IMPROVING RELIABILITY OF STREAMING RADIO

A.J.K. de Vries

MASTER THESIS TELEMATICS
11 DECEMBER 2008

SUPERVISING COMMITTEE
Dr.ir. G. Karagiannis (University of Twente)
Dr.ir. G.J. Heijenk (University of Twente)
Ing. W. Temkov (Mobilaria)

DESIGN AND ANALYSIS OF COMMUNICATION SYSTEMS
FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS, AND COMPUTER SCIENCE
UNIVERSITY OF TWENTE
Abstract

This Master Thesis describes the evolution of a solution for the reliability problems regarding the streaming radio service of Mobilaria. It addresses the problems that were made visible by a preliminary research, and proposes multiple solutions to these problems. One of the solutions, a bitrate switching decision algorithm to prevent buffer underruns caused by insufficient available bandwidth, is chosen to be developed into a proof of concept application, based on the iPhone version of Mobilaria’s multimedia player. This version of the player already contains a bitrate switching protocol that is internally developed by Mobilaria, which will be the basis for the bitrate switching decision algorithm. The decision algorithm is specified using a mathematical model, and the design and implementation of this algorithm into a proof of concept application is discussed. Thereafter a couple of tests are specified and performed to validate the performance improvement of the proof of concept application with respect to the currently used Symbian version of the multimedia player.
Preface

For over a year now, I have been working on my Master’s assignment at Mobilaria in Enschede. In the summer of 2007, my study colleague Martin Prins introduced me at Mobilaria for a brainstorm session regarding a possible Master’s assignment. Immediately, there was a mutual click, and although a precise assignment was not formed yet, we decided to give it a go. Following on this decision the graduation committee was formed, which consisted of Georgios Karagiannis as my first supervisor and Geert Heijenk as my second supervisor from the University of Twente. On behalf of Mobilaria, Wanjo Temkov would guide me through the process of successfully completing the assignment. After forming the committee, it was now time to create an assignment description that would convince both the graduation committee as well as the University’s exam committee. After having entered not less than eight versions of this assignment description for review, the moment was finally there that both committees approved the assignment and that I could finally start. At least, with the first part of the assignment: the preliminary research. Because I was doing a 40 ECTS assignment, it was split into two parts: the preliminary research and the part that would be known as the actual Master’s assignment. In February the preliminary research was completed and I could start with the Master’s assignment, from which the resulting thesis is in now front of you.

First and foremost I am grateful to my academic supervisors Georgios and Geert, for their critical but positive feedback and for their believe in me from the beginning of this assignment, and to Mobilaria for the opportunity they provided me and for their support and patience during this period. I would like to thank Wanjo, who always provided me with lots of ideas and insight, and who stimulated me to produce even better results. I would also like to thank Harreld, who provided me with all the technical support I needed in order to bring this assignment to a success. Besides them, of course my thanks goes out to my colleagues at Mobilaria, Eric, Chiel, Wim and Wouter, without whom it wouldn’t be as much fun working on this assignment as it has been now. Special thanks also goes out to Lucia Cloth for helping me creating the mathematical model.

Furthermore, my gratitude goes out to my parents Sven and Ineke, who not only gave me a good youth, but also supported me through my entire study. Last but certainly not least, I would like to thank my girlfriend Berdi, my family and all my wonderful friends, who have made being me so much fun.

Arne de Vries

Enschede,
11 December 2008
7. TESTING .................................................................................. 41
   7.1 CONTROLLED ENVIRONMENT 41
       7.1.1 TEST CASE 41
       7.1.2 TEST RESULTS 44
   7.2 REAL-LIFE ENVIRONMENT 46
       7.2.1 TEST CASE 46
       7.2.2 TEST RESULTS 48
   7.3 DEMONSTRATION 52

8. CONCLUSIONS AND RECOMMENDATIONS .............................................. 55
   8.1 CONCLUSION 55
   8.2 RECOMMENDATIONS 56
       8.2.1 BITRATE TRANSITIONS 56
       8.2.2 VERTICAL HANDOVERS 56

REFERENCES ......................................................................................... 57
1. Introduction

This chapter describes the reasons for creating this document in section 1.1, the goals that are to be achieved in section 1.2, the approach that is taken to achieve these goals in section 1.3, and finally the structure of the rest of this document in section 1.4.

1.1 Motivation

Mobilaria offers a service with which users can listen to radio stations using the Internet connection of their mobile phone. In order to realize this, Mobilaria has developed a multimedia player that connects to their streaming servers using one of the Internet connections available on the mobile device. Most of the users are using 2G to 3.5G technologies like GPRS, UMTS and HSDPA to listen to Mobilaria’s streaming radio service. These wireless access technologies are currently not reliable and stable enough to match the requirements for a seamless audio experience, especially when being mobile. It often happens that for some reason the Internet connection is disrupted, disconnected, or decreased in quality, resulting in a disruption of the audio and thereby a degradation of the experience of the user. Minimizing the number of times the music that the user is listening to is disrupted should maximize the user’s audio experience.

The multimedia player of the streaming radio service uses TCP to stream audio data. This data is stored in a buffer to be able to overcome fluctuations in the data rate. The buffer should at all times contain audio data: if it runs empty (a buffer underrun occurs), the music will stop and the multimedia player will try to reconnect to the server. Preliminary research performed to find out the reasons for these buffer underruns [7] shows that erroneous horizontal handovers take up one sixth of the number of buffer underruns. Erroneous horizontal handovers are handovers between different cells with the same access technology that in some way disrupt the ongoing data communication of the mobile device. A disconnection of the source input stream at the server side causes one quarter of the buffer underruns. In the case of one sixth of the investigated buffer underruns, a horizontal handover took place, but the handover could not be clearly identified as the reason for the buffer underrun. For the rest of the underruns no clear reason was identified by the preliminary research. Since network operators control the access networks and thus third parties like Mobilaria or end-users cannot influence the way these networks work, solutions for the problem of buffer underruns have to be found in a local or end-to-end approach.

1.2 Goals

The goal of this assignment is to increase the user’s audio experience by providing a solution for the radio stream disruptions caused by buffer underruns and thereby increasing the stability and robustness of the streaming radio service. In order to reach this goal, the assignment is focused on answering the following research question:

*Which solutions are possible to solve the radio stream disruptions caused by buffer underruns?*

After proposing several possibilities for a solution, the following sub question will be answered in order to be able to select an appropriate solution for the problem:

*Which criteria can be identified to efficiently select one of the proposed solutions?*

After selecting the most satisfying solution, it will have to be designed and implemented into a prototype using Mobilaria’s current multimedia player. In order to do this, the following sub question will be answered:
How could the selected solution be designed and implemented into the current multimedia player?

In order to evaluate the improvements of the selected and implemented solution compared to the current version of Mobilaria’s multimedia player, the following sub question will be answered:

What are the performance advantages of using the selected solution within the multimedia player?

1.3 Approach

In order to successfully complete the assignment, at first preliminary research is performed to investigate the causes of the buffer underrun problems of Mobilaria’s streaming service. Using this research, several solutions to the found problems are proposed, from which one will be selected for this assignment. For this solution, an investigation of related work is performed, and after that a detailed specification of the solution is created. From this specification, a design and implementation into a proof of concept is derived, and the proof of concept application will be tested in order to evaluate it and compare it to the currently used multimedia player.

1.4 Structure

The structure of this report is based on the roadmap of the approach given in the previous section. First of all, chapter 2 gives the problem description by presenting an overview of the results of the preliminary research. Different solutions to these problems are proposed in chapter 3, where also the choice for one of the solutions is provided. Chapter 4 presents a study for related work, and chapter 5 gives a detailed specification of the chosen solution. In chapter 6 the process of designing and implementing this solution into a proof of concept application is described, and the testing of this proof of concept application is discussed in chapter 7. Finally, chapter 8 concludes this thesis and gives recommendations for future work.
2. Problem Description

This chapter describes the problem description that is the basis for this Master thesis. Section 2.1 outlines the general problem Mobilaria’s current system is coping with, section 2.2 gives a detailed overview of the reasons for this problem.

2.1 General Problem

As stated in section 1.1, it often happens that Mobilaria’s streaming radio service suffers from buffer underruns that decrease the user’s listening experience. A buffer underrun means that the buffer containing the audio data to play has run empty and hence the music will stop playing. Figure 1 shows a schematic view of the structure of Mobilaria’s streaming radio service.

![Figure 1: Structure of Mobilaria’s streaming radio service](image)

In this figure, there are several points in the chain where something can go wrong. First of all, there is the streaming server of the radio station. If this server for some reason stops working properly the creation of audio data is stopped. Then there is the connection between the streaming server of the radio station and Mobilaria’s transcoding server, which can get interrupted or broken, or Mobilaria’s transcoding server or streaming server can break down. A very important point of failure is the connection of the mobile device to Mobilaria’s streaming server. Since this connection is going through a mobile Internet connection, it suffers from packet loss, handovers, reception issues, etc. Finally, there is the mobile device itself and the client application running on the mobile device that could, although very unlikely, for some reason contain erroneous behavior causing the buffer to run empty.

2.2 Underrun Reasons

Preliminary research performed to find out the reasons for buffer underruns occurring in the multimedia player [7] shows the distribution displayed by Figure 2. For this research, real-life tests were performed and player logs were analyzed, for more information see [7].
Erroneous horizontal handovers (handovers between different cells with the same access technology that in some way disrupt the ongoing data communication of the mobile device) take up one sixth of the number of buffer underruns. Another sixth of the investigated buffer underruns was preceded by a horizontal handover, but the handover could not be clearly identified as the reason for the buffer underrun. In the case of one quarter of the buffer underruns the cause is a disconnection of the source input stream at the server side. For the rest of the underruns no clear reason was identified by the preliminary research.

2.2.1 On-Demand System
Mobilaria uses an on-demand system for content delivery, which means audio streams are started when a user issues a request for it. This design choice is made to save server system resources and to be able to offer a large number of radio stations to the users. The disadvantage of this system is that when a stream is started, a pre-buffer has to be created at the server. In the meanwhile, the multimedia player has to wait for this buffer to become large enough before it can start downloading audio data and start playing. To make sure a user does not have to wait too long, the middle way is taken between having a pre-buffer that is too small and waiting for over a minute before starting to play. This means that when a user requests a stream that is not already started at the server, the time it takes before the multimedia player starts playing is a bit longer than normal, but the maximum percentage the buffer can be filled lies around 40 percent, which increases the risk for buffer underruns because the multimedia player has less time to recover from packet loss and other network errors. Figure 3 shows an example of this behavior.
In Figure 3 can be seen that the buffer size does not grow higher than around 40 percent because of the small pre-buffer at the server side. A disruption in the data flow like at around 15:41:30 can more easily result in a buffer underrun than when the buffer fluctuates between 75 and 100 percent, which is the normal behavior with a large pre-buffer.

### 2.2.2 Source Stream Disconnect

A problem that is impossible to control for Mobilaria, is the disconnection of a source stream. At this moment, when a source stream is disconnected, the server side buffer runs empty and the stream that is send to the client is appended by an end-of-file packet. The buffer at the client’s multimedia player will run empty eventually as well, and the playback of the stream is stopped when the end-of-file packet is reached. This type of buffer takes up one quarter of the total amount of investigated buffer underruns.

### 2.2.3 Loss of Connectivity

One sixth of the investigated buffer underruns is preceded by a handover after which no data is received anymore. Figure 4 shows a situation where two handovers take place after which the connectivity is lost completely.

Another sixth of the investigated buffer underruns is preceded by a handover after which still data was received (but unfortunately not enough) or which occurred too late to identify the
handover as the cause for the buffer underrun. Figure 5 shows a distribution of the decrease rate of the buffer after a handover. The decrease rate is defined as the rate in which the buffer decreases, relatively to the bitrate: a decrease rate of 1 means that the buffer is decreasing with the same rate as the bitrate, meaning no more data is received at all. This distribution shows which percentage of all the analyzed buffer underruns was preceded by a decrease of the buffer fill with the given rate, in steps of 0.05.

![Figure 5: Decrease rate distribution of buffer underruns caused by a handover](image)

What can be seen from this graph is that most buffer underruns preceded by a handover have a decrease rate of 1, which means no data is received anymore after the handover. A decrease rate smaller than 1 means data was still received but not enough to fill the buffer at a rate of at least the bitrate of the stream. Decrease rates larger than 1 are theoretically not possible (this would mean the buffer is emptied at a rate larger than the bitrate), but are occurring because of inaccuracies in the measurement.

For almost half of the investigated buffer underruns no clear reason can be identified. Some of these buffer underruns are caused by a complete loss of connectivity, without a handover preceding it. Figure 6 shows an example of this event.

![Figure 6: Buffer underrun without clear reason](image)
2.2.4 Insufficient Bandwidth

Another problem Mobilaria’s streaming radio service suffers from, is that mobile devices do not always have sufficient bandwidth to be able to stream fast enough to match the bitrate of the stream. This problem is partly solved by buffering, but this is often not enough to prevent the multimedia player from ending up in a buffer underrun. For almost half of the investigated buffer underruns no clear reason can be identified. In the previous section the complete loss of connection was mentioned, but it often occurs that there is data connectivity but, as said before, not enough bandwidth to be able to stream fast enough to match the bitrate of the stream. Figure 7 shows the decrease rate distribution of the investigated buffer underruns that were not preceded by a handover. Again, the decrease rate is defined as the rate in which the buffer decreases, relatively to the bitrate, and plotted in steps of 0.05.

![Figure 7: Decrease rate distribution of buffer underruns not preceded by a handover](image)

What can be seen in this graph is that, like in Figure 5, a lot of buffer underruns have a decrease rate of 1, which means no data is being received anymore preceding the buffer underrun. The difference between the buffer underruns that are preceded by a handover is that in this case a lot of underruns occur with a decrease rate between 0 and 1, meaning that there is data connectivity but not enough to be able to match the bitrate of the stream.
3. Solutions and Choice

This chapter proposes different solutions in section 3.1 to the problems described in chapter 2. Section 3.2 gives identifies the criteria for choosing one of the proposed solutions.

3.1 Solutions

This section proposes possible solutions regarding the on-demand system, source stream disconnects, reconnects, and bitrate switching in sections 3.1.1, 3.1.2, 3.1.3, and 3.1.4. It addresses the following sub research question presented in section 1.2:

*Which solutions are possible to solve the radio stream disruptions caused by buffer underruns?*

3.1.1 On-Demand System

The on-demand system currently in use by Mobilaria increases the risk of a buffer underrun, because of the small pre-buffer at the server side (see section 2.2.1). The reason for this is that the audio streams are not always being buffered at the server side, but only when a user issues a request for it. In order to solve this problem, the size of the pre-buffer at the server should be increased.

A possible way to increase the pre-buffer at the server side is by continually buffering the external streams, and only start transcoding them when a user issues a request. This way there is always a completely filled buffer at the server side, and when a stream is requested the pre-buffer containing the transcoded data can quickly be created. The original idea behind the on-demand transcoders stays intact: reducing the use of system resources. The disadvantage of this solution is that always buffering a lot of external streams produces a lot of traffic, which means higher traffic costs.

Another solution is to prepend a commercial, jingle or some sort of waiting tune to the audio stream. This way a pre-buffer containing transcoded data can quickly be generated at the server side, resulting in a decrease of waiting time or an increasing amount of data that can be buffered at the multimedia player. The disadvantage of this solution is that by presenting something to the user other than the requested stream decreases the user experience of the service.

3.1.2 Source Stream Disconnect

One of the problems causing buffer underruns is that of the source stream disconnects, mentioned in section 2.2.2. When a source stream is for some reason disconnected, the stream ends and the multimedia player stops playing after all data is played and a buffer underrun occurs. After this the player tries to reconnect to the stream, and when the source stream is connected again this will be successful.

A possible solution for this kind of buffer underrun is to append some sort of source disconnection message at the end of the stream at the server side instead of ending the stream. This message could be e.g. a commercial message, a waiting tune, a spoken message or just plain silence. With this solution the buffer at the server does not run out of data, and the player can continue downloading audio data. The disadvantage of this solution is that the user does not hear the actual audio content it should hear, and is therefore a decrease in user experience. However, a disconnection of the source stream means in every case that the user cannot listen to the actual content, because it is not available. Another disadvantage of this solution is that when the source stream is connected again, the disconnection message has to be completely played, because otherwise the problem of a small server side pre-buffer occurs.
Another solution for this problem is when the source stream disconnects, the stream is ended at the server side, but instead of playing until a buffer underrun occurs, the player could try to reconnect to the stream and continue where the stream left off when a reconnection is made. Disadvantages of this solution are that there will be a gap or a jump in the content, the pre-buffer will be smaller than before and there is still a risk of running out of data in the buffer.

### 3.1.3 Reconnect

As stated in section 2.2.3, one of the problems causing buffer underruns is a complete loss of connectivity. At this moment it is unclear what the exact reason is for this problem, but it is likely to assume that something goes wrong somewhere in the underlying network. Since this is a problem that can only be controlled by the network operators and not by third parties like Mobilaria, a different solution for this problem has to be found.

A possible solution when connectivity is completely lost is to close all currently active TCP connections and to reconnect to the server. Optionally even the GPRS or UMTS connection can be closed and setup as well, however this can be too time consuming and is very complex to implement because the way to control connectivity differs for every operating system. If reconnecting to the server is performed within the available playback time in the buffer, the audio playback can continue without gaps and without the user knowing about it, so no degradation of the user experience occurs. It is also possible to temporarily switch to a lower bitrate, which is described in more detail in section 3.1.4.

The main disadvantage of this solution is that it is unknown whether creating a new TCP connection with the server or even a completely new GPRS or UMTS connection is useful to regain connectivity again. Further field research has to be performed in order to investigate this.

### 3.1.4 Bitrate Switching

Section 2.2.4 describes the problem of insufficient bandwidth. When the throughput of the link is lower than the bitrate of the stream, the buffer will slowly run empty and eventually a buffer underrun will occur. A possible solution to cope with this problem is to adapt the bitrate of the stream to the available link quality. Internal research performed by Mobilaria delivered a protocol that is capable of switching between streams containing the same content encoded at a different bitrate.

In case of a throughput insufficient to match the bitrate of the stream, a switch can be made to a stream with a lower bitrate. This way the buffer does not slowly run out of playable audio data. In the same way the bitrate of the stream can be increased when the throughput of the link allows it. This way the audio is of higher quality and thus the user experience is increased.

When the throughput of the link is just enough to match the bitrate of the stream, but the buffer is not completely filled, it is possible to temporarily lower the bitrate of the stream in order to be able to get to a higher buffer fill level. Afterwards a switch to the previous bitrate can be made again to increase user experience.

### 3.2 Solution Choice

From the different solutions provided in section 3.1, a choice has to be made which solution will be further developed for this project. To make this choice, the following sub research question presented in section 1.2 has to be addressed:

*Which criteria can be identified to efficiently select one of the proposed solutions?*
Since this is a scientific project, the chosen solution needs to have some scientific value and cannot be a simple implementation project. The two solutions described in sections 3.1.1 and 3.1.2 are more of a practical nature and thus do not qualify for an academic assignment, and are therefore left outside the scope of this project. The most challenging and ‘intelligent’ solution is definitely the bitrate switching solution described in section 3.1.4. Creating this solution means that there will be some form of reconnecting to different streams, so at the same time the reconnection solution described in section 3.1.3 will be covered.

A protocol that enables synchronizing multimedia streams with different bitrates, which is internally developed by Mobilaria, can be used in order to realize the bitrate switching solution. This so called Switch Protocol gives the possibility to create a new connection to the streaming server, to request an audio stream containing the same content but encoded in a different bitrate. Using sequence numbers embedded into the audio stream, the two different streams can be glued together at exactly the right moment, making it impossible for the human ear to here the switch, except for a change in audio quality. To use this protocol efficiently, a decision algorithm has to be developed which automatically decides when to switch to another bitrate. The following chapters will discuss some background information from related work, give a specification of this decision algorithm and finally the design and implementation into a proof of concept application.
4. Related Work

This chapter gives an overview of other work related to the solution chosen in section 3.2. Sections 4.1 and 4.2 give two examples of standardizations of content adaptation: the MPEG-21 Digital Item Adaptation standard and the 3GPP Release 6 standard. Sections 4.3 and 4.4 give two examples of proprietary solutions currently on the market: the Real Helix Server and Viadator’s Xenon Streamer. Section 4.5 discusses some custom solutions proposed in other research articles.

4.1 MPEG-21 Digital Item Adaptation

The MPEG-21 Digital Item Adaptation standard, specified in the ISO/IEC 21000-7 standard [11], provides a set of tools for managing audiovisual content in a diverse networking environment. It is part of the MPEG-21 multimedia framework, which is developed for transparent multimedia services over a wide range of networks, devices and communities. Muhkerjee et al. [14] show an example of how the MPEG-21 Digital Item Adaptation can be applied on the adaptation of JPEG images. They present a complex decision-taking framework that can contain algorithms on deciding which content is best suited for which situation, however, implementation of these algorithms is left up to the parties that will be using this framework. The standardization for dynamic adaptation of content is still work in progress and will be released as ISO/IEC 21000-7:2004/Amd.2. Therefore the MPEG-21 standard currently is not a good choice yet to be using for a commercial application. However, developments of the MPEG-21 standard should be followed closely in order to be looking into possibilities to be using it in the future.

4.2 3GPP Release 6

The Third Generation Partnership Program (3GPP) has standardized streaming since release 4 specifications [3]. In the newest release specification, release 6 [4], they also standardized a method called adaptive streaming. Content is encoded in different bitrates at the streaming server, and according to parameters like buffer status and available bandwidth signaled by the client application using RTSP [17], the content encoded in the appropriate bitrate for the current situation is sent to the client application. The idea behind this solution is to let the server always keep the buffer of the client application at a specified level, which is chosen by the client application based on e.g. current network characteristics. When the server manages to keep the buffer level of the client application at the specified level by changing e.g. content bitrate or transmission rate, it makes sure that the content will be played back pause-less, i.e. continuously. Performance simulations performed by Curcio and Leon [6] using the 3GPP release 6 adaptive streaming show that the technique is capable of preventing re-buffering and disconnections with a highly fluctuating bandwidth and connection handovers. Mobilaria currently does not use the 3GPP standards for their streaming systems, but the release 6 specification can become important in the future because of the wide deployment of the 3GPP standards in streaming applications. Therefore it is recommended that Mobilaria closely follows the development of the 3GPP standards and will be looking into possibilities of using them in the future.

4.3 Real Helix Server

The Real Helix Server [16] is a streaming server that offers a technique called Enhanced Rate Control [15], which means that the streaming server can change the bitrate of the stream at real-time during streaming of multimedia content. It uses two techniques called rate control and rate adaptation, which together make sure the best available bitrate is streamed to the client: rate control is a technique based on parameters such as packet loss, round trip time and RTP information, and is quite similar as the congestion control offered by TCP but then used for UDP; rate adaptation uses parameters like the clients buffer status and available
bandwidth to adapt the bitrate to match certain predefined condition constraints. The rate adaptation technique is based on the 3GPP release 6 solution presented in section 4.2: it is also based on the idea to keep the client buffer level at a specified value. To ensure this, the server keeps a buffer model for each connected client, which virtually represents the state of the client buffer, based on the packets that are sent out and the information obtained from the rate control technique. Using this virtual buffer model together with thresholds for downshifting and upshifting of the content’s bitrate, the server can maintain the specified client buffer level value. An advantage of the Real Helix Server is that Real has released the source code, which makes it possible to modify the software to ones needs. However, disadvantage of the Helix Server are that it only supports MP3 and Real Audio and Video codecs, and not the HE-AAC codec used by Mobilaria, and that it is based on UDP instead of TCP.

4.4 Vidiator Xenon Streamer

The Xenon Streamer from a company called Vidiator [18] offers a similar solution, called Dynamic Bitrate Adaptation. The bitrate of the offered stream is based on the parameters like e.g. available bandwidth and the resolution and aspect ratio of the mobile devices screen. This way, the user will experience less buffering and disconnections during playback of the streaming media. However, although Xenon Streamer offers support for HE-AAC, which is also used by Mobilaria for mobile streaming, the Dynamic Bitrate Adaptation technique only works for audio and video codecs that support real-time bitrate adjustments, like the MPEG video codecs. Furthermore, the streaming server software offered by Vidiator is not open source and can therefore not be modified to meet the requirements to be integrated in the underlying systems of Mobilaria’s streaming radio service.

4.5 Custom Solutions

Chan and Kok [5] present a system quite similar to the switch protocol used by Mobilaria. The system proposed by Chan and Kok is a client-based protocol, which means the client is in charge of the flow control decisions. Because of this property it is robust against loss of information or negotiation packets between the server and the client. A big difference between the system proposed by Chan and Kok and the system developed by Mobilaria is that Mobilaria’s solution is based on TCP and the solution of Chan and Kok is based on UDP. A big advantage of TCP is that it offers a guarantee that packets arrive, and arrive at the correct order at the receiver, where UDP is a best-effort protocol with absolutely no guarantee of packets arriving at the receiver. Because of this design choice, they had to design a packet loss recovery system as well. By doing this, they created a workaround for a big disadvantage of TCP: when a packet does not arrive in time, the data following the missing packet is also not available to the system. By taking care of the packet recovery themselves, they can make this data available even though some data is missing and fill up the missing data with e.g. predicted or duplicated audio or silence. Another difference is that Chan and Kok embed information about the bitrate in the packet header, making it a very dynamic protocol, which can adapt bitrate at frame level, whereas Mobilaria’s switch protocol has a stream-based bitrate adaptation; each TCP stream has a fixed bitrate. The decision making of the protocol proposed by Chan and Kok is based on the state of the buffer, keeping it at a predefined capacity and thereby trying to use a bitrate with a quality as high as possible. The results of their prototype showed that they could achieve a higher quality than with traditional systems. Unfortunately, this solution is based on multimedia protocols that can cope with scalable content, i.e. content with different bitrates within the same multimedia stream. The audio codec Mobilaria is using for the streaming radio application is HE-AAC, which needs a different instance of the decoder to decode a stream with a different bitrate.

Another approach to adapt multimedia content to available bandwidth is the one presented by Christianson and Brown [2]. They suggest an audio coding scheme where the audio is divided
into different stream components, representing multiple levels of audio resolution. Either the client or server can decide the number of components that will be sent to the client, based on the available bandwidth. A larger number of components means that the total audio resolution will be higher, and therefore the quality of the played back audio will be higher. Because of the non-mainstream way of coding the audio into different stream components, it will be hard to integrate into the current streaming systems of Mobilaria and therefore not a good solution to be using.
5. Specification

This chapter gives a specification of the switch decision algorithm that will be used for improving the reliability of Mobilaria’s streaming radio service. First of all, there is the question whether the algorithm should be placed client-side or server-side. Placing it server-side gives the advantage that the server can immediately start sending data in a different bitrate when a switch decision is made. The server could approximate the connection speed and the amount of data in the clients buffer by looking at which data is already sent to the client, but then again it does not know whether data is being buffered by e.g. a proxy server, or if the data got lost on the way to the client. Therefore, it will need to be provided with measurements by the client application, which means some sort of information messaging protocol has to be created. This introduces another point of failure, because these messages can also get lost, resulting in erroneous or even no switch decisions being made. The advantage of placing the decision algorithm at the client-side means all measured values are directly available to the decision algorithm, and a switch can be issued directly when necessary according to the algorithm. Because of this advantage, the algorithm will be placed client side.

Now we know that the algorithm will be situated at the client side we can define what kind of algorithm we want to use. There are two different variables that can be measured during playback of audio content at the client side: the connection speed and the amount of playable data in the buffer. Besides these two variables, the bitrate is a deterministic value also known within the client application. Based on these three input variables, we want the algorithm to produce a decision whether to switch down to a lower bitrate, keep playing at the current bitrate or switch up to a higher bitrate.

Furthermore, the switch decision algorithm will consist of two different formulas, based on the three input variables mentioned above: the measured connection speed, the amount of playable data in the buffer and the bitrate of the audio stream. These two formulas represent the thresholds for switching down and for switching up. In order to create these formulas, we first have to understand the behavior of the system. Modeling the system in a mathematical model, which is described in section 5.1, can help us understand the system. From this mathematical model a lookup table can be created, which gives insight in the probabilities to run into a buffer underrun given different sets of input variables. The process of creating and filtering the relevant information from this lookup table is described in section 5.2. Finally, from the data filtered from this lookup table the threshold formulas for the switch decision algorithm can be derived; this process is described in section 5.3.

5.1 Mathematical Model

This section provides a specification of a mathematical model of the system, based on a state transition diagram, using performance evaluation. In order to select an appropriate model for the system, the following properties are important:

- The system divides each second of playable audio data into a fixed number of identifiable audio frames. In the used bitrate switching protocol, this rate should be at least 50 frames per second to enable non-audible bitrate switching, and have a maximum of 75 frames per second to stay below a predefined overhead threshold. Therefore the audio frame rate is chosen to be 64 audio frames for each second of playable audio data.
- For each bitrate, these identifiable audio frames have a fixed length.
- These frames arrive at the system as complete frames, i.e. no segmented frames will arrive.
Given these properties the following properties are given for the mathematical model:

- The number of playable and identifiable audio frames present in the system at a given time is discrete: no segmented frames can exist in the system. Because of this property the mathematical model will have a discrete state-space.
- Each identifiable audio frame is processed in the system in 1/64 of a second, since there are 64 frames in a second of playable audio data. Because of this property the mathematical model will be in discrete-time.

Another important aspect of the system is that it is memory-less: the next state of the system (i.e. the number of identifiable audio frames arrived and processed after one step in time) is only based on the current state, and not on the previous states. This property is also called the Markov-property, and holds for all types of Markov models [9]. Since the system is discrete-time and has a discrete-state space, the Markov model that is appropriate for the system is the Discrete Time Markov Chain (DTMC). In this DTMC each state \( i \) represents the number of identifiable audio frames in the buffer currently available for playback. The number of audio frames in the system cannot be lower than zero (buffer underrun) or higher than the number of audio frames able to fit into the buffer \( (buf) \). The audio frames are processed (played) with a given service rate \( (sr) \), i.e. 64 frames per second, and arrive with a measurable bandwidth in frames per second \( (bw) \). Besides these two measurable and one deterministic variable, another variable is introduced in the model to create a probability factor: the loss probability \( (lp) \). This loss probability does not mean that frames actually get lost, but that they do not arrive at the moment they should arrive, and because of that we could say that this piece of bandwidth ‘gets lost’. Because the loss probability is something that is not measurable, it is later on in the process of creating the threshold formulas merged together with the bandwidth, by multiplying the bandwidth with one minus the loss probability: the actual rate at which data is received and measured by the client. Since the model is based on discrete-time, a probability exists between 0 and 1 for each transition from one state to another after one step in time. For each state \( i \) there exists a finite set of transitions to other states \( j \) with corresponding probabilities that can be calculated based on the service rate, the bandwidth and the loss probability. This DTMC model is graphically represented in Figure 8.

![Figure 8: DTMC for the decision model](image)

State zero means that there are no playable audio packets in the buffer, hence a buffer underrun occurs and the system stops: it is an absorbing state, which means that when the state is reached it will always stay there. In the highest state, the arrival of new audio packets is ignored, because it is not really relevant for the purpose of the model on the long run, namely preventing buffer underruns. The state transition probabilities from state \( i \) to another state \( j \) in a time step of 1 second, correspond to the probability formulas 1 to 5 in Table 1, according to the given conditions. These probabilities are based on the properties that every second \( i \) decreases with deterministic rate \( sr \), and increases with a (probable) rate based on \( bw \) and \( lp \). Besides that it is taken into account that state 0 is an absorbing state, and state \( buf \) (the maximum amount of frames in the buffer) is the highest state possible.
The generated lookup table containing transition probabilities is discussed in subsection 5.2.1. The filtering of the relevant data from the generated lookup table is discussed in subsection 5.2.2.

### 5.2 Lookup Table

In order to calculate the probability to end up in a buffer underrun within a certain times period given a certain system state by using this model, it requires matrix multiplications with very large matrices, which is very resource intensive and time consuming. Because of this it is not possible for a mobile device to compute these probabilities at real-time, or for a high-end server to compute these probabilities at real-time for a large number of clients simultaneously. Therefore a lookup table has been generated containing transition probabilities for a large set of possible states, which are given in subsection 5.2.1. The filtering of the relevant data from the generated lookup table is discussed in subsection 5.2.2.

#### 5.2.1 System States

The generated lookup table contains transition probabilities for the following states of the model:

- 11 bitrates: 16, 20, 24, 26, 29, 32, 40, 48, 64, 96 and 128 kbps. This corresponds to a maximum amount of frames in the buffer ($buf$) of respectively 8192, 6554, 5461, 5041, 4520, 4096, 3277, 2731, 2048, 1365 and 1024. Note that the service rate ($sr$) is 64 frames per second at all times.
- For each bitrate 16 different initial states ($i$): from 1/16th of the maximum buffer fill to the maximum buffer fill.
- For each initial state 25 different bandwidths ($bw$): from half the current bitrate up to two times the current bitrate.
- For each bandwidth 21 different loss probabilities ($lp$): 0 to 1 with steps of 0.05.

### Table 1: State transition probabilities

<table>
<thead>
<tr>
<th>Condition</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 0$ and $j = 0$</td>
<td>$\Pr{i \rightarrow j} = 1$ (1)</td>
</tr>
<tr>
<td>$j &lt; 0$ or $j &gt; buf$ or $j &lt; i - sr$ or $j &lt; i - sr + bw$</td>
<td>$\Pr{i \rightarrow j} = 0$ (2)</td>
</tr>
<tr>
<td>$j &gt; 0$ and $j &lt; buf$</td>
<td>$\Pr{i \rightarrow j} = (lp)^{buf - (j - i + sr)} \cdot (1 - lp)^{j - i + sr} \cdot \left(\frac{bw}{j - i + sr}\right)$ (3)</td>
</tr>
<tr>
<td>$j = buf$</td>
<td>$\Pr{i \rightarrow j} = \sum_{k=buf}^{i+buf-sr} (lp)^{buf - (k - i + sr)} \cdot (1 - lp)^{k - i + sr} \cdot \left(\frac{bw}{k - i + sr}\right)$ (4)</td>
</tr>
<tr>
<td>$j = 0$</td>
<td>$\Pr{i \rightarrow j} = \sum_{k=i-sr}^{0} (lp)^{buf - (k - i + sr)} \cdot (1 - lp)^{k - i + sr} \cdot \left(\frac{bw}{k - i + sr}\right)$ (5)</td>
</tr>
</tbody>
</table>

Formula 1 describes the case of the absorbing state 0, where formula 2 makes sure no transitions to states lower than 0, higher than $buf$, lower than the current state minus the service rate or higher than the current state minus the service rate plus the bandwidth are possible. Formula 3 describes the case where a valid transition is made: the probability that frames get lost, times the probability that frames do not get lost, times the total amount of combinations. Formulas 4 and 5 describe these probabilities to states 0 and $buf$, where all probabilities of transitions that would go to states lower than 0 or higher than $buf$ are added. With these transition probabilities, the probability to reach another state after a certain time can be calculated. This is interesting because the probability to reach state zero, i.e. to end up in a buffer underrun under the current conditions, can thus be calculated and a decision can be made whether to switch up or down or to keep using the current bitrate.
For each of these combinations, the following probabilities are calculated:

- One to two times the maximum amount of playable seconds with that bitrate into the future, with steps of 1 second in between.
- For each number of seconds into the future 10 different probabilities:
  - the probability to end up in state 0
  - the probabilities to end up in a set of states between 0 and the initial state, divided in four sets
  - the probabilities to end up in a set of states between the initial state and the maximum state, divided in four sets, only if the initial state is not equal to the maximum state
  - the probability to end up in the maximum state

This results in a lookup table with around 60 million values, with an uncompressed size of 1 gigabyte. By using this lookup table it is possible to analyze the probabilities to e.g. end up in a higher state, in a lower state, or in state 0, from a given initial state, bitrate, bandwidth and loss probability. The filtering of relevant data containing this information and needed to transform this lookup table into threshold formulas is discussed in the following subsection.

5.2.2 Filter Conditions

The data in the lookup tables is filtered with two conditions: a condition that would require a switch to a lower bitrate and a condition that would allow a switch to a higher bitrate. The filter conditions have the following form:

After a certain time, the probability to end up in a certain state or a certain set of states is equal to or larger than a certain value.

In order to define these conditions, choices have to be made for the time to look ahead, the state or set of states to look at and the probability to reach these state or states at the given time. These choices are described below.

Time

First of all there is the time at which to check the probabilities. Because the buffer has a fixed size, the maximum number of playable seconds that fit into the buffer is fixed for each bitrate. With high bitrates, only a small number of seconds fit into the buffer, where with low bitrates the amount of seconds can grow higher than two minutes with a full buffer.

For the lower threshold condition we want to base the time to look ahead on the current bitrate. It can also be based on the current buffer fill percentage, i.e. shorter with a small buffer fill and longer with a high buffer fill, but by using a mean value (per bitrate), the impact for lower buffer fills is higher than average, and the impact for higher buffer fills is lower than average, which will increase and decrease the thresholds respectively. This results in the choice of the time to look ahead for the lower condition to be half of the maximum number of seconds that fit into the buffer at the current bitrate. For a bitrate of 32 kbps, the buffer (of 256 kB) can contain 64 seconds of audio data, thus the time to look ahead 32 seconds.

For the upper threshold condition we do want to base the time on the buffer fill percentage, because the idea behind switching up is that it only takes place when the state of the system can really handle a higher bitrate, i.e. when the bandwidth is high enough and the buffer contains enough data to cope with the possible loss of data during switching. Basing the time to look ahead on the buffer fill also has the advantage that with a low buffer fill, the time to look ahead is short and with a high buffer fill the time to look ahead is longer, respectively.
giving the buffer a smaller and larger chance to reach a maximum buffer fill state within the look ahead time, and thereby respectively a smaller and larger probability that a decision will be made to switch to a higher bitrate. This results in the time to look ahead of simply the current amount of playable seconds in the buffer.

State(s)
The state or set of states we will be looking at is different for each of the two conditions. For the lower threshold condition, we want to prevent running into a buffer underrun, i.e. ending up in state 0. This could be achieved by looking at the probability that under the current conditions the buffer will end up empty after the specified time, but we also want to prevent the buffer to slowly run empty towards state 0. However, with higher buffer fills a small decrease in buffer fill is allowed since there is enough time to recover from this. Therefore we will be looking at the probabilities of the buffer being less filled than half the initial state, so that the allowed buffer fill decrease is dependent on the amount of data in the buffer.

For the upper threshold condition we really want the state of the system to be good enough to allow a switch to a higher bitrate, so we will be looking at the probability that under the current circumstances, the buffer will end up in the maximum state, i.e. a completely filled buffer.

Probability
The last variable in the conditions we need to specify is the probability reaching the given state at the given time. For the lower threshold condition we really want to prevent this behavior to happen, so we will be looking at the case that the probability is larger than 0.

For the upper threshold we really want to make sure this is going to happen, or in other words want this to happen with a high probability. Therefore we will be looking at the case that the probability is 1.

Conditions
From the choices described above, we can construct the two conditions for the data filter. For the lower threshold this is defined as:

\[
\text{After half the maximum amount of playable seconds able to fit into the buffer, the probability to end up in a state that is less than half the initial state is larger than zero.}
\]

For the upper threshold the following condition is defined:

\[
\text{After the current amount of playable seconds in the buffer, the probability to end up in the maximum possible state is equal to one.}
\]

Because the loss probability is something that is not possible to measure apart from complex client-server communication and calculations, from now on we will take the loss probability together with the bandwidth. Since the bandwidth in the mathematical model is deterministic and the loss probability to be a random factor that produces fluctuation in the measured bandwidth at the client, the bandwidth is multiplied by one minus the loss probability. This models the actual bandwidth space that is not lost and thus measured as the actual bandwidth at the client application. It is now possible to filter all the different combinations of variables that comply with one of the two conditions defined in subsection 5.2.2. For each of the bitrates \((br)\), the filtered variables are plotted in a graph with the buffer fill in percentage \((p=i/buf)\) on the x-axis and the measured bandwidth \((bw)\) on the y-axis. If the lower threshold condition is met for the given bitrate, buffer fill and bandwidth a red dot is plotted into the...
graph, and if the upper threshold condition is met for the given bitrate, buffer fill and bandwidth a green dot is plotted into the graph. After plotting a graph for each bitrate, it became clear that each of the graphs have the same characteristics as is shown in Figure 9.

Figure 9: Filtered data from lookup table with threshold formulas

In Figure 9, the data corresponding to the lower threshold condition is plotted in red dots, and the data corresponding to the upper threshold condition is plotted in green dots. Furthermore, two lines are plotted that correspond to the boundaries of the plotted data from the lookup tables. For each of these lines a formula is derived by interpolation, which are shown in the graph. For the lower boundary this formula is defined by formula 6.

\[ bw = 1.1 \cdot br \cdot (1 - p) \quad \text{for} \quad 0 \leq p \leq 1 \]  

(6)

The formula for the upper threshold in Figure 9 is defined by formula 7.

\[ bw = 1.1 \cdot br \cdot \left(1 - \left(\frac{p}{0.75}\right)^5\right) \quad \text{for} \quad 0.3 < p \leq 1 \]  

(7)

5.3 Threshold Formulas

As stated in section 3.1.4 where the switching solution was proposed, it is important that the buffer will have the chance to grow if it is filled with only a small amount of data. Therefore we want the lower threshold to be more ‘tight’ at low buffer fills, so it will sooner switch to a lower bitrate with which it is able to obtain more data than necessary and thereby increase the amount of buffered data. With higher buffer fills we do not want to switch down immediately when no traffic is received for a short while, but still give the system the possibility recover and start filling the buffer again. Therefore we want the threshold to be more ‘loose’ at high buffer fills. To cover these properties in the lower threshold, we have to adjust formula 6 to one that does not go down in a straight line, but stays high for the first part, and then quickly drops down before it reaches the maximum buffer fill. To put this kind of behavior into formula 6, we substitute \( p \) with \( \left(\frac{p}{b}\right)^2 \). By doing this, we can control where the line crosses the x-axis by varying \( b \), and control how long it stays at 1.1 times \( br \) by varying \( a \). This behavior is specified in formula 8.

\[ bw = 1.1 \cdot br \cdot \left(1 - \left(\frac{p}{0.75}\right)^5\right) \quad \text{for} \quad 0 \leq p \leq 0.75 \]  

(8)
Formula 7 conforms to the expectation we had for a good upper threshold. The only thing added to this formula is a safety margin for overhead data like metadata and such. In order to be able to capture possible bursts of this kind of overhead data, it is set at an additional 4 kb/s. This results in formula 9.

\[
bw = \left( \frac{83}{99} \right) \cdot br \cdot (p - 0.3)^{-0.52} + 4 \quad (0.3 < p \leq 1)
\]

To clarify the transformations of these two thresholds they are both plotted together with the old thresholds in Figure 10.

![Figure 10: Transformation of switching condition thresholds](image)

What can be seen in the formulas 8 and 9, is that there are a lot of numerical values that specify the behavior of the threshold, e.g. in formula 8 it says 0.75, which means for this type of formula the threshold will cross the x-axis at a buffer fill of 75 percent. The power of 5 tells us how long the graph stays at 1.1 times the bitrate and how steep it finally goes down: the higher this value the longer it stays at 1.1 times the bitrate and the steeper the graph will go down to zero. We want to substitute all these values into variables that can be dynamically adjusted in the proof of concept application in order to get the best switching decision configuration. This results in formulas 10 and 11, for respectively switching down and switching up.

\[
bw = c \cdot br \cdot \left( 1 - \left( \frac{p}{b} \right)^{a} \right) \quad (0 \leq p \leq b)
\]

with \(a\) = steepness  
\(b\) = zero at x-axis  
\(c\) = multiplication factor

\[
bw = k \cdot br \cdot (p - m)^{r} + s \quad (m < p \leq 1)
\]

with \(k\) = multiplication factor  
\(m\) = minimum value  
\(r\) = steepness  
\(s\) = safety margin
In formula 10, \( a \) defines the steepness of the formula, i.e. how long it stays at \( c \) and how steep it will finally go down towards \( b \); \( b \) defines the value of \( p \) at which the formula will cross the \( x \)-axis; \( c \) defines the multiplication factor, i.e. the formula will start at \( p = 0 \) at the value \( bw = c \). In formula 11, \( k \) defines the multiplication factor of the formula; \( m \) defines the minimum value the formula converges from; \( r \) defines how steep the formula transforms from converging from \( m \) to \( s \); \( s \) is the safety margin introduced at formula 9 and defines the value the formula finally converges to. The default values for all these variables with which we will initially create the algorithm are those specified in formulas 8 and 9.

The last part of the switching decision algorithm we still have to specify is the bitrate to switch to when a decision is made to switch to a lower or higher bitrate. When switching down, we want to switch to the closest available bitrate below the value that would just be above the threshold under the current conditions. To find this value, we specify the new bitrate by formula 12, where variables \( a \), \( b \) and \( c \) are the same as defined in formula 10.

\[
br_{\text{new}} \leq \frac{bw}{c \cdot (1 - (\frac{p}{b})^a)} \tag{12}
\]

When switching up, we want to switch to a bitrate that would not immediately issue a switch down to a lower bitrate again. Therefore the new bitrate at least has to comply with formula 12. Besides that, we want the system to be able to handle the new bitrate, and therefore it is not allowed to switch to a bitrate that is higher than the current bandwidth. These two conditions are specified by formula 13, where variables \( a \), \( b \) and \( c \) are the same as defined in formula 10.

\[
br_{\text{new}} \leq bw \quad (p \geq b) 
\]

\[
br_{\text{new}} \leq \min \left( bw, \frac{bw}{c \cdot (1 - (\frac{p}{b})^a)} \right) \quad (p < b) \tag{13}
\]

For each of the switch decision the last condition that has to be complied with is that if there is no bitrate available lower than the calculated new bitrate, or the first available bitrate is the same as the current bitrate, no switch will be issued.
6. Design and Implementation

This chapter discusses the design and implementation of the proof of concept of the decision algorithm. This way it addresses the following sub research question presented in section 1.2:

How could the selected solution be designed and implemented into the current multimedia player?

Mobilaria is currently developing their multimedia player for the iPhone [1], and the proof of concept will be integrated into this version of the player. Section 6.1 gives an overview of the design how the decision algorithm will be integrated into a proof of concept application. Section 6.2 discusses the implementation process of the proof of concept application on the iPhone.

6.1 Design

The Switch Protocol currently implemented in the iPhone multimedia application has to be extended with functionality that collects the necessary parameters to decide whether or not to switch, processes the parameters according to the formulas specified in section 5.3, and issues a switch when considered necessary according to the algorithm. Subsection 6.1.1 gives an overview of the currently available implementation that can be used to automate the process of switching between bitrates. The functionality of the new implementation is given in section 6.1.2. The process of collecting and processing the necessary parameters and the possibility to alter the variables mentioned in the threshold formulas is described in subsection 6.1.3. Subsection 6.1.4 describes the different states of the switching algorithm and the transitions between these states. The algorithm behind the actual switch point, the moment the audio from the new stream is sent to the audio playback instead of the audio from the old stream, is discussed in subsection 6.1.5. Subsection 6.1.6 describes the design of a streaming proxy to make sure that the amount of unneeded data downloaded by the client is reduced to a minimum.

6.1.1 Current Implementation

The multimedia player is already equipped with Mobilaria’s Switch Protocol algorithm, which enables it to switch between streams with the same content but encoded in a different bitrate. In order to issue such a switch, a button can be pressed which makes the application switch to the stream with the nearest bitrate below or above the current bitrate. When issuing such a switch, the client creates a new connection to the streaming server and requests a stream with the required bitrate. The two audio streams the client is then receiving contain the same content, but are encoded in different bitrates. In order to be able to synchronize the two streams, sequence numbers are embedded into the different streams during the encoding process. These sequence numbers are placed within the data at locations that represent the same position in time for each of the streams. With these sequence numbers, the client can determine a position in time for which it has data from both streams and switch playback from the old stream to the new stream, without the user knowing about it except for the difference in quality. Currently, this actual switch point is chosen to be at the end of the old stream. Subsection 6.1.5 gives an algorithm that improves this process to increase the quality of the audio the user is listening to, by depending the actual switch point to the quality of the different streams and the amount of data in the buffers. Figure 11 gives a schematic view of the current network topology of the switch protocol. In this figure, the network of the radio station the user is listening to broadcasts the radio stream over the Internet. The transcoder server located in Mobilaria’s network is receiving this stream and transcodes it to an audio codec that is best suitable for mobile networks, in this case HE-AAC. For each bitrate available to and requested by mobile clients a stream is created and sent to the Icecast
Streaming Server [10]. This server broadcasts the appropriate streams to the connected clients.

Figure 11: Schematic network topology of existing switch protocol

For example, when a user wants to listen to radio station A and no other users are listening to this station, the transcoder server connects to the streaming server of radio station A and starts transcoding the radio stream to the bitrate requested by the user, e.g. 32 kbps. This transcoded stream is sent to the Icecast Streaming Server, which in turn broadcasts it to the mobile device of the user. After some time the user decides to switch to a higher bitrate, let’s say 48 kbps. The transcoder already has a connection to the stream of radio station A, so only has to start transcoding the same stream to the newly requested bitrate of 48 kbps and send it to the Icecast Streaming Server, which in turn broadcasts it to the mobile device of the user again. After being connected to the new stream and the buffering of data has started, the mobile device disconnects the old stream of 32 kbps. Mobilaria’s network now discovers that no users are listening to this stream of 32 kbps anymore, so the transcoder stops transcoding the stream of radio station A to 32 kbps. At the mobile device, the sequence number of the last audio data in the 32 kbps stream is looked up in the new stream of 48 kbps, and when the last data is played of the 32 kbps stream, a switch is made to the data of the 48 kbps stream.

6.1.2 Functionality

In the current implemented version of Mobilaria’s Switch Protocol, the user has to decide when to switch, and has to issue a switch by pressing a button. To automate this process, the decision algorithm described in section 5.3 has to be integrated into the multimedia player. To do so, the behavior of the multimedia player has to be extended to the functionality block diagram given in Figure 12. In this figure the different blocks have the following functions:

- **Core**: the main part of the application. Is called on startup and creates a SwitchingManager and a graphical user interface (GUI). It also acts as a bridge between these two blocks, i.e. forwards user input calls from the GUI to the SwitchingManager and forwards information from the SwitchingManager to the GUI.
• GUI: the graphical user interface. It offers a list containing the test stream(s) and displays all necessary information about the playing stream and the visualization of the switch protocol. Besides that, it offers controls like stop and volume, and for manipulating the decision algorithm.

• SwitchingManager: the part of the application that manages the connection, data parser and the data buffer for each stream, and also manages the AudioPlayback. It also initializes the DecisionManager and makes sure it is provided with the necessary information for making switch decisions.

• DecisionManager: the part of the application that contains the switch algorithm. It is provided with the necessary information by the SwitchingManager, and based on this information decides whether or not a switch is needed.

• HTTPConnection: handles the connection to the stream. Provides the SwitchingManager with the connection speed and provides the AudioParser with the received data.

• AudioParser: parses the received audio. Filters metadata when available and looks up the switch protocol header information. The audio data is provided to the AudioBuffer together with the corresponding switch protocol sequence number.

• AudioBuffer: handles the buffering of audio data. Audio data is stored together with the corresponding switch protocol sequence numbers. Provides the SwitchingManager with the current buffer status, which is forwarded to the DecisionManager and used for making switch decisions. Provides audio data to the audio playback, based on calls from the SwitchingManager.

• AudioPlayback: manages the playback of audio data. Provides the SwitchingManager with buffer status information, so the SwitchingManager can in turn tell the appropriate AudioBuffer to provide the AudioPlayback with data.

![Figure 12: Functionality block diagram](image)

6.1.3 Parameters and Variables

In order to make a decision based on the formulas specified in section 5.3 the following parameters have to be available for the decision algorithm:

- Buffer fill percentage
- Available bandwidth
- Current bitrate

Two options are available to get these parameters: first of all we can use polling, which means the algorithm class actively collects the necessary parameters at the appropriate classes; the other option is that the appropriate classes pass through all changes that occur to the parameters. The latter option gives a more detailed overview of the parameters, since the
decision algorithm can review all combinations of all parameters; with the polling technique the algorithm only gets instantaneous samples of the parameters and thereby incomplete information. Therefore the choice is made to use the latter option: the necessary classes will provide updates of the parameters to the decision algorithm.

The AudioBuffer that maintains the audio data and the corresponding switch protocol headers provides the amount of data that is available for playback to the decision algorithm. From this amount of data, and the corresponding bitrate of the stream, the number seconds of playable audio data can be calculated. Together with the information that all audio buffers of the Icecast Streaming Server are 256 kB, independent of the bitrate, the fill percentage can be calculated. It has to be taken into account that when switching to a different bitrate, the amount of 256 kB can not be taken anymore as a full buffer, but the number of seconds that fit in the buffer of the first stream has to be taken as the 100 percent buffer fill. E.g. when starting with a stream at 32 kbps 64 seconds of playable data fit into the 256 kB buffer. A full buffer at the client of this stream will be around 256 kB and contain these 64 seconds of playable audio data. However, when switching to e.g. 64 kbps, the number of playable seconds that can be buffered is still 64 seconds, since no jump in time will be made during playback. Therefore a full buffer of this audio stream is twice the size of that of the 32 kbps stream, thus 512 kB.

The HTTPConnection that maintains the connection to the streaming server provides the decision algorithm with the available bandwidth. The class is only able to measure the speed of the connection to the streaming server. Therefore, a problem arises when all the audio data available at the streaming server is buffered by the client: the measured speed will maximally be that of the bitrate of the stream, since new data only becomes available at that rate. In order to be able to measure bandwidth peaks, two different connection speeds should be provided: one measured over a long period of time and one measured over a short period of time. The speed measured over a long period will give an average of the connection speed and can be used for decisions to switch down. The speed measured over a shorter period contains information about bandwidth peaks, and can be used for decisions to switch up. These two periods should be adjustable, together with the interval in which the connection classes update the speed parameters.

Besides the parameters mentioned above, section 5.3 also specifies seven different variables in the formulas used to base the switch decisions on. These variables will be set to the initial values provided by section 5.3, but are adjustable at real-time in order to be able to test different setups of the variables.

### 6.1.4 State Transitions

The core of the application is based on a state machine that has five states that are important to the switch algorithm. These five states are depicted in Figure 13.

![Figure 13: Main state transition diagram](image-url)
The states in the state machine depicted in Figure 13 correspond to the following system states:

- **IDLE**: The system is idle and waiting for user input.
- **CONNECTING**: The user has selected a stream to play and the system is creating a connection to the stream.
- **BUFFERING**: The connection to the stream is created and the system is buffering data for playback.
- **PLAYING**: The buffer threshold has been reached or the system is done switching, and the system is playing audio.
- **SWITCHING**: A switch has been issued either by the user or by the decision algorithm, and the SwitchingManager is handling the switch. Audio is still playing in the meanwhile. When the switch is completed the system will go back to the PLAYING state.

The SwitchingManager maintains a local state machine for each of the streams, which is illustrated in Figure 14.

![State transition diagram per stream](image-url)

*Figure 14: State transition diagram per stream*

With a successful setup of the first stream, the first four states of the newly connected stream will transit parallel to the main state machine in the **Core**: IDLE, CONNECTING, BUFFERING and PLAYING. When the main state enters the state switching, it means that a connection is being made to a new audio stream and this stream will go into the state SWITCH_CONNECTING. When the connection is setup it will transit to the state SWITCH_BUFFERING. Then the switching algorithm will decide on the appropriate switch point, which is discussed in the next subsection, and when the actual switch is made it will transit to the state PLAYING, together with the main state in the Core. The previously active stream will transit to the state IDLE at the same moment.

### 6.1.5 Deciding Switch Point

After issuing a switch and having set up the new audio stream, the point when to switch exactly between the two existing streams has to be calculated. In order to this, the buffers of the two different streams, located in the AudioBuffer of each stream, have to be compared. Each buffer contains a set of pieces of data with corresponding sequence numbers. By comparing these sequence numbers, the position within the data can be found which contains the same audio, but encoded in a different bitrate. By stopping to send data from the old stream to the audio playback and starting to send data from the new stream to the audio playback at the sequence number where we left off, the audio will switch from the old to the new quality without any audible gaps or transitions, except for a difference in quality. Comparing the two different buffers can be done with temporal relations, described in [12]. When deciding the switch point, we keep in mind that data that is already available in a higher quality can be deleted, and playing audio encoded in the highest quality is always preferred above audio encoded in lower quality. For determining the right switch point, we define
sixteen different temporal relations between the two buffers. Table 2 gives an overview of the defined temporal relations. In this overview, the block called A defines the buffer of the old and currently playing stream, and the block called B defines the buffer of the new stream. The buffers are shown respectively to each other over time (horizontal). The conditions called \textit{Up} and \textit{Down} respectively mean switching to a higher bitrate or to a lower bitrate.

<table>
<thead>
<tr>
<th>Temporal relation</th>
<th>Decision(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 A precedes B</td>
<td>None</td>
</tr>
<tr>
<td>2 A meets B</td>
<td>None</td>
</tr>
</tbody>
</table>
| 3 A overlaps B    | Close connection A  
\textit{Down}: remove x-y from B |
| 4 B finishes A    | Close connection A  
\textit{Down}: remove x-y from B |
| 5 A includes B    | Close connection A  
\textit{Down}: remove B |
| 6 A starts B      | Close connection A  
\textit{Up}: switch  
\textit{Down} \& x = y: switch  
\textit{Down} \& x \neq y: remove x-y from B |
| 7 A equals B      | Close connection A  
\textit{Up}: switch  
\textit{Down} \& x = y: switch  
\textit{Down} \& x \neq y: remove B |
<table>
<thead>
<tr>
<th>Case</th>
<th>Temporal Relation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>B starts A</td>
<td>Close connection A &lt;br&gt;&lt;em&gt;Up &amp; b reached threshold: switch&lt;/em&gt; &lt;br&gt;&lt;em&gt;Down: remove B&lt;/em&gt;</td>
</tr>
<tr>
<td>9</td>
<td>B includes A</td>
<td>Close connection A &lt;br&gt;&lt;em&gt;Up: remove 0-x from B&lt;/em&gt; &lt;br&gt;&lt;em&gt;Down: remove 0-y from B&lt;/em&gt;</td>
</tr>
<tr>
<td>10</td>
<td>A finishes B</td>
<td>Close connection A &lt;br&gt;&lt;em&gt;Up: remove 0-x from B&lt;/em&gt; &lt;br&gt;&lt;em&gt;Down: remove 0-y from B&lt;/em&gt;</td>
</tr>
<tr>
<td>11</td>
<td>B overlaps A</td>
<td>Close connection A &lt;br&gt;&lt;em&gt;Up: remove 0-x from B&lt;/em&gt; &lt;br&gt;&lt;em&gt;Down: remove B&lt;/em&gt;</td>
</tr>
<tr>
<td>12</td>
<td>B meets A</td>
<td>Close connection A &lt;br&gt;Remove B</td>
</tr>
<tr>
<td>13</td>
<td>B precedes A</td>
<td>Close connection A &lt;br&gt;Remove B</td>
</tr>
<tr>
<td>14</td>
<td>B, no A</td>
<td>Reconnect</td>
</tr>
<tr>
<td>15</td>
<td>A, no B</td>
<td>Reconnect</td>
</tr>
<tr>
<td>16</td>
<td>No A, no B</td>
<td>Reconnect</td>
</tr>
</tbody>
</table>

Table 2: Temporal relations

All the decisions depicted in Table 2 are based on the two rules mentioned above. Because it is not known from a certain sequence number what the next sequence number will be (they are not based on automatic increment or anything similar), we can never determine the relations where ‘a meets b’ or ‘b meets a’ (case 2 and 12). Therefore a switch decision is not based on these temporal relations. Furthermore, when removing a part from a buffer, this is
always done excluding the last element, e.g. removing x-y will remove everything from x (including) up to y (excluding). This way we do not intentionally end up in temporal relation case 2 or 12. The decision to reconnect with temporal relation case 14 is made because there will most likely be a gap or disruption in audio since it is very unlikely that the buffers exactly meet each other. Because of this gap, the position in time where the audio is playing is changed and therefore it is no longer possible to determine the buffer fill percentage anymore, which is necessary for the decision algorithm to work correctly. The following example gives us an idea how the buffers change respectively to each other and the decisions that belong to these temporal relations.

The system is playing a stream at 64 kbps and a decision is made to switch to 48 kbps. After the new connection is setup the first data is received. Let’s say that we start at temporal relation case 1, where the buffer of the current stream precedes that of the new stream. For this relation nothing has to be done, so the system will keep buffering both streams. After a while, the buffer of the current stream has grown to the moment that the sequence number corresponding to the last data in the buffer is also present in the buffer of the new stream. This means that the system is now in temporal relation case 3. This means that the connection of the current stream will be closed. Since we are switching down, the data in the buffer of the new stream can be deleted up to the last sequence number of the data in the buffer of the current stream. At this moment, the current stream is still playing and its connection is already closed, which means the amount of data in the buffer will slowly decrease. The buffer of the new stream is slowly growing, since no audio data is played and the connection is still alive and receiving audio data. At a moment in time the buffers will transit to temporal relation case 6, where the buffer of the current stream will only contain one single sequence number, which is the same as the first sequence number in the new buffer. Corresponding to Table 2 this means the x=y condition is true, and together with the fact that we are switching down this means that the switch point is reached and the data from the new buffer is from now on sent to the audio playback. The old stream can be closed, and the system will transit to the PLAYING state again.

6.1.6 Streaming Proxy

When the Switch Protocol developed by Mobilaria was implemented into the prototype and used for the first time, some unwanted but unforeseen behavior became visible. When connecting to the server to request a stream, the data is always sent starting from the beginning of the buffer. When switching to a stream with another bitrate, there is a lot of data that is unnecessarily downloaded by the client because of this property: it is already present in the buffer of the old stream. This problem is increased when switching down, because at the server side buffers have the same absolute size (in bytes), and therefore the buffer of the stream encoded in a lower bitrate contains more seconds of playable audio than that of a stream encoded in a higher bitrate. Also the fact that switching down means the bandwidth is scarce makes it an even bigger problem since a lot of bandwidth and thereby time to buffer is wasted on unnecessary data that is already available in a higher bitrate.

In order to work around this problem, the best option would be to alter the server that delivers the streams, the Icecast Streaming Server, in a way that it can identify Switch Protocol sequence numbers and make it possible for the client to request a stream starting from a certain sequence number. However, the Icecast Streaming Server is a pretty complicated piece of software and altering it would take a lot of time. In order to create a workaround that would not take up a lot of time, a streaming proxy is developed, based on a standard Apache Server [8]. It is possible for the client to connect to this streaming proxy, thereby specifying a bitrate and optionally a sequence number from where it would like to start receiving data. This streaming proxy server starts a Perl script that handles all the functionality for that specific connection. This Perl script connects to the actual stream (when it exists in the specified
bitrate), and when a sequence number is not provided it simply proxies all the data from the stream to the client. When a sequence number is provided, the Perl script will start looking for Switch Protocol headers and compare the sequence numbers to the sequence number specified by the client. When the specified number is not yet reached, the data according to this Switch Protocol header will not be proxied to the client. When the sequence number or a sequence number higher than the one specified is found, all the data from there on will be proxied to the client. This way, the client downloads almost no unnecessary data and thereby no time is lost, which is a huge advantage for the algorithm to work correctly. Only the duration of the connection setup is causing a slight overlap in data between the two different audio streams. Figure 15 shows a detailed view of the design of the streaming proxy server. In this figure, the mobile device is connected to multiply streams with different bitrates via the streaming proxy server.

*Figure 15: Streaming proxy server*

### 6.2 Implementation

The implementation of the proof of concept is integrated into the iPhone version of Mobilaria’s multimedia player. This version of the multimedia player runs on the first generation iPhone and on the iPhone 3G, with iPhone OS version 2.0 and up. The programming language for this device is Objective-C, which is a superset of C. The iPhone SDK is used in combination with Xcode 3.x, and can only be used on an Intel based Apple computer. This section provides the class diagram of the implementation in subsection 6.2.1. The implementation of the decision algorithm is discussed in subsection 6.2.2. The implementation of the decision on the actual switch point is detailed in subsection 6.2.3, and the implementation of the streaming proxy is discussed subsection 6.2.4. Finally, section 6.2.5 gives some suggestions for improvements areas of the proof of concept application.

#### 6.2.1 Class Diagram

Figure 16 gives an overview of the implementation of the decision algorithm within the proof of concept. The class structure is based on the functional block diagram given in Figure 12. Since the complete multimedia player is a quite complex piece of software, only the classes and methods that are important for the algorithm are displayed in this class diagram: from the block diagram, the *Core* and the *GUI* are left out. The rest of the block diagram corresponds to the classes in the class diagram with the same name. The names of the methods, variables and classes may have been altered for simplicity reasons.
The SwitchingManager class is the main class that handles all the switching functionality, like creating and maintaining other necessary classes and deciding when the actual switch between two streams should take place. It also the class that collects bandwidth updates from the HTTPConnection class and buffer updates from the AudioBuffer class and sends them to the DecisionManager class when necessary.

The DecisionManager class is the class that contains the decision algorithm. It is provided with the necessary variables by the SwitchingManager class. Based on these variables, each time the connection speed is updated, the DecisionManager calculates whether a switch decision is necessary, and if so, it will issue a switch at the SwitchingManager.

The HTTPConnection class handles a connection to the streaming server. It has functions to open and close a connection, a function to set the bandwidth update interval and a function to set the periods over which the bandwidth of the connection should be measured. The class updates the two different bandwidth values to the SwitchingManager class through the updateSpeeds() method. All received audio data is forwarded to the AudioParser class using the parseData() method.

The AudioParser class parses the data that is received by the HTTPConnection class, and collects all metadata and Switch Protocol headers from it when present in the audio stream. The sequence numbers found in the Switch Protocol headers are then forwarded to the AudioBuffer class together with the corresponding data using the bufferData() method.

The AudioBuffer class stores the data in an array, together with the corresponding sequence numbers. Each time the audio buffer array is changed, the amount of data in the buffer is updated to the SwitchingManager class using the updateAudioBuffer() method. With the enqueueAudio() method, the SwitchingManager class orders the AudioBuffer class to send data to the AudioPlayback class for playback based on the available buffered data in the latter class. The AudioBuffer class uses the playAudio() method to enqueue the data for playback in the AudioPlayback class.

The AudioPlayback class is the class that decodes the audio data when necessary and maintains a handle to the audio playback hardware of the device. It updates its buffer status to
the SwitchingManager using the updatePlaybackBuffer() method. Based on this value the
SwitchingManager can decide to order the AudioBuffer class to send data to the
AudioPlayback class for playback. This buffer will be kept above 4 seconds of audio data at
time, in order to assure a smooth audio playback.

6.2.2 Decision Algorithm

The decision algorithm is implemented in the DecisionManager class, using formulas 10 and
11 specified in section 5.3. The DecisionManager class is initialized with the maximum
number of seconds able to fit into the buffer, and after that constantly updated with the buffer
status and the connection speed of the active stream. Each time the buffer status is updated,
this value is stored in a local variable. Each time the connection speed is updated, the values
of the buffer status and the connection speed are checked against the decision algorithm
formulas, and when a switch is necessary according to the algorithm, a new bitrate is
calculated and a switch decision is issued. The values are not checked every time the buffer
status is updated, because this can result in an extremely high load of the system. The
connection speed update interval can be adjusted and in that way the load on the system can
be controlled and reduced if necessary. The following pseudo code gives an idea of how the
check against the algorithm is performed:

```java
if (buf < B && bwLong < (C*br*(1-(buf/B)^A)) && bwShort < (C*br*(1-(buf/B)^A))) {
    if (br > lowestBr && newBr < br) {
        switchToBitrateBelow(newBr);
    }
}
if (buf >= M && bwShort > ((K*br*(buf-M)^R)+S)) {
    if (br < highestBr && newBr > br) {
        switchToBitrateBelow(newBr);
    }
}
```

This peace of pseudo code checks in the first if-statement whether the buffer fill is below the
value of B, and whether both the connection speed measured over a long period and the speed
measured over a short period are below the threshold boundary for switching down. If this is
the case, the current bitrate should be higher than the lowest bitrate and a new bitrate is
calculated and checked whether it is below the current bitrate. When all this checks out to be
true, a switch is issued to the next available bitrate below that of the newly calculated bitrate.

In the second if statement, the code check whether the buffer fill is higher than or equal to the
value of M, and whether the connection speed measured over a short period is higher than the
threshold boundary for switching up. If this is the case, the current bitrate should be below the
highest bitrate, and a new bitrate is calculated and checked whether it is above the current
bitrate. When all this checks out to be true, a switch is issued to the next available bitrate
below that of the newly calculated bitrate. The following pseudo code explains how the new
bitrate is calculated:

```java
if (switchingDown) {
    newBr = (bwLong/(C*(1-(buf/B)^A)));
}
if (switchingUp) {
    if (buf < B) {
        newBr = MIN(bwShort , (bwShort/(C*(1-(buf/B)^A))));
    } else {
        newBr = bwShort;
    }
}
```

The first if-statement calculates the new bitrate for switching down; the second if-statement
does the same for switching up. These calculations are done according to formulas 12 and 13
specified in section 5.3. If a switch is issued, the switch will be made to the next available bitrate below this newly calculated bitrate.

6.2.3 Deciding Switch Point

As stated in subsection 6.1.5, after a switch has been issued and the connection has been setup, the actual switch point within the buffers has to be calculated in order to obtain a smooth transition from one stream to another, together with obtaining the maximum possible quality. Table 2 in the aforementioned subsection gives a set of decision based on temporal relations between the buffers of the two streams. The following pseudo code located in the SwitchingManager class encapsulates these decisions:

```java
UInt64 fromFirstHeader = fromBuffer.getFirstHeader();
UInt64 fromLastHeader = fromBuffer.getLastHeader();
UInt64 toFirstHeader = toBuffer.getFirstHeader();
UInt64 toLastHeader = toBuffer.getLastHeader();

if (fromFirstHeader == 0) {
    if (toFirstHeader == 0) {
        //Temporal relation case 16
        reconnect();
    } else {
        //Temporal relation case 14
        reconnect();
    }
} else if (toFirstHeader == 0) {
    //Temporal relation case 15
} else if (fromFirstHeader < toFirstHeader) {
    if (fromLastHeader < toFirstHeader) {
        //Temporal relation case 1
    } else if (fromLastHeader == toFirstHeader) {
        //Temporal relation case 2
        closeConnection(fromConnection);
    } else if (fromLastHeader > toFirstHeader) {
        if (fromLastHeader < toLastHeader) {
            //Temporal relation case 3
            closeConnection(fromConnection);
            if (switchingDown)
                toBuffer.deleteDataExcluding(fromLastHeader);
        } else if (fromLastHeader == toLastHeader) {
            //Temporal relation case 4
            closeConnection(fromConnection);
            if (switchingDown)
                toBuffer.deleteDataExcluding(fromLastHeader);
        } else if (fromLastHeader > toLastHeader) {
            //Temporal relation case 5
            closeConnection(fromConnection);
            if (switchingDown)
                toBuffer.deleteDataExcluding(fromLastHeader);
        }
    }
} else if (fromFirstHeader == toFirstHeader) {
    if (fromLastHeader < toLastHeader) {
        //Temporal relation case 6
        closeConnection(fromChain);
        if (switchingDown)
            toBuffer.deleteDataExcluding(fromLastHeader);
        if (switchingUp || fromFirstHeader == fromLastHeader)
            switch();
    } else if (fromLastHeader == toLastHeader) {
        //Temporal relation case 7
        closeConnection(fromConnection);
        if (switchingDown)
            toBuffer.deleteDataExcluding(fromLastHeader);
        if (switchingUp || fromFirstHeader == fromLastHeader)
            switch();
    } else if (fromLastHeader > toLastHeader) {
        //Temporal relation case 8
        closeConnection(fromConnection);
        if (switchingDown)
            toBuffer.deleteDataExcluding(fromLastHeader);
        if (switchingUp && toBuffer.tresholdReached())
            switch();
```
6.2.4 Streaming Proxy

The implementation of the streaming proxy is realized using a standard installation of the Apache web server. A Perl script is used to accept incoming connections, analyze the requested parameters like bitrate and sequence number and create a connection to the streaming server. After this connection is created, the audio data is analyzed for Switch Protocol sequence number information, and when no sequence number is specified, or when the sequence number found is higher than or equal to the sequence number requested, the audio data from the streaming server is proxied without any modifications to the client.

A problem encountered after implementation was that the burst speed of the Icecast streaming server was not high enough in order for the streaming proxy to work as efficient as possible. In order to create a workaround for this problem, another instance of the Icecast streaming server was installed on the same machine as the Apache web server, and setup as a relay server: a relay server simply copies the buffers from another Icecast server. Using this local Icecast server, the burst speed was now limited by the speed of the script instead of the bandwidth between the servers, and thus high enough for the Perl script to find the requested sequence number in the stream in an acceptable time interval.

6.2.5 Improvement Areas

Since the implementation of the switch decision algorithm is still in a proof of concept phase, the application still needs a lot of debugging to fix shortcomings like deadlocks and null pointers. Besides that, the algorithm itself also has some areas of improvement, which are discussed below.
Multiple Subsequent Switches

One of the major shortcomings is that only one switch at the time can be issued, and that only when this switch is completely finished another switch can be issued. For example, then a switch is issued to a higher bitrate and directly after the switch is issued the bandwidth drops to a very low value, this would result in a the loss of a lot of valuable buffered data since the bandwidth will not be high enough to keep the buffer at the same fill. Besides that it is very likely that this situation would end up with a buffer underrun.

It would be a major improvement to the algorithm to be able to constantly monitor the total amount of playable seconds in the buffer, also during switches, to be able to issue multiple subsequent switches to adapt even better to the available bandwidth. In order to add this functionality, the SwitchingManager would have to be extended with functionality to maintain more than two sets of connection, buffer and decoder, and it would have to provide the complete amount of data of all the different buffers to the DecisionManager during switches. Besides that, the switching algorithm of the DecisionManager would have to be improved so that it takes into account the connection setup time and similar variables, so it would not be issuing too many switches in a short period of time.

Bandwidth Measurement

Another shortcoming is the lack of possibility to measure low-level data traffic on the iPhone platform. Currently, the bandwidth measurement is implemented in the HTTPConnection class, which is on application level of the OSI stack [13]. This class is provided with blocks of received data by the underlying connection APIs of the iPhone platform. When for instance packet loss occurs, all the data that is received that should be behind the lost packet is kept in a buffer within these connection APIs, and this is not notified to the HTTPConnection class. Therefore it looks as if no data is being received anymore, possibly resulting in erroneous switching decisions. When the packet finally arrives, all the data that is buffered that is behind this lost packet is send to the HTTPConnection class at once, making it look as if the connection speed is suddenly very high.

It would be a major improvement to the system to be able to measure the available bandwidth at transport level of the OSI stack. Besides that being aware of packet loss could help in making the correct switching decisions for these situations. For example, when packet loss occurs and the buffer only contains a small amount of data, this could be translated into a switching decision. But when the buffer contains a large amount of playable data, the packet loss could be ignored and taken into account when issuing a switch.

Another possibility to workaround this problem is a little more radical: using UDP instead of TCP. By using UDP, this problem of the data being delivered in the correct order does not exist. So when a packet is lost, data that is behind this packet is still being delivered to the application layer. However, making the choice of using UDP as the transport layer also introduces the disadvantage that all valuable properties of TCP are lost, like e.g. automatic retransmission of lost packets and flow control. Besides that, there is no guarantee that the packets are delivered in the correct order, so this function has to be implemented as well. Last but not least, this change in transport layer would also mean a rigorous change at the server side as well, since the Icecast Streaming Server that is used at the moment to deliver the streams to the clients only works with TCP.

Processor Load

The proof of concept application is currently producing a high load on the system while switching, because the SwitchingManager is constantly notified with every change in the buffers by the AudioBuffer classes, and with every update the temporal relation between the buffers for that particular situation is calculated and acted on appropriately.
To reduce the processor load produced during switches, this can be changed into a way that the **AudioBuffer** knows when an update of the sequence numbers is necessary and only then notifies the **SwitchingManager** of changes. For example, when the buffers are in temporal relation case 3 (see subsection 6.1.5), where the buffer of the currently playing stream overlaps the buffer of the new stream, the **SwitchingManager** can calculate how long it takes for the buffers to end up in temporal relation case 6, and it can tell the **AudioBuffer** of the currently playing stream to stop notifying buffer updates until this moment arises. When this is used next to the proposed feature that enables multiple consequent switches, it can simply calculate the amount of playable seconds in the buffer of the currently playing stream, since its connection is closed (because of the decision corresponding to temporal relation case 3 that closes the connection of that stream) and one second of playable data will be send to the **AudioPlayback** every second.

### Connection Setup Delay

At this moment, the proof of concept application does not take connection setup delay into account when switching. Taking this into account, it could influence the switching decisions in a way that e.g. when switching down the switch should be made to a lower bitrate than in the current situation, because the amount of data in the buffer will decrease caused by the connection setup delay. This issue can also be improved in another way: let the **SwitchingManager** adapt the requested sequence number at the streaming proxy to the connection setup delay. For example, when the currently playing stream is still receiving data it would request data starting from a sequence number at the streaming proxy that corresponds to a moment in the future according to the rate at which the connection of the currently playing stream is receiving data and the average connection setup delay. This way the amount of overlap in data of the different buffers is minimized and thereby the amount of playable seconds in the buffer can be optimized and theoretically no buffer fill is lost due to the connection setup delay.

Another option that could be introduced to solve the problem of connection setup delay is to use the same connection for audio data from different streams with different bitrates. This would mean the introduction of some sort of information embedded into the stream that contains information about the bitrate of the next piece of data. This could be implemented into a streaming server like the Icecast Streaming Server, or the functionality could be added to the streaming proxy server that was introduced during this assignment. The decision to alter the bitrate within the stream could be pull based, i.e. made by the client application like it is implemented at the moment, which would mean the introduction of some sort of feedback or request system between the client and the server. Another possibility is to put the responsibility of deciding which bitrate is best suitable for a client at the server side. This would take away the requirement of a feedback or request system, but it could still be introduced to aid the decision making process or to create a combination of the two options. Of course, when the same stream is used for different bitrates, the decision algorithm would no longer cover the loss of connectivity problem, so a separate reconnection algorithm would have to be introduced as well.
7. Testing

This chapter describes the tests of the proof of concept for the switch decision algorithm. Section 7.1 describes a test that is performed in a controlled environment, to find the best parameter settings and to verify the behavior of the decision algorithm formulas. Section 7.2 describes a comparison between the currently used Symbian version of the player which does not contain the decision algorithm, and the proof of concept. Section 7.3 describes a test that can be used for demonstration purposes.

7.1 Controlled Environment

This test is performed under controlled conditions, to find the best possible settings for the decision algorithm threshold formulas and to verify the correct functionality of the algorithm. With this test, the bandwidth provided to the application is limited to certain values. This test only includes the proof of concept player on the iPhone.

7.1.1 Test Case

The controlled environment test case is specified as described below.

Goals

The goal of this test is twofold: first of all we want to find the best possible parameter settings of the threshold formulas specified in section 5.3. Secondly, we want to verify the decision algorithm by creating situations where normally a buffer underrun would occur because of insufficient bandwidth, and verify that the proof of concept prevents the stream from stopping by switching to a lower bitrate.

Network Topology

The topology of the network exists of a single mobile device connected via Wi-Fi to a laptop computer that provides a network connection to the mobile device and controls the network parameters like e.g. the bandwidth. An Icecast streaming server together with the streaming proxy server on the server side provide the radio streams to the mobile device. A schematic view of the network topology for this test case is given in Figure 17.

```
Figure 17: Controlled environment test case network topology
```

Configuration

For different settings of the parameters for the threshold formulas, we follow different patterns of network parameters to compare the behavior of the algorithm under the same
circumstances with different decision parameters. Five different decision parameter sets will be used, which are listed in Table 3.

<table>
<thead>
<tr>
<th>Switch Up</th>
<th>Switch Down</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>k</th>
<th>m</th>
<th>r</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Medium</td>
<td>5.0</td>
<td>0.75</td>
<td>1.1</td>
<td>83/99</td>
<td>0.3</td>
<td>-0.52</td>
<td>4</td>
</tr>
<tr>
<td>Optimistic</td>
<td>Early</td>
<td>2.2</td>
<td>0.7</td>
<td>1.1</td>
<td>83/99</td>
<td>0.2</td>
<td>-0.52</td>
<td>4</td>
</tr>
<tr>
<td>Conservative</td>
<td>Delayed</td>
<td>5.0</td>
<td>0.75</td>
<td>1.2</td>
<td>83/99</td>
<td>0.5</td>
<td>-0.52</td>
<td>4</td>
</tr>
<tr>
<td>Eager</td>
<td>Early</td>
<td>5.0</td>
<td>0.75</td>
<td>1.2</td>
<td>83/99</td>
<td>0.2</td>
<td>-0.52</td>
<td>4</td>
</tr>
<tr>
<td>Lazy</td>
<td>Delayed</td>
<td>2.2</td>
<td>0.7</td>
<td>1.1</td>
<td>83/99</td>
<td>0.5</td>
<td>-0.52</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3: Decision parameter configurations

These parameter settings are visualized in Figure 18, where the green lines are the threshold boundaries for switching up, and the red lines are the threshold boundaries for switching down. The labels next to the lines correspond to the configurations in Table 3.

For the bandwidth two different patterns will be used: decreasing and increasing. The bandwidth for the decreasing pattern is chosen in a way that the currently used Symbian player would encounter a buffer underrun when playing a stream at 64 kbps. First, the bandwidth is set to be 80 kb/s, which is just enough for the player to play the stream and in the meanwhile fill the buffer with some data. After one minute, the bandwidth is decreased to 50 kb/s, at which it will keep playing for 1.5 minutes. After this time, the Symbian player would have encountered a buffer underrun. Now the bandwidth is decreased even more, to 32 kb/s, at which the player will keep playing for another 2 minutes. The theoretical behavior of the Symbian player is displayed in Figure 19, assuming a best-case scenario, i.e. all bandwidth is efficiently used, no packet loss occurs, etc.
The bandwidth for the increasing pattern is chosen in a way that the player will first start at a bitrate at which the player is able to create a small buffer of data and can keep playing at that bitrate and at that bandwidth. First, the bandwidth is set to be 80 kb/s, which is just enough for the player to play the stream and in the meanwhile fill the buffer with some data. After one minute, the bandwidth is increased to 160 kb/s, which creates the possibility for the algorithm to switch to a higher bitrate. The theoretical behavior of the Symbian player is displayed in Figure 20, again assuming the best-case circumstances, i.e. all bandwidth is efficiently used, no packet loss occurs, etc.

**Figure 20: Theoretical behavior Symbian player with increasing bandwidth**

The patterns of the bandwidth provided to the mobile device is chosen so that the shortcomings of the proof of concept are taken into account, e.g. the lack of possibility to switch to another bitrate while a previous switch is not completed yet and the problems that still exist with packet loss.
Performance Measures
The performance measure in order to find the best parameter settings will be the average bitrate the stream is played under different circumstances, together with the number of switches needed to achieve this bandwidth. The other performance measure is a comparison between the provided bandwidth to the proof of concept and the actual bitrate the content is played at.

Monitoring and Presentation
During this test, the behavior of the proof of concept application is logged using the built-in logging functionality of the multimedia player. Because the logging takes place on different systems, the clocks have to be synchronized prior to the test runs. The actual bitrate of the playing stream can be presented against the provided bandwidth, and to compare the behavior of the decision algorithm using the different parameter settings, the behavior of the decision algorithm is presented against the behavior with different parameter settings under the same network circumstances.

Statistical Accuracy
Since the bandwidth provided to the proof of concept is completely controlled, all variables in the system are deterministic, and nothing is random. Therefore there is no need for any statistical analysis.

7.1.2 Test Results
Figure 21 till Figure 30 display the typical bitrate measurements for each of the tested configurations against the provided bandwidth: the figures on the left show the results from the tests with the decreasing bandwidth; the figures on the right show the results from the tests with the increasing bandwidth. Each of the bitrate configurations is tested with all of the parameter configurations: default, optimistic, conservative, eager and lazy. The characteristics and decision parameter settings of these configurations are given in Table 3 in subsection 7.1.1. Because all variables in the system are deterministic (i.e. there is no random behavior) each of the test runs is performed only once.
Figure 23: Decrease, optimistic

Figure 24: Increase, optimistic

Figure 25: Decrease, conservative

Figure 26: Increase, conservative

Figure 27: Decrease, eager

Figure 28: Increase, eager

Figure 29: Decrease, lazy

Figure 30: Increase, lazy
What can clearly be seen from these results is that with the configurations where the decision thresholds are relatively close to each other, a lot of switches are made. The runs with switching thresholds that are relatively far from apart, show much less switches. Table 4 gives the average bitrate and the number of switches measured for each of the configurations:

<table>
<thead>
<tr>
<th></th>
<th>Decrease</th>
<th></th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average bitrate (kb/s)</td>
<td>Number of switches</td>
<td>Average bitrate (kb/s)</td>
</tr>
<tr>
<td>Default</td>
<td>36</td>
<td>8</td>
<td>93</td>
</tr>
<tr>
<td>Optimistic</td>
<td>46</td>
<td>2</td>
<td>104</td>
</tr>
<tr>
<td>Conservative</td>
<td>40</td>
<td>5</td>
<td>83</td>
</tr>
<tr>
<td>Eager</td>
<td>41</td>
<td>7</td>
<td>91</td>
</tr>
<tr>
<td>Lazy</td>
<td>45</td>
<td>2</td>
<td>93</td>
</tr>
</tbody>
</table>

*Table 4: Measurement statistics of controlled environment tests*

What can be seen from these measurements is that the optimistic parameter configuration came out best with the highest average bitrate for both the increasing and decreasing bandwidth configuration. Second best is the lazy configuration. Although this lazy configuration does not provide the highest bandwidth, the characteristic that it switches up later than the optimistic configuration is a big advantage when taking in mind that we want as few switches as possible, because of the lack of multiple subsequent switches of the proof of concept application described in section 6.2.5. Besides that, ensuring a somewhat lower bitrate means that the buffer fill will be somewhat higher, making the lazy configuration a more safe one according to buffer underruns. Therefore in the real-life test, described in the following section, the lazy configuration is used.

### 7.2 Real-Life Environment

This test is a real-life comparison between the currently used multimedia player for the Symbian OS and the proof of concept containing the switch protocol and the decision algorithm. To make this a fair test, the conditions of the two players should be as equal as possible, and since the proof of concept player is based on the iPhone player that is still in development, the bugs and shortcomings that are still present in this proof of concept version will be left out of the test results, e.g. when the proof of concept stops playing because of a deadlock, it will not be counted as a buffer underrun.

#### 7.2.1 Test Case

The real-life test case is specified as described below.

**Goals**

The goal of this test is to verify the improvement of the proof of concept application compared to the currently used multimedia player for the Symbian OS. It is used to answer the last sub research question presented in section 1.2:

*What are the performance advantages of using the selected solution within the multimedia player?*

**Network Topology**

The topology of the network first of all exists of two devices: a Symbian OS device running the currently used multimedia player, and an iPhone running the proof of concept application. In order to take out the factor of different network connections, both devices are connected to a laptop computer via Wi-Fi, which provides them with a connection to the Internet. The laptop computer is connected to a mobile network using GPRS, UMTS and HSDPA through
another mobile phone. Sharing one Internet connection for two devices could introduce the fact that the two different players would have to compete for the available bandwidth and there might be a difference in TCP implementation that could influence the results. However, each device having its own mobile network connection could mean a lot more difference in available bandwidth. An Icecast streaming server together with the streaming proxy server on the server side provide the radio streams to the mobile devices. A schematic view of this network topology is given in Figure 31.

![Real-life test case network topology](image)

**Figure 31: Real-life test case network topology**

**Configuration**

The laptop computer fully routes the mobile Internet connection to the mobile devices, without any limitations. The Symbian device is connected to a stream at a fixed bitrate of either 29 or 64 kbps, but the proof of concept application is able to switch between streams at different bitrates. The parameter setting of the decision algorithm is the one that delivered the best results during the test with the controlled environment: the lazy parameter settings. During the test the complete and running client setup will be transported at random routes, either by car or by train.

**Performance Measures**

The performance measures of this test are twofold: first of all, the number of buffer underruns that occur is compared between the two different players. Secondly, the bitrate the proof of concept is playing content at is compared to the (static) bitrate of the Symbian player.

**Monitoring and Presentation**

During this test, the behavior of the proof of concept application and that of the Symbian player are logged using the built-in logging functionality of the players. Because the logging takes place on different systems, the clocks have to be synchronized prior to the test runs. The behavior of the two players will be compared to each other with respect to buffer underruns and stream quality.

**Statistical Accuracy**

We are aiming for a stochastic 90% confidence interval for the mean for the comparison of the average bitrate of both the players. Since we are working under real-life and non-controllable circumstances, we want to have enough test results in order to minimize the probabilistic bounds corresponding to this confidence interval. Therefore we will be testing for several hours.
7.2.2 Test Results

Figure 32 displays one of the performed real-life test runs. In this figure, the black line displays the buffer fill of the Symbian player; the dashed purple line displays bitrate of the Symbian player; the dashed red line displays the bitrate of the proof of concept application. Note that the legend explains the dashed red line as being the iPhone bitrate: this is because the proof of concept application is running on an iPhone. Unfortunately, the buffer fill of the proof of concept is not available from this test.

![Figure 32: Real-life test run (1)](image)

Figure 33 displays a close-up view of Figure 32 from 22:10:00 till 22:13:00, which gives an example of a situation where the currently used Symbian player experiences a buffer underrun, and where the proof of concept application manages to switch down to a lower bitrate in order to prevent this from happening.
What can be seen from this figure is that shortly after 22:11:00 the buffer fill of the Symbian player is decreasing fast. The same is true for the buffer fill of the concept application: this is not shown in the figure, but it can be seen from the bitrate the proof of concept application is switching to. First it switches down from 128 kbps to 64 kbps, and quickly after the switch is complete it switches down to 16 kbps, just enough to keep buffering enough data and prevent the buffer from running empty. The buffer of the Symbian player runs out of data, and the stream is reconnected and the music starts again. The proof of concept application switches back to 128 kbps when it detects enough data is in the buffer and the connection speed has increased again.

Figure 34 displays another one of the real-life test runs. Again in this figure, the black line displays the buffer fill of the Symbian player; the dashed purple line displays bitrate of the Symbian player; the dashed red line displays the bitrate of the proof of concept application, explained as iPhone bitrate in the legend because the proof of concept application is running on an iPhone. Besides that, the buffer fill of the proof of concept application is also displayed: with a blue solid line. This buffer fill is not measured during switches, since multiple buffers then exist and the implementation did not take this into account. During the switches, the buffer fill is interpolated, i.e. a line is drawn from the last known value before the switch to the first known value after the switch.
Figure 34: Real-life test run (2)

Figure 35 displays a close-up view from Figure 34 from 21:20:00 till 21:25:00. This close-up shows an example of adaptation from the proof of concept application. In this case, both the Symbian player and the proof of concept application experience a decrease in connection speed and thereby a decreasing amount of data in the buffer. Although no buffer underrun occurs in both players, it gives a clear view of how the bitrate of the proof of concept application is adapted to the available bandwidth. Would the Symbian player have been playing at a bitrate of 40 kbps or higher, according to these values a buffer underrun would have occurred.

Figure 35: Buffer decrease with both players

In total, eight test runs of between 35 minutes and 1 hour and 20 minutes were performed. Table 5 shows the statistics of the first four test runs, with regard to the average playing
bitrate of the proof of concept application (the bitrate of the Symbian player was 29 kbps at all times during these test runs), and the number of buffer underruns of both the Symbian player and the proof of concept application. The values in the lower row of Table 5 show the totals for the duration and the buffer underruns of both players, and the average bitrate of the proof of concept player, weighted against the duration of the tests.

<table>
<thead>
<tr>
<th>Duration (h)</th>
<th>Underruns (Symbian)</th>
<th>Underruns (iPhone)</th>
<th>Symbian Bitrate (kb/s)</th>
<th>Average iPhone Bitrate (kb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test run 1</td>
<td>0:52</td>
<td>1</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>Test run 2</td>
<td>0:43</td>
<td>1</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>Test run 3</td>
<td>0:38</td>
<td>2</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>Test run 4</td>
<td>0:34</td>
<td>2</td>
<td>2</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>2:47</td>
<td>6</td>
<td>4</td>
<td>29</td>
</tr>
</tbody>
</table>

Table 5: Measurement statistics of real-life tests with 29 kbps Symbian player

Table 6 shows the same statistics as Table 5, except for the last four test runs. The difference between the first and the last four test runs is that with the last four runs, the bitrate of the Symbian player was 64 kbps at all times.

<table>
<thead>
<tr>
<th>Duration (h)</th>
<th>Underruns (Symbian)</th>
<th>Underruns (iPhone)</th>
<th>Symbian Bitrate (kb/s)</th>
<th>Average iPhone Bitrate (kb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test run 5</td>
<td>0:44</td>
<td>2</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>Test run 6</td>
<td>0:30</td>
<td>6</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>Test run 7</td>
<td>0:40</td>
<td>7</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>Test run 8</td>
<td>1:20</td>
<td>5</td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>3:14</td>
<td>20</td>
<td>3</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 6: Measurement statistics of real-life tests with 64 kbps Symbian player

From the measured bitrates it is possible to create a confidence interval for the mean bitrate. The specification of the real-life test stated we are aiming at a confidence interval for the mean of the bitrate of 90%. The probabilistic bounds for a 90% confidence interval for the mean are given by formula 14.

\[
\left( \bar{x} - t_{[1-\alpha/2,n-1]} \frac{s}{\sqrt{n}}, \bar{x} + t_{[1-\alpha/2,n-1]} \frac{s}{\sqrt{n}} \right)
\]

with \( \bar{x} = \text{average bitrate (115)} \), \( \alpha = \text{significance level (0.1)} \), \( n = \text{number of samples (4)} \), \( t_{[1-\alpha/2,n-1]} = \text{t-quantile (2.353)} \), \( s = \text{standard deviation (8.45)} \)

The average bitrate (\( \bar{x} \)) and the standard deviation of the measured bitrates (\( s \)) are both weighted against the durations of the tests. From filling in all these values, we can now claim with a 90% confidence interval that the mean of the measured bitrates of test run 1 to 4 is between 105 and 125 kbps, and that the mean of the measured bitrates of test run 5 to 8 is between 119 and 129 kbps. We can also use this formula the other way around to calculate the probability of the bitrate of the iPhone player being more than 29 kbps or more than 64 kbps, by looking up the value for the t-quantile belonging to the bounds of the mean being between 29 and 201 (115-86 and 115+86) kbps or between 64 and 184 (124-60 and 124+60) kbps. The value found for \( t_{[1-\alpha/2,n-1]} \) for these probabilistic bounds corresponds to a confidence interval of over 99.95% that the measured bitrate of the iPhone player will be more than 29 kbps in test
runs 1 to 4, and more than 64 kbps in test runs 5 to 8. Concluding from these results, we can say with 99.95% confidence that the proof of concept application is capable of playing content at a significantly higher average bitrate than the currently used Symbian player, while in the meantime experiencing an equal amount or fewer buffer underruns.

Using formula 14 for determining a confidence interval for the mean of the number of buffer underruns, we can say with a 90% confidence interval that the mean of the number of buffer underruns from the Symbian player playing at 29 kbps is between 0.9 and 2.1, and playing at 64 kbps is between 2.8 and 7.2. Doing the same for all the test runs of the iPhone player, we can say with a 90% confidence interval that the mean of the number of buffer underruns is between 0.135 and 1.165. From these numbers we can conclude with a 90% confidence interval that with the Symbian player playing at 64 kbps, the number of buffer underruns is higher than the number of buffer underruns the proof of concept application is experiencing, while it is playing at a lower average bitrate.

7.3 Demonstration

The last test case is created for demonstration purposes to be able to show the behavior of the decision algorithm. Since this test case is for demonstration purposes only, no statistical analysis and no performance measures are specified for this test case.

Goals

The goal of this test is to be able to demonstrate the behavior of the decision algorithm when it is used in a real-life environment.

Network Topology

The network topology for this test exists of a mobile device (iPhone) running the proof of concept application, connected to a laptop computer that provides an Internet connection to the mobile device via Wi-Fi. Furthermore, the mobile device is connected to a 3G network. An Icecast streaming server together with the streaming proxy server on the server side provide the radio streams to the mobile device. A schematic view of this network topology is given in Figure 36.

![Figure 36: Demonstration test case network topology](image)

Configuration

The mobile device is connected to both Wi-Fi and a 3G access networks, from which the Wi-Fi network has to be controllable by the person giving the demonstration. By actively switching on and off the Wi-Fi network, the reconnecting functionality of the proof of
concept can be demonstrated. By controlling the bandwidth of the Wi-Fi network the bitrate
switching functionality can be demonstrated.

**Monitoring and Presentation**
During this test no monitoring is performed, since it is for demonstration purposes only. For
the presentation, a user interface is shown that displays the status of the buffers of the active
stream and the stream the player is switching to if applicable. It also shows a graph of the
buffer switching thresholds, together with the current position of the active stream during
playback. On this graph, the buffer fill is displayed on the x-axis, against the connection speed
on the y-axis.
8. Conclusions and Recommendations

This chapter concludes this Master Thesis in section 8.1 and recommends future work for the switching algorithm and the proof of concept application in section 8.2.

8.1 Conclusion

The goal of this assignment as stated in section 1.2 was to increase the user’s audio experience of Mobilaria’s streaming radio service by providing a solution for the radio stream disruptions caused by buffer underruns and thereby increasing the reliability, stability and robustness of the streaming radio service. Preliminary research preceding this Master Thesis, [7] shortly described in chapter 2, uncovered the underlying reasons for these buffer underruns: the on-demand system used by Mobilaria, source stream disconnects, loss of connectivity either caused by horizontal handovers or not, and insufficient bandwidth are identified. This provided the opportunity to answer the first research question:

\textit{Which solutions are possible to solve the radio stream disruptions caused by buffer underruns?}

Chapter 3 proposes multiple possible solutions for the problems identified in chapter 2. Because this assignment could only cover one of the proposed solutions, the following research question was introduced to select one of the proposed solutions:

\textit{Which criteria can be identified to efficiently select one of the proposed solutions?}

At the end of chapter 3, the choice is made to select the solution of bitrate switching, since it would have the most scientific value over all the other proposed solutions. The bitrate switching solution would at the same time partly cover the reconnection problem, so potentially even more buffer underruns can be prevented by this solution. A proprietary protocol developed by Mobilaria can be used for the bitrate switching solution, which has to be extended by a decision algorithm. Chapter 4 discusses other work related to this topic, and chapter 5 addresses the creation of the bitrate switching decision algorithm. The following research question reads:

\textit{How could the selected solution be designed and implemented into the current multimedia player?}

The design and implementation of a proof of concept application is discussed in chapter 6. During the design and the implementation, a disadvantage of the switch protocol was uncovered, which would extremely influence the successful operation of the decision algorithm. Because of this, a streaming proxy server was introduced that is necessary to optimize the bitrate switching protocol and prevent the client from buffering huge amounts of data twice, and thereby even create a higher probability to end up with a buffer underrun. After the design and the implementation of the proof of concept were completed, the following research question could be addressed:

\textit{What are the performance advantages of using the selected solution within the multimedia player?}

Chapter 7 validates the performance improvement of the proof of concept application regarding the buffer underruns by specifying multiple test cases in both a controlled environment and a real-life environment. Concluding from the validation tests performed, we can say that the bitrate switching decision algorithm ensures a significant improvement to the streaming radio service offered by Mobilaria. The proof of concept application containing this
decision algorithm is able to stream the same content in a significantly higher bitrate than the currently used Symbian player, while in the meantime experiencing fewer buffer underruns. This claim is made with a 99.95% confidence interval. The decision algorithm showed that it was able to keep the amount of buffered data at an acceptable rate by switching down to a lower bitrate during periods of a decrease in available bandwidth. The most efficient algorithm configuration is the one where both switching to a higher bitrate and switching to a lower bitrate is done relatively late, which ensures a low number of switches.

8.2 Recommendations

In addition the other solutions to the general problem proposed in section 3.1 that were left out of scope of this thesis and the improvement areas of the proof of concept player discussed in subsection 6.2.5, this section introduces some new interesting areas of future research that could even more improve the switching functionality of Mobilaria’s streaming radio service.

8.2.1 Bitrate Transitions

Currently, when switching between to streams of a significant difference in quality, the transition between the two streams is audible in a way that the quality is improving or deteriorating. For example, when switching from a 64 kbps or higher stream to a 32 kbps or lower quality stream, the audio becomes dull, contains artifacts and high frequency tones are missing. The same for the other way around: the audio suddenly sounds clearer and contains more high frequency tones. When multiple switches are issued, this could decrease the quality of experience for the listener.

To solve this issue, one could think of the e.g. letting the system create a slight overlap in audio from different quality from the two different streams, which are mixed together in a fading way to prevent a sudden transition between two different qualities. This way a smooth transition from one quality to another is made, which is less audible to the listener and thereby improves its listening experience. A possible area of future research is to find the best way to cover such a transition in quality.

8.2.2 Vertical Handovers

At this moment, when a network connection is lost and a switch is made to another network, or when another network is found, the algorithm does not know the characteristics of this new network where it will be switching to. For example, the player is playing at 128 kbps using a Wi-Fi network, the device travels out of reach of the network, and now the only available network is a GPRS network which would only be able of supporting a stream up to 32 kbps. The algorithm does not know this and creates a new connection to the 128 kbps stream, which would result in a very fast decrease of the buffer fill. Therefore it would be useful to know the characteristics of network regarding e.g. maximum data throughput or costs, or to be able to in someway measure some of these characteristics before a new connection is made.
References


