an RTT above the threshold. If, after a certain amount of requests no time-stamp with an RTT below the threshold is found, the threshold level is increased. This way, an as accurate as possible time-stamp is obtained without spending a long time to obtain it. If the amount of requests exceeds a maximum retry threshold, no time-stamp and thus no measurement is obtained. Furthermore, as a measure for the accuracy of individual measurements, the RTT of the time-stamp request is individually stored in the database for each measurement.

5.4.6. NTP server

To use NTP as time reference for the measurements, an accurate NTP server is needed. This server needs to return consistent timestamps as close as possible to the actual time. One way to obtain NTP timestamps would be to use NTP servers that are part of the publicly available cluster of NTP server of the NTP pool project [31]. A disadvantage of this method would be however that no control over the accuracy of the provided timing information can be exercised. This is a real problem, because experience shows that picking a random NTP server from the NTP pool project can provide a time that is hundreds of milliseconds off from other NTP servers in the same pool. Because of this inaccuracy, this would require a single server from the project to be used, instead of of using an address that references to a random server in this pool. This would not be conform with the philosophy of the NTP pool project which tries to load balance requests over different NTP servers.

So these two reasons lead to setting up a controlled NTP server which obtains its time from a GPS device. This GPS device is considered a stratum-0 time source, and by using an accurate Pulse Per Second (PPS) [32] signal to connect this GPS to the NTP server, a stratum-1 server was created. This signal is extremely accurate and can have an accuracy ranging from values in order of picoseconds to microseconds per second [33]. More detail on the performance of the NTP server can be found in the chapter regarding the performance of the measurement system, chapter 6.

5.5. Implementation of the app

5.5.1. Class structure

The following list provides a short overview of the classes and their functionality.

- **Fingerprinter**. Handles the process of fingerprinting and handles callbacks from the Audio Content Recognition (ACR) subsystem and the fingerprint server.
- **GPS**. Class handling GPS functionality in the Android device.
- **SntpClient.** Implements SNTP functionality.
- **MeasurePlayoutDiff**. Main class responsible for combining, handling and controlling EPG, Fingerprinter, GPS and all Fragment classes. PrefsActivity, HelpActivity. It takes care of creating the GUI and handling GUI events. It also has logging functionality and handles communication with the back end server for storing results. Has private classes for handling JSON and XML communication.
- **FingerprintFragment, ResultFragment, TimeSyncFragment, TVDataFragment**. Fragments corresponding with the several GUI tabs.
- **EPG**. Constructs messages in the right format to communicate with the Electronic Programming Guide (EPG) server. Note that the actual sending of messages is done by MeasurePlayOutDiff.
- **GnCircularBuffer**. Helper class for Fingerprinter.
- **PrefsActivity**. Activity for displaying and handling preferences and saved values of the settings menu.
- **HelpActivity**. Activity for displaying and handling the help menu.
- **ExpandableListAdapter**. Helper class for HelpActivity.

Furthermore, the app makes use of the ActionBarSherlock support library [34]. This allows the app to use some GUI related functionality unavailable on lower Android versions, such swiping between several tabs. The main functionality of the app is divided over these tabs. The first tab provides some information about the app and app usage. The next tab allows for manually entering TV and TV subscription related data such as TV channel, channel quality, TV provider and TV subscription. The fingerprint tab allows for obtaining a time-stamp and audio fingerprinting TV content using the microphone on the device. The results tab allows for querying the EPG for EPG data, combining this with the previous results and submitting this to the back-end server for storing the results. Also in the upper right corner there is a menu that allows for changing some app settings (developer related) or viewing the help menu for a more detailed explanation of how to use the app.

5.6. Screenshots

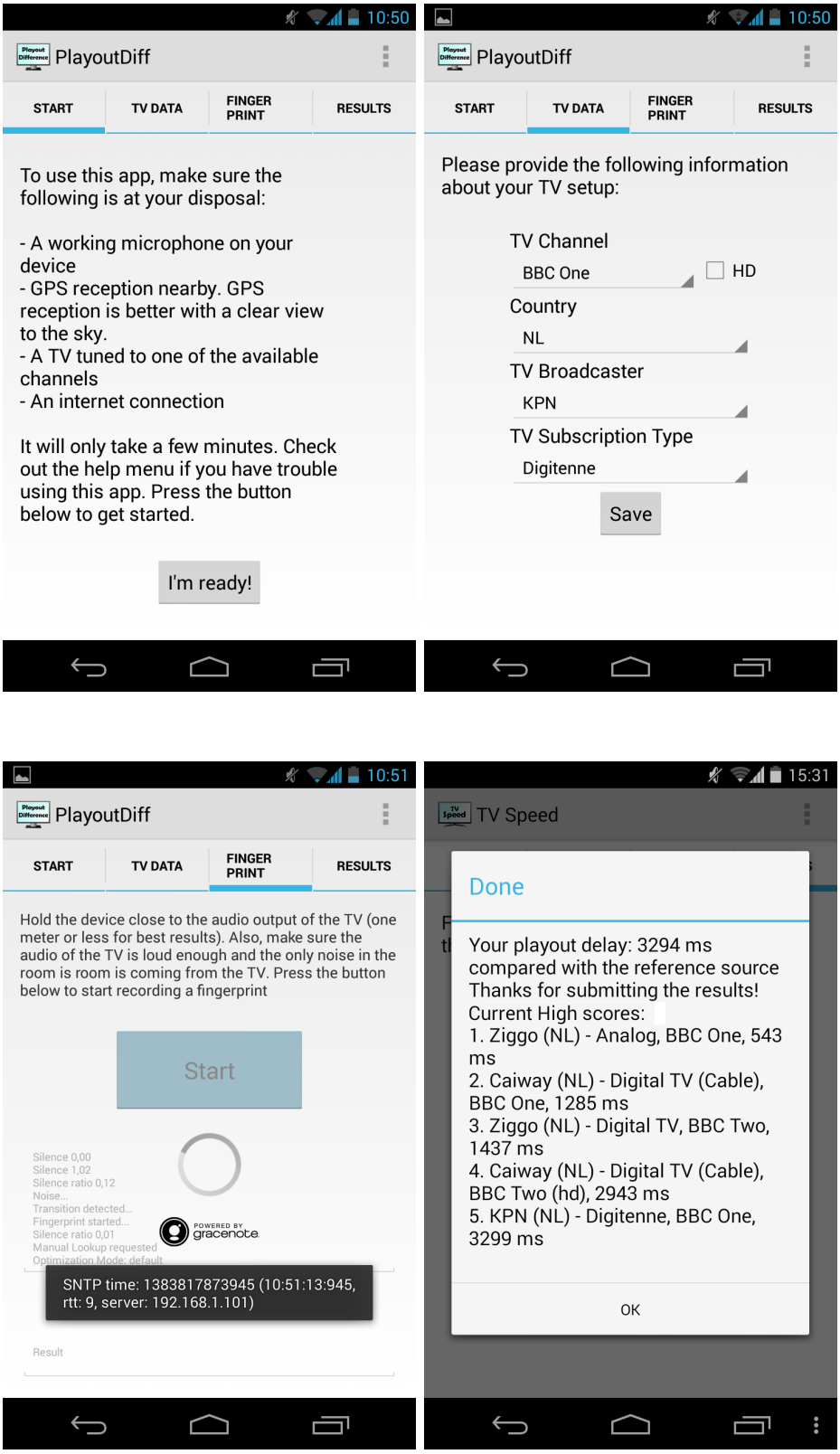


Figure 5.13.: Screenshots of the app

6. Performance analysis of the playout measurement system

6.1. Introduction

This chapter examines how precise and accurate the measurement system is. Tests are performed on the whole system, but also on parts of the system. To test the performance of the measurement system, a very accurate clock is needed. Without an accurate clock reference it would not be possible to measure time differences accuracy. To overcome this problem, an accurate NTP server was set up. Using this NTP server in optimal conditions, it is possible to obtain accurate timing information on an Android device that is at most a few milliseconds less accurate than the NTP server. Using this as an accurate timing reference on the Android device, it will in turn be possible to measure the accuracy of the internal clock on Android. Knowing how accurate the internal clock is on Android will give insight on the accuracy of the rest of system. In the same way as NTP is tested against the internal clock of Android, GPS will also be compared with the internal clock of Android. This should give an idea of the accuracy of GPS on Android.

Furthermore, the accuracy of timestamps from the fingerprint system will be tested. Finally, based on the accuracy results tests of NTP compared with GPS on Android, the most accurate of these two will be used as a time-source in the measurement app to perform the tests that measure the performance of the system as whole. The system as a whole will be tested both for precision and accuracy.

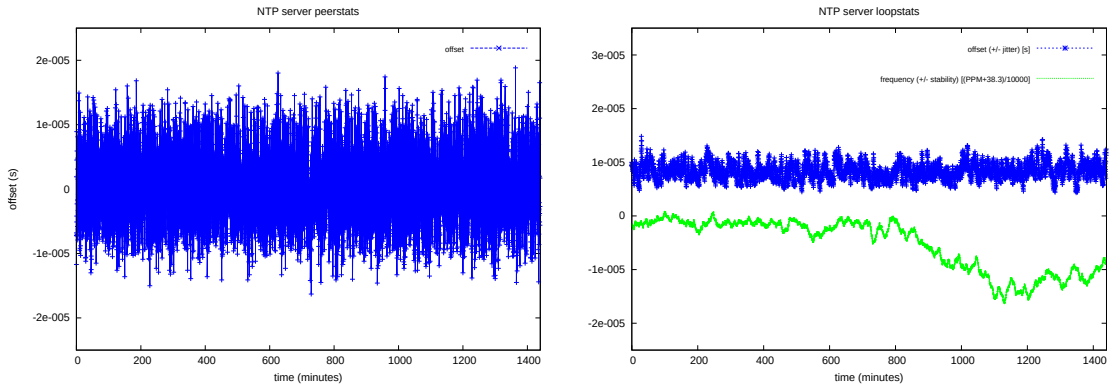
6.2. NTP Server

This subsection reviews the accuracy of the NTP server. Not only can this server be used for testing the accuracy of the app itself, but it can also function as time source for measurements using the app. This NTP server has its own GPS hardware receiver on which the NTP server synchronizes its time using the PPS signal. To see how accurate this NTP server is, the NTP daemon can be monitored by using the ntpq -p command, which shows the performance of the connected peers. A screenshot of this command executed is given in Fig. 6.1. This shows the jitter between the NTP server and its time sources. It also shows that it uses the PPS signal to synchronize its clock to.

```
hans@gps-ntp-server:~$ ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
-85.12.35.12     193.79.237.14    2 u   59   64  377    2.871    1.201   0.033
-ns3.proserve.nl 193.79.237.14    2 u   56   64  377    3.145    0.411   0.032
+ns1.proserve.nl 193.67.79.202    2 u   53   64  377    4.186   -0.079   0.618
+panoramix.linoc 193.79.237.14    2 u   21   64  377    5.614   -0.013   0.132
xSHM(0)          .GPS_.           0 l    3   16  377    0.000  -90.010  29.920
*SHM(1)          .PPS.            0 l    2   16  377    0.000    0.018   0.004
hans@gps-ntp-server:~$
```

Figure 6.1.: NTP Daemon performance. The values of delay, offset and jitter are in milliseconds.

This is a snapshot of a single moment however. For a better performance measurement the server can also log performance statistics. This was done and these results are plotted in Fig. 6.2. The peerstats offset estimation shows how disciplined the internal clock is compared with the PPS clock. The system logs the offset for each system clock update [35]. 6.2a shows that the NTP server is estimated to be within an accuracy of 20 microseconds of the PPS signal. This means that the time that the NTP server is keeping the system clock accurately synchronized with the PPS signal, which was identified as an accurate time-source in sec. 5.4.6. Furthermore, 6.2b shows the loopstats statistics. This shows the stability of clock frequency of the NTP server. the frequency of the clock adjusts itself over time, the reason that it has to be adjusted is that it has to compensate for changes in temperature and clock imperfections. The frequency (+/- stability) shows the wandering over time, not an absolute frequency value. The value of 38.3 Parts Per Million (PPM) added to the frequency plot was added to obtain a value around the y=0 axis. This allows for plotting it in one figure combined with the offset. This fixed offset value compensates for the delay between the moment the GPS signal was received and when it is actually processed, as calculated by the NTP protocol.



(a) Offset estimation of the NTP server with the PPS signal.

(b) Estimated offset and frequency error.

Figure 6.2.: Accuracy of the NTP server

6.3. Android internal clock (elapsedRealtime)

Now that there is a sufficiently accurate time reference, the next step is to test the accuracy of the internal clock in Android. The most reliable and accurate clock that can be used in Android is the internal clock that keeps track of the amount of time passed since the device was booted. The value of this clock can be obtained in milliseconds (android.os.SystemClock.elapsedRealtime()) or in nanoseconds (android.os.SystemClock.elapsedRealtimeNanos()) using the API.

The test setup used to test this internal clock is displayed in Fig. 6.3. In this test, the smartphone is polling the NTP server periodically for its NTP time. The smartphone used here is a LG Nexus 4, which is running a pure version of Android without any additional manufacturer specific soft- or firmware installed. To prevent congestion from having much impact on the timestamp requests, the NTP server and smartphone were connected through a wireless router in a local area network. No other devices had access to this LAN during the tests, so there was no other traffic reducing the performance. Also a wireless channel with as little interference as possible was chosen (using a Wifi analyzer app for Android). This did not exclude other wireless signals from interfering with the test setup, but looking at the RTTs from the NTP timestamps, this did not seem to be a problem. The RTT was in almost all the requests a few milliseconds, with the exception of two RTTs of 145 and 251ms. In Fig. 6.4 the test results can be found. This graph shows the difference in clock values of both clocks using a chosen starting point. Each data point corresponds with an SNTP request to the local NTP server. For each SNTP request, the obtained SNTP time and the Android clock time since boot were recorded. For both these values the difference with the first measurement is calculated. These differences are the values of the red dots in Fig. 6.4. If the clocks on both devices were completely identical and without any drift, differences between these clocks would be zero and the plotted values should overlap the y=0 axis (the red line should overlap the green line).

Based on these results, there seems to be a monotonic, linear drift between the SNTP time and the Android clock. Based on this, the drift of the Android clock can be calculated. Namely, this value would be the slope of line that is formed. Calculating this gives a slope of $84.23\mu s/s$. So the elapsedRealtime clock in Android on this specific test device has a drift of $84.23\mu s$ drift per second; it is running $84.23\mu s$ per second too fast. Extrapolating this to a 24 hours would mean that the elapsedRealtime clock would be 7.28 seconds off. However other factors such as heat might influence this drift, so more measurements over time would be required to say anything about this drift over a longer period. Since the app only makes use of this value for a short amount of time (in the order of seconds up to around 3 minutes at most), this internal clock is accurate enough for these purposes. In 3 minutes it would only be 15 microseconds off. The value measured here is only applicable for the exact test device, but it would seem logical to assume that this value would be an indication for the accuracy of the internal clock of medium to

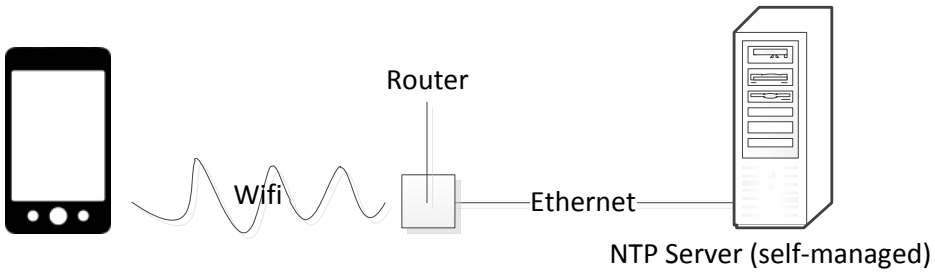


Figure 6.3.: Android time since boot clock accuracy test setup.

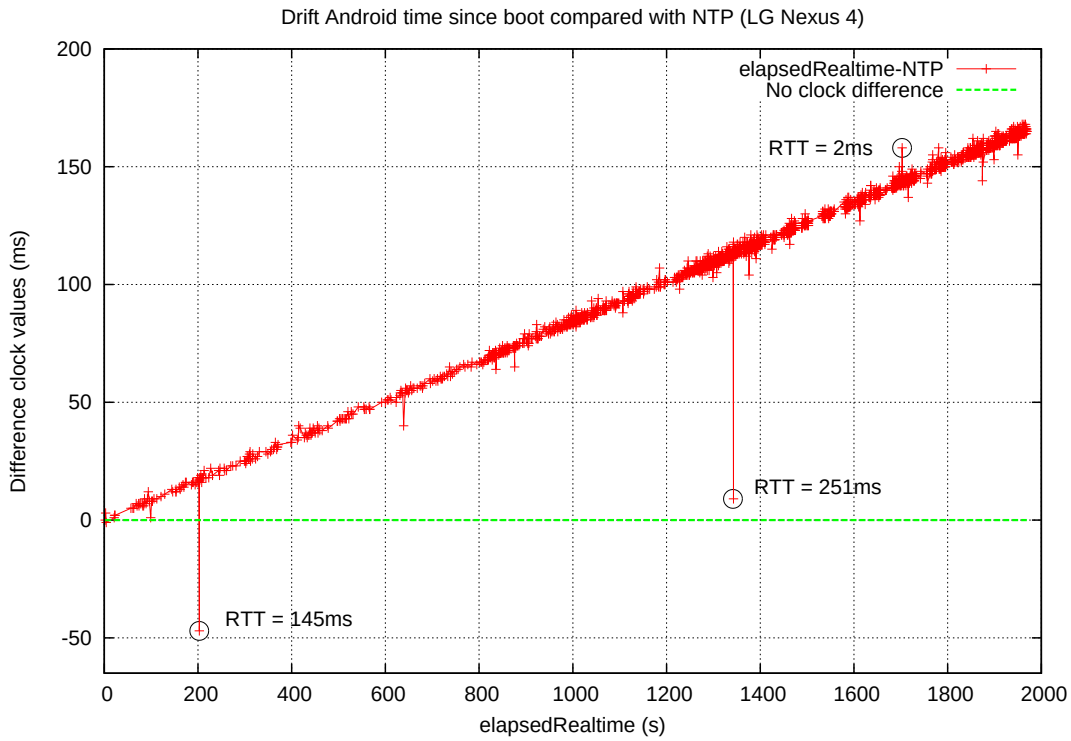


Figure 6.4.: Android time since boot (elapsedRealtime) clock drift on a LG Nexus 4.

high-end smartphones. This drift will probably differ per device, but on the other hand there are tolerance boundaries in the manufacturing process that would limit this value to be within certain acceptable boundaries.

6.4. GPS time in Android

Now that the internal clock of Android is measured to be rather accurate and monotonous, the GPS time on Android can be compared with this value. If the GPS time on Android is accurate within milliseconds or less, it would overlap the internal clock of Android within a few milliseconds. This is tested in this subsection.

6.4.1. Testing accuracy

To see how accurate the GPS time is, an Android app test was written that logs values obtained from these methods. Over time, several GPS fixes are obtained and then these methods are called right after each other. The GPS time is then calculated and compared with Androids internal clock (elapsedRealtimeNanos()). In sec. 6.3 this method is shown to be accurate enough to serve as a reference time source. Only GPS times from unique GPS fixes are used in this figure.

In Fig. 6.5, the results of this test can be found. This plots the difference between the internal clock of Android (see sec. 5.4.3) and the GPS time based on the first measured elapsedRealtime and GPS time tuple value (start of the measurement). In sec. 6.3, it was shown that the internal clock of Android (elapsedRealtime) is indeed monotonic as was described in the API. So the fluctuations in the graph do not indicate that the internal clock of Android fluctuates, but that the GPS time values in Android do. This is surprising because GPS technology can be much more precise than this. The GPS on Android provides time values within an accuracy that might be at least half the difference between the highest and the lowest measured value off from the actual time. So GPS on Android can provide a timestamp that is at least $\frac{156+954}{2} = 555ms$ up to possibly double this amount off from the actual time on the test device (an LG Nexus 4).

The next code snippet provides the code used to measure the values displayed in Fig. 6.5:

```
1 Location gps;
2 ...
3 //measure the values right after each other
4 long now = android.os.SystemClock.elapsedRealtime();  //value
      in milliseconds
5 long last_gps_fix = gps.getElapsedRealtimeNanos();  //value
     in nanoseconds
```

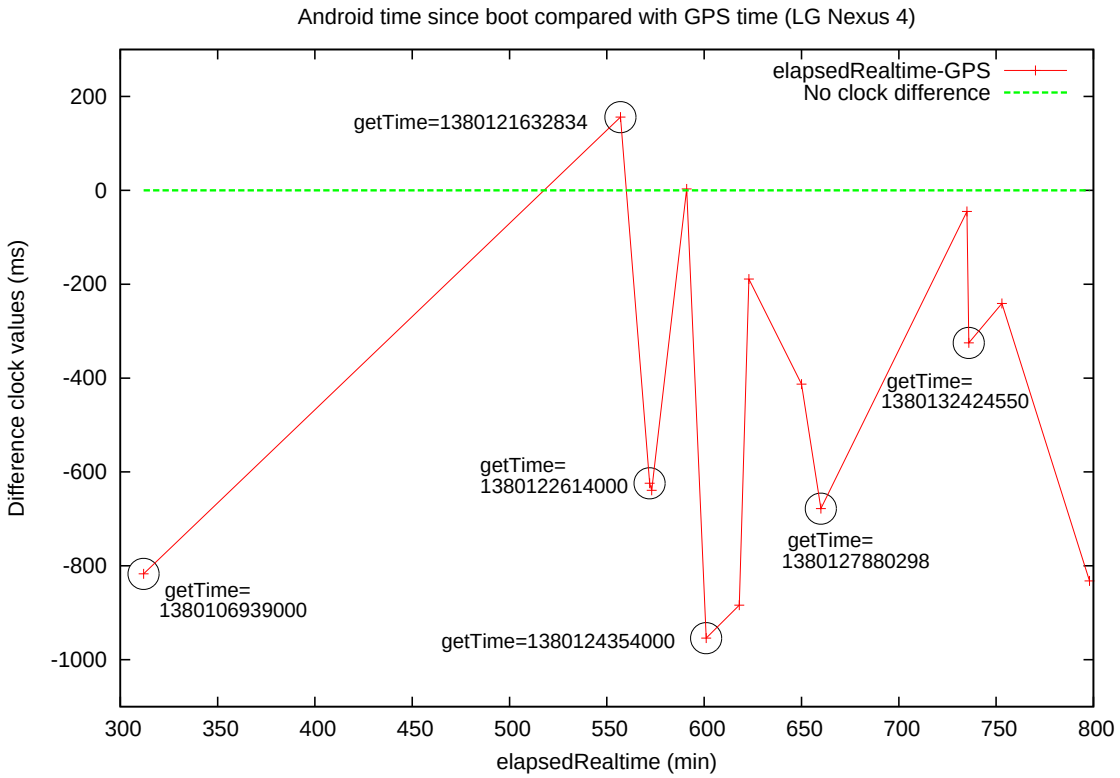


Figure 6.5.: Error in GPS time on Android and elapsedRealtime

```
6  long age_gps_fix = now - (last_gps_fix / 1000000);  //1ms =
       1000000ns
7  long gps_timestamp = gps.getTime();
8  ...
9  long gps_time = age_gps_fix + gps_timestamp;
10 ...
11 //log the obtained values
```

Two types of timestamps Another interesting result from this test is that the values of the getTime() method used to construct the GPS value seem to be rounded. These values are very often ending in three zeros. Some of these values are displayed in Fig. 6.5 next to their corresponding measured points. 9 out of the 15 measured GPS fixes (used for 14 difference measurements) have a rounded value of getTime. Assuming that these values are more or less random and can take on each value between 000 and 999 with an equal chance, the chance that 9 out of 15 measurements end with 000 is 0.06% so this is probably no coincidence. It could be that some satellites send the message part containing the GPS timing information exactly at complete seconds, while others do not. However this would not explain the variance in Fig. 6.5. There are several factors that might cause this, and it might even be GPS receiver hardware dependent. A reason might be that the GPS receivers uses

the GSM or Wifi network to obtain its time or it might be caused by the buffering or caching mechanisms in the device. A test was set up to find factors that cause these timestamps to be rounded. This test tests the influence of the following factors on obtaining rounded timestamps:

- The Location access setting
- Wifi setting
- GSM Network (turned off by enabling Airplane mode)
- SIM Card in device

This was done by setting the settings of the mentioned factors to all available combinations followed by a reboot of the device. The reboot is performed to ensure that the settings are actually applied. Performing this test did not show any correlation between a rounded getTime() value and any of the above mentioned factors. So there is something in the GPS receiver hardware or low level processing that introduces these values, possibly some sort of caching mechanism.

A next step that was done in discovering where these values get rounded is inspecting the message on the NMEA[24] data communication messages outputted by the GPS receiver. By manually inspecting these NMEA messages both rounded values and non-rounded values were obtained.

6.4.2. Possible causes of inaccuracy

In sec. 5.4.2, it was explained how GPS in Android works. An important thing that was remarked was that GPS receiver soft- and hardware is not the same on different types of Android devices. So the inaccuracies found here are not for all types of Android devices. In this section some causes for these inaccuracies is looked at.

GPS Receiver

Determining possible causes for these inaccuracies, is partly speculating. Not all inner workings of Android are clearly documented or publicly available [37]. In figure Fig. 6.6 an overview is given of the GPS architecture in Android. What can be observed from this figure is that the Android Location Services is built upon a so-called GL Engine which has multiple possible inputs of timing information. Namely, the GL Engine can obtain a timestamp from a Secure User Plane Location (SUPL, part of A-GPS), through an NTP server, via the Network Location provider (GSM) or from the NVRAM or XTRA Data. How exactly this is done in different Android devices is vendor specific.

What the GL Engine then basically does is detecting satellites for which the GPS driver was programmed and then combine this with with timing information from one of these other sources to construct a fix. Alternatively, it can download the

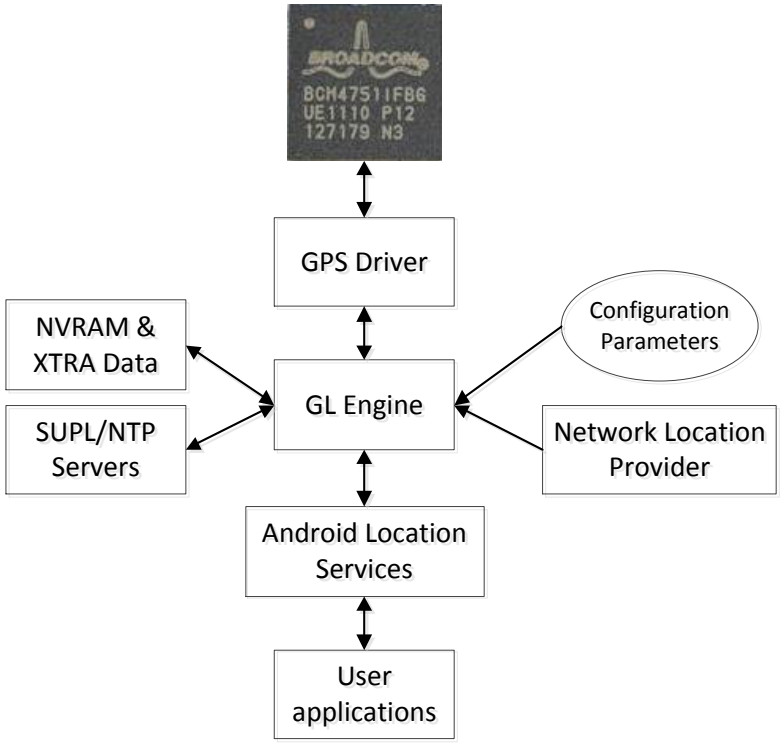


Figure 6.6.: Android GPS architecture [36]

additional required information from the satellite, which is very slow. To download this required timing information from the satellite, a so called almanac must be obtained which contains coefficients to calculate UTC time from the GPS time. This almanac is part of a 25 frames long navigation message. Since each of these frames consist of 1500 bits, 25 frames consist of 37500 bits. The navigation message is transmitted at a rate of 50 bit/s, therefore downloading a navigation message from a satellite would take 12.5 minutes[38].

The first option, using one of the other non-GPS sources to obtain, is a lot faster. However, this has the downside that the timing information is less accurate, because (1) it might be using a different combination of sources which is different between different device models and/or vendors, (2) the accuracy of these sources is less and (3) the sources of these non-GPS timing sources might differ (for example a different NTP server, a different Network Location Provider).

One of these sources, NVRAM is different than the other two sources. NVRAM functions as a cache for storing timing information. Again, just like the other two sources, it is device/vendor specific whether or not this is passed onto the Android Location Services. If this is used by the GL Engine, then this is another source of possible inaccurate timing information.

Overview

So the GPS receiver can cause inaccuracies to the provided timing information. Beneath the level at which GPS can be accessed through the Android API, there are several elements that might be the cause of these inaccuracies. Causes might be one or more of the following:

- The emphasis of a GPS receiver in a smartphone is on power saving and not performance. Because of this, the receiver might for example not log or receive all broadcasted GPS signals. Instead, it might for example interpolate time values obtained from GPS fixes.
- GPS on Android is probably designed to be used with Assisted GPS (A-GPS). It might be the case that A-GPS is used "underwater" to obtain the GPS time in some cases where no GPS time was obtained.
- GPS on Android is not designed for timing information in the first place. The main reason GPS in Android is used is for positioning purposes.
- The so called ZDA message (providing a very accurate Pulse-Per-Second (PPS) timing message) of the NMEA protocol that provides extra timing information is not supported in all smartphones. This might be reserved for more dedicated GPS receivers where GPS as a time-source is the main purpose. It might not be implemented for energy savings or economic reasons. For example, on the LG Nexus 4 test device, no ZDA messages were observed to be outputted by the GPS receiver. On the contrary, other messages, such as the GPRMC message where received very often. This last message type also provides timing information, but its not as accurate as the ZDA message because the sending moment of this message is not synced with the PPS clock in the GPS satellite.

And again, these causes might be different for different kind of Android smartphones which have different hard-, soft- or firmware. So it is not so easy to pinpoint one single or common cause. One thing is clear at least; GPS timing information on Android is not nearly as accurate as the GPS technology would allow for. In fact, obtaining the time from an NTP server in optimal conditions with a single timestamp request is much more accurate.

6.5. Fingerprint offset matching

Another individual part of the measurement system that can be tested for accuracy is the fingerprinting part, both client side and server side. The Gracenote Entourage system that is used for fingerprinting is only available on mobile devices. It makes use of a proprietary implementation of the MediaRecorder.AudioSource class from the Android API. In other words, the microphone on the Android device must be used to record a fingerprint. This is an obstacle when trying to measure the consistency of

the fingerprint offset returned by the fingerprint server. In an attempt to overcome this obstacle and check if the fingerprint time offset values from the server are consistent, the following was done:

1. Record audio from a piece of TV content
2. Connect the Android device with a PC using a 4-pins jack plug to have the audio output of the PC act as audio input for the Android device. The cable used here was soldered manually and proved also useful for other tests in this chapter.
3. Start the fingerprint engine when no sound is being played yet.
4. Start playing the recorded piece of audio content from the PC, feeding it as input to the Android device. Having the input source recorded through a cable avoids environment noise being picked up by the microphone.
5. Send a request for fingerprint recognition and note the match time offset.
6. Repeat this multiple times and compare the values.

This basically measures the combined accuracy of the audio classification engine contained in the fingerprinting engine on the local device and the server side matching process of the fingerprints. The test result for 25 values obtained in the described manner using a random piece of recorded content of a broadcast of a random TV program on BBC One are presented in Fig. 6.7. Computing the average and the standard deviation based on these results, yields an average of 161847 ms and a large standard deviation of σ=896.

Based on these results, it seems that the server side matching process of the fingerprints is not quite accurate or maybe the local fingerprint engine is not consistent in creating exactly the same fingerprint. Based on the status of what the fingerprint engine detects or analyzes, the engine starts a fingerprint or not. The engine provides callbacks if it detects noise, speech, music or silence. Based on this detection it independently decides if it should start a fingerprint or not. Another factor that might contribute to the variance in the results is the moment the playout is manually started relative to the moment the fingerprint engine was started, namely the moment when step 4 is performed. There was no clear correlation to this however, if the playout was manually started if the fingerprint engine was running for more or less half a minute or a few seconds did not seem to matter. Both ways returned a time offset in the range of on average hundreds of milliseconds up to 3 seconds between each other. Due to the nature of the test, this is not easy to test in the range of (tens or hundreds of) milliseconds. Namely it would be required to play the recorded piece at a fixed time interval from the moment the fingerprint engine is started. This could be done, but for example load difference at both machines might fluctuate, still having an uncertain factor.

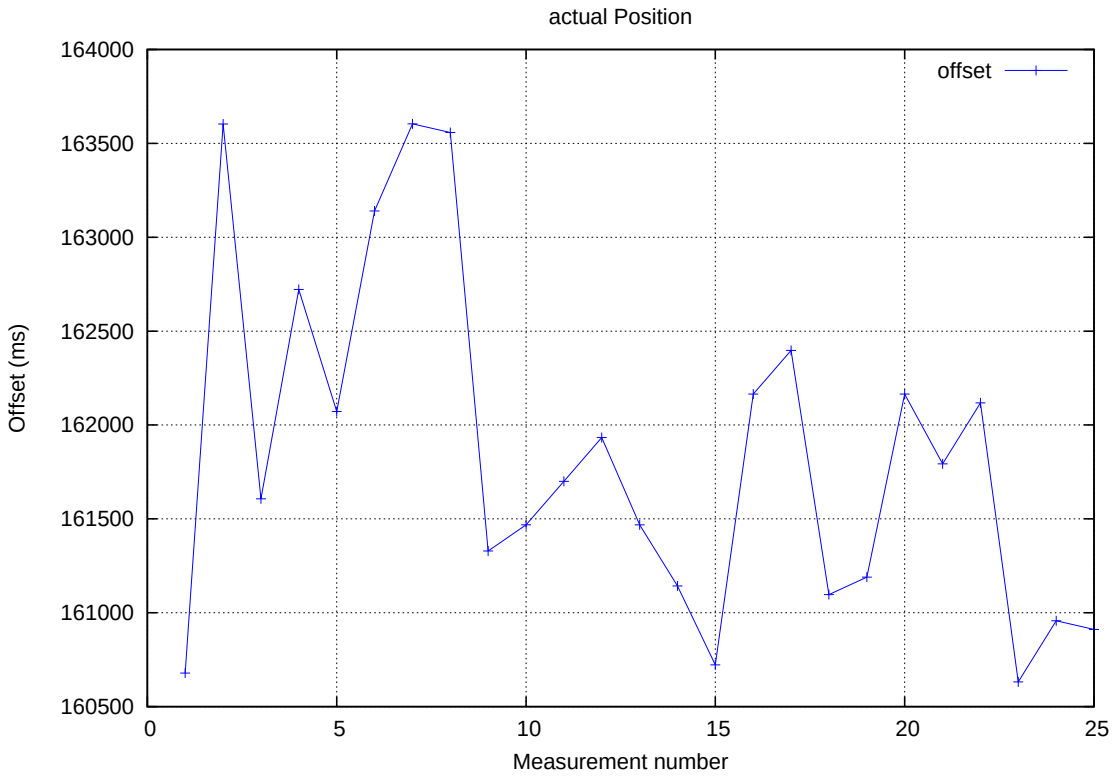


Figure 6.7.: Measured time offsets of a local fingerprint match inside the server side reference audio using a single piece of pre-recorded TV audio. The pre-recorded piece of audio is fingerprinted by feeding it manually to the fingerprint engine.

6.6. Overall precision

6.6.1. Test setup

This subsection measures the overall precision of the developed measurement system. In the measurements included from here on, a small algorithm that seemingly slightly improves the performance of the outliers of the measurement is implemented in the measurement system. Even with this small algorithm, there are still outliers. This algorithm performs multiple regular fingerprint measurements and based on the difference between local elapsed time between individual measurements on the Android device and between the differences obtained from multiple timing measurements from the reference server, inconsistent measurement are ignored and the whole measurement sequence is started again. Experience showed that this was not always consistent and this algorithm ignores inconsistent values. Translating this into a formula, a measurement is marked as inconsistent if:

$$abs\left((T_{\text{Start,FP}n} - T_{\text{Start,FP}(n-1)}) - (T_{\text{Result,FP}n} - T_{\text{Result,FP}(n-1)})\right) > 300$$

Here, $T_{\text{Start,FP}n}$ and $T_{\text{Start,FP}(n-1)}$ are the moments fingerprints n and fingerprint $(n-1)$ are started according to the local clock of Android (which is described in sec. 6.3). Furthermore, $T_{\text{Result,FP}n}$ and $T_{\text{Result,FP}(n-1)}$ are the server-side fingerprint matching results time offsets. All values are in milliseconds.

Precision is defined[39] as the "*Closeness of agreement between indications or measured quantity values obtained by replicate measurements on the same or similar objects under specified conditions*". In other words, the precision of the complete playout measurement system is the degree to which playout difference measurements started at the same moment produce the same values.

To test the precision of the system, the setup as displayed in Fig. 6.8 was used. In this setup, two smartphones measure the audio from a single TV. The measurements on both smartphones are started manually more or less at the same time (within less than around 200 ms of each other). This results often in exactly the same piece of audio being fingerprinted, looking at the callbacks from the audio fingerprint engine on both phones (speech, noise and music classification happen often exactly at the same time). This also often results in the recognition moment happening exactly at the same time (the moment the match results are received from the server). However this is not always the case. Sometimes, for example, the start of the recognition process is the same, but during the recognition process one smartphone recognizes the audio a little later. This might be caused by the fact that the audio fingerprint engine only allows for listening to real-time audio and not accepting chunks of pre-recorded audio. This makes it impossible for two smartphones to start recording the same content at *exactly* the same moment. Furthermore, there might be inaccuracies introduced in the process of playing the audio from the TV, traveling over the air to the microphones of the smartphones and going through the processing in Android. If one smartphone did not return a match at all, the measurement was marked as invalid and the measurement was done again.

The smartphones used during this test were a LG Nexus 4 with Android version 4.3 (API level 18) and a Samsung Galaxy S3 4G with Android version 4.1.2 (API level 16). The TV was playing BBC One with a Ziggo Digital TV (DVB-C) subscription. Both smartphones are connected to the NTP server (and the internet) in the same way as the test setup described in sec. 6.3 (see Fig. 6.3), so the NTP timestamp obtained is at most few milliseconds less accurate than the accuracy of the stratum 1 NTP server itself (which has been discussed in sec. 6.2).

6.6.2. Test results

The results measured in this test are presented in Fig. 6.9. This shows the playout differences between the local TV and the reference TV on the fingerprint server as measured by the measurement app ran on both devices at the same time.

Fig. 6.10 shows the overall precision error of the system. This is simply the difference of the values shown in Fig. 6.9. This results in a graph with less "spikes" than the

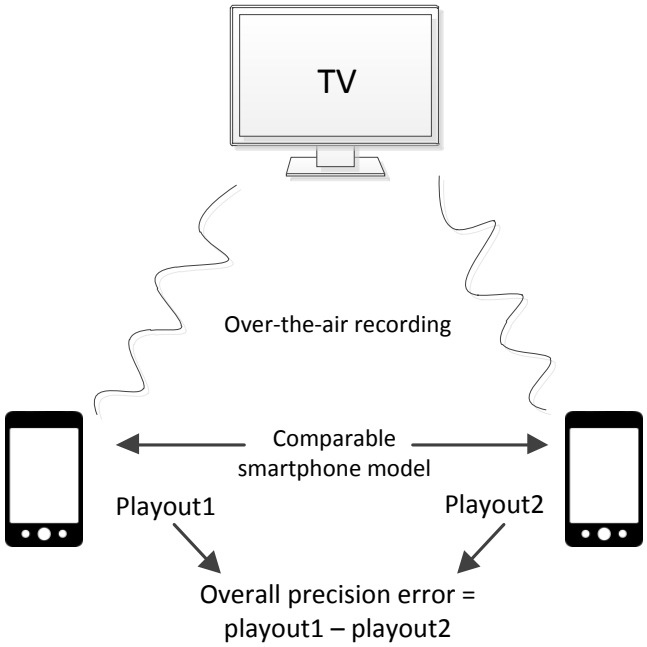


Figure 6.8.: Overall precision test setup.

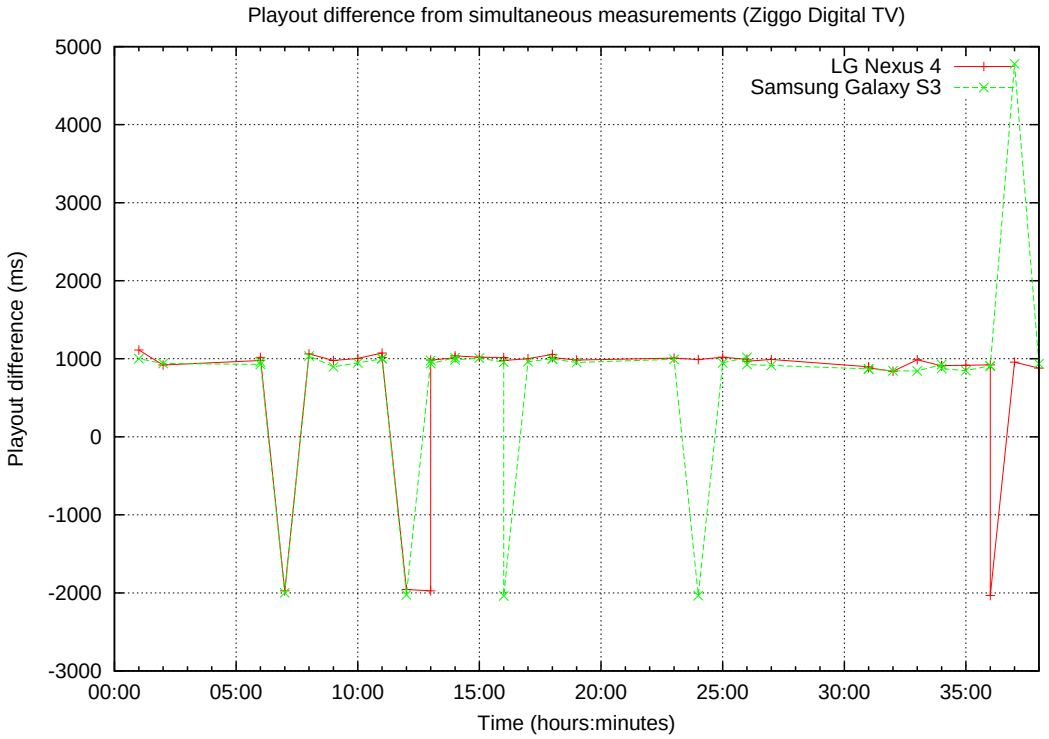


Figure 6.9.: Playout differences using simultaneous measurements Ziggo Digital TV.

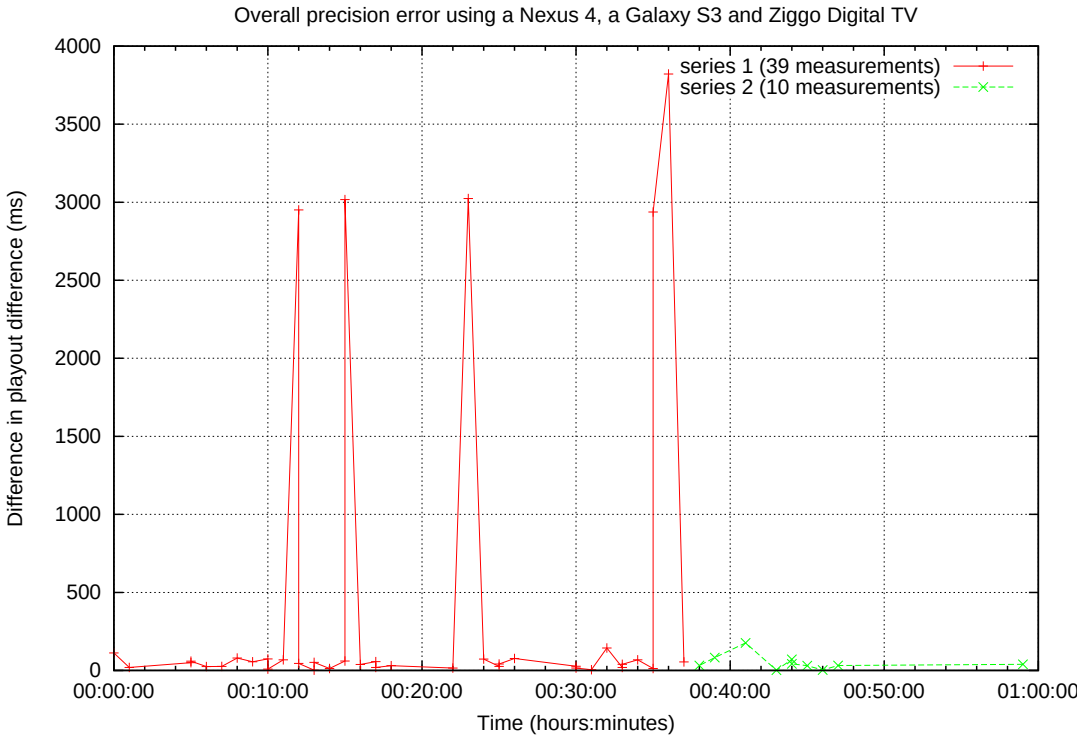


Figure 6.10.: Overall precision error of the system. Difference in playout difference using simultaneous measurements using a Nexus 4 and a Galaxy S3.

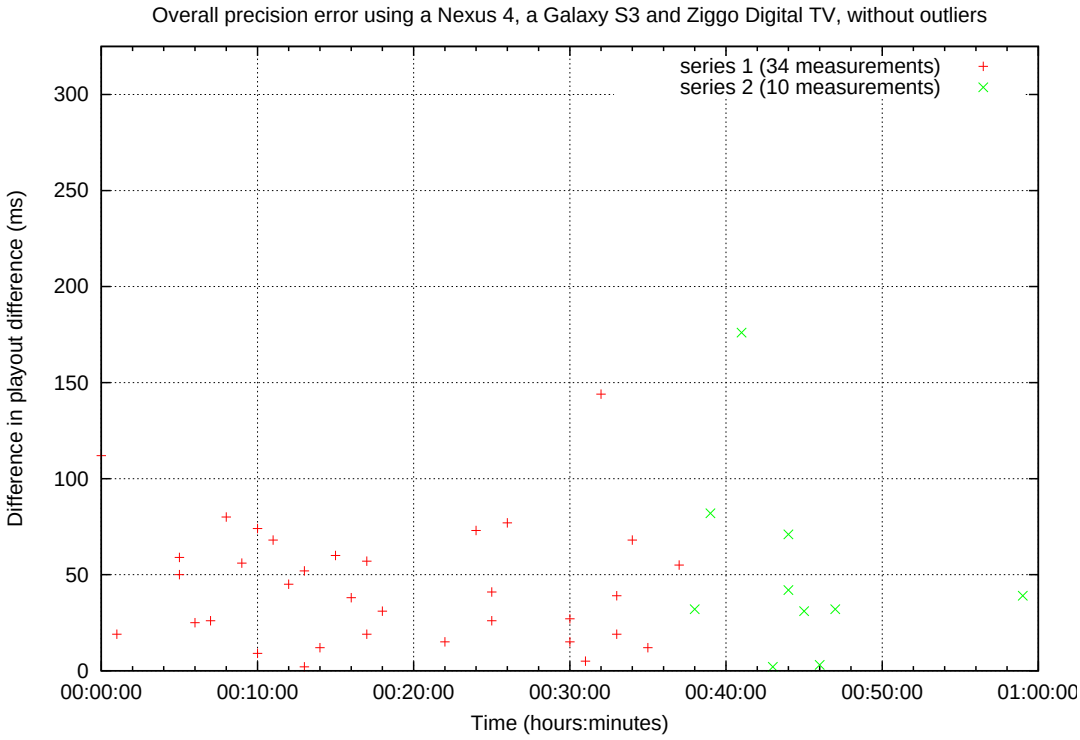


Figure 6.11.: Fig. 6.10 without outliers

one in Fig. 6.9 because the moments where both devices measured an outlier are not shown here, because these are not precision errors. Both measurements "agree" more or less on the measured value, when speaking in terms of the definition of precision introduced at the beginning of this section.

6.7. Overall accuracy

6.7.1. Test setup

Accuracy is defined [39] as the "*closeness of agreement between a measured quantity value and a true quantity value of a measurand*". In other words, the total accuracy of the playout system is the degree to which the obtained values correspond with the actual values. Since there is no access to the reference TV source, it is not possible to compare these values directly. However, it is possible to calculate the difference between playout differences obtained from two different TVs and compare these with the actual audio difference between these TVs. This is exactly what was done and the setup is presented in Fig. 6.12 and Fig. 6.13.

In this setup, two smartphones measure the playout difference using the app on two different TVs simultaneously. One smartphone measures the playout difference of TV1 and the other smartphone measures the playout difference on TV2. Both TVs obtained their TV content from a different provider (KPN Digitenne on TV1 and Ziggo Digital Cable on TV2) but are tuned to the same channel (BBC One). Smartphone 1 records TV1 using a 3.5mm jack plug connected directly to the headphone output of the TV. Smartphone 2 records TV2 over-the-air. Furthermore, in between these measurements, the actual audio difference is measured by connecting the headphone output of both TVs to an RCA connector obtaining a mono signal from both TVs. These different mono signals from both TVs are then combined using a audio interface to create a stereo signal where the left signal represents the audio of TV1 and the right signal the audio of TV2. This stereo signal is then recorded using the software "Audacity"[40] on a laptop connected with a USB cable to the audio interface. This recorded audio signal is then manually analyzed to compute the difference in audio between both TVs. A screenshot of how this is done is can be found in Fig. 6.14.

Now the overall accuracy is obtained by subtracting the values of the playout differences measured with the different smartphones from each other and comparing the resulting value with the actual difference in audio. Note that the difference in audio playout between TV1 and TV2 will be more or less constant, but not completely. Differences in audio playout might arise due to frameskipping or buffering techniques in the TV itself.

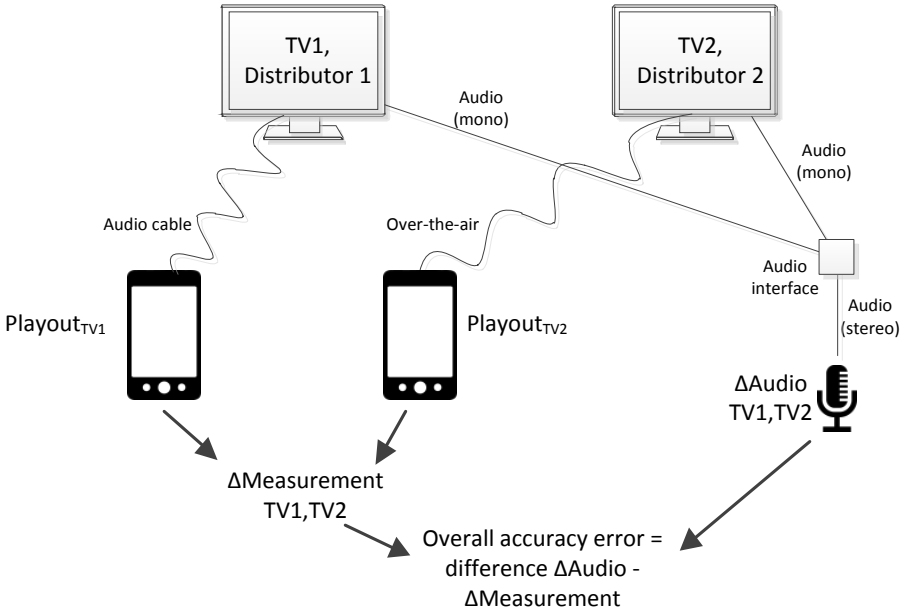


Figure 6.12.: Overall accuracy test setup

6.7.2. Test results

The playout differences measured during testing are displayed in Fig. 6.15. Just like in earlier measurements, some outliers can be seen. However, the majority of the playout difference measurements from TV1 are around 3000ms and the majority of TV2 around 1000ms. In Fig. 6.16 the overall accuracy error is presented. This is the difference between the values from Fig. 6.15 and the actual audio measurements carried out in between the app measurements. When one of the measured playout differences is an outlier (from either TV1 or TV2), the measured accuracy will most likely also be an outlier. So the overall accuracy test will have more outliers than just a series of single playout measurements with the same amount of measurements. Fig. 6.17 provides a close-up on the measured overall accuracy error, without the obvious outliers. It can be seen that the non-outliers measure the playout difference pretty well. The playout difference is often within less than 125ms. However, starting from around measurement number 60, the accuracy becomes less with 2 datapoints showing an accuracy error of around a little over 400ms. Near the end of measurement series the accuracy becomes better and approaches the accuracy level at the start of the measurement series.

The inaccuracies shown here cannot be the client side timestamp because the client makes use of an NTP server in optimal conditions. So the inaccuracies are often at most the RTT of the SNTP request, which is around 5-15ms, and at most 25ms.

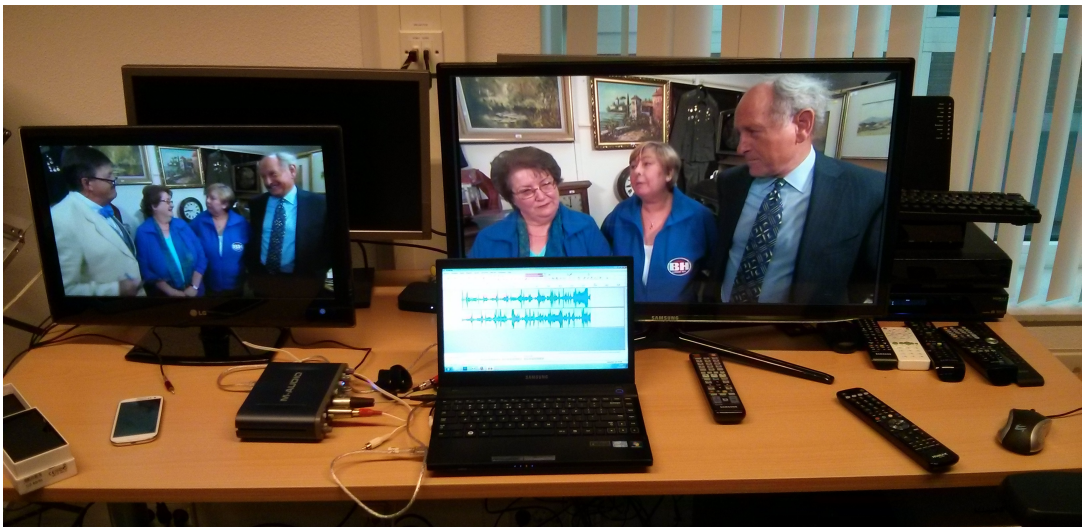


Figure 6.13.: Overall accuracy test setup photo

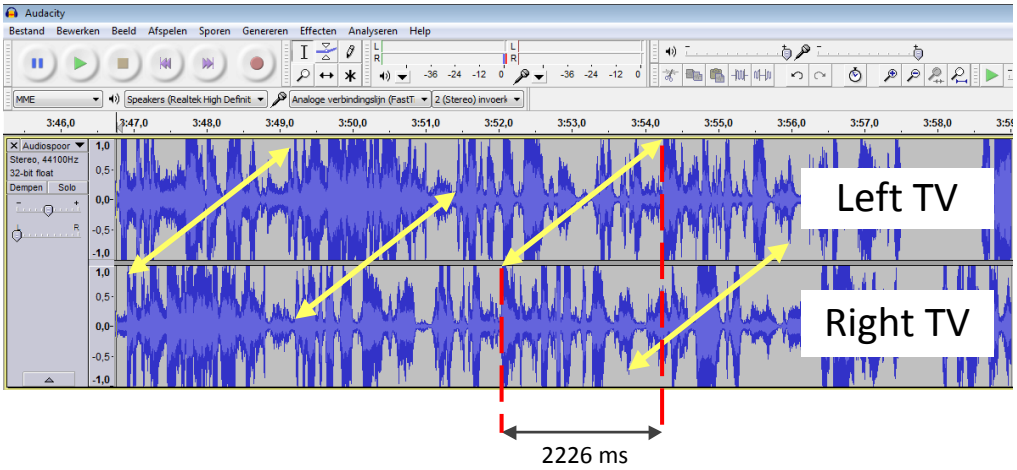


Figure 6.14.: Manually measuring audio difference between two TVs

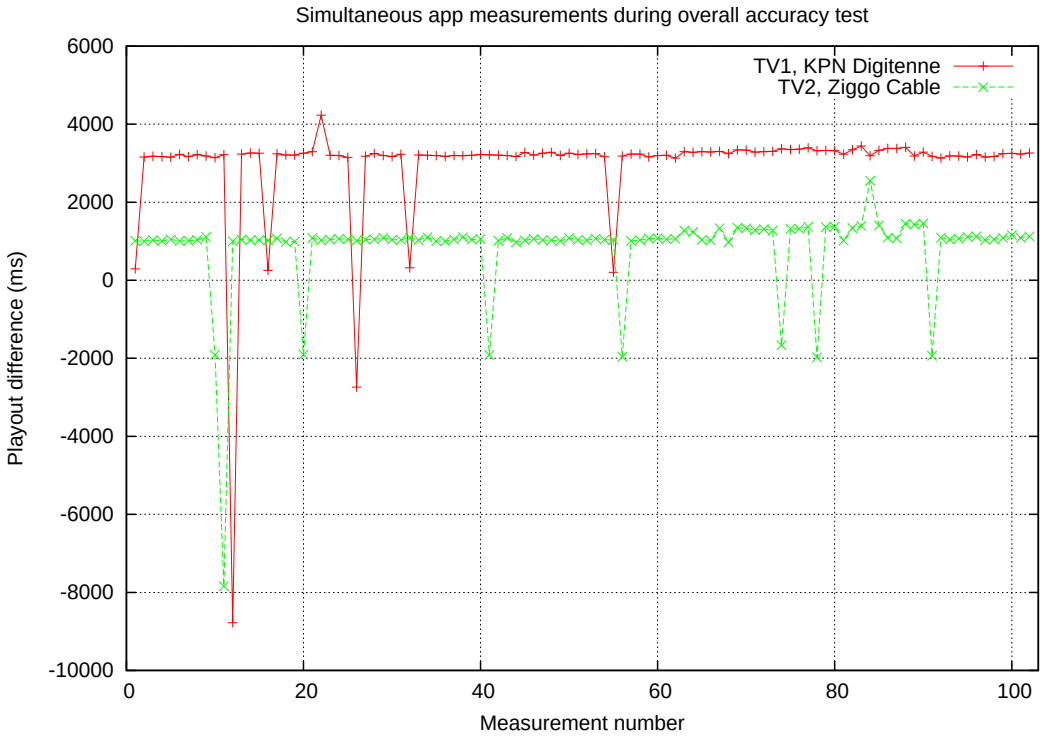


Figure 6.15.: Playout differences measured during the overall accuracy test

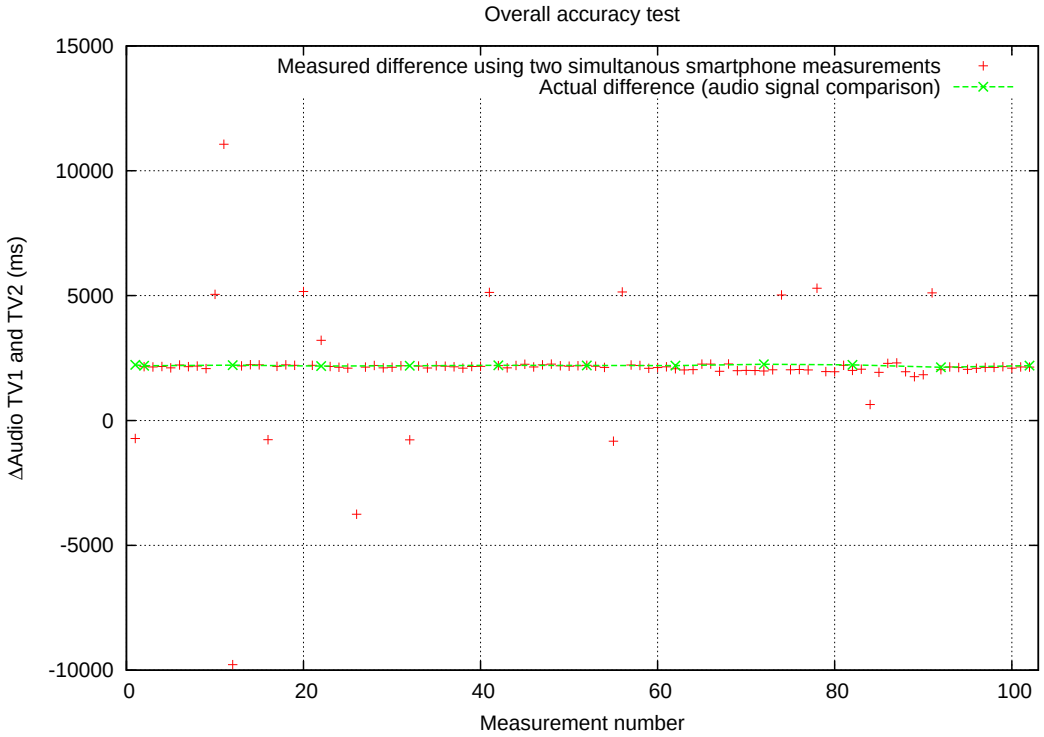


Figure 6.16.: Overall accuracy test results

Neither can it be Android internal clock drift, which was shown in sec. 6.3 to be magnitudes less than the inaccuracies shown here. So there are only 2 remaining factors left that could cause this drift, namely the fingerprint engine on the Android device and the server side internal clock drift. These are both black boxes. It would seem less probable that this would be caused by the fingerprint engine on the Android device, because there are no real factors to be pointed to that might clearly cause such an inaccuracy. Server side clock drifting is a much more logical cause. Looking at which TV programs were currently broadcasted during the measurement also slightly reinforces this idea. There seems to be a loose correlation between the program currently on and the accuracy of the measurement. It might for example be that the server side clock gets corrected at the start of a new program. However to make any real conclusions about this relation, more datapoints spread out over much more different TV programs would be needed. On the other hand, if server side clock drift is causing this, it is strange that the difference between the measured values is becoming less. If the clock is drifting linearly, both app measured values would drift in the same direction, not changing their mutual difference much. If the clock is drifting non-linearly over short periods of time, then it might be possible that the difference between these app measured values becomes less. But with an equal chance, these values should become less. This is not something that is observed in the figure, however. So it remains speculating what is really causing this small change in accuracy.

Nevertheless, the app seems to provide an accurate value if the outliers are filtered out. Without removing outliers, 83% of the measured differences are within 400 ms accuracy and 81% are within 300 ms accuracy of its real value based on this test. Once the outliers are removed, the measured differences are in 98% of the cases within 300 ms of their actual value difference and 100% within 400 ms accurately.

These values representing the differences are actually based on two measurements each, and each introducing inaccuracies. So the actual amount of outliers will have more or less half the amount of outliers. This is seems logical when looking at Fig. 6.15 again. The KPN Digitenne measurement shows 7 outliers out of 103 measurements (6.80%) that deviate more than 300ms from the median (3221 ms). The Ziggo cable measurement shows 17 outliers out of 103 (16.50%) that deviate more than 300ms from the median (1047 ms). Combining the numbers from both measurements would indicate a 24 out of 206 outliers more than 300 ms from the mean. This equals an accuracy of 88.35%.

6.8. Filtering outliers

In the previous results, it was seen that the measurement results sometimes contain outliers. More specifically, in Fig. 6.16 81% of the values obtained in a test were accurate within 300ms of the actual value. When the values that differ more than 300ms from the actual value are considered to be outliers, 19% of the values returned

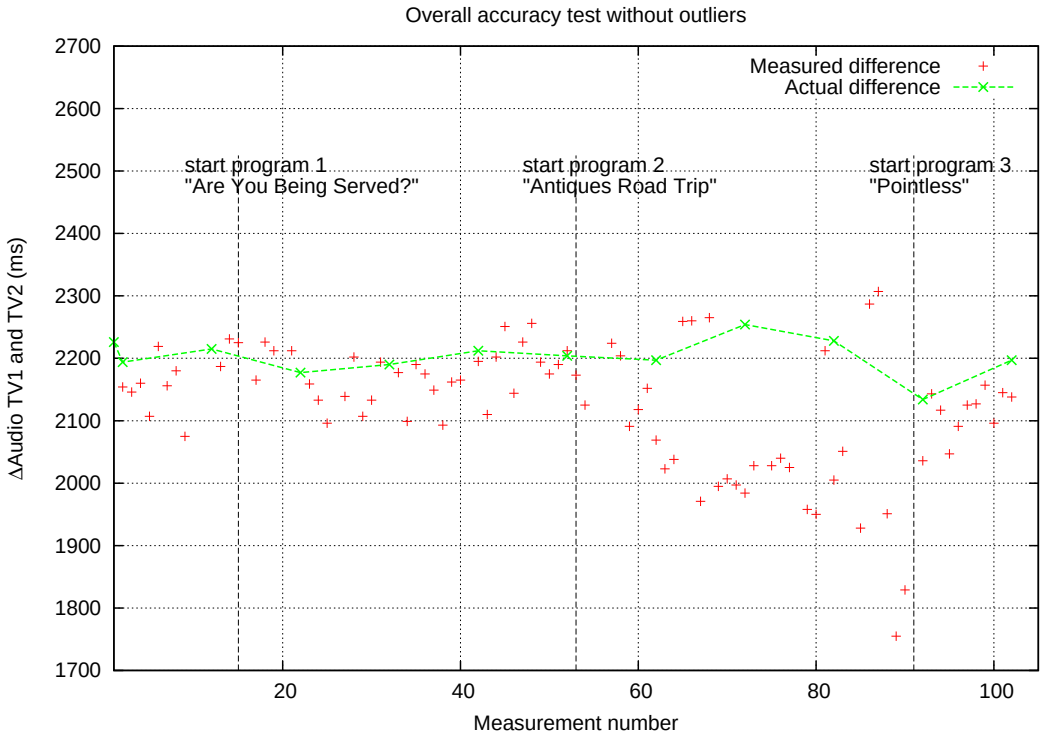


Figure 6.17.: Overall accuracy test results without outliers

by the measurement system are outliers. To increase the tolerance, the outlier border could also be set to 400ms, in which case 17% of the values are considered to be outliers.

However, looking at this same figure, a distinction can be made between the outliers. First, there are outliers that are close to the actual value. These do not differ more than 400ms from the actual value. Second, there are outliers that are a fixed value (more or less 3000 milliseconds or a multiple of this) off from the actual value. These are the outliers that might possibly be a sub-fingerprint mismatch. To filter these last type of outliers out, a boundary larger than 400ms but smaller than half the difference of this fixed value might be used.

Outlier test

An algorithm for removing outliers could for example use the standard deviation of each measurement to determine if a measurement is an outlier. It could for example remove the values that deviate the most from the mean and then return the mean of the non-discarded values as the actual playout. This first part is exactly what the z-test does for detecting outliers. This test calculates a z-score of a sample and compares this with a fixed threshold. If the z-score is higher than the threshold, the sample is marked as an outlier. If not, the sample is a non-outlier. The z-score is

calculated as shown in 6.1:

$$z - score \quad = \quad \frac{abs(mean - sample)}{standard\ deviation} \tag{6.1}$$

The next question is then what a good value of a reference threshold would be to compare the z-scores with. Different measurement group sizes have different optimal values the z score reference threshold. The determine candidate z score reference thresholds, tests were performed using Matlab scripts. More specifically, the following was done:

1. Select a dataset as a reference sample.
2. Manually tag the outliers.
3. Select a random permutation of n-values in this reference dataset, obtaining measurement group samples. Here, n is the measurement group size that represents the amount of individual measurements to be done that together could agree on which of these values would be an accurate measurement of the actual playout difference.
4. Calculate the z-score for each of the values in the measurement group. This results in n z-scores per measurement group sample. So, for n=5, 5 z-scores are calculated with a mean and standard deviation over these 5 values.
5. Calculate the false and true positive rate for a range of candidate z score reference threshold values. The true positives are the values that are classified as non-outlier (where the z-score would be below the z score reference threshold value) and are also in fact, real outliers (as manually marked). False positives are values marked as outliers based on the z-score, while they are not in reality.
6. Repeat this for multiple values of n.
7. Choose a reference threshold value for z that has a low false positive rate combined with a high true positive rate for a given value of n. Logically, the higher the true positive rate, the higher the false positive rate.

Results

The true- and false positive rates obtained this way for two datasets and n=5 can be found in Fig. 6.18. What can be seen is that the Ziggo dataset has more false positives than the KPN dataset for a given z threshold value (logically, this does not mean that Ziggo is somehow worse than KPN, but that the fingerprinting system returned an outlier). This is something that reflects the fact that the Ziggo dataset has more outliers than the KPN set. Furthermore, for almost all values, the Ziggo dataset has more true positives for a given value of the KPN dataset. This might

(a) Dataset: KPN/Digitenne/NTP, z=1. This dataset contains 103 measurement, of which 7 are outliers (more than 300 ms deviation from median, 6.80%). Results based on 100000 random permutations for each measurement group size.

	Measurement group size					
	3	4	5	7	10	15
No result rate	1.0e-05	0	0	0	0	0
False positive rate	8.1e-04	3.6e-03	8.2e-04	8.0e-05	0	0
True positive rate	0.9919	0.9964	0.9992	0.9999	1	1
Improvement	+5.99%	+6.44%	+6.72%	+6.79%	+6.80%	+6.80%

(b) Dataset: Ziggo/Cable/NTP, z=1. This dataset contains 103 measurement, of which 17 are outliers (more than 300 ms deviation from median, 16.50%. Results based on 100000 random permutations for each measurement group size.

	Measurement group size					
	3	4	5	7	10	15
No result rate	1.0e-05	0	0	0	0	0
False positive rate	6.95e-02	3.35e-02	1.76e-02	4.30e-03	4.10e-04	0
True positive rate	0.9305	0.9665	0.9824	0.9957	0.9996	1
Improvement	+9.55%	+13.15%	+14.74%	+16.07%	+16.46%	+16.50%

Table 6.1.: Outlier detection algorithm results using datasets 1 and 2, using 100000 random permutations of playout difference measurements two datasets. Results of classification are obtained by calculating z-scores, discarding z-scores higher than 1 and returning the median of the playout values corresponding with the non-discarded values. If the non-discarded values are even and there are at least 2 values, the mean is calculated by ignoring the last value. Test were performed using MATLab R2013b.

(a) Dataset: KPN/Digitenne/NTP, z=1. This dataset contains 103 measurement, of which 7 are outliers (more than 300 ms deviation from median, 6.80%). Results based on 100000 random permutations for each measurement group size.

	Measurement group size					
	3	4	5	7	10	15
No result rate	0	0	0	0	0	0
False positive rate	8.4e-03	5.0e-04	6.0e-04	0	0	0
True positive rate	0.9916	0.9995	0.9994	1	1	1
Improvement	+5.96%	+6.75%	+6.74%	+6.80%	+6.80%	+6.80%

(b) Dataset: Ziggo/Cable/NTP, z=1. This dataset contains 103 measurement, of which 17 are outliers (more than 300 ms deviation from median, 16.50%. Results based on 100000 random permutations for each measurement group size.

	Measurement group size					
	3	4	5	7	10	15
No result rate	0	0	0	0	0	0
False positive rate	3.52e-02	4.00e-02	7.7e-03	1.3e-03	3.2e-04	0
True positive rate	0.9648	0.9600	0.9923	0.9987	0.9997	1
Improvement	+12.98%	+12.50%	+15.73%	+16.37%	+16.47%	+16.50%

Table 6.2.: Outlier detection algorithm by selecting the mean of the values in the measurement group sample. Results obtained by using datasets 1 and 2, using 100000 random permutations of playout difference measurements for both datasets. If the non-discarded values are even and there are at least 2 values, the mean is calculated by ignoring the lowest value (this gives slightly better results than using the highest value, since outliers are more often negative outliers in the used datasets).

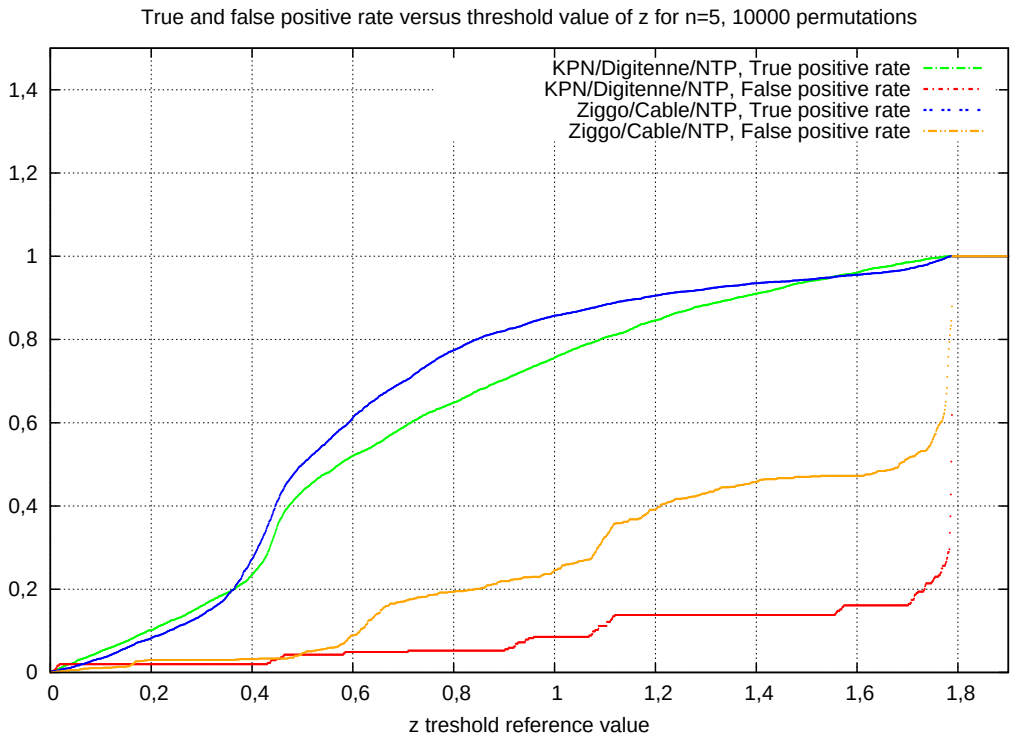


Figure 6.18.: Reference threshold of z versus true positive and false positive rates for n=5 using the datasets from Fig. 6.15.

sound unlogical, because a dataset with a large amount of outliers might be expected to have a small amount of true positive, but this is not the case. Since the Ziggo dataset has more outliers than the KPN dataset, the Ziggo dataset has fewer non-outliers and thus less chance that a non-outlier is marked as a false positive. Thus the ratio of true positives for Ziggo is larger, but the absolute amount of true positives is not.

In Fig. 6.19, a graph with the false positive rate subtracted from the true positive rates can be found. This might not be the best way to determine an optimal true/-false positive rate. It might for example be better to choose a z reference threshold value that has a low false positive vs a true positive *ratio*. Furthermore, in this same graph, the same values but for a measurement group size of 10 can be seen. Here it can be seen that the optimal value of z depends on the group size used. Although there are values of z which perform well for both group sizes and both dataset, such as n=1. For the dataset of KPN and n=5, a value of the z threshold reference around 1.7 would be slightly better. But for the Ziggo dataset with also n=5 this would not be the case, because this optimum lies at around a z threshold reference of 1. So the performance of the outlier detection mechanism (logically) depends on the amount of outliers in the dataset and thus on the optimal. So, based on this figure, a z threshold reference value of 1 is a good candidate for at least the two datasets used, but might not be optimal. To find a better z threshold reference

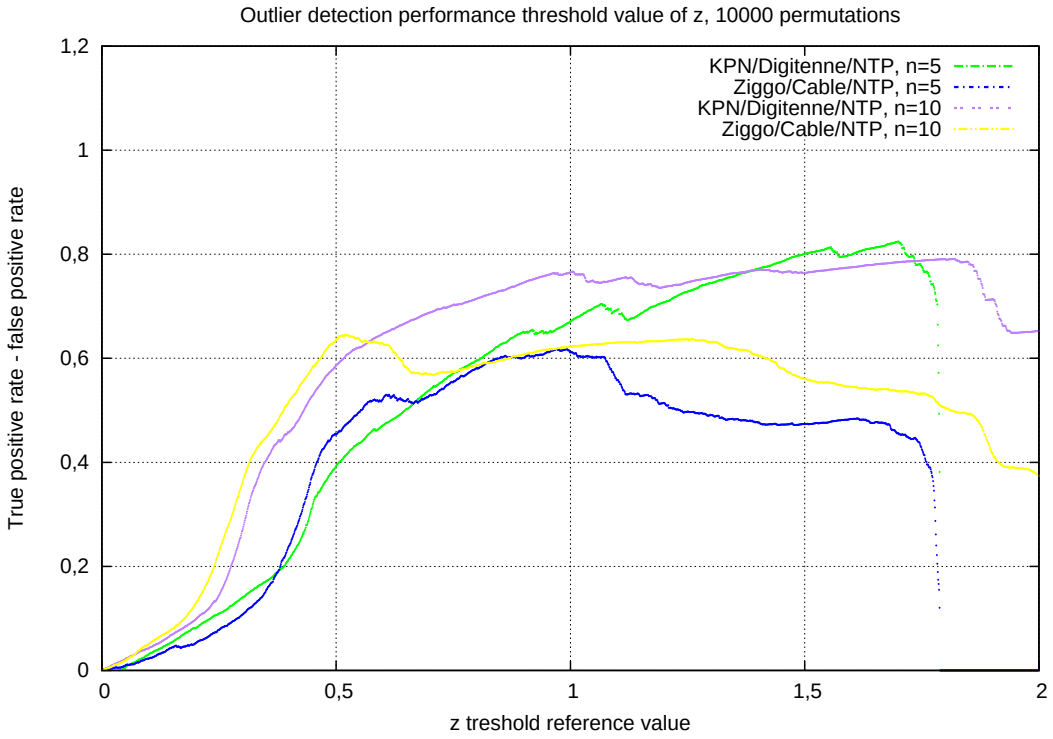


Figure 6.19.: Reference threshold of z versus true positive minus false positive rates using the datasets from Fig. 6.15.

value, more and larger datasets are required. For now, z threshold reference value of 1 will be used.

Median

The above described method describes the performance of detecting individual outliers based on a series of measurements (n). If the information for each of the individual outliers in this measurement group is combined and the outliers are discarded, it is possible to obtain a value of the playout difference that is no outlier with much more certainty. This is what has been done next. Using a fixed z threshold reference value of 1, again different permutations for different measurement group sizes (n) have been randomly picked. But this time, not a false/true positive rate for *individual* measurements in a measurement group is calculated, but for the *median* of the non-outliers (as detected by comparing the z-score with the z score reference threshold value). The results are presented in Tab. 6.1.

The results show that even with a small group size, outliers can be filtered out reliably in the used datasets. Based on these results, one might wonder why the rate that the outlier detection algorithm returns no results is so low. Although unlikely, it might for example be the case that the random permutations select values that

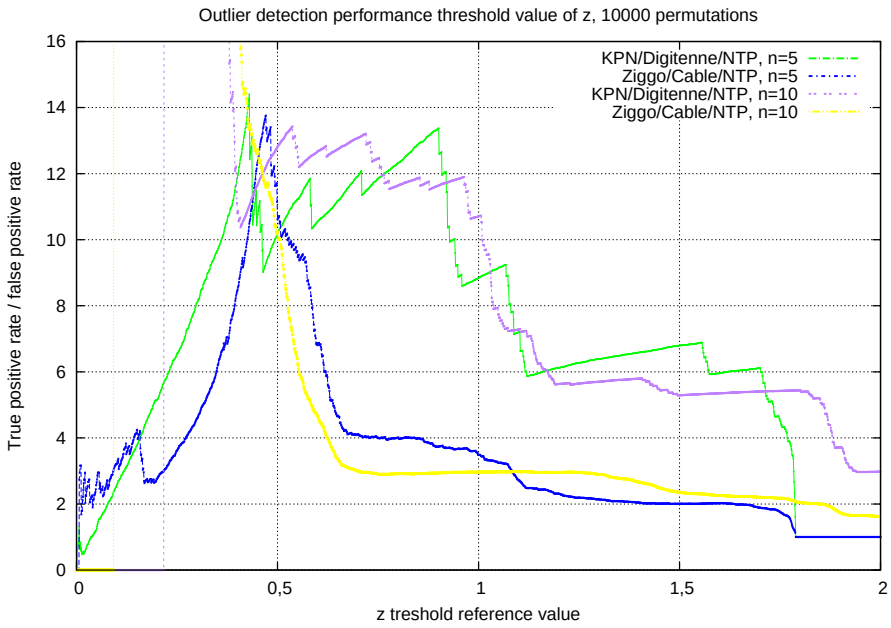


Figure 6.20.: Reference threshold of z versus True positive rate divided by the false positive rate using the datasets from Fig. 6.15.

are only outliers. For n=4, this might be possibly, since the chance that 4 out 4 in the dataset using the Ziggo measurement (which has the most outliers) are outliers is $\left(\frac{17}{103}\right)^4 = 7.42.10^{-4}$. Multiplying this by the amount of random permutations gives a chance of 74.21% that this would have happened. Starting from n=5, this chance is much lower, namely 12.2% for n= 5 and 0.33% for n=7. For the other dataset from KPN, these numbers are fractions lower because there are only 7 outliers in that dataset.

Note that for these assumptions to hold, outliers are assumed to be statistically independent. It is not known if this is the case. Since the values returned by the fingerprint reference system are coming from a system that can be seen as a black box, there might not be many other ways to say anything useful about sample sizes and accuracy this way. One way to determine this would be to test if the distance between the outliers are distributed normally. To do this however, a much larger dataset would be needed to get results that are accurate enough, since outliers are only a fraction of the measurements.

Nevertheless, based on these results, outliers can be filtered out with an accuracy of very closely approaching 100%, depending on the measurement group size used.

6.9. Conclusion

One important and maybe remarkable conclusion here is that NTP is a better way to obtain accurate timing information on Android. Also it is more easy to use, since there is no need for the device to have a open view to the sky or having to wait for the timing part of the GPS message to be received. So NTP seems to be the clear winner to use as a time-source for the measurement app, unless the internet connection of the Android device is really bad or not available. But then again, the app doesn't work anyway without internet.

In optimal conditions NTP provides timestamps which are within a few milliseconds accuracy range of the accuracy of the used NTP server. In more or less optimal settings, as used in the test performed in this section, this results in obtaining timing information on the Android device within at most 5-15ms accuracy bounds of the accuracy of the server on average. In turn, the queried stratum 1 NTP server has a clock that represents the actual time with an accuracy bound that is fractions smaller than this range. So the NTP timestamps are also more or less on average at most 5-15 ms inaccurate in representing a time value, which fits well within the required accuracy bounds.

On the contrary, timing values obtained using the GPS receiver on Android are far less accurate, not within the required accuracy bounds. The smartphone having a cheap GPS receiver seems to be a cause of this. Also the GPS sometimes returns a cached value, making it inconsistent and less accurate than NTP.

The largest part of inaccuracy in the measurement system is caused by the time-keeping and time-correlation with generated fingerprints on the live TV recording server provided by Gracenote. This server mostly introduces inaccuracies of up to 300ms, but also seemingly random outliers of with more or less a fixed offset of around 3 seconds, negative or positive of the actual playout difference.

Depending on the needs, the app can measure playout differences well enough. The app is able to measure a playout difference within 300 ms accuracy of its actual value within 81% of the cases. Doing more measurements in succession allows for obtaining the playout difference in this accuracy range with much more certainty. Doing so however would require a good method to detect whether a measurement is an outlier or not. This would involve measuring multiple times in succession and to obtain enough samples to have a satisfactory certainty that the value obtained after filtering out the outliers is the actual playout difference (within the 300ms accuracy range). The measurements done here are based on a limited amount of samples, so more measurements will strengthen the obtained accuracy bounds.

7. Playout measurement results and analysis

In this section, the results from the app measurements using NTP as a reference timesource are presented. The amount of data used for plotting these figures is based on a snapshot of measurements performed by volunteers that have installed the app from the Google Play Store. At the time of writing, additional measurements can still be performed for providing more measurement data. With even more measurements, not only more absolute values of different TV setups will be obtained, but also the influence of factors such as geographical distribution and the moment time on playout differences can be estimated better.

Currently, measurements results are received mainly from the Netherlands and Great-Britain, but there are also some measurements performed in Germany, France, Belgium and Swiss. The results of these measurements, independent of geographical location or time can be found in Fig. 7.1. This figure displays the average of the measured combination of broadcaster, subscription (technology) and quality (HD or SD). What can be seen from these measurements is that analog broadcasts are faster than other broadcasts from the same broadcaster. The reason for this is probably in the encoding part, which makes digital TV slower than analog TV.

Furthermore, HD broadcasts are slower than their SD equivalent in general. This is in accordance with the theory in chapter 4 and seems logical, because HD broadcasts introduce more encoding delay (due to multipass encoding for example) and also use more bandwidth and are therefore more likely to introduce delay in the broadcast. This does not have the be the case necessarily, because something that broadcasters might do is create the SD content based on the HD content, in which case it would be expected that SD is the slower one.

Looking at the absolute values, it can be seen that delay differences of up to almost 5 seconds are possible in TV broadcasts in the Netherlands. International delay differences are larger when measurements from the UK are also included. These measurements are the fastest, and compared with the slowest measurement in the Netherlands (excluding internet streaming TV), delay differences can become almost 6 seconds.

Furthermore, with the help of the BBC, a measurement was also performed at the broadcasting chain prior to coding and multiplexing. This is indicated as “internal” in Fig. 7.1. Comparing the playout delay difference between this internal measurement and the fastest measured average delay difference, we see that there is a delay

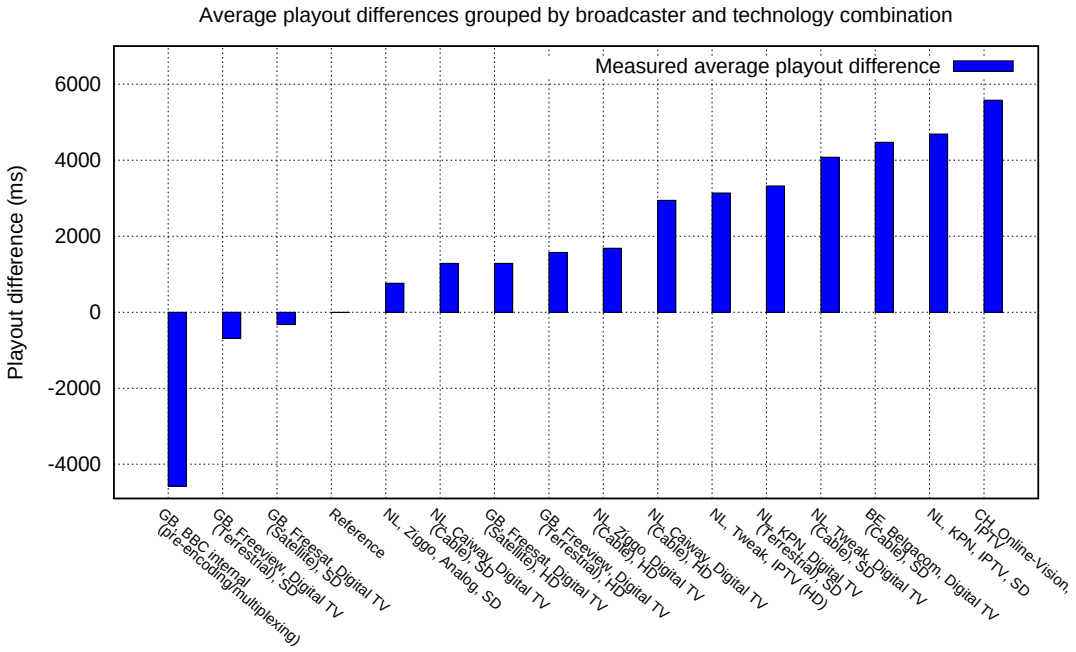


Figure 7.1.: Average playout differences grouped by broadcaster/technology combination.

difference of around 4 seconds. This value consists of the delay caused by encoding, modulation and also distribution to the fastest receiver (which is terrestrial and SD quality). Since it is not known which part of this value is broadcasting delay, we can only conclude that in this case the encoding and modulation at the BBC takes at most around 4 seconds.

Moreover, the measurements clearly show that the playout delay in the United Kingdom is lower than in the Netherlands. This is not surprising as it is broadcasted from the UK itself, although analog broadcasting can be faster than the slower broadcasts in the UK (such as HD or Satellite).

A geographical delay dispersion analysis at a national level, where exactly the same broadcaster and setup combination (subscription type/quality) are directly compared with each other was not performed due to a lack of data. We did notice that there is definitely some playout delay differences between geographically distributed measurements with the same setup. But without sufficient geographically distributed measurements of the same broadcaster and setup combination, no clear conclusion can be drawn. Apart from this, the fact that different TVs or settopboxes will cause some variance in the measured delays will only make it harder to draw a conclusion on this.

Apart from only regular TV, some internet streams broadcasting TV were also

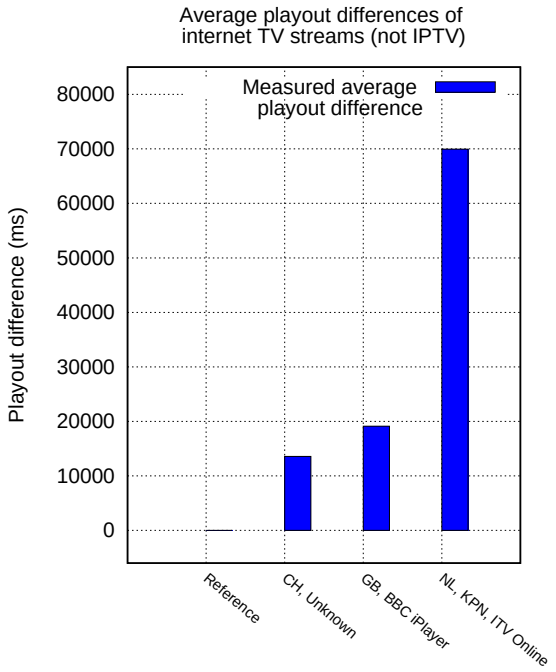


Figure 7.2.: Playout differences of TV Internet streams grouped by broadcaster/technology combination.

measured. Fig. 7.2 present these measurement results. What can be seen here is that internet streams (KPN ITV Online, BBC iPlayer) are much slower than normal television. This is not very surprising and is something that could be expected since content for internet streams is normally prepared separately and is distributed separately as well. The underlying techniques and systems that deliver these streams can vary greatly, both in terms of architecture as well as in terms of delay. E.g. in case MPEG-DASH is used for delivering the internet streams, the content is segmented and encoded in several versions, and usually delivered using Content Delivery Networks.

Apart from measurements using the channel “BBC One”, “BBC Two” was also measured for playout difference delays. Comparing the results from these two different channels, no differences were found that could not be attributed to accuracy fluctuations of the measurement system itself.

Furthermore, multiple measurements using a fixed setup and source were also performed. The results can be found in Fig. 7.3. Here, measurements using a single broadcaster, technology and television (LG 22LE5500) are plotted against time. Note that the distance between different measurement points do not have a fixed value between them in reality. These measurement datapoints are however grouped by the day these measurements were performed. This should give an idea of how

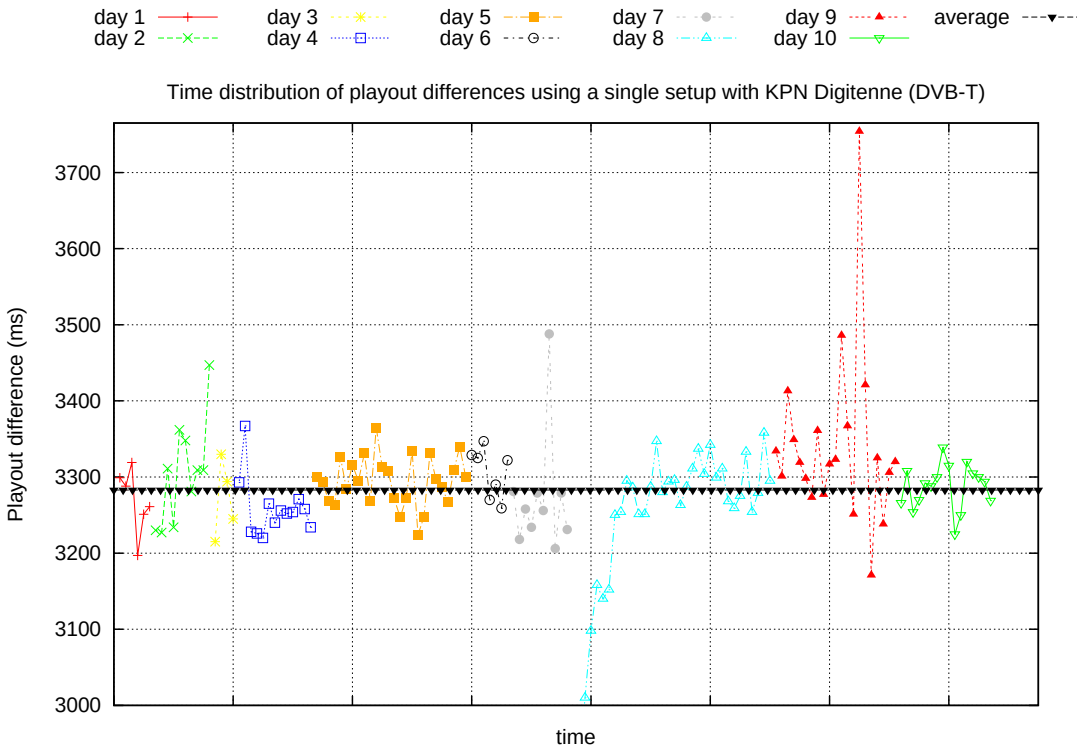


Figure 7.3.: Time distribution of playout differences with a single setup using KPN Digitenne (DVB-T) and the same TV .

the playout difference changes over a period of a few days in this specific case using DVB-T; it remains quite constant. The peaks in the graph are likely to be a inaccuracies from the measurement system itself, since the fluctuations fall more or less within the accuracy of the system, as discussed in chapter 6.

As mentioned, to obtain a better overview of the playout differences, more measurements are needed. To summarize, there are four reasons why more measurements are needed:

- Determine **extremes**, i.e. get a better overview of the smallest and largest possible delays introduced. Obtaining more measurements using different broadcaster and technology combinations (which have not been measured before) will allow for this.
- Determine the **influence of geographical location** on the playout differences. Multiple geographical distributed measurements for a single broadcaster and technology combination are needed to determine what the correlation between geographical location and the playout delay at that location is. Ideally, the measurements obtained for this purpose should all be performed at the same time to exclude the influence of the moment of time of the measurement as much as possible.
- Better determine how playout differences **differ over** an extended period of

time. Multiple measurements distributed over time for a single broadcaster and technology combination are needed to determine the influence of time on a playout delay.

- Ruling out **outliers** as much as possible. Despite the outlier filtering mechanism, there still is a small chance that the measurement system might give an outlier. More measurement will make it possible to easily pick out the outliers (either manually or automatically) and strengthen the results.

8. Conclusions and future work

8.1. Conclusions

In this report, a measurement device to measure relative playout difference (relative to the reference server) was designed, developed and tested. Also, a theoretical background was given of the architecture of TV broadcasting systems the delay introduced in these systems. The measurement system has been developed for the Android platform and is using fingerprinting technology to recognize audio content and NTP to obtain accurate timing information. These fingerprints are then compared with a live TV recording at a backend provided by the company Gracenote. Using this system, it is possible to measure playout differences within an accuracy of less than 300ms of their actual values, as was shown during the testing of the system. Sometimes however, the system measures an outlier. Using a series of multiple successive measurements, it is possible to filter out these outliers very well, and deliver a single playout value that is much more certain to be the actual value and no outlier. However, to completely verify this, a larger dataset is needed to test the assumption that the outliers are indeed statistically independent.

Furthermore, another interesting thing that was found is that GPS timing information on Android is not nearly as accurate as GPS technology would allow. GPS timing information can be hundreds of milliseconds off from the real time. Reasons for this might be that the smartphone devices have a focus on energy saving and power efficiency and that this comes at a cost of limited accuracy of the timing information provided by its internal GPS device. As opposed to GPS on Android, NTP in optimal conditions does provide an accurate reference time-source for the measurement system.

8.2. Answers to research questions

Sub research questions

1. Which factors in TV content delivery networks are introducing delay in the content delivery process and what is their influence?
2. How can delay in the content distribution chain of TV content be measured and how accurate can this be done?

3. What is the difference of playout times of television content between different content distribution techniques, channels, distributors and geographical locations in the Netherlands?

The answer to research question 1 is given in chapter 3 and chapter 4. In chapter 3, the elements in a TV broadcast chain were identified and in chapter 4, the most important elements that introduce delay were examined more closely. Encoding and decoding of video content are the largest factors in the delay introduced in these TV broadcasting chains. This is for example 3-4 seconds for HD content in the KPN broadcast delivering chain.

The second research question has been answered in chapter 5 and chapter 6. In chapter 5 is explained how the system is designed and created while in chapter 6s the system is tested for its accuracy. In this chapter is shown that the system can measure playout times with an accuracy of within 300 milliseconds of the actual value.

Finally, the last subquestion has been answered in chapter 7. In that section was shown that the playout times (playout differences compared with the reference) can range between around 200ms up to more than 4000ms. In other words, there can be a difference of almost 4000 milliseconds between the moments TV content is displayed on different setups. This can be more when more measurements are performed. To answer the last element mentioned in this subquestion, large scale measurements are needed to determine the geographical influence on playout delay.

Based on the information the main question can be answered. Recall that the main question was the following:

- Is it feasible to synchronize different types of broadcasted television signals of different television distributors by modifying broadcasting delays in a television broadcast chain?

Using the developed app it would be possible to measure playout delays of all TV broadcasters in the Netherlands within an accuracy of 300ms. If the fastest broadcaster would delay its broadcasting to allow for syncing with the slowest one, this would definitely possible. This way it would be possible to synchronize TV broadcasting in the Netherlands within less than 300ms from each other. So technically this would be possible. However, if this is policy wise possible, is a second question. Faster broadcasters would probably prefer to broadcast their own content as fast as possible. To really accomplish anything this way, an overarching initiative is necessary.

8.3. Future work

There are several things that are left to be done. First, a larger dataset with playout difference samples must be obtained to be able to verify the statistical independency

of the outliers provided by the measurement system. Furthermore, the measurement system can be publicly released to obtain playout difference measurements from multiple locations and countries in the world. Based on this, a comparison on the playout times between different locations and TV broadcasters can be made.

A. Instructions for measuring playout difference

A.1. Requirements

This appendix explains how the developed app can be used to measure playout differences. To app can be downloaded from the Google Play Store[1] by searching for the name "TV Speed". It can be installed like any other app from the Play Store.

To use the app, there are a few requirements:

- The TV needs to be tuned to one of the available channels. See the TV Channel dropdown menu under the TV Data tab for a list of available TV channels.
- An internet connection on the Android device. This is needed to communicate with the NTP-, Fingerprint-, EPG- and backend servers. A faster internet connection will allow for obtaining a more accurate reference timestamp, so a fast and stable connection is preferred.
- A microphone on the Android device.

A.2. Usage

In Fig. A.1, simple instructions for using the app can be found. When the app is started, the screen as displayed in the first item (1) in Fig. A.1 is shown. To get started the button on this screen can be clicked or the screen can be swiped to the right. This will show the screen that allows for inputting TV Data (2). Next, a TV channel has to be selected to be measured. The preferred channel to be measured is BBC One. This channel has the largest coverage of the available channels. Check the corresponding checkbox if this channel is broadcasted in HD quality. Also provide the TV broadcasting details here. TV broadcasters are arranged by country. If the subscription is not listed, select "other" and provide the right information. Make sure the information provided here is correct, otherwise the measurement has not much use. Press save when this is done. When done correctly, this will automatically make the screen go to the fingerprint part of the app, numbered as (4) in Fig. A.1.

[1] http://play.google.com

Before pressing the Start button, hold the device close to the TV which is playing the previously chosen supported channel with the audio of the TV enabled. Make sure the audio of the TV is loud enough and there is not much background noise. Holding the app very close to the TV speakers should provide the best result. Press the "Start" button to start fingerprinting. This will first query the NTP server a few times to obtain a time reference and after this it will start fingerprinting immediately. In optimal conditions, the app will recognize the channel being played pretty quickly. If everything is done correctly, the app will fingerprint and recognize the channel within less than 20 seconds. For better accuracy however, it will fingerprint the channel multiple times, and the total measurement time is around 2-3 minutes if everything goes well.

Next, the app will show a dialog whether the TV audio channel was recognized or not. In case no match was found, make sure you have followed the instructions correctly. Make sure you have the TV tuned to a supported channel, the audio of the TV is loud enough, there is not too much background noise, the Android device has an internet connection and the microphone of the Android device is not broken.

If a match is successfully found, confirm this is in the dialog and this will present the last screen of the app, which is numbered (5) in Fig. A.1. Press the "Calculate and submit" button to obtain the playout difference between your local TV and the TV reference. By clicking this button, the app will query an EPG (Electronic Programming Guide) server to obtain an absolute timestamp of the fingerprint match timestamp. Next, the timestamp obtained from the NTP server is compared with the timestamp from the fingerprint match combined with the EPG timestamp. This difference is the playout difference. Furthermore, the results are send to a database so they can be compared with playout differences from other channels, broadcasters, locations or other moments in time.

Help Us Gather Data On TV Delays!

[Android users only]

What?

We want to ask you to help us measure TV playout delays. We are gathering data on the playout delay differences of TV broadcast chains. In other words, what is the delay difference between countries, technologies and providers? By participating, you will not only help us, but also yourself, since the results will be published this year.

Why?

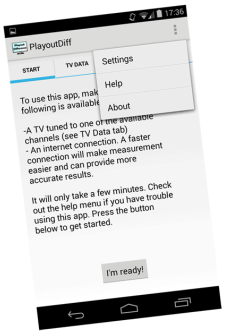
The results will be useable e.g. to see how much buffering is needed for second screen sync, for social TV, or in designing interactive game shows, etc.

How?

Measuring the playout difference can be done with an app we have created. A TV (with audio) and at least one of the channels "BBC One", "BBC Two" or "TV5 Monde Europe" is needed. Also an internet connection, preferrably Wifi is needed. The app is only available for Android.

Where?

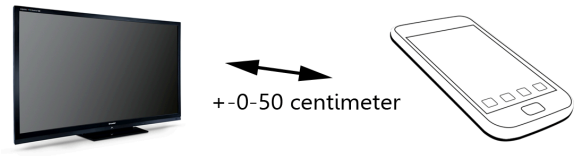
Download the app by scanning the QR code, going to tinyurl.com/tvspeed or by searching for "TV Speed" in the Google Play Store.



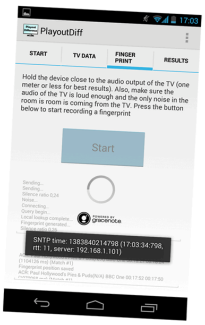
1. Measuring the playout difference with this app is really simple. First, install and start the app. You will see the screen on the left.

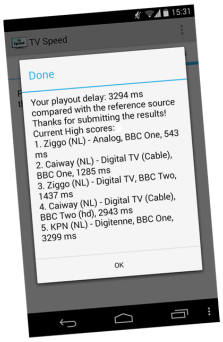
2. Provide the information of your TV subscription.

3. Hold your device close to the speakers of your TV.



4. Now press the start button and let the device perform the measurement. Make sure there is little environment noise when the telephone is recording the TV audio. When done, confirm the match.





5. If everything went ok, you can now press the calculate result button and obtain the playout difference. If something went wrong, check out the help menu in the app for more detailed feedback.

6. Now repeat the same for other TV subscriptions or TV on other devices (such as tablets) if you have them. Don't forget to change the TV data!

Thank you for participating!

Figure A.1.: Simple instructions for using the app

B. Testing Echoprint

B.1. Overview

This appendix gives insight in the process how Echoprint was tested if it can be used as an audio fingerprint engine for a TV playout difference measurement system. As has been mentioned in [41], Echoprint [19] is an open-source framework for audio identification. It allows for identifying audio based on fingerprints created from audio. Part of the goal of Echoprint is to identify audio based on microphone recordings of audio, to be able to provide a service such as Shazam or Soundhound. Information regarding the testing approach and the test results are presented here.

B.2. Testing criteria

Several requirements of a mobile playout measurement system have been identified in [41]. Based on these requirements, Echoprint was selected as a possible candidate. There are some more specific requirements for Echoprint however to be able to be of use, namely:

- An accurate over-the-air matching rate. A matching rate as high as possible would be desired. Having a matching rate of around 90% or more would be ideal, however a lower match rate of around 70-90% might also be sufficient. As long as not many false positives are provided, it would be possible to submit a new query when a match is not found. Or multiple queries might be submitted right after each other, allowing a match to be found faster.
- A low false positive rate. A false positive, i.e. when a matching is given when there is not an actual match, is very undesirable. Having false positives will give wrong measurements results and this is of course not desired.
- Ability to deal with a small sample size of audio fragments. The smaller the sample size the better. This would allow for quicker measurements. A sample size of 20 seconds or less would be desirable, however higher sample sizes could be acceptable if this could significantly improve the over-the-air matching rate.
- Matching partially overlapping fragments. Comparing (fingerprints of) fragments that are not based on the exact same piece of content (i.e. the same start and end-moment in the content) should not be a big obstacle for a fingerprint matching system.

B.3. Test setup

To be able to test the above criteria, several test-cases are defined. The tests consist of the following elements:

- An Echoprint server. This consists of a custom Apache SOLR (a server architecture with search capabilities) component, a database management system (Tokyo Cabinet) and interface (Tokyo Tyrant) and a fingerprinting matching API (written in Python)
- A code generator for creating fingerprints of audio. These fingerprints can be submitted to the server using JSON.
- A reference sample (without noise) file, consisting of a piece of audio that is split into pieces and ingested as references in the server database.
- A recorded sample (with noise) file of the same content as the digital sample, split into multiple pieces and used for querying the database. To systematically perform tests with varying sample length of both the digital and the recorded sample, several Linux bash scripts were written. A short description of these scripts is given next. These scripts are added in following sections.
- ffsplit.sh. This script takes as input a media file and splits it into multiple smaller media fragments of specified length and type.
- delete_and_split.sh. This script uses the ffsplit.sh script to split both the reference (digital) and the recorded sample into a specified amount of smaller ones. It also deletes files which are the result of a possible previous split action in the relevant directories.
- ingest_db.sh. This script first removes possible earlier ingested fingerprints on the server, it subsequently creates a list of the files created by the delete_and_split.sh script. Next, the files in this list are put into the code generator and the output in the form of JSON strings are saved and ingested in the server.
- get_matches.sh. This script fingerprints all the split files that were created from the recorded sample using the delete_and_split.sh script, queries these fingerprints to the server and saves the result.
- perform_test.sh. Finally, in this script, all of the above scripts are "glued" together to allow for performing multiple testing scenarios after each other in an automated fashion.

Different tests are performed with each having a different combination of lengths of the reference, length of the recorded sample and the duration of the reference file. This results in a different amount of reference samples in the database (i.e. amount of fingerprints), which will have an impact on the matching rate of system. This is because matching of fingerprints depend on the comparison of scores between multiple candidate matches. If the difference between multiple candidate matches is low, a fingerprint is less likely to be declared a match by the algorithm.

B.4. Test results

Journaal

In this test, the public broadcasting television news show, of the "Journaal" is used as a test subject. This is one of the most popular news broadcasts in the Netherlands. The length of these broadcasts is around 15-25 minutes. For this test, the length of the reference and also the recorded samples is varied. The reference sample file is split up into pieces of a length of 5, 10, 15, ..., 50, 55, 60. For each of these 12 reference sample sizes, the recorded sample size is also split up into pieces of the same length. So there are 12*12 = 144 scenarios for which the database is ingested and audio is matched against it. The sample of the Journaal is from the broadcast of 26-05-2013 and has a length of 16 minutes and 2 seconds. The fact that the Journaal is split into pieces of different lengths results in a database with different amount of reference pieces. The exact amounts are given in Tab. B.1: Sample length and amount of reference pieces . Note that the amounts mentioned here are slightly different than could be expected when dividing the duration by the sample length. This is because the splitting of the large recording and/or reference file into smaller pieces introduces inaccuracy. This should not have much impact on the results, because such a small difference in length of the reference sample pieces does not have a large impact on the matching rate.

Sample length	5	10	15	20	25	30	35	40	45	50	55	60
Reference piece count*	193	97	65	49	40	33	29	25	23	22	20	18

Table B.1.: Sample length and amount of reference pieces

By putting these amount of references into the horizontal and vertical axis of a table, multiplying each of these amounts with each other, the total number of fingerprints in these tests can be calculated as a total of 376996.

Matching vs. original source

The first test is matching parts of the reference vs. parts of the same reference. This will probably give a higher matching rate than matching versus a recorded version. The results are given in Tab. B.2. The values of the columns are the reference sample length, i.e. the sample length of fragments of which the fingerprints are ingested in the database. It should be noticed here that the diagonal axis has a 100% unique matching rate (except for 1 case where it is 97%, when matching fragments of 20 seconds vs. itself). This is because the matched fragment are completely the same. The exact same parts of the same length and spectrogram are matched. Another thing that is remarkable here is the fact that the matching rate for samples that are not the same isn't very high; almost all unique matching rates are below 50%

for these cases. This indicates that the matching algorithm has trouble to match partly overlapping fragments. Slight peaks can be noticed when the reference is a multiple of its query sample, or vice versa. However this correlation is not always present and is not very strong. The average matching rate in this table is 23,0%.

Recorded sample length (s) \ Reference sample length (s)	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**	**55**	**60**
5	100	39	24	21	15	12	12	9	8	7	6	6
10	56	100	60	12	26	10	14	7	8	6	9	5
15	7	46	100	32	35	40	26	15	10	12	13	20
20	4	34	36	97	24	44	18	46	18	22	10	30
25	5	7	20	25	100	17	17	12	15	51	10	10
30	6	6	21	27	21	100	18	27	36	24	9	48
35	3	3	7	7	14	17	100	14	14	10	10	10
40	8	8	8	8	12	32	16	100	12	20	12	36
45	4	4	9	9	4	22	18	13	100	13	9	27
50	10	10	5	5	15	10	10	20	15	100	10	20
55	5	5	5	5	5	5	5	16	11	11	100	11
60	11	5	11	11	5	17	9	17	23	23	11	100

Table B.2.: Unique matching rate of the reference vs itself of the Journaal of 26-05-2013

Matching vs. recorded version

The results of the measurements for this test are presented in Tab. B.3.

Something that can be noticed directly from this table are the low unique matching percentages. The highest matching percentage is 28% for the scenario with a reference sample length of 20 seconds and a recorded sample length of also 20. This value is not sufficient for a playout difference measurement system. The average matching rate of the whole table is 3,7%, this is a lot lower than the average of the matching of the original copy vs. itself (23,0% Tab. B.2), this seems to indicate that it is not able to deal very well with added noise obtained from the recording (over-the-air matching). Another thing that can be noticed is that a recorded sample length of 15-25 seems to be giving the best matching rates, although they still remain low. Furthermore the values that are in the diagonal axis seem to be also somewhat higher than other diagonal axes. This is logical however, because the values of the reference and the recorded sample have the same length. This means that the 20 seconds in the recorded sample represent the same 20 seconds of content in the reference sample, hence it gives a higher score. This is something that was also observed when matching against the original source.

Recorded sample length (s) \ Reference sample length (s)	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**	**55**	**60**
5	0	0	1	1	1	2	1	0	3	1	1	1
10	0	10	10	5	4	3	7	7	6	3	5	3
15	0	4	16	12	13	13	10	12	6	9	4	10
20	0	2	10	28	10	10	12	14	6	6	8	8
25	0	0	0	14	14	7	4	14	9	9	7	7
30	0	0	0	0	2	7	7	2	4	4	2	9
35	0	0	0	3	0	0	6	6	3	3	0	3
40	0	0	0	0	3	0	3	15	0	3	3	0
45	0	0	0	4	0	0	4	8	8	4	0	4
50	0	0	0	0	0	0	0	0	0	13	5	0
55	0	0	0	0	0	0	0	0	0	5	5	0
60	0	0	0	0	0	0	0	0	0	0	0	0

Table B.3.: Unique matching rate of a recording and a reference of the Journaal of 26-05-2013

"Journaal" fragments

This test uses a smaller part of a broadcast of the Journaal used above, namely the first 90 seconds. This is a more representative scenario, because broadcast playout difference will likely not be very large and 90 seconds is more likely to be the maximum playout difference between multiple TV sources. In Fig. B.1, the spectrogram of the original, digital and the recorded sample is given. Even though the sample was recorded at a "normal" audio volume and at a very close distance of around 20 centimeters to the source, the amplitudes in the spectrogram are much smaller in the recorded sample than in the original. This does not necessarily imply that information here is less useful because the frequency information might still be there.

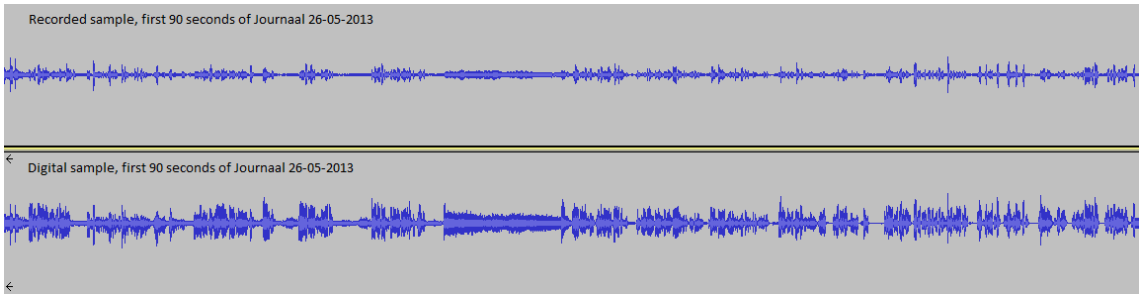


Figure B.1.: Spectrogram of the original, digital audio (below) and the recorded sample (above)

In Tab. B.4, the unique matching rate of a recording and the reference of the first 90 seconds of the Journaal of 26-05 are given. Again, the unique matching rate

percentages are low here. Note that the total amount of comparisons here is lower than in the previous test (24025 vs. 376996). This is because the total length of all the fragments is much smaller (90 seconds vs. 16 minutes and 2 seconds). In this test, the fragments were split up in smaller intervals. For example, for the query fragments of 6 seconds versus the reference sample length of 10 seconds, fingerprints of 90/6=15 fragments were matched against 90/10=9 reference fingerprints. Which gave a matching rate of only 6%. Despite the fewer matches, the matching rate still remains low. The total average matching rate of the whole table is 11,1%. This is a lot more than the average of the matching rate of matching fingerprints of fragments of the original version of the Journaal versus its recorded copy (Tab. B.3, 3,7%). Even though this includes many combinations of query and reference fragments sizes which are not efficient, this is still a very low percentage.

Query fragment length \ Reference sample length (s)	2	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**	**26**	**28**	**30**
2	11	0	0	0	0	0	0	0	2	0	0	0	2	0	0
4	0	0	4	4	4	0	4	0	0	0	0	4	4	13	4
6	6	16	0	6	6	13	6	13	6	0	6	0	0	6	0
8	8	0	8	8	8	33	25	0	25	0	0	8	8	16	8
10	0	12	0	11	10	0	11	22	22	11	0	11	11	0	11
12	0	14	12	12	25	37	12	0	25	12	0	12	12	0	12
14	14	14	14	14	0	28	28	28	0	14	0	0	14	42	14
16	16	16	16	16	33	33	0	66	0	16	33	0	16	33	0
18	0	0	0	20	0	0	40	20	40	60	20	0	40	40	40
20	0	0	0	0	20	20	0	40	20	60	0	20	20	0	0
22	20	0	0	0	0	0	0	0	20	0	20	20	20	0	0
24	0	0	0	0	0	0	0	25	0	25	25	25	25	25	25
26	0	0	0	0	25	0	0	25	25	25	0	25	0	25	25
28	25	25	25	25	25	25	25	0	0	0	0	0	0	25	0
30	0	0	0	0	0	0	0	33	33	33	0	33	33	33	0

Table B.4.: Unique matching rate of a recording and a reference of the first 90 seconds of the Journaal of 26-05-2013

Conclusions

The results of the tests above show that Echoprint lack some properties that are required for a good playout measurement system. The test where the original copy of the Journaal is matched against itself shows that Echoprint is not able to deal very well with fragments that are only partially overlapping. It did show that it was very good at matching fragments that are the same, but when the fragments were different, the matching rate was quite low. Also, when comparing the results of the matching of the Journaal with itself vs. the results of the matching of the original

copy with a recorded version, the results are much lower (3,7% vs. 23,0% average matching rate). This seems to indicate that it is not able to deal with added noise very well at all. It could be however that this big difference is also (partially) caused by the fact that it is not good at handling fragments that are not exactly overlapping, since the recorded version might not be 100% overlapping since it was recorded and cut manually. On the other hand, Tab. B.3 does show that the diagonal though the middle where recorded fragments have exact the same size as the fragments of the original has a higher matching rate than any other diagonal. That shows that it is somewhat "outlined". Either way, these matching percentages do not indicate that Echoprint in the current status is suitable for creating a reliable playout difference measurement system. Even though these results were not positive, another test was executed which was more suitable to the situation of a playout measurement system. This test consisted of matching fingerprints created from fragments of only the first 90 seconds of the same Journaal broadcast. These results were better however, but no not that change the conclusion. Echoprint is open-source however, and there are ways to improve this accuracy by modifying the matching algorithm. However, doing so would require more time and a more in-depth study of the workings of audio fingerprinting algorithms and that would be out of the scope of my research.

C. Backend API

For the purpose of saving information in the database on the backend server, the server makes use of PHP/MySQL and JSON to communicate with the Android app. The API created for this purpose and corresponding parameter information can be found in Tab. C.1 and Tab. C.2 respectively. The URL and API key to access this service are not published here.

Function	Result, JSON
Submit results	**Input:** action="submit"
	Required parameters: action, playout_diff, country, provider, channel, program, sub_type, hd, loc_lat, loc_long, gps_age, device_info, app_version
	Output: result="succes"
	Comment: submits the provided measurement results to the database
Request most recent	**Input:** action="request", req_type="most_recent"
	Required parameters: action, req_type
	Output: JSON Array of length 0 ... 10. Contains values: id, timestamp (format: YYYY-MM-DD HH-mm-ss), playout_diff (milliseconds), provider, channel, sub_type, hd (true/false)
	Comment: Requests the most recent playout measurements recorded in the database
Request same distributor	**Input:** action="request", req_type="same_provider"
	Required parameters: action, req_type, provider
	Output: see output of request most recent
	Comment: Requests the most recent playout measurements recorded in the database for the given provider
Request same subscription	**Input:** action="request", req_type="same_sub"
	Required parameters: action, req_type, sub_type
	Output: see output of request most recent
	Comment: Requests the most recent playout measurements recorded in the database for the given subscription type
Request same channel	**Input:** action="request", req_type="same_channel"
	Required parameters: action, req_type, channel
	Output: see output of request most recent
	Comment: Requests the most recent playout measurements recorded in the database for the given channel
Request same setup	**Input:** action="request", req_type="same_setup"
	Required parameters: action, req_type, country, provider, sub_type, channel, hd
	Output: see output of request most recent
	Comment: Requests the most recent playout measurements recorded in the database for the given country, provider, sub_type, channel and hd combination
Request nearby	**Input:** action="request", req_type="nearby"
	Required parameters: action, req_type, loc_lat, loc_long, radius, id
	Output: see output of request most recent
	Comment: Requests the most recent playout measurements recorded in the database in the given radius for the given latitude and longitude (decimal notation) coordinate pair. id can optionally contain the value of the id to exclude from being delivered by this request
Request fastest	**Input:** action="request", req_type="fastest"
	Required parameters: action, req_type
	Output: Array of length 0 ... 5, see output of request most recent for format
	Comment: Returns the fastest setup combinations

Table C.1.: Result server API

Parameter	Comments
playout_diff	Measured playout difference, value in milliseconds.
country	Land code of the country the measurement was performed in. Format: ISO 3166-1 [42].
provider	Provider of the TV signal.
channel	Measured channel.
program	Program broadcasted on the channel during the measurement.
sub_type	Subscription type/technology provided by the provider.
hd	Indicates if the measured was performed on a HD or SD signal. A value of false indicates an SD signal, a value of true indicates an HD signal.
loc_lat	Latitude of the coordinate where the measurement was performed.
loc_long	Longitude of the coordinate where the measurement was performed.
gps_age	Age of the last obtained GPS fix, value in milliseconds.
device_info	Information about the device used to obtain the measurement.
app_version	Version number of the app used to obtain the measurement.

Table C.2.: Parameter information

Bibliography

[1] R. Mekuria, P. Cesar, and D. Bulterman, "Digital TV: The Effect of Delay when Watching Football," in *Proceedings of the 10th European conference on Interactive TV and video*, 2012.

[2] STEER, "STEER: Exploring the dynamic relationship between social information and networked media through experimentation." Internet: `http://www.fp7-steer.eu/`, 2013. Retrieved October 21, 2013 from http://www.fp7-steer.eu.

[3] van Deventer, O. and Stokking, H.M. and Niamut, O.A. and Walraven, F.A. and Klos, V.B., "Advanced Interactive Television Services Require Content Synchronization," *International Conference on Systems, Signals and Image Processing*, vol. 14, pp. 1–6, 2008.

[4] E. D'heer, C. Courtois, and S. Paulussen, "Everyday life in (front of) the screen: The consumption of multiple screen technologies in the living room context," in *EuroiTV '12 Proceedings of the 10th European conference on Interactive tv and video*, pp. 195–198, 2012.

[5] M. Lochrie and P. Coulton, "Tweeting with the telly on!," in *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, 2012.

[6] D. Geerts, I. Vaishnavi, R. Mekuria, O. van Deventer, and P. Cesar, "Are we in sync?: Synchronization requirements for watching online video together.," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, (New York, NY, USA), pp. 311–314, ACM, 2011.

[7] F. Boronat, R. Mekuria, M. Montagud, and P. Cesar, "Distributed media synchronisation for shared video watching: Issues, challenges and examples," in *Social Media Retrieval* (N. Ramzan, R. Zwol, J.-S. Lee, K. Clüver, and X.-S. Hua, eds.), Computer Communications and Networks, pp. 393–431, Springer London, 2013.

[8] R. Mekuria, P. Cesar, and D. Bulterman, "Digital tv: The effect of delay when watching football," 2012.

[9] R. Mekuria, "Inter-destination synchronization for tv-broadcasts," Master's thesis, Delft University of Technology, 2011.

[10] Civolution, "Civolution." Internet: `http://www.civolution.com/`, december 2013. Retrieved December 11, 2013 from http://www.civolution.com/.

[11] TvTak, "Tvtak." Internet: http://www.tvtak.com/, december 2013. Retrieved December 11, 2013 from http://www.tvtak.com/.

[12] Gracenote, "Gracenote." Internet: http://www.gracenote.com/, 2013. Retrieved November 11, 2013 from http://www.gracenote.com.

[13] R. ter Horst, "KPN ITV television distribution chain." Personal communication, july 2013.

[14] A. Troost, "KPN television distribution chain." Personal communication, july 2013.

[15] U. Reimers, *DVB The Family of International Standards for Digital Video Broadcasting*. Springer, 2006.

[16] R. Schreier and A. Rothermel, "A Latency Analysis on H.264 Video Transmission Systems," in *International Conference on Consumer Electronics (ICCE)*, 2008.

[17] A. Wang, "Whitepaper - an industrial-strength audio search algorithm," 2003.

[18] C. L. Y. Duong, N.; Howson, "Fast second screen tv synchronization combining audio fingerprint technique and generalized cross correlation," in *IEEE International Conference on Consumer Electronics*, 2012.

[19] Echoprint, "Open Source Music Identification." Internet: http://echoprint.me, 2013. Retrieved October 4, 2013 from http://echoprint.me.

[20] Audible Magic, "Audible Magic - Digital Fingerprint Content & Media Recognition." Internet: http://audiblemagic.com/, 2013. Retrieved November 11, 2013 from http://audiblemagic.com.

[21] Gracenote, "Gracenote Entourage." Internet: https://developer.gracenote.com/entourage, 2013. Retrieved November 11, 2013 from http://developer.gracenote.com.

[22] http://www.norad.mil/, "North American Aerospace Defense Command." Internet: http://www.norad.mil/, 2013. Retrieved November 18, 2013 from http://www.norad.mil.

[23] W. Lewandowski, J. Azoubib, and W. Klepcynzski, "GPS: Primary Tool for Time Transfer," in *Proceedings of the IEEE*, 1999.

[24] National Marine Electronics Association, "NMEA 0183 standard." Internet: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp, 2013. Retrieved November, 2013 from http://www.nmea.org/.

[25] K. Yaghmour, *Embedded Android*. O'Reilly Media, 2013.

[26] Android Developers, "SystemClock." Internet: http://developer.android.com/reference/android/os/SystemClock.html, 2013. Retrieved October 4, 2013 from http://developer.android.com.

[27] Android Developers, "Parcel." Internet: http://developer.android.com/reference/android/os/Parcel.html, 2013. Retrieved October 4, 2013 from http://developer.android.com.

[28] D. Mills, "Internet Time SynchrSynchroni: The Network Time Protocol," *IEEE Transaction on Communications*, vol. 39, 1991.

[29] Mills et al., "RFC: 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification," 2010.

[30] D. Millis, "Executive Summary: Computer Network Time Synchronization." Internet: http://www.eecis.udel.edu/~mills/exec.html, May 2012. Retrieved November 7, 2013 from http://www.eecis.udel.edu.

[31] NTP Pool Project, "The internet cluster of NTP servers." Internet: http://www.pool.ntp.org/en/, 2013. Retrieved November 11, 2013 from http://www.pool.ntp.org.

[32] D. Mills, "RFC 1589: A Kernel Model for Precision Timekeeping," 1994.

[33] e. a. Mogul, "RFC 2783: Pulse-Per-Second API for UNIX-like Operating Systems, Version 1.0," 2000.

[34] J. Wharton, "Actionbar Sherlock." Internet: http://actionbarsherlock.com, 2013. Retrieved November 18, 2013 from http://actionbarsherlock.com.

[35] W. Kelly, "Monitoring Commands and Options." Internet: http://www.eecis.udel.edu/~mills/ntp/html/monopt.html, 2013. Retrieved November 11, 2013 from http://www.eecis.udel.edu/ mills/ntp/html/monopt.html.

[36] xda developers forum, "Understanding Android GPS Architecture." Internet: http://forum.xda-developers.com/showthread.php?t=2063295, 2013. Retrieved October 4, 2013 from http://xda-developers.com.

[37] ars technica, "Google's iron grip on Android: Controlling open source by any means necessary." Internet: http://arstechnica.com/gadgets/2013/10/googles-iron-grip-on-android, 2013. Retrieved December 19, 2013 from http://arstechnica.com.

[38] Navstar, "Navstar GPS user equipment introduction," tech. rep., 1996.

[39] Joint Committee for Guides in Metrology (JCGM), "International vocabulary of metrology," 2008.

[40] Audacity, "Audacity." Internet: http://audacity.sourceforge.net/, 2013. Retrieved November 11, 2013 from http://audacity.sourceforge.net/.

[41] W. Kooij, "Second screen device synchronization techniques and frameworks." 2013.

[42] International Organization for Standardization (ISO), "ISO 3166-1 alpha-2." Internet: http://www.iso.org/iso/country_codes/iso_3166_code_lists/country_names_and_code_elements.htm, 2013. Retrieved October 7, 2013 from http://www.iso.org.