CHALMERS

Media Streaming For Infotainment

Predictive streaming using adaptive bitrate and buffering

Master of Science Thesis in the Programme Networks & Distributed Systems

Andreas Lilleste
Lukas Lundgren

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, June 2009

Media Streaming For Infotainment
Predictive streaming using adaptive bitrate and buffering

Andreas Lilleste
Lukas Lundgren

Examiner: Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2009

Abstract

This master thesis will investigate if it is possible to vary the bitrate of a media stream in real time to make it fit over a link with varying quality. The varying bitrate of the media stream is based on different strategies for when to buffer and how much needs to be buffered. Strategies for buffering uses an estimated bitrate for the link to be able to predict when and where the link has high bandwidth or low bandwidth. The estimated bitrate is constantly updated and based on previously measured bitrates. Streaming is done between a client and server where the client is connected against the server trough a mobile broadband or a wireless accessed point.

Sammanfattning

I den här rapporten kommer vi att undersöka om det är möjligt att variera bandbredden på en mediaström i realtid så att den kan skickas över en länk som har en varierande kvalité. Den varierande bandbredden på mediaströmmen är baserad på olika strategier över när buffring ska ske samt hur mycket som behövs buffra. Strategierna använder sig av en estimerad bandbredd för länken för att på så sett kunna prediktera när och var länken har lägre eller högre bandbredd. Den estimerade bandbredden uppdateras kontinuerligt och är baserad på gamla uppmätta bandbreddsvärden. Mediaströmmen skickas mellan en server och en klient där klienten är uppkopplad mot servern via ett mobilt bredband eller ett öppet trådlöst nätverk

Preface

The original idea for this Master Thesis came from the company Mecel AB. The work was carried out by Andreas Lilleste and Lukas Lundgren. We are both grateful to have been given the opportunity to develop and put the original idea into practice. We want to thank our supervisor at Mecel, Lars-Christian Aadland and Jan Jonsson, our examiner at Chalmers. We also would like to thank Lars Matsson and Anders Eliasson at Mecel for all expertise and helpful feedback.

Contents

Abstract i

Sammanfattning ii

Preface iii

1 Introduction 3
1.1 Background . . . 3
1.2 Thesis description . . . 5
1.3 Delimitation . . . 6
1.3.1 System layout . . . 6
1.3.2 Client route . . . 6
1.3.3 Data collection . . . 7
1.3.4 Open Source . . . 7
1.4 Organization . . . 7

2 Method Description 8
2.1 Design . . . 8
2.2 Implementation . . . 8
2.3 Testing . . . 8
2.4 Measuring . . . 8

3 Analysis 10
3.1 Coverage deviations . . . 11
3.2 Target system . . . 12
3.3 Operating System . . . 12
3.4 Development Software & Environment . . . 13
3.5 Programming languages . . . 13
3.6 Transport method . . . 13
3.7 Open Source Software and the Automotive Industry . . . 13
3.8 Libraries & Software . . . 14
3.8.1 GStreamer . . . 14
3.8.2 libavcodec & libavformat . . . 14
3.8.3 Clutter . . . 15
3.8.4 Twisted . . . 15
3.8.5 D-Bus . . . 15
3.8.6 x264 . . . 15
3.8.7 Cairo . . . 15
3.8.8 PostgreSQL . . . 15
3.8.9 PostGIS . . . 16
3.8.10 gpsd . . . 16
3.8.11 Apache HTTP Server . . . 16
3.8.12 Open Street Map . . . 16

3.9 Market Survey . . . 17
3.9.1 Techniques . . . 17
3.9.2 Summary . . . 18

4 Implementation and Analysis of the Client 19
4.1 Client Backend . . . 19
4.1.1 GPS . . . 20
4.1.2 Wifi . . . 20
4.1.3 Cellular . . . 20
4.1.4 Networks . . . 21
4.1.5 Simulation . . . 22
4.2 Client Frontend . . . 23
4.2.1 Playback . . . 24
4.2.2 Tools . . . 24
4.3 Packet Routing . . . 26

5 Implementation and Analysis of the Server 27
5.1 Open Street Map . . . 27
5.2 Database . . . 28
5.3 Coverage Map . . . 30
5.4 Streaming . . . 32
5.4.1 Video Codec . . . 32
5.4.2 Transcoding . . . 33
5.5 Measuring bandwidth . . . 36
5.6 Estimating available bandwidth and velocity . . . 37
5.7 Strategies for adapting bitrate and buffering . . . 39
5.7.1 Identifying intervals to buffer . . . 40
5.7.2 Finding intervals where buffering is possible . . . 40
5.7.3 Implementation . . . 41

6 Results 42

7 Conclusion & Discussion 43

Glossary 48

Index 49

Appendices 50

A Cellular provider comparison 50

B D-Bus API 51

List of Figures

1 System overview 11
2 Reference route 11
3 Throughput at different times on the same route 12
4 Backend overview 20
5 GStreamer pipeline client-side 24
6 Screenshot of a coverage map 31
7 Data throughput VS. encoding bitrate 33
8 GStreamer pipeline server-side 35

List of Tables

1 System Parts 10
2 Client hardware table 12
3 Different network states 22
4 Data sent to server 24
5 Table of what velocity a certain road type has 27
6 Table in the measurement database 29
7 Bitrate to colour table 30
8 Tested processors 32
9 Cellular provider comparison 50

1 Introduction

1.1 Background

The Internet has changed people's communication habits. Just decades ago when people wanted to communicate they had to send a letter or make a phone call through a land line connected telephone. Letters could take days to arrive but with the revolution of Internet, email has become a fast and reliable communication channel. As the wireless nets becomes faster and covers larger areas people will be able to stay connected to the Internet anywhere and at any time. With a modern mobile phone it is possible to browse the Internet and read emails nearly anywhere. Today people use the Internet for business, entertainment, education and information seeking. A new trend is that home appliances such as TV sets and stereos come equipped with network plugs and Wifi. This enables streaming of MP3 music, movies, Internet radio and YouTube clips direct to the TV set or stereo. The streaming could be done from a home computer where all media such as pictures and movies and music is stored on. A new actor on the streaming market is Spotify [1]. They deliver a vast library of music to anyone who is connected to the Internet and has a Spotify account. They have succeeded in establishing contracts with some major record companies to legally stream copyrighted music. All this streamed digital media will soon lead to a demand for streaming media to cars as well.

Infotainment systems in cars today come with many features such as satellite navigation, Bluetooth and even the possibility to play DVD movies. As these systems become more available and used, the demand for streaming media to the car grows. A main advantage with being able to steam the media to the car is that you get media on demand. You don't have to buy a DVD or a music CD in advance to be able to listen to it in the car. With streaming passengers will be able to watch their favorite YouTube clips while heading down the highway. A problem with streaming to mobile units is the varying bandwidth and the bandwidth you get depends on many different factors such as how congested the network is and on the physical surroundings. There could be a building in the way or a low valley where the coverage of the wireless link is bad. This leads to an unreliable link with a lot of bandwidth fluctuation. When streaming media to mobile units it is not possible to guarantee any good Quality of Service (QoS) because that would require every Internet service provider to priority the stream in their network. As wireless networks depend on many different factors, it is hard to guarantee QoS over a wireless network.

This Master Thesis will try to stream media over wireless networks with some QoS. The main focus of this master thesis will be to present a way

to predict the expected bitrate over a link for a specific position. The link will be between a server, which is connected to the Internet with a fast connection, and the client which is connected to the Internet using a 3G connection or a local wireless access point. Previously measured bitrates at a specific position between the server and client are stored in a database which is located on the server. With the help of these measurements and the route of the client it is possible to make good buffering decisions such as when to buffer and how much buffering is needed. The media will be encoded to an estimated bitrate to make it fit on the link.

To be able to stream media on such a link buffering of the media is needed. As a result of the buffering you would have to wait for the movie or music to start playing or worse the media will have to pause to buffer. Another problem with buffering is to determine how much to buffer to be able to play the media without having to pause or wait to long for the media to start playing. To be able to determine how much to buffer you will have to be able to predict what bitrate you can get out of the link at a certain time and place.

This report will try to give a solution how to estimate the bitrate for a link at a certain time and place, and with the help of this estimated bitrate determine a good strategy for buffering and adapting the A/V bitrate in real time. With good strategies it could be possible to achieve higher reliability and better quality on the media stream. If the estimated bitrate and the buffering strategies work well, the result of this thesis could be interesting to use in the future where it would be possible to watch a movie or listen to music in the car without having to download the media in advance. It will truly become media on demand.

This master thesis will also briefly discuss already existing solutions for streaming media to mobile units such as Multimedia Broadcast Multicast Service (MBMS) and Digital Video Broadcasting Handheld (DVB-H). An assessment of how the market for mobile streaming looks today will also be made and how these techniques could evolve in the future.

1.2 Thesis description

The original preliminary thesis description stated:

> The thesis work includes setting up a server environment which can receive streamcast/YouTube/Joost from the Internet as well as DVB-T/DVB-S and stream these via 3G cellular networks in a compressed format suitable for viewing in an Infotainment environment. The server should also be able to stream locally stored media such as DivX/MPEG/MP3. The receiving side, which should be Linux based, suitable open source software should be used to either store the media stream for later playback or live playback.
>
> - Suggest possible suggestions for the client and the server. The focus is on the client.
> - Define requirements for QoS for live and stored playback. Is respective scenario viable?

Before the work on this thesis begun the goals was reworked to incorporate some ideas Mecel AB had regarding predicting coverage and how to use it to optimize streaming. The original goals were extended to reflect this:

- Suggest possible solutions for the client and the server. The focus in on the client.
- Define requirements for QoS for live and stored playback. Is respective scenario viable? How can compression, buffering and dynamic buffering be used to increase availability. Is predictive availability/coverage a possible solution? Can roaming between 3G and Wifi help QoS?
- As an overview, a short market survey of other techniques for mobile A/V, should be included: DVB-H, IP-streaming, MBMS.

As the work began at Mecel AB, we started to investigate on what parts of the problem to focus on together with our supervisor. It was decided to focus on adapting buffering and the A/V bitrate using predictions of the coverage environment for a client. The project shifted somewhat from the original description which described the server requirements in detail but stated the client as the point of focus, to how to best stream an A/V stream using techniques to predict coverage.

1.3 Delimitation

This master thesis contains many different parts such as the collecting of measurement to the database, compute strategies for streaming, streaming the media, testing of all parts and to visualize it in a good and understandable way. Due to the size of the many different parts of this master thesis some delimitation were necessary.

1.3.1 System layout

The proposed solution in this Master Thesis only has one server and a client where the media is streamed from the server to the client. This kind of solution has a drawback that it does not scale very well. As the number of clients grows the server will have to do a lot of heavy calculations such as encoding and decoding. Other system like DVB-H uses multicast which scales better. A drawback with DVB-H is that it does not deliver truly media on demand. You only have a selection of channels which you can watch. Therefore you will not be able to watch or listen to a specific tune or movie whenever you want. To be able to deliver true media on demand a server and client structure is needed. A drawback with this kind of system layout is that the number of clients the server can handle is limited by the hardware performance of the server and the Internet connection to the server. To be less dependent of the server hardware performance the system could possibly be modified to use Peer to Peer (P2P) network but with today's mobile 3G nets the upload is limited. For this reason P2P is currently not a practical solution. A P2P solution could also have difficult legal issues such as copyrights.

The proposed system has only been tested with one client due to limited access to clients. But the system has been implemented with many clients in mind. And to make the system scale a home media server would stream to the client. Nearly every household today has a computer where they have stored pictures music and videos. This computer could act as the streaming server and this would make the server client layout scale very well.

1.3.2 Client route

A delimitation is that the client knows which route it will travel and that it never deviates from the route. At the beginning the client sends which route it will travel to the server. Calculation of the clients route is out of this master thesis scope and therefore predetermined routes are used which are stored at the client. Due to limited time for implementation the system does not handle if the client deviates from the specified route. But a possible solution would be to look at what position the client previously had and use the gathered information about this position. With the help of this data

it would be possible to make some rough estimations about were the client might be and what bitrate it might get.

1.3.3 Data collection

When collecting measurements of the cellular networks, only data along specific routes were collected. Instead of collecting data from a very large area, data were collected from specific smaller routes many times. This gives more data for specific routes and leads to higher accuracy when computing estimations of the bitrate.

1.3.4 Open Source

Mecel required that Open Source software and libraries were used. Because they believe Open Source will become more used within the Automotive industry.

1.4 Organization

The rest of this report is organized as follows: In Section 2 we discuss the method which is followed with an analysis in Section 3, 4 and 5. Finally in Section 6 and 7 the results and conclusions of this Master Thesis is discussed and presented.

2 Method Description

2.1 Design

We chose to use a server client architecture because we found this system type the most appropriate for this project. A P2P[2] solution was discarded because of the limited uplink bandwidth of cellular networks. A server client architecture does not scale very well as the system is limited by the server hardware performance and the Internet connection of the server. But we believe this could be solved because nearly every household has a computer at home which could act as a streaming server to the client.

2.2 Implementation

The implementation phase overlapped somewhat with the design phase. Hardware handling on the client was implemented first as it was vital for testing the networks during the development.

2.3 Testing

Before testing of the system could be made we had to gather measurement data for the cellular networks. This required several trips out on the field where the client where rigged in a car and set to measure the bandwidth of the different cellular networks. All the measurement was logged and later inserted into the server database.

To test our system we implemented a simulation application on the client which simulated a route. A problem with just simulating the route is that we get the same bandwidth on the wireless cellular link all the time. But to truly test the system we had to bring it out on the field and go between areas where the bandwidth changed from good to worse. This was time consuming to do so only a few tests were made. Tests were made on the route between Landvetter and Hindås in Sweden. This route was chosen because the bandwidth went from good to worse along the route. Here we could see if the system would compensate for the changing bandwidth. The result of the test is found in section 6 on page 42

2.4 Measuring

A few different programs were tested to make accurate bitrate measurements between the server and client. Measuring was done from the server to the client. A problem might be that the client is behind a firewall. This could be solved by making the connection from the client to the server. We tested a program called *iperf*[3] which is a client server based program which can measure bandwidth on both the uplink and downlink. This program was

discarded due to problems with stability. A program called *tptest*[4] was also tested. This program is also a client server based and was discarded due to lack of stability and problems with connectivity. A simple server application was written which listens for connections and as clients connect sends as much garbage data as possible. This program is described more in section 5.5.

3 Analysis

The primary task of this project has been to implement a system for adaptive streaming of an A/V stream over various transmission media such as cellular Universal Mobile Telecommunications System (UMTS) and Wifi networks. Adaptive streaming is to adjust the stream to be sent according to the unique, constantly changing, environment of each client. This can be done by changing the bitrate of the A/V stream in real time or buffer intelligently. Most adaptive solutions today choose a bitrate at connection for each client which may not be optimal on a link which varies heavily in available bandwidth depending on different external factors. The project requires the creation of both a server application to stream data as well as a client to receive it.

The proposed system can be divided into several parts:

Table 1: System Parts

System	Task
Client	Hardware control (Modems, GPS) Network handling Routing Measure bandwidth Receive & buffer stream Play stream
Server	Transcoding Streaming Database (Measurements) Estimation (Bandwidth, Velocity) Strategies for streaming Coverage maps

The proposed system deviates from a standard server/client streaming solution as the server needs to estimate what conditions the client will experience in the near future to be able to adapt both bitrate and buffering. This will be done by collecting bandwidth measurements consistently on the client which will be sent to the server. Given a geographical route the server should then produce a realistic strategy to stream provided there is enough previous collected data along the route.

As the availability of different public networks increases so does the potential networks available for use to vehicles. The application proposed should be able to use both cellular networks as well as Wifi networks interchangeably

and roam between available networks as the vehicle moves. A crude overview of the system can be seen in figure 1.

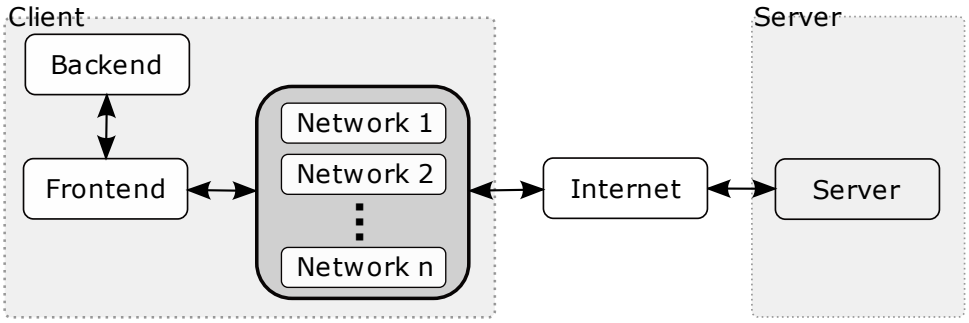

Figure 1: System overview

3.1 Coverage deviations

The basis for this thesis is handling coverage deviations for different networks, the major one being cellular networks. A reference route was used for measurements during the later parts of the development, as can be seen in figure 2. Several separate measurements were made on this route at different times to show that coverage deviates during this route. Figure 3 shows the different measurements of bandwidths at five different times along with a combined estimate of available bandwidth using the data from all previous measurements.

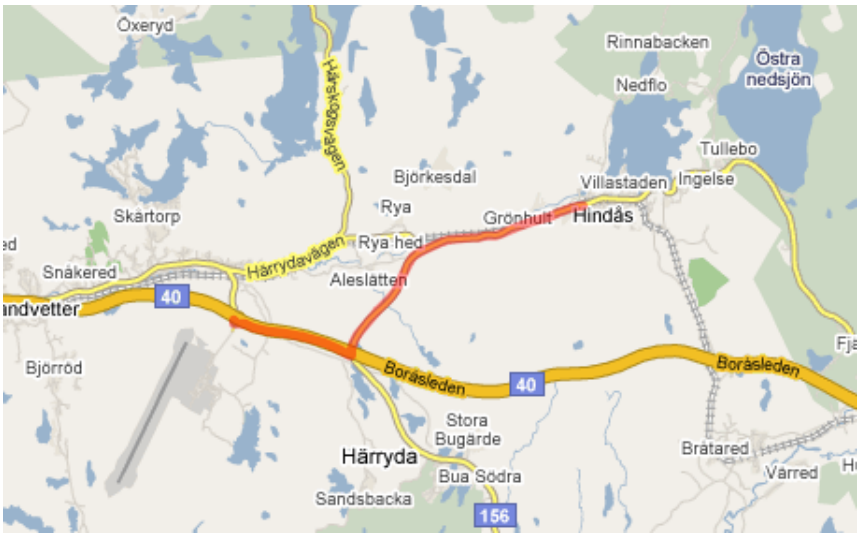

Figure 2: Reference route

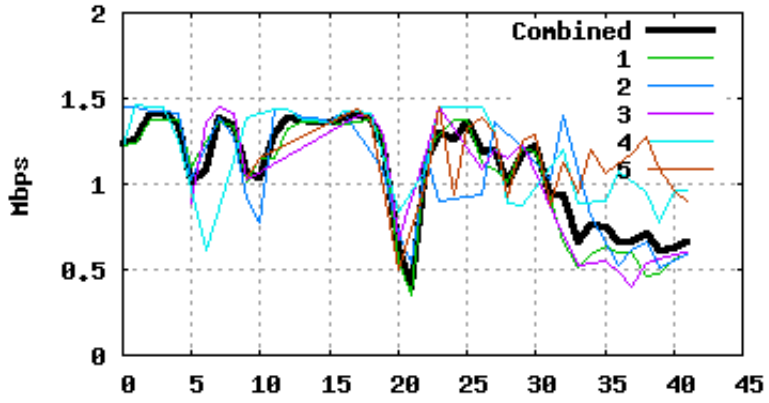

Figure 3: Throughput at different times on the same route

3.2 Target system

Mecel AB proposed a client with an Intel Atom processor due to the fact that new infotainment platforms will be based on this processor type. See section 3.7 for more information about mobile platforms. If the client had similar hardware as these upcoming infotainment platforms an integration of the proposed system would hopefully be easier. The client needed to be a laptop because of the measurement test would take place in a moving car and therefore would a laptop make the testing easier. An Asus 901 EEE PC was chosen as the client system because it is flexible and powerful enough for decoding media stream. For more detailed information about the hardware see table 3.2.

Component	Description
Display	8.9“
Processor	1,6 GHz Intel Atom N270
Memory	1024 MB SO-DIMM DDR2 RAM
Disk	20 GB SSD
Wireless card	802.11n, Bluetooth
Weight	1.1 kg

Table 2: Client hardware table

3.3 Operating System

Mecel AB believes that Open Source operating systems will play a prominent role in future automotive Infotainment systems. Therefore we chose to use *Arch Linux*[5] which is a light weight and flexible Linux distribution which

we also had some previous experience of. We chose to use *Arch Linux* both on the client and on the server for smoother usage.

3.4 Development Software & Environment

IBM Rational ClearCase was used for *version control* of the code created. ClearCase is used at Mecel AB and was the reason why it was used for this Master Thesis. The UCM[6] layer of ClearCase was also used. The report was written using *Latex* and a separate external *Subversion* repository was created to provide easy access to the report from outside Mecel AB.

3.5 Programming languages

Several programming languages were considered for both the server side as well as the client side. Due to time constraints a managed higher level language was preferred. The choice fell on *python* as the main language to be used. The libraries used, see section 3.8, all have language bindings for python even though they are primarily written in *C*. Some *C* was used to modify the libraries when necessary.

3.6 Transport method

The client and server communicate via two channels. A *data* channel which is only used to send the A/V stream to be played by the client and a *communication* channel. TCP was chosen over UDP due to the need of an *ordered*, *reliable* protocol for both the *data* channel and the *communication* channel. The client does not handle error correction or fault tolerance of the A/V stream received.

3.7 Open Source Software and the Automotive Industry

Open Source is a computer program or source code which is available for anyone to use and modify. An open source code project is usually maintained by a group of people. The modifications on the original source code or program which is done by other people is usually passed back to those who maintain the project. This is done so the maintainer of the original project can decide if the modifications should be part of the original program or source code.
Mecel AB required that Open Source was used through out the master thesis because they believe Open Source will play a big role in future In-Vehicle Infotainment (IVI) systems to cut development times and costs.

GENIVI

GENIVI is a non-profit industry alliance with the goal of driving the adoption of an IVI open source development platform. It was founded by the BMW Group, Wind River, Intel, GM, PSA, Delphi, Magneti-Marelli, and Visteon. By developing a reusable platform consisting of the essential core services of an IVI system, such as media and graphics, it is believed this will result in shorter development cycles, quicker time-to-market and reduced costs for companies developing IVI systems. GENIVI supplies a reference software architecture platform called the *GENIVI Platform*. It contains Linux based core services, middleware and an open application layer interface. Automobile manufacturers and suppliers can add their products on top of this architecture.

Moblin

Moblin is a Linux platform specifically designed for mobile devices such as netbooks, Mobile Internet Devices (MIDs) and IVI systems. The architecture contains a layer called the *Moblin Core*. This layer is hardware and usage model independent which provides a uniform way of developing for mobile devices. The layer below the *Moblin Core* contains the Linux kernel together with specific device drivers for the target hardware platform. The uppermost level contains a specific user interface for the target device. The *Moblin Core* is built upon the *GNOME Mobile Platform*[7]. Moblin will act as an independent distribution mechanism for the first *GENIVI* reference implementations.

3.8 Libraries & Software

A number of different open source libraries and software was evaluated for inclusion into the project. The main reason for the use of them in the project was to cut development times and make development easier. The libraries are listed below with a description and why each was used or not used.

3.8.1 GStreamer

GStreamer[8] is a cross platform multimedia framework written in C. Audio and video components are chained together into pipelines which allows for playback, recording, editing and streaming. GStreamer is free software, licensed under the *GNU Lesser General Public License*. It is used within the project to transcode, stream and finally play the stream on the client.

3.8.2 libavcodec & libavformat

libavcodec and *libavformat* are both part of the open source FFMPEG[9] project. *libavcodec* is a library containing codecs for encoding and decoding

video and audio and *libavformat* contains *muxers* and *demuxers* for a number of different formats. Both are licensed under the *GNU Lesser General Public License*. The two libraries was considered for use in the transcoding part of the project, see section 5.4.2, but *GStreamer* was chosen instead as it offered a more versatile solution.

3.8.3 Clutter

Clutter[10] is an open source graphics library for creating hardware-accelerated user interfaces. It is used to present the A/V stream on the client. Licenced using the *GNU Lesser General Public License*.

3.8.4 Twisted

An event-driven network programming framework written in Python. Twisted[11] is licensed under the *MIT License* and is used on the client to simplify handling of both the communication channel as well as the data channel.

3.8.5 D-Bus

A message bus system providing interprocess communication. It's widely used within the project for:

- Receive notifications of hardware changes (3G modems).
- Backend/Frontend communication.
- Queuing creation of coverage maps.

3.8.6 x264

A free software library for encoding H.264/MPEG-4 AVC video streams. Licensed using the *GNU General Public License*. Used by *Gstreamer* to encode H.264 video.

3.8.7 Cairo

Cairo[12] is a free software library for creating and modifying graphics which was used on the server for rendering coverage maps.

3.8.8 PostgreSQL

PostgreSQL[13] is a open source database which was used on the server for storing the bitrate measurements and the OSM data.

3.8.9 PostGIS

PostGIS[14] is an extension to PostgreSQL which adds support for geographic objects. This was used on the server for easy and fast calculation of distance between two positions.

3.8.10 gpsd

Gpsd[15] is a software daemon which allows other programs to access gps data without contention or loss of data. A Gps unit could be connect trough a Serial port or a USB port. Gps data is queried from gpsd via TCP. Gpsd was chosen because of it easy use and it exists a Python wrapper class. It is used on the client to make easy gps position requests.

3.8.11 Apache HTTP Server

Apache[16] webserver was used on the server so the client could access the coverage maps.

3.8.12 Open Street Map

OSM[17] is a open source map which is open for use and contribution by anyone and it covers the whole world. OSM is used on the server for creating coverage maps.

3.9 Market Survey

The Internet has become a massive distribution channel for media and people are storing pictures, movies and music on their computers. Today it is possible to stream that media direct to a TV. As this becomes more used a demand for playing media on mobile unit will grow. Mobile service providers use different approaches to deliver media to mobile units. One approach is to download the media in advance to the mobile unit before playing it. This enables the viewer to start watching or listening on the media from the beginning. Another technique is to stream the media direct to the mobile unit.

3.9.1 Techniques

It exist a few different techniques for streaming media either trough unicast, multicast or P2P. If unicast is used a dedicated stream is setup between the streaming server and the mobile unit. This enables every mobile unit to watch whatever they like whenever they like. A drawback with unicast is that it is not an efficient way to deliver the same stream to many units because of one stream is needed for every mobile unit.

An efficient way of delivering the same stream to many mobile units is to use multicast. With multicast one stream from the server can be delivered to many mobile units at the same time. The MBMS[18] and DVB-H are both multicast techniques but they use different distribution networks. MBMS uses the UMTS network for delivering the same stream to many users. While DVB-H requires a whole new distribution network to be built and new dedicated frequency bands. Today it exist working DVB-H networks in Finland, India and Italy. Recently the EU commission decided that DVB-H is going to be the official standard for delivering mobile TV within Europe. Today it does not exist any working MBMS network but Ericsson AB which develops MBMS believes that MBMS will come to play dominant role in the future.

Another technique for streaming media is P2P[2]. In a P2P network the clients pull fragments of a file from other clients which posses the same file. In this way the client gain a higher total bitrate than if it would pull the file from only one source. Joost[19] is a system which uses the P2P technique and if it does not exist any other clients with the same file the client can pull the file from a main server. P2P requires a good uplink from the clients and therefore it is not a practical solution with today's slow upload bitrate of UMTS networks.

3.9.2 Summary

A few different techniques exist for distributing media as described above. Some of them are working today already and some will maybe come in the future. According several surveys found in [20], people are willing to pay for mobile streaming services. These surveys clearly shows it exist a market for streaming to mobile units. A combination of unicast and multicast would be the optimal solution for streaming media. MBMS or DVB-H would deliver TV channels to the great mass of mobile units and unicast will enable individual media streams to the units. Unicast could be delivered trough a cellular or Wifi network.

4 Implementation and Analysis of the Client

The purpose of the client is twofold: it should not only play an A/V stream produced by the server but also gather information about the client's environment. This information would then be passed on to the server which does all the actual calculations.

It was decided to control the necessary hardware, such as modems and Wifi cards, in a separate application on the client. This application would handle available networks along with the hardware needed for each network. Other applications may communicate with this application to retrieve information about different networks as well as GPS location and velocity. In a *GENIVI/Moblin* context, this subsystem can be seen as a service with a set of Application Programming Interfaces (API) for other applications to use. This application is hereby referred to as the *backend*, see section 4.1. The application responsible for playing the A/V stream is referred to as the *frontend*, see section 4.2.

4.1 Client Backend

The main purpose of the backend is to decouple the handling of hardware and networks from the actual streaming and user interfaces. The backend is a service which runs in the background keeping track of the state of hardware, networks and other resources needed for the streaming application. Other applications may query the backend, or receive signals when changes occur, using *D-Bus*. D-Bus is used by the majority of Linux distributions today and is also part of the Moblin platform[21]. The use of D-Bus enabled us to develop the backend early and use it continuously during development, testing and also as a help for researching what bandwidths to expect using different cellular operators. Simple tools could be developed to extract the information necessary. For an API reference of the implemented *D-Bus API*, see appendix B. As the backend acts as a service it has no user interface, important information and events are logged to a simple log file. The user interface is instead implemented as a separate *frontend*. Some use of the devices handled by the backend is also controlled by different frontends, see section 4.2.

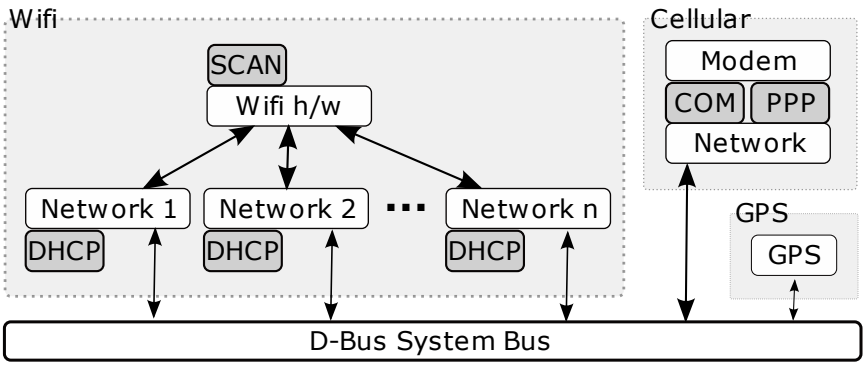

Figure 4: Backend overview

4.1.1 GPS

The backend handles the *GPS unit* by the use of *gpsd*, see section 3.8.10. The GPS daemon is polled frequently for longitude, latitude, velocity and connection status to satellites. When the data has changed the backend emits a signal notifying frontends that the GPS data has changed. It is also possible to ask the backend for the latest data retrieved from the GPS.

4.1.2 Wifi

The *Wifi interface* of the client is continuously used to scan for new access points. Only open access points, that are not using WEP/WPA or any other cryptographic protocol, are considered. Available access points are added as offline and are not checked if they are actually useable. The networks are not verified until the backend is told to try to take the network online. When a Wifi network is brought online the following steps are taken: The Wifi interface is configured and a *DHCP client* is launched to try to receive network information such as a valid IP address. If an address is received the backend tries to reach the project server. If it succeeded the net is considered valid and added to the list of valid networks of the backend. Should the procedure fail the network is considered invalid. If consecutive scans shows that the net has disappeared it is removed as a valid net. A program which connects to the Wifi networks to determine is they can be used was also written, see section 4.2.2

4.1.3 Cellular

To be able to handle cellular networks the backend first needs to control an *UMTS modem*. One modem is required per network and the backend supports multiple cellular networks when more then one modem is plugged in.

Two different models from *Huawei* were used: *E220*, *E169*. The behavior of these devices varied in some details, mostly in the initializing phase of the devices. The modems exports two serial communication devices, one for establishing the actual data link and the other one for querying the device for information. Each modem has an *identification number* which is used to map each modem to a specific cellular provider. Communication is done using the *Hayes command set* protocol detailed in [22]. As the backend must cope with cellular network changes, such as roaming or degraded signal the state of the modem is monitored for changes. If the modem looses the connection to the base station the backend invalidates the network until the connection is online again. The current bandwidth reported by the modem is also saved.

When a modem is plugged into the client a signal is emitted by HAL and received by the backend. The backend proceeds by:

1. Resetting the communication device
2. Queries the device for an *identification number*
3. Authenticates using a PIN code if necessary
4. Asks the modem to send information asynchronously

When the modem reports a viable state the backend tries to take the network online by issuing a dial command as well as starting a PPP daemon. If the daemon receives an IP address the network is considered useable and is added to the list of valid networks just as a Wifi network would. The net is only removed if the modem is unplugged.

A simple interface to send raw *AT commands* via *D-Bus* to the modems was also implemented. This was used during testing to simulate faults in the cellular network by telling the modem directly to disconnect from the current network cell and begin scanning for a new.

4.1.4 Networks

As nets are added, removed or their state changes, signals are emitted via *D-Bus*. It is also possible to issue a state change via *D-Bus*: The backend could for example be told to try to take a net online which is currently in the offline state. The different states are:

State	**Wifi**	**Cellular**
Invalid	DHCP failed / Server unreachable	Cellular network lost
Offline	-	
Connecting	Acquiring DHCP lease	PPP negotiation
Routing	Routing initializing	
Verifying	Server reacheable?	Not used
Online	Network available for use	
Establishing	Contacting server	Contacting server
Measuring	Measuring in progress	
Streaming	Streaming in progress	

Table 3: Different network states

Depending on the available networks and which states they are in the backend keeps one network as the preferred communication net to the server. This network is the net in which all non streaming communication will take place, although streaming is possible too. If the preferred network goes offline or becomes unavailable in some way the backend tries to find a new suitable network for communication with the server. The viable types of networks for the communication net can be configured to allow for only Wifi networks, only cellular networks or both. As the communication protocol does not require any significant bandwidth a reliable network is preferred over a network which might be short lived, such as a Wifi network.

4.1.5 Simulation

To help development of the frontend and the server a *simulation* feature was implemented to the backend. If the backend is started in simulation mode a configured *gpx* file with a route is read. The physical GPS is disabled and the GPS signal is emitted as if the client would be traveling the route. The client would report the positions to the server as they were real allowing tests of the server as if the client really was travelling a route. The measurements reported by the client would still be from the actual location of the client. Simulation of measurements would have been preferred but was not implemented due to time constraints.

4.2 Client Frontend

The main purpose of the frontend is to receive the A/V stream and play it. It also reports its state to the server like where it is or what networks are available. The frontend communicates with the *backend*, see section 4.1, to receive information.

The initial goal of the frontend of the client was a graphical interface with three parts: A media player, a coverage map viewer and an interface to the backend. The media player would simply play the stream and present a simple interface to control playback such as pausing and playing. The second part of the frontend would download map tiles from the server and show them as map, displaying expected bitrates at various parts of the route. The third part would display information supplied by the backend, information such as available networks, states and GPS position. The different parts would be implemented using *Clutter*.

Due to time constraints which became more evident as the work progressed there would not be enough time to implement the frontend as originally intended. The map viewer had to be discarded, although the maps are still available to the client by using a web browser, see section 5.3. A user interface to the backend was implemented during the development of the backend for testing purposes. The decision was made to keep this implementation as is and not integrate it into the main frontend as previously planned. The media player was implemented using *GStreamer* and the pipeline detailed in table 5. The player is very basic, it is only possible to toggle pausing. Incoming data is buffered before playback.

The frontend needs two different connections to the server: One communication channel and a data channel where the raw media stream is received. Both channels were built on top of the *Twisted framework* for an asynchronous message system. The client first establishes the communication channel to the server by asking the backend which is the preferred communication network. When connected the client identifies itself, sends the route it will travel, and asks for a list of available streams. The server needs to have some information about the client to be able to make decisions regarding the stream, therefore the following data is sent frequently:

Table 4: Data sent to server

Type	Frequency	Comments
Network	Event driven	Networks added, removed or changes of state
Player state	Timed	Position in stream and bytes buffered
GPS	Timed	Longitude, latitude and velocity

As soon as the server retrieves information about the network states on the client side it can make an informed decision of what network the client should use while streaming. The client receives the name of the network to use along with a port number to connect to. The data connection is established and the stream is played back.

4.2.1 Playback

GStreamer was used for decoding the stream and presenting it to the user. The pipeline used is depicted in figure 5. The intermediate buffer when receiving data is used to cope with small fluctuations of bandwidth and as a buffer for parts of the stream which is prebuffered to cover areas with less or no coverage. The size of the queue affects how much time it is possible to buffer and at what quality.

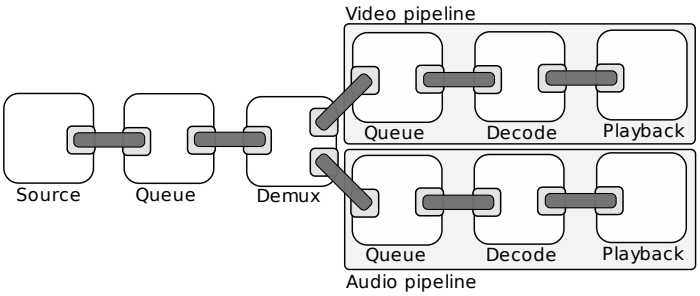

Figure 5: GStreamer pipeline client-side

4.2.2 Tools

Many small applications were written during development which interacts with the *backend*. Some were just used during development while some are considered a part of the final system.

ip

Given the name of a network, returns the current IP address mapped to the network.

measure
: Connects every online network not currently used for streaming to the measuring server and starts to measure the bandwidth of the network. If a network is brought offline or used for streaming, the measuring process is aborted until the network becomes available again.

wifi
: The purpose of this tool is to verify that the available Wifi networks are actually useable for streaming. The tool tries to bring every available Wifi network online and if a network fails it is regarded as invalid and not useable.

gps
: Simple visualization of the state of the GPS. Displays current position and velocity.

logger
: Used during testing to log measurements for networks before the *server* was implemented. Saves measurements to a local database which can be imported on the server later.

4.3 Packet Routing

Most of the traffic for the client will be incoming data, such as the A/V stream. The available networks still need to be able to send outgoing data to communicate with the server. As the client handles multiple networks simultaneously outgoing traffic must be able to flow independently for each network. The system must be able to route traffic for a number of 3G modems as well as a Wifi card. In traditional systems routing is only based on the destination of the packet which is not sufficient when multiple interfaces exists which all has the ability to route to a specific destination, namely the project server. To overcome this limitation *source based routing* is required. With *source based routing*, routing decisions are based on the source of the packet which can be set by binding to a specific interface. So to send data over a specific network you must first bind to the corresponding device before connecting to the remote location. The procedure to set up routing for a networks works like follows:

1. Create a *routing table*
2. Define a rule declaring that all traffic originating from the interface must use the new table.
3. Specify a *gateway* for the table.

The routing procedure is performed every time a network is brought online, after link specific protocols such as DHCP or PPP. Routing tables and rules for each network is removed when it enters the *offline* state.

5 Implementation and Analysis of the Server

The main application on the server handles communication with the client. It keeps track of clients and information received from the clients, such as position, available networks and what it currently displays of the stream. All decisions regarding the stream for each client is taken within this program.

5.1 Open Street Map

To be able to visualize the bitrate coverage for the wireless nets a map was needed and the choice fell on Open Street Map (OSM) which is an open source project where any one is allowed to contribute or use the map in any way. To gain access to the map data, a database file with all map information were downloaded from [17]. Since the tests in this master thesis only were conducted in Sweden, only map information for Sweden was downloaded. Once the database file was downloaded the file was loaded into a PostgreSQL database for easy and fast access.

To make maps from the database information, an open source Python program called *Mapnik*[23] was used. Mapnik generates tiles which are png files and the tiles are saved in a hierarchy according to a zoom level, latitude and longitude. When the tiles are generated they are set side by side to form a map. Tiles are used to form coverage maps 5.3.

The estimation program described in section 5.6 needs to be able to estimate velocity at a given positions. With the help of OSM it is possible to find out what type of road a position is on and from the road type a rough estimations of the velocity is made. See Table 5 for what velocity a certain road type in Sweden has.

Table 5: Table of what velocity a certain road type has

Road type	Velocity (km/h)
Motorway	110
Trunk	90
Primary	80
Secondary	60
Tertiary	70
Residential	50
Unclassified	50

5.2 Database

All measurement data which the client collects is stored in a SQL database. An API was built for the measurement database to enable fast and easy accesses to the measurements. Estimating functions which is described in section 5.6 uses the API to get measurements for a specific position. To find which measurements lies within a specified range some calculation is needed. Therefore was PostGIS[14] used in the measurement database. PostGIS allows adding a geometry object to a table and PostGIS also offers functions for distance calculations on the geometry objects.

Geometry objects can be a point or a polygon and in the measurement database we used the point object which stores an x and y value. In the measurement database each entry has two geometry objects, one has a WGS84 projection and the other has a Google projection. A projection is a method on how to represent the surface of a sphere on a plane. In this case the sphere is the Earth and the plane is the map. Two geometry objects were used because the coverage map program needs the coordinates to be in Google projection due to the coverage maps are defined in pixels. Meanwhile the estimate functions need the coordinates in latitude and longitude. This makes it easier due to the client is reporting its positions in longitude and latitude.

The database API offers an insert function for inserting measurements and a get function which given a longitude and a latitude finds the measurements which lies within a specified radius. The measurement database consists of a table which can be seen in Table 6.

Table 6: Table in the measurement database

Column	Type	Description
osm_id	integer	Not used.
client_id	integer	Client id.
name	character varying	Network name.
net_type	character varying	Network type (wifi,utms)
tim	bigint	Timestamp in unixtime
bitrate	integer	The measured bitrate
bitrate_is_max	boolean	Client in measuring/streaming mode.
quality	integer	Network quality
velocity	double precision	Client velocity
extra	character varying	Cell id
geom	geometry	WGS84 mercator geometry object.
the_geometry	geometry	Google mercator geometry object.

5.3 Coverage Map

To easily see what estimated bitrates are available at different positions some sort of coverage maps is needed. The idea is that the bitrate estimation function takes data from the measurement database and hands it to a program which will generate coverage maps. A coverage map program will run as a daemon in the background and when new measurements are registered at the server it will check which tiles needs to be updated. After that the program takes one tile at a time and renders this tile. The rendering process collects the tile in question from the Open Street Map and divides this tile in small squares. For each square it asks the estimation function for which bitrate is possible to achieve in this square and translates it to a colour according to table 7. Color is then applied to the square with some opacity this makes its possible to see the original Open Street Map tile. At last the program saves the tile in a hierarchy accordingly to zoom level and position. The maps are accessed through a web server. To see the bitrate coverage maps you simply connect to the server trough a web browser. From here you can choose which service providers 3G net you want to see. Figure 6 is a screenshot depicting how the coverage map looks like.

A possible extension would to integrate the coverage maps into the client fronted. Then it would be possible to follow the client route on the coverage maps. But there were not enough time for implementing this. Generating coverage maps could be optimized by using several threads to generate the coverage tiles by pulling tiles from the queue simultaneously instead of just one thread taking one tile at the time.

Table 7: Bitrate to colour table

Bitrate (Mbit/s)	Colour
Measurements missing	No colour
0.0 - 1.0	Red
1.0 - 2.0	Pink
2.0 - 3.0	Orange
3.0 - 4.0	Yellow
4.0 - 5.0	Turquois
5.0 - 6.0	Blue
>6.0	Green

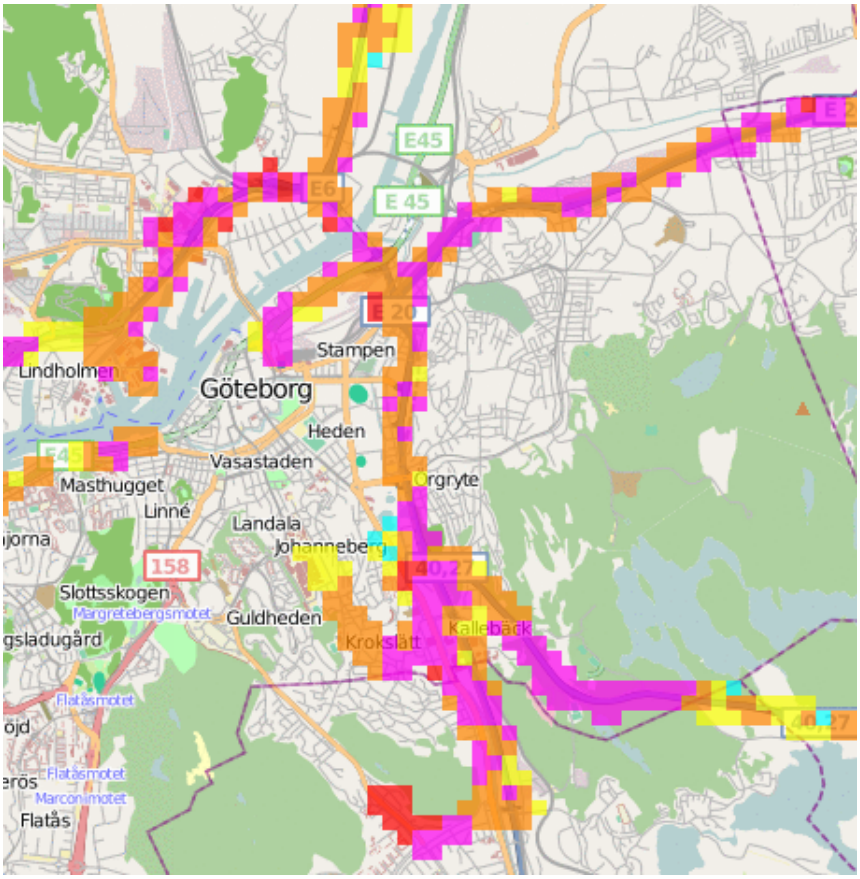

Figure 6: Screenshot of a coverage map

5.4 Streaming

5.4.1 Video Codec

H.264, also known as MPEG-4 Part 10 and MPEG-4 AVC, was the preferred codec by *Mecel* due to the growing number of decoding hardware acceleration implementations. With hardware accelerated decoding a less powerful CPU is needed to decode the video as the decoding is done completely in hardware, usually by a graphics card. The intent of H.264 was to provide good quality with lower bitrates compared to previous standards, such as MPEG-2, H.263, or MPEG-4 Part 2, while still maintaining good quality at higher bitrates.

To test the performance of encoding on the target server hardware, a throughput test was performed with different settings regarding bitrate. Three different systems were tested labeled as *System A, B, C & D*, see table 8. Both systems A and C are dual core systems, system D uses a quad core processor while system B has a single core. The test for system C and D was performed within a *wmvare machine* which negatively affects the results. Figure 7 shows the results where the bars represents throughput for each system at different encoding bitrates and the lines how many times faster than real-time it is possible for this system to operate.

Table 8: Tested processors

System	Processor
A	Intel(R) Pentium(R) Dual CPU E2180 @ 2.00GHz
B	Intel(R) Pentium(R) 4 CPU 3.20GHz
C	Intel(R) Xeon(TM) CPU 2.80GHz (Virtualization, 2 cores)
D	Intel(R) Xeon(TM) CPU 2.40GHz (Virtualization, 4 cores)

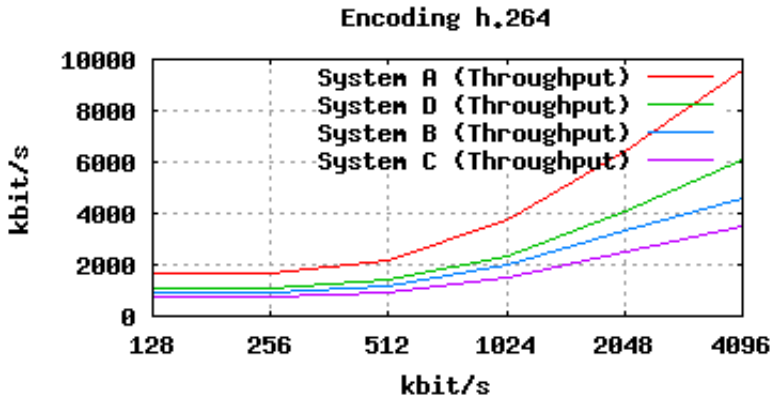

Figure 7: Data throughput VS. encoding bitrate

5.4.2 Transcoding

As the stream must be modified to fit the environment of each unique client the source to be streamed must be transcoded to allow real-time modification of the A/V bitrate. Transcoding is a very CPU intensive task and multiple steps must be performed which must be performed faster than or equal to real-time to guarantee the client always has data to play. Transcoding includes:

- Demultiplex the source into a video stream and an audio stream
- Decode the streams
- Encode the streams according to specification
- Multiplex the streams

Two fundamentally different approaches to performing the transcoding were considered: *In advance* and *real-time*. When transcoding in advance multiple versions of the source with different bitrates is created and cached locally on the server. An obvious disadvantage of this technique is that it requires the source to be of a non-live nature, i.e. a file on disk. If a some what static collection of sources is to be offered by the system this approach can be advantageous as the CPU intensive transcoding is only performed once, albeit in several versions. Disk space could become a factor if a large number of versions need to be kept. When the actual streaming is performed the different versions needs to be interleaved to a stream. The system can only choose from the relatively limited set of versions created and try to adapt to the network environment as good as possible.

By transcoding the source in real-time the bitrate can be adjusted to match the limitations of the network more precisely. This comes with the cost of higher CPU usage during streaming. Live sources, such as the signal from a Digital Video Broadcasting (DVB) source, could also use this as no initial caching or preparation of the media is performed. Some optimizations of non-live sources could be advantageous to minimize CPU usage during streaming. An example would be to re-encode sources encoded with a codec which requires large amounts of resources to decode to make the resources needed to decode less.coded to a less CPU intensive format.

From an implementation point of view, transcoding in advance is the more complex alternative due to the fact that the different version needs to be interleaved when streaming. This needs to be done on a frame level when considering video as not all frames are stored in full. When researching candidate software using a technique similar to the proposed method nothing viable was found. Research showed that transcoding in real-time would be a less complex task to implement. By modifying the parameters for the codec in real-time, the codec would hopefully pick up the new parameters and adjust the bitrate. Some testing of the *x264* encoder showed that it was possible to feed it new parameters by reinitialize it during runtime. For a description of *x264*, see section 3.8.6. Real-time transcoding was chosen as the best solution as it is less complex to implement and switching bitrate can be done with a higher resolution. It's possible to increase or decrease the bitrate with just a few Kbit.

Another requirement of the transcoding process is the ability to encode faster than real-time. By keeping the bitrate as it is and as much data buffered as needed before an area with bad or non existent network coverage the user will not see a deterioration of the quality of the media streamed nor will any pausing for buffering be necessary. The server must be able to encode data fast enough when buffering is possible to fully utilize the available bandwidth. As seen in figure 7 the project hardware was not enough to ensure network throughput higher than the theoretical maximum throughput of a High-Speed Downlink Packet Access (HSDPA or Turbo 3G) connection.

Both *libavcodec* and *GStreamer* were considered for the task of transcoding, both being open-source and widely used on the linux platform. See section 3.8 for a brief description of the two libraries. Both libraries were tested with the x264 encoder and could produce a media file with sections containing different bitrates. The plugin for x264 in GStreamer had to be modified, to allow the bitrate property to be changed after initialization. GStreamer further abstracts the transcoding compared to libavcodec, Graphs dictating the flow of the audio and video streams can be constructed to easily modify streams. For a graph depicting the transcoding of the source as

well as the sending via TCP, see figure 8. As GStreamer would make the implementation simpler, it was chosen over libavcodec.

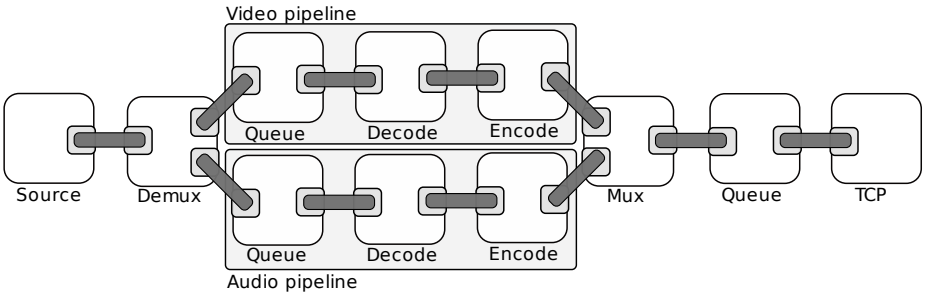

Figure 8: GStreamer pipeline server-side

GStreamer pipelines can be in four different states[24]: *null*, *ready*, *paused* and *playing*. When starting the server pipeline for a stream it is initially set to *paused*. When a client connects the state is temporarily switched to *playing* until the client disconnects again. To control how fast transcoding the process flows, and ultimately control to what degree the client's connection is saturated, it is possible to change the *synchronization property* of the *TCP element* of the pipeline. This property decides whether the element should synchronize to the pipeline's clock or not. Testing of this property showed that it worked per expectation if it was set during the initialization of the pipeline. If the element was set to not synchronize to the clock at start up and later told to synchronize, the element would stop sending data until it reached the position if would have had if it would have synchronized the whole time. This is a problem as it voids the whole buffering process. It was finally solved by reinitializing the pipeline during runtime:

1. Set pipeline state to *ready*
2. Set the *synchronization* property
3. Set pipeline state to *playing*

This is not an optimal solution but was chosen due to the fact that the problem was discovered late in the development phase and time was a constraint. A problem with this solution is that the clock of the pipeline is reset. A correct clock is needed to determine the length of the buffered data on the client. The stream position of the server and the client is compared to estimate how many seconds of the stream currently is buffered on the client. The reset of the clock was solved by saving the current value of the clock before the reset and then add this value to the new clock. A better alternative solution would have been to control the data rate by a separate component in the pipeline or to modify the pipeline clock appropriately.

5.5 Measuring bandwidth

The only way of retrieving the available bandwidth is to actually send or receive data as fast as possible. This is a problem as transmitting data could have an associated economical cost or disturb other vital transmissions. When streaming and not buffering the system does not know if the bandwidth is the maximum available or not. Due to the CPU requirements of transcoding, encoding H.264 in particular, even when buffering as fast as possible it is not certain that the available bandwidth is fully used as the CPU on the server may be the bottleneck and not the connection to the client. As can be seen in figure 7, the performance of the test system's hardware is not enough to be neglected. The most accurate measurements are obtained when no streaming is performed and arbitrary data can be sent to the client as fast as possible. These measurements can be reported as fully utilizing the link as the network is guaranteed to be the limited resource. When streaming the utilization is reported as *unknown* by the the client so that the server can determine the accuracy of the measured value by evaluating whether the streaming is done in real-time or buffering. Another possibility would be to look at CPU utilization to determine if the CPU is limiting buffering speeds or not.

To be able to measure bandwidth on links which are not being streamed to a simple server streaming data was needed. The contents of the data are not important as long as the data is being generated fast enough to saturate the link on the client-side. A simple server was written in *C* which accepts connections on a configured port. As soon as a client is connected the server sends random data as fast as possible until the client disconnects. The server reads the data to send from `/dev/random` which is a pseudo random number generator [25] as the client connection may use a compression protocol, such as *PPP Deflate*[26], which would provide incorrect measurements. As data would have been compressed, bytes transferred would have been reported higher than bytes actually sent.

5.6 Estimating available bandwidth and velocity

As a client moves a long a route the system must estimate how much bandwidth is available in different parts of the route before the client reaches the waypoints. It is also a benefit to estimate what velocity the client will have at each part to be able to deduct how long the client will stay in every part of the route.
A number of different factors needs to be taken into account when estimating the available bandwidth and expected velocity of the vehicle at a given point. These factors need to be weighted according to relevance to create a realistic value as possible. By retrieving all previous measurements within a configured radius of the estimation point, the values can be merged into an estimate by weighing according to the following factors:

Distance
: Measurements nearby the estimation point will give a more accurate estimate. The distance between where the particular measurement was made and the estimation point is calculated and weighted to give close points more weight.

Intermittence
: Due to the fact that the available bandwidth can be affected by the number of users of the cellular network, the possibility exists that the bandwidth might decrease during certain parts of the day. For instance in a traffic jam during rush hour when there is a large number of cellular clients over a small area. Such a scenario would also lower the velocity of the vehicle with the result that the client would spend more time in that specific area.

Age
: Fresh measurements are more relevant than old measurements as base stations may change, buildings built or demolished which could affect coverage. Wifi access points may have been closed.

Velocity
: If tests show that there is a correlation between the velocity of the vehicle and measured bandwidth, this need to be taken into account.

How these factors are weighed is defined in a configuration file which allows different configurations depending on the type of value to be estimated. For every measurement retrieved from the database, a weight per factor is calculated. This weight reflects how relevant the measurement is according to the factor. These weights are in turn weighted according to configuration file to calculate a final weight for the measurement. An estimate can then be calculated using all measurements and their corresponding weights.

By estimating the bandwidth and velocity for every point along a route it is possible to estimate how long, and at what intervals, the client will have certain bandwidth.

5.7 Strategies for adapting bitrate and buffering

There are many different motives for adapting bitrate and buffering depending on available bandwidth and conditions. Different users may prioritize different factors. One user may want to minimize data traffic as a mean to lower costs and another user might want to maximize quality of the A/V stream as much as physically possible. To accommodate these cases a number of strategies will be presented. The strategies make decisions based on the estimations detailed in section 5.6:

Cost

Bandwidth costs money and some ISP's do not offer a flat rate pricing model. The user usually pays per megabyte transferred. Some flat rate subscriptions has a cap on how much data may be transferred over a given amount of time, after the cap is reached the connection is throttled to a fraction of the speed previously available. See appendix A for a comparison of Swedish cellular providers. The purpose of this strategy is to minimize total data sent. The idea is to choose as low bitrate as the A/V source permits, identify areas which have a lower estimated bitrate than the chosen bitrate and try to buffer enough for these areas before they are reached. If the user closes the stream the buffer is disregarded, as such, it might be a benefactor to buffer data as late as possible to reduce the risk of streaming unused data.

Average

With the *average* strategy an A/V bitrate is chosen early and is permanent as long as it's still possible to buffer areas with less or no coverage.

Quality

If the cost of the bandwidth is a non-issue it's feasible to try to maximize the quality of the video streamed. The goal is to maximize the bitrate on the A/V stream while still buffering to cope with areas where the coverage is worse. The A/V bitrate is updated regularly to reflect the estimated bitrate. Areas with low or no coverage at all is still buffered by lowering the bitrate in an earlier part of the route.

5.7.1 Identifying intervals to buffer

Given a desired bitrate b and a function $e(t)$ defining the estimated bitrate at time t, all continuous intervals where $e(t) < b$ is saved. For each interval $[t_i, t_j]$ the number of bytes B needed to buffer before t_i is:

$$B = (t_j - t_i) * b - \int_{t_i}^{t_j} e(t)\, dt.$$

By letting $B = 0$ and t_i be unknown a new interval starting at t_x can be calculated:

$$t_x = t_j - \frac{\int_{t_i}^{t_j} e(t)dt}{b}$$

$t_{i,b} = (t_j - t_x)$ is the number of seconds which **must be** buffered before the deadline t_i. Let D contain all identified deadlines and number of seconds which must be buffered:

$$D = \{(t_i, t_{i,b}), ...\}$$

5.7.2 Finding intervals where buffering is possible

Given an interval $[t_i, t_j]$, the number of seconds which can be buffered during this period depends on chosen bitrate, available bitrate and maybe even the cpu. The buffered video over the interval will have the same A/V bitrate as the video considered *live*. As such, different strategies may want to choose intervals differently. A strategy which tries to optimize quality may want to choose the period with the highest A/V bitrate while a strategy which tries to lower costs may want to buffer as late as possible to avoid wasting bandwidth if the user decides to stop the stream prematurely.

All intervals $[t_i, t_j]$ where $e(t) > b+\epsilon$ are intervals where buffering is possible. ϵ is a safety margin to prevent buffering when the A/V bitrate is very close to the estimated bandwidth. For every interval the number of seconds possible to buffer within the interval is calculated:

$$s_j = \frac{\int_{t_i}^{t_j} e(t)\, dt - \int_{t_i}^{t_j} b(t)\, dt}{\int_{t_i}^{t_j} b(t)\, dt} * (t_j - t_i)$$

Let I contain all calculated available seconds along with t_j which marks when the seconds are available:

$$I = \{(t_j, s_j), ...\}$$

For the streaming strategy to be successful the required number of seconds buffered must be available at each deadline. If the strategy fails it may retry with more conservative values for the bitrates during some intervals until it

succeeds.

Buffering during an interval with a certain bitrate results in the same bitrate for the buffered stream. As such, choosing suitable intervals is dependent on the selected strategy. The required CPU resources is also higher during intervals with a high bitrate which means it is possible to buffer more seconds of the stream per second the lower the bitrate.

5.7.3 Implementation

When the client first connects it will send a predetermined route to the server. Based on this route the server calculates which networks will be available to the client. For each network available, estimations of the available bandwidth are calculated using the technique described in Section 5.6 on page 37. The best network is selected for each waypoint along the route. What A/V bitrate and buffering mode used for each waypoint is decided by the different strategies. Three different strategies were implemented:

Cost
: The A/V bitrate is chosen to be as low as allowed. Buffering may still be necessary as some intervals may have no coverage at all or very low. The buffering should occur as late as possible to avoid sending data which would be unused.

High Quality
: The A/V bitrate is set to the maximum allowed for the entire route. All intervals which need buffering are identified. If the identified intervals which need buffering can't be buffered in time a lower bitrate is selected for the route and the calculations are repeated. This is done until a viable buffering strategy is found.

Smooth Quality
: The A/V bitrate is set to follow the estimated bandwidth as close as possible. The same strategy as *High Quality* is used for buffering.

The strategies are executed every time the client reports a new position. This allows the server to cope with unexpected bandwidth to some extent. The client may not have buffered the expected amount of the stream. New decisions are made to ensure the expected behavior. A number of scenarios were not taken in to account which may improve the strategies. Buffering is done as early as possible which is not optimal for some strategies as buffering at a specific interval means the buffered data will have the same A/V bitrate as the interval. A strategy interested in minimizing data sent might want to select an interval with a lower A/V bitrate.

6 Results

The result of this thesis is a *proof of concept* system including a set of client applications as well as a server environment. The server accepts measurements from clients and stores the values in a database. Estimations based on these values are calculated and used to construct coverage maps and bandwidth graphs for use in strategies to stream efficiently. The configuration used to estimate values was not researched for efficiency in calculating optimal values, it was considered well enough for testing of the system. Simple versions of the strategies mentioned in the analysis are also implemented. The transcoding implementation works as expected with the possibility to control the A/V bitrate of the video in real time. The control of the data rate when streaming, that is, the ability to either send the stream in real time speed or to burst the stream as fast as possible, is somewhat lacking due to the choice of using the synchronization method of the components in *GStreamer*.

The system has a number of areas where it can be improved to provide a more reliable and better solution: The rate control of the streaming needs a better implementation compared to the one currently implemented. The *strategies* implemented could be improved together with some research of how the most accurate estimations of the available bandwidth are calculated.

Testing of the bandwidth provided by cellular providers in different areas and at different times has showed that it does exist areas with greatly decreased bandwidth. As seen in figure 3 on page 12 the coverage deviations is constant enough to make qualified estimations of the available bandwidth.

To summarize the resulting system it provides a framework for future development within the area. The client provides access to both a Wifi card and to a number of cellular networks in a uniform way. Coverage testing of networks can easily be performed. The server application provides an interface to test buffering and dynamic bitrate adaptation strategies for various use cases. Streaming and playback is implemented but with a few limitations as mentioned previously.

7 Conclusion & Discussion

This has been a very interesting and rewarding project to work with. It covered a lot of different areas such as databases, media streaming and hardware handling. Because of the many different areas we had to solve many different and interesting types of problems. We have therefore gained a whole lot of new knowledge about how to vary the bitrate of a media stream in real time, how generate maps using OSM and handle hardware such as GPS devices and cellular modems. The work has been free in the sense of planning the work and designing the system. This master thesis was divided into several phases which was, research, designing the system, implementing the different parts of the system, testing the system and finally writing the report.

Smart buffering is an interesting concept as it allows for smoother playback of the stream, while it helps to minimize unnecessary traffic to the client. The system knows when to buffer to use the network link optimally for user. This might mean to delay buffering until roaming to a cheaper network is possible, or to just delay it as long as possible if the user decides to cancel playback. To further help the client to always have data from the stream to play, the concept of adapting the bitrate in real time is a viable solution. The resource requirement on the server side is a drawback, but in turn it's possible select a bitrate tailored to the client which is updated frequently. If the client would spend more time in an area with lower coverage than intended lowering the bitrate temporary is possible to still provide a smooth playback.

As mentioned in the results the final implementation has some shortcomings which were not resolved due to time constraints. Some aspects of the development took longer than expected which lead to some compromises in the implementation. These compromises should be resolved for future use of the application. This would result in more accuracy in the stream control and how much of the stream really is buffered in the client application. The client also has some shortcomings mentioned in 4.2. The client use several programs to provide all the desired functionality. This should be merged into one process as was the original purpose of the client frontend.

As for estimating values for maximum available bandwidth at a location a limiting problem is the fact that it is necessary to maximize throughput on the link to measure it. When receiving a stream in real time and not performing any buffering it is only possible to say that the available bandwidth is at least what the measurement reported. Receiving data for the only purpose of measuring the link is associated with a cost for the user. To provide reliable estimations of available bandwidth a large dataset of measurement is required over a large geographic area. This is a time consuming and expensive job to do for company or organization. If the collecting of

measurements were community driven then anyone could contribute with bitrate measurement from any where in the world. With enough members contributing with bitrate measurements this would be a cost effective solution for gathering measurements.

The major problem with the resulting system concept is not to predict where the coverage will be suffering or to adapt the A/V bitrate but how to get enough measurement data to predict with efficiently. The data need to cover a large area for the service to be appealing for a market and the data must constantly be updated to reflect changes in different providers networks.

Future Work

The estimation part of the server implementation could be generalized into a separate service. This could be useful as this feature is interesting for other applications which require mobile Internet access. This service would export an API for retrieving estimations of values based on a geographical location and a point in time. This separate system could then be used by this application and by other systems in the need of information regarding available bandwidth.

The resulting system would benefit from various solutions regarding routing. The route sent to the server could use some interpolating of the waypoints for higher accuracy when calculating where the client is on the route. The system also does not handle client route deviations. If the client would deviate from the prepared route, the system could calculate a temporary route by using data from the GPS as well as information from OSM.

Some form of predictive routing would alleviate the need of the client to provide a route. Based on usage patterns of the user a likely route could be calculated and used by the server. The system could use a magnitude of information to make intelligent guesses of the route. A current project at *BMW*[27] researches such a system. Using information such as date, time, passengers and driver a probable route is calculated.

As described in Appendix A the mobile Internet Service Providers has a max limit of how much data one can download under a month and after the max limit is reached they will reduce your bandwidth next to nothing. It would be desirable to take this into count when streaming so one tries to hold down the amount of sent data. Amount of sent data could be reduced by decreasing the quality of the movie or music. This allows one to watch or listen to more media per month. Another possible extension to the streaming strategies would be to choose the cellular network with the lowest price per data transferred. This is interesting when traveling between different

countries where it is some times possible to choose what cellular operator to use.

More data in the measurement database will lead to better accuracy when estimating the bitrate and to more accurate coverage maps. To get more measurements one have to go by car and run the test system. This is a time consuming job but it could be interesting to know what bitrates you really get. This could evolve into a service where you could sell coverage maps either to the Internet Service providers or to others who are interested.

References

[1] "Spotify, streaming music." http://www.spotify.com/en/.

[2] M. Frick and E. Steen, "Streaming Media Using Peer-to-Peer Technologies," 2008.

[3] "Iperf Project." http://sourceforge.net/projects/iperf.

[4] "TPTest." http://sourceforge.net/projects/tptest.

[5] "Arch Linux, a Linux Distribution." http://www.archlinux.org/.

[6] "IBM Rational Clearcase UCM." http://en.wikipedia.org/wiki/IBM_Rational_ClearCase_UCM.

[7] "GNOME Mobile." http://www.gnome.org/mobile/.

[8] "GStreamer Project." http://www.gstreamer.org.

[9] "Ffmpeg project." http://www.ffmpeg.org.

[10] "Clutter Toolkit." http://www.clutter-project.org.

[11] "Twisted." http://twistedmatrix.com.

[12] "Cairo Graphics Library." http://cairographics.org.

[13] "PostgreSQL." http://www.postgresql.org.

[14] "PostGIS." http://postgis.refractions.net.

[15] "gpsd." http://gpsd.berlios.de/.

[16] "Apache HTTP Server Project." http://httpd.apache.org.

[17] "Open Street Map." http://www.openstreetmap.org.

[18] F. Hartung, U. Horn, J. Huschke, M. Kampmann, T. Lohmar, and M. Lundevall, "Delivery of broadcast services in 3g networks," MARCH 2007.

[19] "Joost, P2P Streaming." http://www.joost.com/about.

[20] Statens Offentliga Utredningar, "Tillgänglighet, mobil tv samt vissa andra radio- och tv-rättsliga frågor," SOU 2006:51.

[21] "Moblin Services." http://www.FIXME.

[22] 3rd Generation Partnership Project, "AT command set for User Equipment (UE)," 2008.

[23] "Mapnik." http://www.mapnik.org.

[24] W. Taymans, S. Baker, A. Wingo, R. Bultje, and S. Kost, "GStreamer Application Development Manual." http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html.

[25] Z. Gutterman, B. Pinkas, and T. Reinman, "Analysis of the Linux Random Number Generator," March 2006. http://www.pinkas.net/PAPERS/gpr06.pdf.

[26] J. Woods, "RFC1979 - PPP Deflate Protocol," August 1996. http://www.ietf.org/rfc/rfc1979.txt.

[27] "Navi mit künstlicher intelligenz." http://www.all-electronics.de/news/31673-Navi+mit+kuenstlicher+Intelligenz.

[28] "Tre price list." http://www.tre.se.

[29] "Tele2 price list." http://www.tele2.se/mobilt-bredband.html.

[30] "Telia price list." http://www.telia.se/privat/produkter_tjanster/mobilt/mobiltbredband/.

[31] "Bredbandsbolaget price list." http://www.bredbandsbolaget.se/wps/portal/privat/bredband.

[32] "D-Bus Data Types." http://dbus.freedesktop.org/doc/dbus-python/doc/tutorial.html#data-types.

Glossary

A/V Audio/Video. 4, 5, 10, 13, 15, 19, 23, 26, 33, 39, 41, 42, 44

API Application Programming Interface. 19, 44

CPU Central Processing Unit. 32–34, 36, 41

DHCP Dynamic Host Configuration Protocol. 26

DVB Digital Video Broadcasting. 34

DVB-H Digital Video Broadcasting Handheld. 4, 6, 17, 18

GPS Global Positioning System. 43

H.264 Video codec, also known as MPEG-4 Part 10 and MPEG-4 AVC. 15, 32, 36

HAL Hardware Abstraction Layer. 21

HSDPA High-Speed Downlink Packet Access, also known as Turbo 3G. 34

IVI In-Vehicle Infotainment. 13, 14

MBMS Multimedia Broadcast Multicast Service. 4, 17, 18

MID Mobile Internet Device. 14

OSM Open Street Map. 15, 16, 27, 43, 44

P2P Peer to Peer. 6, 8, 17

PPP Point-to-Point Protocol, a data link protocol. 21, 26

QoS Quality of Service. 3, 5

TCP Transmission Control Protocol. 13

UDP User Datagram Protocol. 13

UMTS Universal Mobile Telecommunications System. 10, 17

Index

Backend, 19

Cellular
    Usage, 20
Clutter, 15
Coverage Map, 30

D-Bus
    Description of, 15
    Usage, 19
Database, 28

Estimating, 37

Frontend, 23

GENIVI, 14
GPS, 20
    gpsd, 16
    Usage, 20
GStreamer, 34
    Client, 24
    Description of, 14
    Pipeline
        Client, 24
        Server, 35
    States, 35

H.264, 32

libavcodec, 14

Market survey, 17
Measuring, 36
Moblin, 14
Modems, 21

Open Source, 13
Open Street Map, 27

PostgreSQL
    Description of, 15
    PostGIS, 16

Routing, 26

Simulation, 22
Strategies, 39
Streaming, 32

Transcoding, 33
Twisted, 15

Wifi
    Usage, 20

x264, 15

Appendices

A Cellular provider comparison

This market investigation was done 2009-05-18 and changes could have been made since then. Swedish Internet Service Providers market their mobile broadband with bitrates up to typically 7.2 Mbit/sec. Consumers has filed complaints against this because they feel they have been fooled due to that the real download bitrate just reaches around 1-3 Mbit/sec. During our tests we had an average bitrate of 1-3 Mbit/sec down to the client. The upload bitrate of the networks are fairly poor with around 0,38 Mbit/sec. Se table 9 for a comparison between Swedish mobile providers. The information has been taken from [28], [29],[30] and [31].

Table 9: Cellular provider comparison

Provider	Bitrate down/up, (Mbit/sec)	Max data/-month, (Gbyte)	Reduced bitrate (Mbit/sec)	Price (SEK/-month)
Tre	7,2/0,384	20	0,2	199:-
	21/5,76	20	0,2	249:-
Telia	10/1 *	3	0,057	229:-
	10/1 *	5	0,057	249:-
	6/0,7 *	2	0,057	99:-
Tele2	14,4/-	10	0,064	249:-
	7,2/-	5	0,064	219:-
	7,2/-	1	0,064	129:-
Telenor	7,2/0,384	Unlimited	-	199:-
Bredbandsbolaget	7,2/0,38	1	0,03	199:-

* Telia has 2500 zones in some bigger Swedish cities where it is possible to reach bitrates of 22 Mbit/sec.

B D-Bus API

D-Bus API reference. For a description of the various D-Bus data types, see [32].

org.mecel.gps

An interface to the GPS device. Enables retrieval of GPS data as well as signalling when the data changes.

Name	**Arg.**	**Ret.**	**Description**
get	None	nddd	Returns mode, latitude, longitude and velocity
update	nddd	None	Signals a change in mode, latitude, longitude or velocity

org.mecel.netmanager

This interface handles information regarding networks. Used to retrieve all available networks.

Name	**Arg.**	**Ret.**	**Description**
netExists	s	b	Returns True if network exists
getNets	None	as	Returns all networks
getPreferedCommunicationNet	None	s	Return current communication net
netAdded	s	None	New network available
netRemoved	s	None	Network removed
preferedCommunicationNetChanged	s	None	Communication net changed

org.mecel.net

Base interface for all networks. Contains methods and signals which all types of networks implements.

Name	Arg.	Ret.	Description
makeValid	None	b	Validate network manually
measureStarted	None	b	Mark network as currently measuring
measureStopped	None	b	Mark end of measuring
streamingStarted	None	b	Mark network as currently used for streaming
streamingStopped	None	b	Mark end of streaming
activate	None	b	Bring network online
deactivate	None	b	Take network offline
getName	None	s	Return name of network
getState	None	n	Return state of network
getType	None	s	Return type of network, "wifi" or "umts"
isOnline	None	b	Return True if network is online and useable
getQuality	None	n	Return quality of link
getBitrate	None	t	Return current bitrate
getIP	None	s	Return IP address bound to network
getGateway	None	s	Return current gateway
getDNS	None	ss	Return primary and secondary DNS servers
setState	n	None	manually set state
stateChanged	n	None	State changed
qualityChanged	n	None	Quality changed
bitrateChanged	t	None	Bitrate changed
ipChanged	s	None	New IP assigned
gatewayChanged	s	None	New gateway assigned
dnsChanged	ss	None	Primary or Secondary DNS has changed

org.mecel.net.umts

All cellular networks implements this interface

Name	Arg.	Ret.	Description
getCell	None	s	Return current cell the modem is bound to
getOperator	None	s	Return current operator
getMode	None	n	Return current mode
getSubMode	None	n	Return current sub-mode
getLAC	None	s	-
getRSSI	None	n	Return RSSI as reported by modem
getBER	None	n	Return BER as reported by modem
getTx	None	t	Return total transmitted data
getRx	None	t	Return total received data
getTxFlow	None	t	-
getRxFlow	None	t	-
getTxRate	None	t	Return current transmitting bitrate
getRxRate	None	t	Return current receiving bitrate
cellChanged	s	None	Modem has roamed to a new cell
operatorChanged	s	None	Modem has roamed to a new network
modeChanged	nn	None	Mode of modem changed
signalChanged	nn	None	Signal quality has changed
flowChanged	tttttt	None	New information regarding flow of data

org.mecel.net.wifi

Used by wifi networks.

Name	Arg.	Ret.	Description
getESSID	None	s	Return ESSID
getAP	None	s	Return MAC address of current Access Point
getMaxBitrate	None	t	Return max theoretical bitrate
apChanged	s	None	New Access Point registered
maxbitrateChanged	t	None	Max theoretical bitrate changed

org.mecel.modem

Simple interface to allow sending of custom AT[22] commands to modems.

Name	Arg.	Ret.	Description
syncAT	s	s	Send AT command manually and wait for reply
asyncAT	s	None	Send AT command manually and return immediatelly