Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Sports Reporter Application for future Mobile Networks

Master thesis

Author:	Remo Niedermann
Supervisor:	Prof. Dr. Bernhard Plattner
Advisors:	Ariane Keller & Dr. Franck Legendre

Zurich, May 26, 2010

Abstract

In the last few years mobile devices and applications for these mobile devices got more and more popular. Users like to create videos, music songs and pictures and share them with their friends. The distribution is done via Facebook, YouTube or just between two mobile devices. For users it would be nice to have an application which shares the content and publishes it in the Internet.
Todays mobile devices allow network connections to the Internet using WiFi in infrastructure mode and 3G. Further between two mobile devices a connection in Bluetooth and WiFi in ad-hoc mode can be established. A module on mobile devices, which keeps connected with all these networks at the time would give the user the ability to use the best network connection for its needs: 3G and WiFi in infrastructure mode for the Internet connection and a WiFi ad-hoc connection for the communication between two mobile devices to exchange data.
In the TCP/IP Internet architecture for the identification of two devices the IP protocol is used. In a network without routing, the IP protocol is not necessary because the identification of the two devices can be done with the MAC address. Removing the IP protocol can increase the payload throughput between the two devices. The common operating systems do not allow to remove protocols to create an alternative protocol stack. ANA allows to set up the protocol alternative stack and further allows to add functionalities such as encryption when it is needed.
In this master thesis an application is implemented on an Android phone which gives the user the ability to create podcasts and episodes which are distributed to his friends via WiFi in ad-hoc mode and to publish the episodes on the Internet. This application is based on ANA to use an alternative protocol stack in a WiFi in ad-hoc mode connection. As an extension there is information on how to implement a network switch on an Android phone.

Acknowledgment

At first I want to thank Prof. Dr. Bernhard Plattner for the possibility to write my master thesis at his institute at the Communication System Group of the Swiss Federal Institute of Technology (ETH) Zurich.
Further I am very grateful to Ariane Keller and Dr. Franck Legendre because for their continuous help and support as advisors during the whole master thesis.

Contents

Abbreviation List

ANA	Autonomic Network Architecture
ANM	Answer Message
API	Application Programming Interface
AuC	Authentication Center
BSC	Base Station Controller
BSS	Base Station System
BTS	Base Transceiver Station
DTN	Delay Tolerant Network
EIR	Equipment Identity Register
FB	Functional Block
FDD	Frequency Division Duplex
GMSC	Gateway MSC
GPRS	General Packet Radio Service
HLR	Home Location Register
IAM	Initial Address Message
IC	Information Channel
IDP	Information Dispatch Points
IDT	Information Dispatch Table
IMEI	International Mobile Equipment Identity
IMF	Integrated Monitoring Framework
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol

IWF	Interworking Function
JNI	Java Native Interface
Kc	Cipher Key
Ki	Secret Authentication Key
KVR	Key-Value Repository
ME	Mobile Equipment
MIS	Multi Interface Switch
MM	Mobility Management
MS	Mobile Station
MSC	Mobile Switching Center
MSRN	Mobile Station Roaming Number
OSS	Operations and Support System
RNC	Radio Network Controller
RNS	Radio Network Subsystem
RTT	Round Trip Time
SGSN	Serving GPRS Support Node
SIM	Subscriber Identity Module
SMSC	Short Message Service Center
SS	Switching System
TRAU	Transcoding and Rate Adaption Unit
VLR	Visitor Location Register
WCDMA	Wideband Code Division Multiple Access

List of Figures

1. Introduction

In the last few years smartphones got more and more popular mainly iPhone and Android phones. The advantage of smartphones is that independent developers have the possibility to implement applications and sell them in the App Store for the iPhone or the market for Android phones.
A lot of these applications use an Internet connection or a Bluetooth connection. Bluetooth connections are used for the communication between two mobile devices to exchange files whereas the Internet connection is used to get content from the Internet or to publish content in the Internet. For a connection to the Internet there are two ways: 3G and WiFi in infrastructure mode. The advantage of WiFi in infrastructure mode is that it is quite fast and often for free, whereas a 3G connection is much slower and usually the user must pay for the data exchange. The disadvantage of WiFi is that it is dependent of an access point, but 3G is almost available everywhere. For a connection between two mobile devices there is Bluetooth or WiFi in ad-hoc mode. A Bluetooth connection is very slow compared to a WiFi ad-hoc connection. But smartphones do not support WiFi in ad-hoc mode by default.
Today two mobile devices are mainly connected to exchange files such as videos, music songs or pictures. These media files are created by an owner of a smartphone, for example by taping a football game in a stadium with the smartphone camera or by taking a picture of a football player, which the user met before the game etc. Users, which are the authors of these media files like to share them with their friends. Today this is done by sending the file from a smartphone to a smartphone via Bluetooth or uploading the file to YouTube, Facebook or another web site in the Internet.
The social platforms like YouTube, Facebook etc. are very popular because people can upload their created content and share it with their friends. Not everybody is interested in the same hobbies such as sport, music, theater, etc. It would be nice if the content is categorized. Users then can subscribe for a category they are interested in and get informed when new content is published in this category. This is exactly what is done in podcasts. A podcast is a series of media files, for which a user can subscribe. A user can subscribe for different podcasts, for example a sports interested user can subscribe for a football podcast and an ice hockey podcast etc. Within a podcast the media files are grouped into episodes. An episode contains one or more media files. For example different videos from goals in the same football game are grouped in one episode. For a user with a smartphone it would be nice to create its own podcast channel and episodes and to distribute the content to his friends. The friends of the author of the podcast channel can be followers of the podcast and get the episodes.
For the author of such podcasts it would be nice to have an application which sup-

ports the creation and exchanging of episodes and podcast channels. The application should exchange the episodes between two mobile devices via WiFi in ad-hoc mode, which is a lot faster than the exchange via Bluetooth. The exchange of episodes with WiFi in ad-hoc mode has also the advantage that the distribution of the episodes is independent. The distribution can take place in remote areas such as the desert. The distribution is delay tolerant. Further good episodes such as nice football goals are also interesting for people which are not subscribed for the football podcast. Therefore good episodes should be published on the Internet.
Such an application is going to be implemented in this master thesis which allows the creation of podcasts and episodes, which are distributed via WiFi in ad-hoc mode. Good episodes are published on a web server. Further there is a new technology used in this application which is called ANA (Autonomic Network Architecture [1], see section 2.3). ANA allows to exchange the common TCP/IP stack with an own protocol stack. In WiFi in ad-hoc mode, without routing, the IP protocol is not necessary to identify the devices because the devices can be identified by the MAC address. The advantage of this alternative protocol stack is that the payload throughput can be increased. Further encryption of the payload of a packet can be added if it is necessary in ANA.

1.1. Task Description

The task of this master thesis is to implement an application, which allows the user to create a new podcast, subscribe to a podcast and to create a new episode. Further the application should exchange episodes with neighbours, who are subscribed to the same podcast. Finally it should be possible for the subscriber to open the file of the episode and rate the episode. Good episodes are interesting for other people, therefore good episodes should be published on a web server. Finally this application should run on mobile devices.

1.2. Contributions

The contributions of this thesis are:

- Implementing an application which uses ANA and PodNet
- Porting of ANA and PodNet to Android
- Design of a Multi Interface Switch

1.3. Outline

The next chapter discusses related work. It gives a brief background on ANA, PodNet and other important knowledge for this thesis. Chapter 3 is about the architecture and design of the application. The implementation of the application is described in chapter 4. The verification of the application, how it works and which features it provides is written in chapter 5. Chapter 6 is about how to set up a multi interface switch. Finally there is a summary of this master thesis in chapter 7. In the appendix there is a step by step guide how to set up a mobile phone for the application, run the application and set up a server for the file upload.

2. Related Work

2.1. Mobile Communication Interfaces

2.1.1. GSM

GSM is the European standard for digital cellular systems. It consists of the Switching System (SS), the Base Station System (BSS) and the Operations and Support System (OSS). The BSS consists of the Base Station Controller (BSC) and the Base Transceiver Station (BTS). In an ordinary configuration several BTSs are connected to a BSC and several BSCs are connected to the Mobile Switching Center (MSC). GSM uses the 850, 900, 1800 and the 1900 MHz frequency bands.

2.1.1.1. GSM Network Architecture

The Mobile Equipment (ME) (see figure 2.1) like a mobile phone, has a Mobile Station (MS), called antenna, which is connected to the BTS. Each mobile phone has a SIM card. The SIM Card contains user specific information, mainly the phone number and other subscribers's authentication information. One or more BTS are connected to a BSC. The BSC provides a number of functions related to the radio-resource management, some functions related to the Mobility Management (MM) for subscribers in the coverage area. BSCs on the other hand are connected to MSC. There are one or more BSC connected to the same MSC. MSC controls the call setup, call routing and the tear down of a call. In fact the subscribers are mobile, the MSC must provide a number of mobility management functions. One function is the Visitor Location Register (VLR). The VLR is a database containing subscriber related information for the duration the subscriber is in the overage area of a MSC. Between MSC and BSC is a module called TRAU (Transcoding and Rate Adaption Unit) which converts from or to speech with 13kbps (Full rate), 12.2kbps (enhanced Full rate) or 5.6kbps (half-rate) from the BSC to the standard 64kbps for the MSC.
Another module is the Home Location Register (HLR). It contains the subscriber's data e.g. the service for which the user has subscribed. Further it stores the location of the user's cell phone. By location it is meant to which BSS the subscriber is connected to. Attached to the HLR is the Authentication Center (AuC). The AuC contains the subscriber's authentication data like the secret authentication key (Ki) and several sophisticated authentication algorithms. For a given subscriber, the algorithm in the AuC and the Ki are also found on the SIM card. The AuC sends a challenge to the ME. If the returned value matches the user is authenticated.

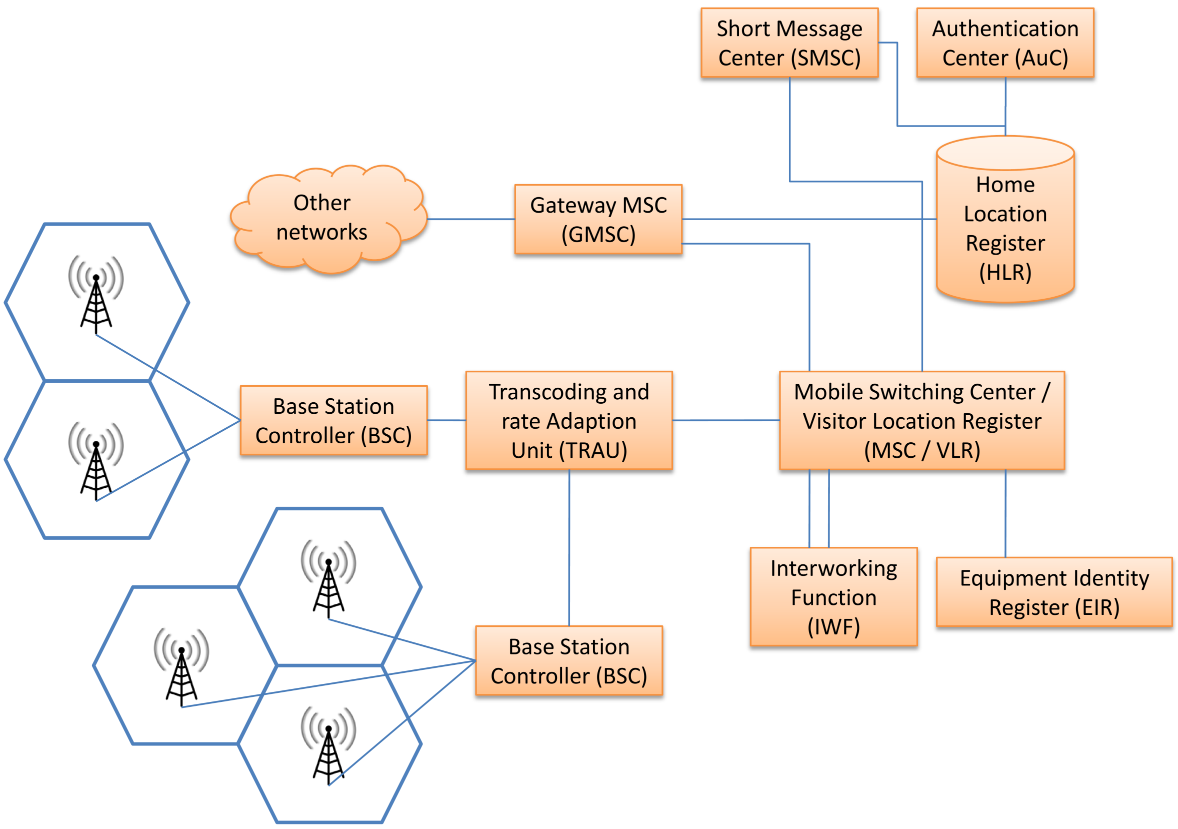

Short Message Center (SMSC)
Authentication Center (AuC)
Other networks
Gateway MSC (GMSC)
Home Location Register (HLR)
Base Station Controller (BSC)
Transcoding and rate Adaption Unit (TRAU)
Mobile Switching Center / Visitor Location Register (MSC / VLR)
Interworking Function (IWF)
Equipment Identity Register (EIR)
Base Station Controller (BSC)

Figure 2.1.: GSM Architecture

Calls from a different network, e.g fixed line network, first get to the Gateway MSC (GMSC). The GMSC then queries the HLR to get the MS's location and forwards the call to the ME.
One more component is the Short Message Service Center (SMSC). At the SMSC all SMS (text messages up to 160 characters) are stored and forwarded if the ME has a connection to the network.
Equipment Identity Register (EIR) is used to identify the ME, which contains the SIM card. This EIR is used to restrict access from a stolen ME. The identification of the ME is done by the International Mobile Equipment Identity (IMEI). It is a unique identifier for each mobile device. It is stored on the ME and can be used to identify a stolen ME. If an IMEI is blocked (stolen ME) in the EIR the call is not accepted from the network.
Finally there is the Interworking Function (IWF). It is used for circuit switched data and fax services and is basically a modem bank. In the early days of the Internet the dial-up modems (28.8kbps) were connected to the IWF.

2.1.1.2. Location Update

When a MS is turned on the position of the ME is probably different from the last one. In this case the location needs to be updated (see figure 2.2) Before the update can

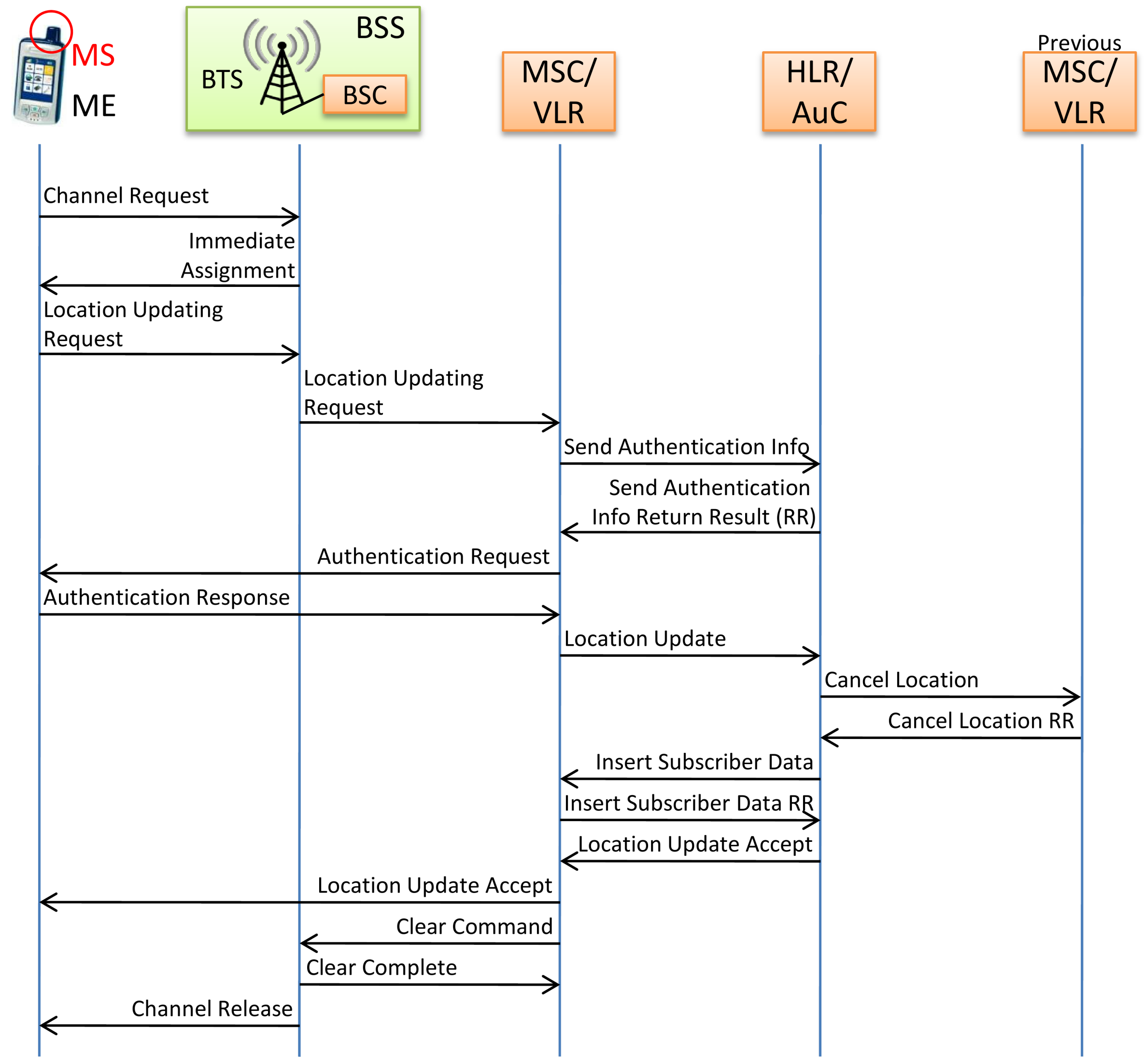


Figure 2.2.: Location Update

be done, the ME needs to be authenticated. For the authentication the MS sends the IMSI (International Mobile Subscriber Identity) or TMSI (Temporary Mobile Subscriber Identity) to the MSC/VR, which forwards it to the HLR/AuC. In return the MSC/VR gets a result with up to five authentication vectors. Each vector contains a random number RAND and a signed response SRES from the HLR/AuC. The MSC sends an Authentication Request to the MS containing the random number. If the response (*SRES) is equal to the SRES from the AuC, the MS is authenticated.
After the authentication the MSC sends the HLR the new location. The HLR informs

the previous MSC to delete all data about the subscriber in the VLR. Finally the HLR updates the new location and sends the new data back to the actual VLR of the new MSC. When the VLR has added the subscriber and its data, the VLR sends an ACK back to the HLR and next an ACK to the MS.

2.1.1.3. Mobile Originated Voice Call

In case a user with an ME wants to call a fix net phone, the MS first makes a Connection Management Service request to the BSS, which forwards the request to the MSC (see figure 2.3). After the authentication, the MSC sends the Cipher Key (Kc) to the BSS. Then the BSS instructs the MS to encrypt the data. The MS generates the Kc itself. This ensures that the key never gets transmitted over the air. The MS then informs the BSS and MSC that the data will be sent encrypted.
The MS then sends a setup message to the MSC with the information, which number should be dialed. When the MSC has all the needed information to connect the call, the MSC sends a Call Proceeding message to the MS. The MSC sends an Assignment Request to the BSS to establish a channel, between the BSS and the MSC, to transmit the voice data of the call. BSS sends an Assignment Command to the MS to establish a voice data channel. When everything is set up the BSS sends an Assignment Complete to the MSC to inform it, that there is a voice data channel from the MS to the MSC. The MSC initiates the call setup to the fix net phone (Initial Address Message: IAM). As a response to the IAM the MSC gets the Address Complete Message (ACM), which indicates that the dialed phone is ringing. The alerting informs the caller that the dialed number is called. When the call is answered, the phone sends an Answer Message (ANM) to the MSC. The MSC informs the MS with the Connect message that the connection is established. The MS acknowledge the message.

2.1.1.4. Mobile Terminated Voice Call

A call can also be initiated from a fix net number. From the fix net arrives an IAM (see figure 2.4) at the GMSC containing the dialed number of the ME. Because the ME is mobile the GMSC does not know the position of the mobile device. Therefore it needs to ask the HLR of the position and the corresponding MSC for the location. The HLR sends a message to the MSC to allocate a temporary number (Mobile Station Roaming Number MSRN) at the MSC. The MSC sends the MSRN to the GMSC, which forwards the IAM to the MSRN at the MSC.
Next the MSC requests the BSS to page the subscriber, which stores the location area in which the subscriber should be paged. The MSC sends the Cipher Mode command containing the Kc to the BSS to get an encrypted channel. After the MS has generated the Kc itself, it signals the MSC that the ciphering is on. Afterwards MSC sends a Setup message to the MS with the information of the caller. The MS confirms the call. Then the channel gets assigned. Now the phone is ringing and the MS informs the MSC of the ringing whereas MSC informs the fix net phone with an ACM that the phone is ringing. When the subscriber picks up the phone, the MS

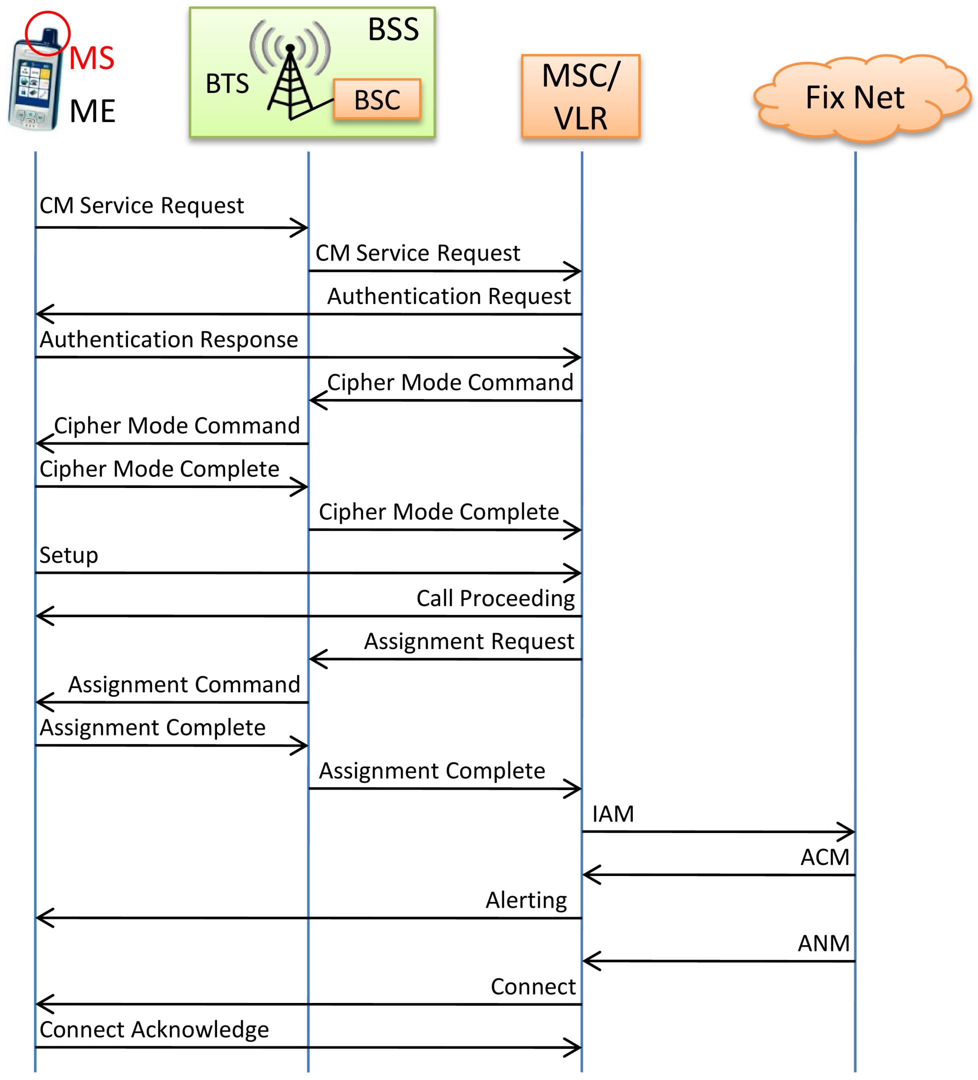


Figure 2.3.: Mobile Originated Voice Call

sends a Connect message to the MSC, which on the other side sends an ANM to the GMSC and the dialer. Finally the MSC sends a Connection ACK to the MS. The connection is established.

2.1.1.5. Handover

A mobile subscriber can walk around during a call. It happens that the connection gets worse and there is a BTS with a better reception. In this case it is useful to hand over the call to the other BTS. The call can be handed over between cells of the same BTS, between cells of different BTS but connected to the same BSC, between cells of different BSC or between cells of different MSC. This process is called handover. If a handover takes place between two cells on the same BSC, the BSC will handle the handover and then informs the MSC. In case of a handover between two BSC the MSC must be involved because there is no direct connection between different BCS. In GSM the handover is also called „mobile-assisted handover“, which means that the network decides if, how and when a handover should take place.

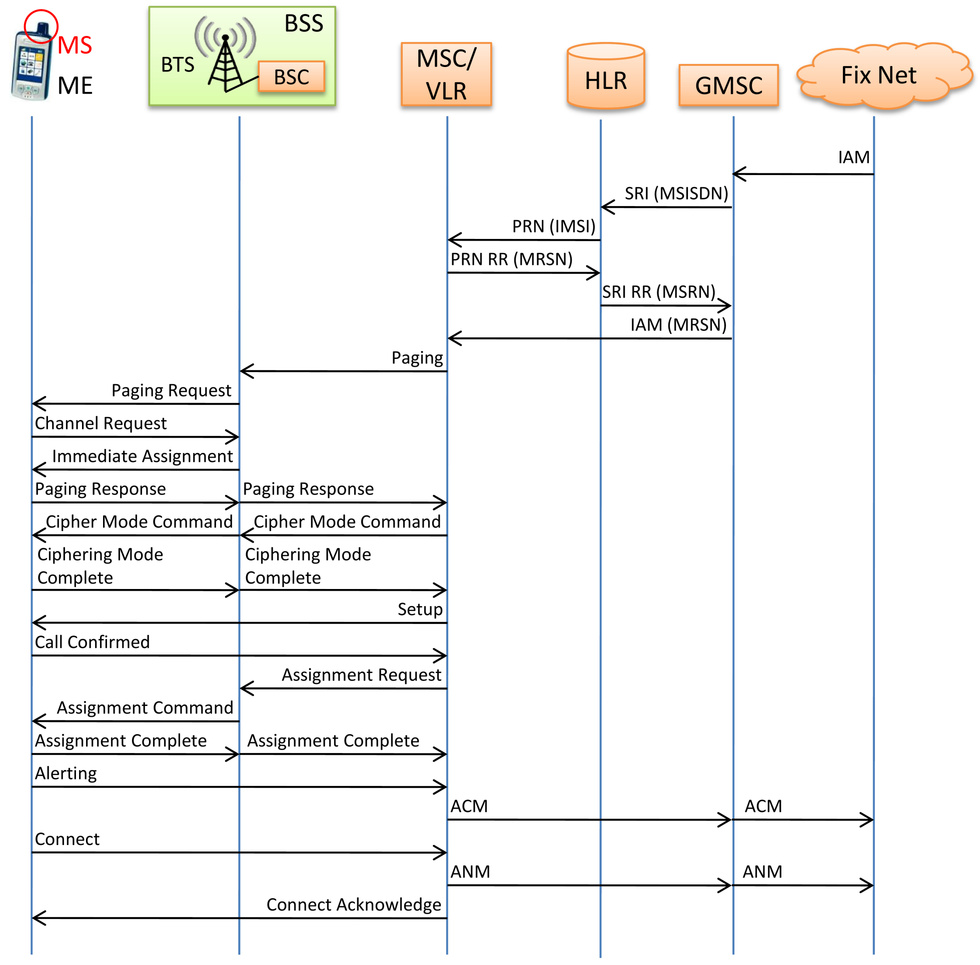


Figure 2.4.: Mobile Terminated Voice Call

GSM uses TDMA (Time Division Multiple Access) with eight time slots per frame. During the time, where a MS is not sending or receiving data, it has time to listen to other carrier frequencies from other BSCs and taking signal measurements. The results are reported back to the BSC (see figure 2.5). A good scanning result does not automatically mean the MS has to synchronize with this BTS. It can be that the BTS does not have a good reception of the MS. Therefor the BTS makes signal measurements of the signal from the MS and reports them back to the BSC. With the results from the MS and the BTS, the BSC can determine if it is necessary to handover the MS to another BTS of the same BCS or even to a BTS of another BCS. In case a handover must occur between two BSC, the current BSC informs the MSC with a Handover Required message. This message contains the desired target cell

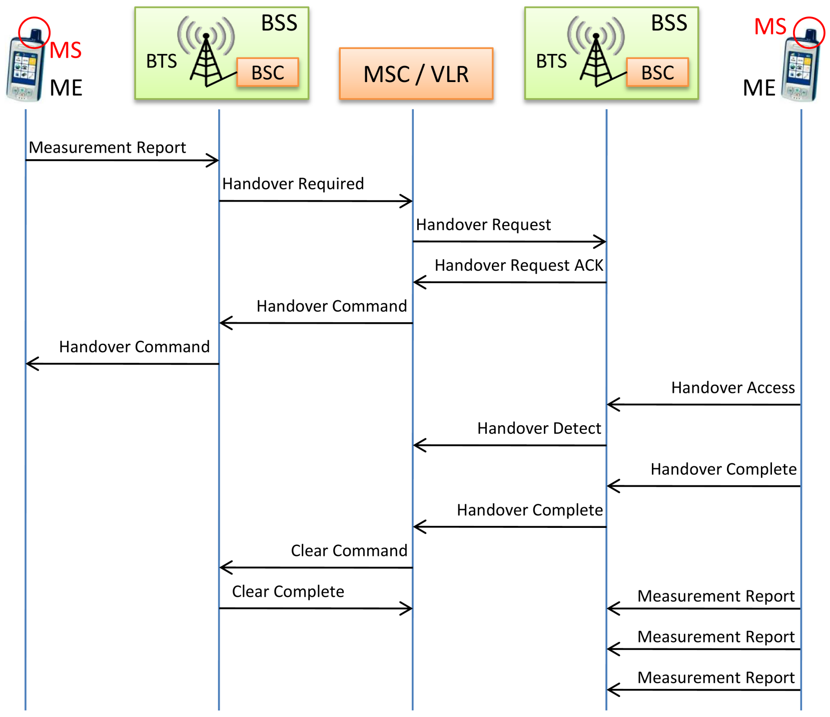


Figure 2.5.: Handover

and the current channel the MS is using. The MSC then determines which BSCs are involved and then sends a Handover Request message to the new designated BSC. If the BSC accepts the handover request, the MSC sends a handover command to the old BSC, which passes on the request to the MS. The MS switches to the new BST and BSC and sends a Handover Access and Handover Complete to BSC. The BCS detects the handover and informs the MSC. Finally the MSC informs the old BSC about the completed handover and that the subscriber's data can be deleted on the BSC.
The handover process is now completed and the BTS and the MS start again with signal measurements and the reporting to the BSC.

2.1.2. Third Generation (3G)

The third generation is based on the Internet Protocol (IP), which is a world wide used standard in the Internet. Development in the BTS are also made for the third generation. These changes of technology gives a lot of advantages:

- Compatibility of service between different networks mainly between the mobile phone network and the Internet
- High quality
- Worldwide common frequency bands

- Small MEs for worldwide use
- Worldwide roaming possibility
- Multimedia application services on MEs
- Flexibility for evolution to the next generation of wireless
- High speed data packet rates

For the third generation and future generations it is important that they can be integrated into the existing network of older generations (1G and 2G).

2.1.2.1. Universal Mobile Telecommunication Service (UMTS)

For the UMTS [3] the standard technology in Europe is WCDMA (Wideband Code Division Multiple Access) [4] combined with FDD (Frequency Division Duplex) [5]. Because GSM [6] is very popular UMTS needs to be an evolution of GSM. The first UMTS release used the same network infrastructure as GSM and GPRS (General Packet Radio Service) [7] with some enhancements. The BST got further development such as the Enhanced Data Rates for Global Evolution (EDGE) [8]. GSM and UMTS (WCDMA, FDD) are specifications in Europe from the European Telecommunications Standards Institute (ETSI) [9]. The first release of the UMTS of ETSI it is known as „3GPP Release 1999“ [10].
In UMTS the data rate is much higher than in GSM or GPRS. It is up to 2Mbps. In UMTS there are four service classes which have different characteristics:

Conversational: This class has a low delay, jitter and error tolerance. The data rate can be high but only in one direction. For example Voice is very delay sensitive in a typical conversational application but it requires low data rates. Video conference is also conversational. It is delay sensitive, less error tolerant and requires high data rates.

Interactive: Interactive applications consist of request and response transactions. It is error sensitive but delay tolerant. Jitter does not harm interactive applications very much. Interactive applications can have high and low data rates but the high data rates are just in one direction at the time not typically in both ways at the same time.

Streaming: Streaming is a one-way service which can use high data rates. Streaming is sensitive for errors but can be tolerant to delays and jitter. A streaming application can be delay and jitter tolerant because it uses buffers to adjust the delay and jitter of the incoming packets. Streaming is used for Audio and Video applications.

Background: Background traffic has, if any, a delay constraint. For SMS and Emails, it is not important if a user gets the message now or 5 seconds later. It is important that the user gets the notification.

2.1.2.2. Overview of 3GPP Release 1999

The 3GPP Release 1999 (see figure 2.6) is the first release of UMTS. The RNC (Radio Network Controller) controls the resources of the RNS (Radio Network Subsystem). RNC is the equivalent of BSC in GSM and RNS is the equivalent to BTS. RNC are connected to MSC/VR and to the SGSN (Serving GPRS Support Node). The link to the MSC/VR is used for calls, location updates, etc. whereas the link to the SGSN is used for data packets to the Internet. Packets to the Internet are forwarded from the SGSN to the GGSN (Gateway GPRS Support Node).

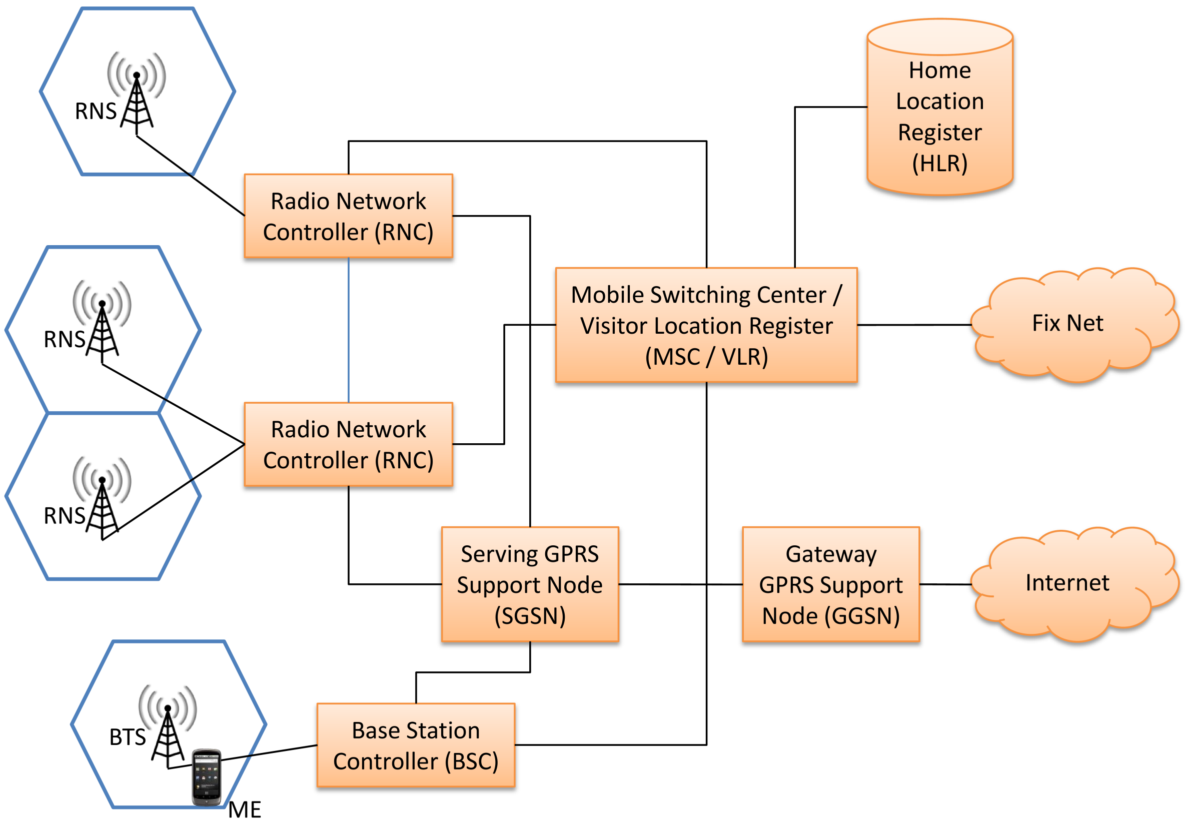

RNS
RNS
RNS
Radio Network Controller (RNC)
Radio Network Controller (RNC)
Home Location Register (HLR)
Mobile Switching Center / Visitor Location Register (MSC / VLR)
Fix Net
Serving GPRS Support Node (SGSN)
Gateway GPRS Support Node (GGSN)
Internet
BTS
ME
Base Station Controller (BSC)

Figure 2.6.: UMTS Architecture

2.2. Multiple Wireless Interfaces

Today most mobile devices like iPhone, Android phone, Blackberry, . . . have different wireless connections such as 3G and WiFi at the same time. The throughput of a 3G link is much smaller than the throughput of a WiFi link. The aggregation of the throughput is not much higher than the throughput of the WiFi. Is there a better way then just the aggregation of the throughput of a WiFi and a 3G link to transfer data to a server?

Via a wireless link only one device can send data at the time. It is either the base station or a mobile device. A mobile device send some packets over the wireless link with a TCP connection to a server. The server then acknowledges the packets with ACKs (one ACK per packet or one ACK for more packets). The ACK packages are sent back to the mobile device. The mobile device has a sliding window (see figure 2.7). This means that the sending device can only send a number of packets before it has to wait for the ACKs. When an ACK arrives the window is slided forward for the number of packets the ACK acknowledges. Then the sender can continue to send packets. The mobile device can not sending packets permanently. It needs to stop from time to time to receive the ACKs before it can continue to send more packets.

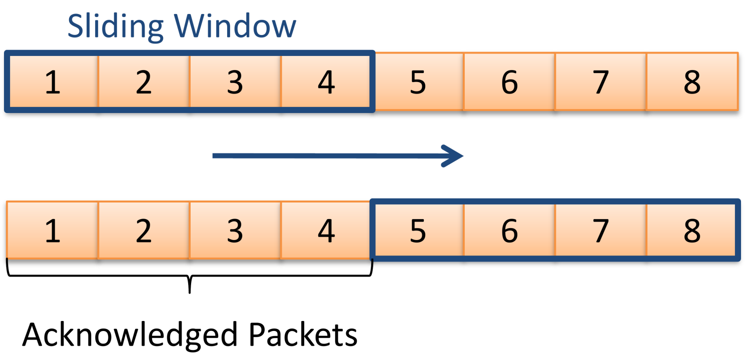


Figure 2.7.: Sliding Window

To increase the throughput one can use a second link for example the 3G link. Sending additionally packets via 3G increases the throughput. It is just the addition of the two throughputs but it can be done better with „Super Aggregation“ [12].

2.2.1. Super Aggregation

Super Aggregation uses a WiFi and 3G link to send data from a client to a server. If the client has a 3G and WiFi connection it is possible to send the big packets (data packets) permanently via the faster link (WiFi) and receive the small packets (ACKs) on the slower link (3G). The client can send data to the server and does not need to stop sending to receive ACKs because they arrive at the 3G link.
For the implementation of the super aggregation only the client needs to be modified. The data packets of the client which sends the data of the WiFi link contains the IP of the 3G link. When this packet arrives at the server it sends an ACK to the source IP of the arrived data packet. This source IP is the client's 3G connection. The ACK arrives automatically at the 3G interface of the client.

2.2.2. Results

A simple aggregation of a 3G link and a WiFi 802.11g sending data with TCP gives an increase of the data throughput of 3% compared to a WiFi 802.11g. With the

WiFi 802.11b standart the increase is 16%. If one uses the Super Aggregation the increase of the throughput is 37% in the g standard and 152% in the b, compared to the simple WiFi.

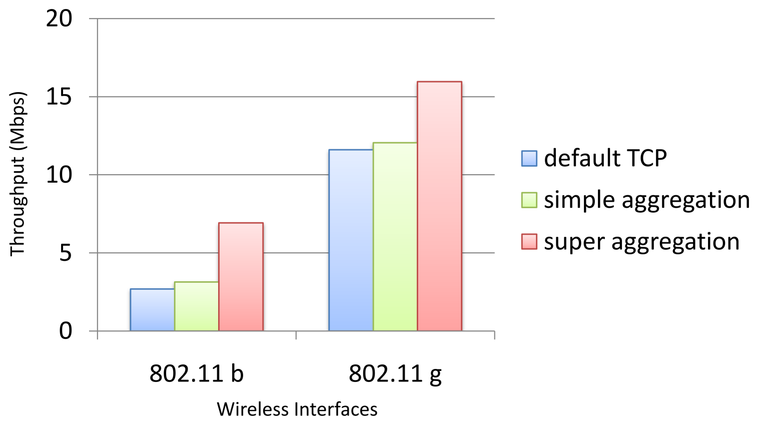


Figure 2.8.: Performance Results

2.3. ANA

ANA (Autonomic Network Architecture) [1] is a network research project which universities from Europe and North America are participating. The goal of ANA is to develop a new network architecture, where the Internet protocol stack is adjustable to the needs of the user. Until now the TCP/IP protocol stack is fixed and can not be changed. In ANA it is possible to use only the protocol layers which are needed and functionalities such as encryption can be added whenever it is necessary.
ANA does not consist of layers, it consists of Functional Blocks (FB). A functional block implements a protocol such as IP, TCP or a functionality like encrypting the payload. The advantage of functional blocks compared to layers is that functional blocks can be arranged in different orders and are independent of the layer beneath. In the TCP/IP stack the TCP layer is usually above the IP layer. In ANA it is possible to remove the IP layer which is unnecessary in a network without routing.
ANA consists of a core called „minmex“ and the FBs called „bricks“. The bricks are loaded into ANA with the „mxconfig“ program. ANA supports an API for the communication between the functional blocks.

2.3.1. Functional Blocks

A FB is a small program which can be plugged-in ANA as a brick. It can be the implementation of a protocol such as TCP or IP. Further it can be a program like a chat

program or PodNet (see section 2.4). Each FB can provide services and use services from other FB. A service is published within this ANA node.
The minmex knows which bricks are running on this ANA node. For the communication between two bricks, minmex uses IDPs (Information Dispatch Points) (see figure 2.9). The first brick publishes a service in the KVR (Key-Value Repository) and in return gets an IDP which has an entry in the IDT (Information Dispatch Table). The second brick, which wants to use the service, resolves the service and gets back the IDP to which the brick can send the data.
If a brick gets removed from the ANA node the services need to be unpublished and the bricks, which used this service, must be informed which have a connection to the brick.

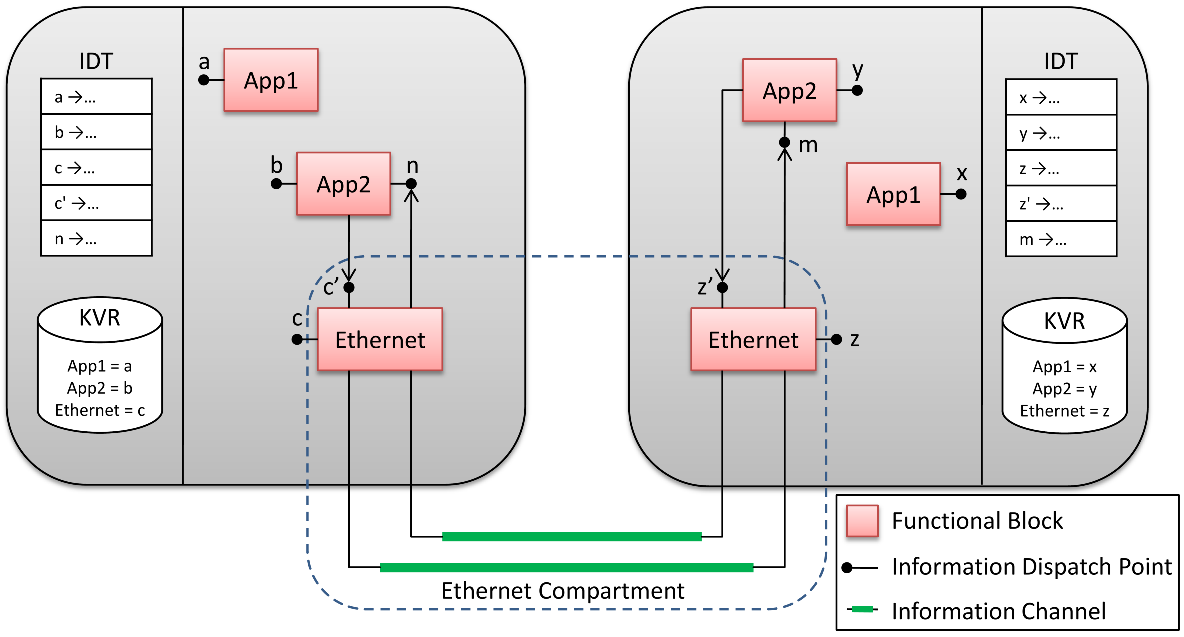


Figure 2.9.: ANA Communication

2.3.2. Information Channel

Between two ANA nodes there are Information Channels (IC) used as communication channels. The information channels can be wired, wireless or virtual links which are used for the communication between the nodes such as forwarding data, resolve IDPs of the a service in the other node etc. ICs also have IDPs for the identification.

2.3.3. API

The API (Application Programming Interface) in ANA is used in every FB to have a standard communication between the FBs. It simplifies the publishing and unpublishing of services, the start and stop of the bricks and the communication between the bricks.

2.4. PodNet

Today mobile devices are more and more used to watch or listen to episodes from podcasts from the Internet and upload self created videos and music songs to YouTube, which everyone can view or listen to. A lot of the applications use the Internet to distribute content and cause Internet traffic. A lot of people create such media files and want to share them with their friends but mostly these friends meet each other at work, in a bar, at sports, etc. It can be easier to share the media files directly with these friends.
PodNet provides this functionality. It allows a user to create media files and attach them to episodes. Each episode belongs to a channel which users can subscribe for. Each channel contains some specific content. For example a football channel contains interviews of football players, videos of goals etc. If two users of PodNet meet each other, their mobile devices exchange episodes of the same channels on their own. Afterwards the users can view the new episodes.
PodNet works on a delay tolerant network (DTN). This means that the mobile phone of the author of an episode does not need to have an end-to-end connection to all of its clients. The advantage of DTN is that mobile devices with no Internet connection in a remote area can receive the episodes from another mobile device than the authors mobile device.

2.4.1. PodNet on ANA

PodNet is a FB in ANA, which is used to create an episode or a podcast channel and store it. It uses different other bricks such as the TCP, IMF (Integrated Monitoring Framework) and Syncdiscovery. TCP is used for a reliable transmission of data packets between two devices running PodNet. IMF is used to monitor neighbours by the evaluation of the measurements of the neighborhood. Syncdiscovery keeps track of all the neighbors. A neighbor is called stable when at least three consecutive packets arrived. The second task is to get informed by PodNet or other Syncdiscoveries on different devices if something has changed (new channel or episode added). PodNet verifies if a neighbor is subscribed to the same channels and if it is the case, checks if the neighboring PodNet has new episodes to exchange.
PodNet has two modules to communicate with: a GUI for Linux like Ubuntu or a command line interface. In this master thesis PodNet is extended with a third module (see section 3.3.2). It is an interface which communicates via Unix sockets. The interface is implemented as a C library and can be used by bricks or other programs.

A list of the commands in command line interface of PodNet:

channel_list: Displays all the channels (*channel_list*)

channel_add: Adds a new channel (*channel_add <channel_name> <channel_name>*)

channel_remove: Removes a channel (*channel_remove <channelID>*)

channel_removeall: Removes all channels (*channel_removeall*)

episode_list: Displays all the episodes of a channel (*episode_list <channelID>*)

episode_add: Add a new episode to a channel (*episode_add <channelID> <episodeName> <filePath>*)

episode_remove: Remove an episode from channel (*episode_remove <episodeID>*)

2.5. Android

Android is an operating system for mobile devices such as smartphones, mobile phones and netbooks. This operating system was initially programmed by Android Inc., which Google purchased in 2007. Now it is the Open Handset Alliance [13], consisting of 34 enterprises, which develops Android. Most of the code from Android is published under the Apache License [14]. This is an open source license, which means that everyone can get, extend or change the code.
The basis of Android is a Linux 2.6 kernel (see figure 2.10), which is used for memory and process management and the network communication. Further the kernel is the hardware abstraction layer for all the other software and provides the device drivers.
The runtime environment of Android is the Dalvik Virtual Machine (DVM). It is similar to a Java VM. Both execute byte code. The DVM is designed to run efficient multiple instances on the same device. This is necessary because each application runs in its own process and DVM.
Applications are written in Java. For performance critical calculations there exist a Native Development Kit (NDK) which allows the user to write some functions in C or C++. Android itself has some performance critical calculations written in C and C++ such as codecs for media player, web browser or a database (SQLite).
To develop its own applications the Java SDK and the Android SDK are needed. The code gets compiled from the Java compiler and afterwards from the cross assembler for the DVM. For the development of an application there exist a plug-in for the editor Eclipse which handles the compilation and puts the application on an emulator or the Android phone.

2.5.1. Android GUI Functionalities

In the GUI of an application there are several GUI design elements such as multi language, long clicks, layouts, menus, dialogs and styles. In this section it is described

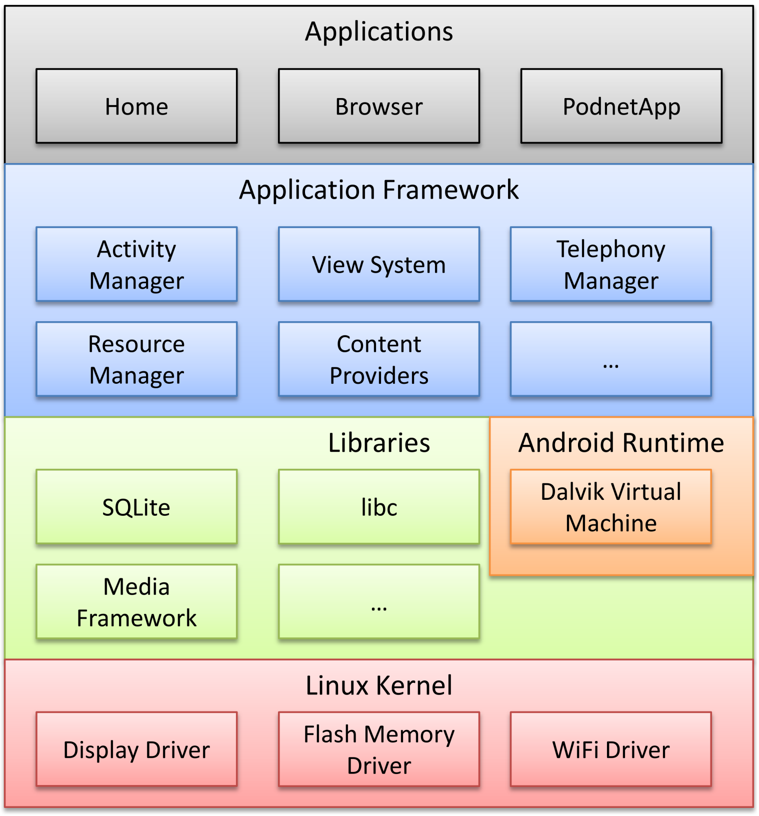


Figure 2.10.: Architecture of Android

where and how to use these elements. The code for an Android application is written as an Android project which is similar to a Java project. An example of a folder structure of an Android project can be found in the appendix E. In the */res* folder are all the resources such as the multi language files, layout files, pictures, etc. The folder */src* contains all the Java classes. A page in Android, which is displayed on the mobile device, is called view. Each view is implemented in an own Java class which must inherit from the „Activity“ class. All activities are declared in the AndroidManifest.xml file (see in the appendix F.1).

2.5.1.1. Multi Language

An application can be used all over the world but not everyone speaks the same language. For this reason it is important that the application is translated to the same language which the user has defined on his mobile phone. If the application is developed properly Android offers a simple solution. In Android it is possible to store strings, string arrays, integers, etc. in the *strings.xml* file, which is in the */res/values* folder. Each value can be defined by a variable name. The variable name can then be used in all the classes but mainly in the activities. The application in German or English differs only in the values in the *strings.xml* file. For multi language support a new folder *values−de* must be created in the */res* folder. Second a new *strings.xml* file must be created in the new folder, containing all the variables for the new language. Finally all the variables which differ from the English version are redefined in the *strings.xml* file in the */res/values* folder. The variables which are not defined in the new file are taken from the default version in the *strings.xml* file.

2.5.1.2. Layout

The layout of a view can be defined in two different ways: implementing the view in an XML file containing the body of the view or it in the activity for the corresponding view. The best way to implement the view is to use the XML file version. Later changes in the view are very easy to do in the XML file. Further the code of the view is easy to understand. The disadvantage of this view implementation is that the hard coded view is not dynamically. If the final view is not known at compile time it is better to implement the view in the activity.
In the PodnetApp the code for the layout is in the XML file.

2.5.1.3. Menu

Each Android mobile phone has a *Menu* button. When an application runs and the user presses this button, the menu is displayed in case there is one implemented. The layout of the menu can be defined the same way as a view: in an XML file in */res/menu* or dynamically in the function *onCreateOptionsMenu*.
If a user clicks on an item in the menu, it calls the *onOptionsItemSelected* to determine which menu item was selected. An example of a menu is implemented in the *ChannelListView*.

2.5.1.4. Dialog

Dialogs are very helpful to ask a user a question and get a „yes“ or a „no“ such as „Do you want to delete this episode?“ and two buttons a *Yes* and *No* button. A dialog is shown with the command: *showDialog*. The definition of the dialog is in the *onCreateDialog* function. A dialog with buttons is done with the „AlertDialog“ class.

2.5.1.5. Styles

Styles are useful for an application that the same elements look the same and if the style needs to be changed there is only one place to change the values. Further styles can also be different in different languages. These reasons make it reasonable that the styles are defined in the *strings.xml* file of the different languages (see section 2.5.1.1). Buttons, text, tables etc can have styles. In this application the styles are used to define text sizes in text views.

2.5.2. Permissions

In Android an application uses permissions to access contacts, establish a network connection, start a phone call, sending and receiving SMS, etc. To get the required permissions they need to be declared in AndroidManifest.xml (see appendix F.1). For example to send or receive SMS the $SEND\_SMS$ and $RECEIVE\_SMS$ are required.

3. Application Architecture / Design

Today social applications get more and more important. In this thesis a new application is implemented to exchange multimedia files. Today's applications use Bluetooth [15] or WiFi[1] [16] in infrastructure mode to exchange files between two mobile devices. The data throughput in Bluetooth is quite low and the connection can only be established within short range [17]. WiFi in infrastructure mode has a much higher data throughput and the communication radius is longer than in Bluetooth. But WiFi in infrastructure mode has the disadvantage that it always needs an access point. Without the access point there is no communication between two mobile devices. WiFi in ad-hoc mode solves this problem. In WiFi in ad-hoc mode the devices have a fast connection with a high data throughput and to establish the connection there is no access point needed.
The social application developed in this thesis is used to create and subscribe for podcast channels, which contain episodes. An episode contains media files such as videos, music songs or pictures and can be added to a podcast channel in the application. The episodes are exchanged between two mobile devices via the WiFi in ad-hoc mode. Afterwards a user can watch or listen to the content of the episode and finally he can rate the episode. The rating is sent to the author of the episode and is stored at his mobile device. If the episode has more than a required number of ratings and the ratings exceed a threshold the content of the episode will be published on a web server. Now the content of the episode is public and it is no more just available in the podcast channel.

3.1. Requirements

The implementation of the application is done on Android. To use this application the requirements are:

- Two or more mobile phones running on Android 1.5 or newer and having WiFi and 3G
- A web server to publish the content of the episodes (must allow file upload)

[1]WiFi: 802.11 standard

3.2. Setup

The application is designed for Android phones. The basis of the application is ANA and PodNet. Those, ANA and PodNet, run also on Linux devices like Ubuntu. This gives the possibility to use the application with different setups, but the rating system can only be used on Android devices.

- Multiple Android mobile phones
- Multiple laptops with Ubuntu
- Any combination with Android mobile phones and laptops

3.3. Application

The application consists of different code parts. At first the application is based on ANA and PodNet. ANA is used for the basic communication between the mobile devices and to handle the received packets. The function of PodNet is to store and to exchange episodes and the media files. The code of ANA and PodNet existed at the beginning of the thesis but the PodNet Interface needed to be added. The architecture of the application is shown in figure 3.1.

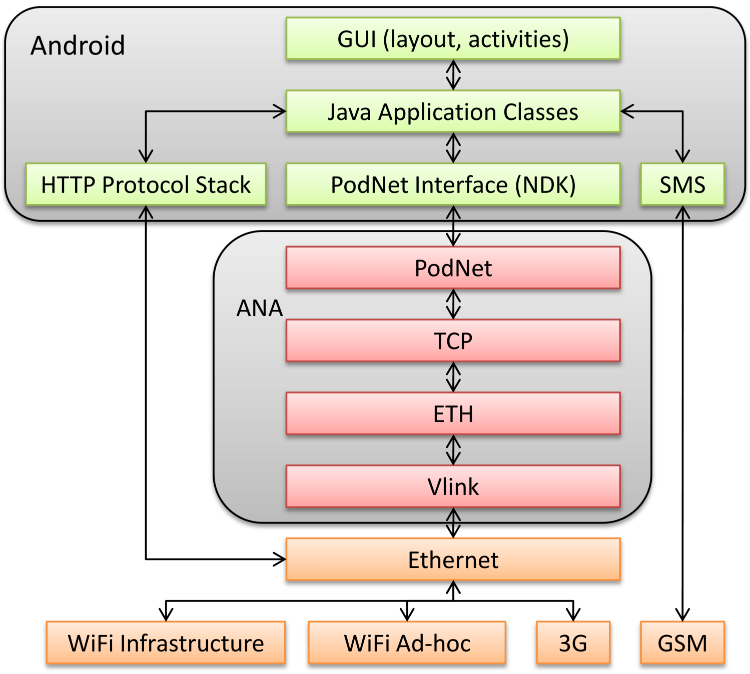


Figure 3.1.: Architecture of the Application

3.3.1. GUI

All applications on Android need to be implemented in Java. The code of the GUI is organized in an Android project. In the root folder of the project are the AndroidManifest.xml and default.properties files and the bin, gen, libs, res and src folder.

bin: The bin folder contains all the *.class* files, which are the compiled files of the *.java* files.

gen: Contains the R.java file. It initiates static variables (this file is generated automatically).

libs: For our application one needs two libraries (libana-app.so and libpodnet-interface.so). Libana-app is used by the application to communicate with ANA. Libpodnet-interface contains the interface to exchange information with PodNet (for more detail see section 4.2.2)

res: This folder contains all static resources such as layouts, menus, pictures and a file of strings values (see section 2.5.1.1).

src: This folder contains all the *.java* files. In these files all the functionalities of the GUI are implemented.

/src/ana/bricks The files in this folder are used to load the libraries from the *libs* folder and to call the functions in the libraries.

/src/ch/ethz/tik/ana containts the main code of the application. The folder *view* contains all the different views. The main view (mainView.java) is shown at the start up of the application. The user clicks on a button or an item and a new view is opened for example if the users selects the settings. In this case the view for the settings is shown. In this case the code is SettingView.java.

AndroidManifest.xml: AndroidManifest contains all the important information about the project. All views are declared in this file. Further the package name, receivers, permissions, etc. are defined in this file.

The whole folder structure of the project is found on Page 62.

3.3.2. PodNet Interface

PodNet is used to exchange the episodes in the podcast between two devices that are running PodNet for example two Android phones. For a user there are two ways to communicate with PodNet: one is to use the command line interface and the second one is a GUI for Linux and Windows Mobile hosts. To integrate PodNet in our application there must be an additional interface to talk to the PodNet. The interface must also be available to other bricks in ANA.

The GUI of the application is implemented in Java whereas ANA and the PodNet Interface are written in C. To communicate between Java and C a so called Java Native Interface (JNI) is required.

The PodNet interface finally consists of three parts:

- Extension of PodNet to allow other bricks and C programs to communicate with PodNet
- C library, which bricks or C programs can include to talk to PodNet
- The JNI between the C library and the GUI

The code of the three parts are at different places:

- The code of PodNet is in the podnet.c, podnetMain.cpp, podnetMain.h, datastore.cpp and datastore.h. These files are located in the folder */ana−core/devel/C/bricks/dtn/podnet*.
- The C library and the JNI are in the Android NDK [2] environment. The NDK is used to cross-compile the library for the Android phone architecture. JNI itself consists of two parts: the jni-brick.c file in the NDK environment and the file PodnetNDK.java in the application project environment at */src/ana/brick*.

3.3.3. Rating System

In the application is a rating system included. It is used to rate episodes. Important for a rating system is the authentication of the evaluator, the integrity of the rating and that a user can only rate once. One way to send a rating is via the Internet architecture, using 3G or WiFi in infrastructure mode. In 3G or WiFi in infrastructure mode, the packets are routed on the bases of IP and not on the phone number. The IP address changes every time the user connects to a base station. This implies that IP can not be used to identify the evaluator. Sending the phone number or IMEI within the packet does not help. A user can change the mobile phone or just insert a wrong phone number or IMEI in the message payload. The receiver of the rating has no chance to authenticate the evaluator.
The simplest solution is to rate via SMS (see figure 3.2). The mobile phone of the evaluator sends an SMS with the rating to the mobile device of the author of the episode. The author's device can not determine if the rating is sent from the mobile device of the evaluator because SMS can also be sent from an Internet homepage. Therefore it sends back a challenge to the evaluator to determine if the evaluator is really the owner of the mobile phone number. If it sends back the correct challenge the evaluator is authenticated.
The integrity of the SMS is given by the fact that SMS are transmitted encrypted via GSM or GPRS (see section 2.1.1.2). This means that if the evaluator is authenticated

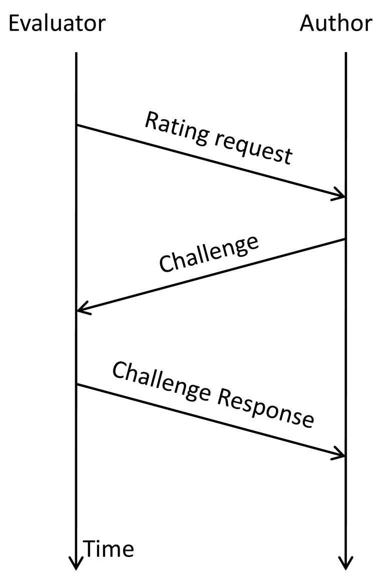


Figure 3.2.: Rating System

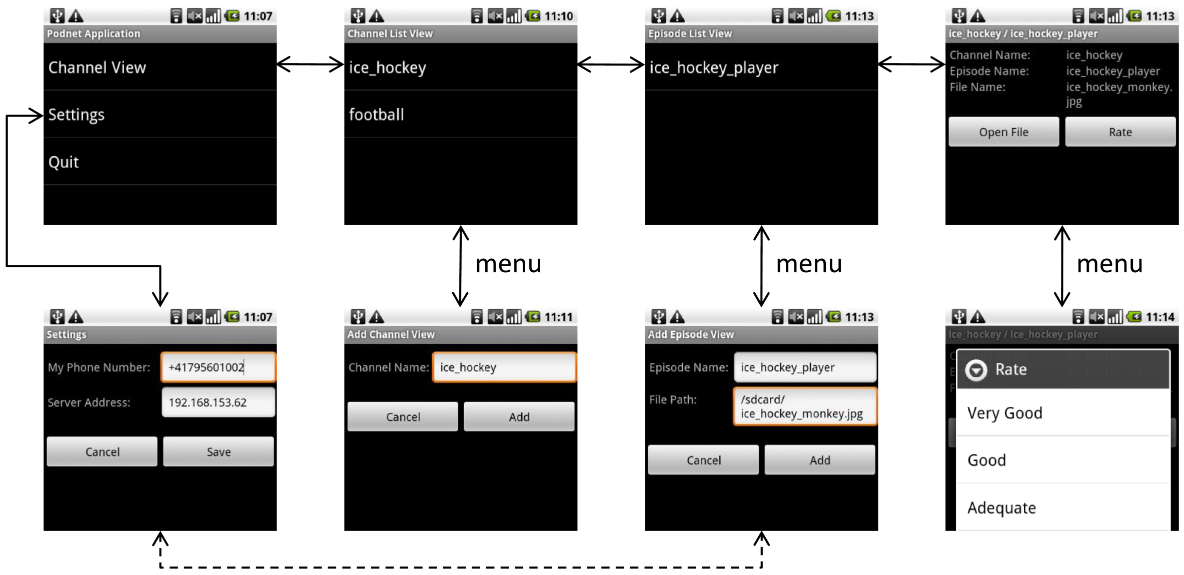
(the integrity is given by default) the rating can be automatically accepted.
The author of an episode gets the ratings from other users. For the authentication of the evaluator, he sends a challenge back in an SMS. The author pays for each SMS he sends. This financial aspect is also a mechanism which reduces bad content in the podcast channels. An author does not want to pay for ratings of an episode in which nobody is interested.

3.4. GUI

3.4.1. Application Design and Functionalities

3.4.1.1. Flow Chart

The application consists of different views. Each view is like a page on a web server. To get from one view to another, the user clicks on a button or selects a menu action. The chart flow of the application is shown in figure 3.3.
The entry point of the application is the main view (top left in figure 3.3). At the first start of the PodNet Application, there are a few settings, which should be set in the first place: the phone number and the address of the server (IP or domain name). When a user creates a new episode, the file and his phone number are stored in the episode. The phone number is used to send a rating from the evaluator to the author of the episode. The server address is used to upload the episode, when there are enough ratings and the rating exceeds a threshold. To get to the settings one need to click on the list entry called *Settings* in the main view.
From the main view a user can click on *Channel View* to get the list of all the channels



Channel Name: ice_hockey
Episode Name: ice_hockey_player
File Name: ice_hockey_monkey.jpg

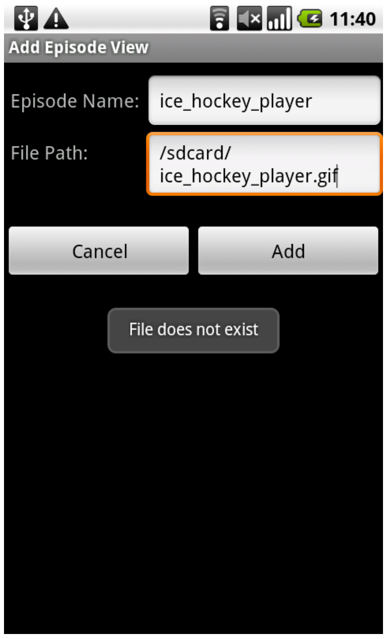
My Phone Number: +41795601002
Server Address: 192.168.153.62

Channel Name: ice_hockey

Episode Name: ice_hockey_player
File Path: /sdcard/ice_hockey_monkey.jpg

Figure 3.3.: Application Flow Chart

he is enrolled. A user can add/remove or open channels. To add a new channel the user needs to press the menu button on the phone and then select *Add*. In this view one can add a channel.
The episode list is opened, when in the channel list, the user clicks on a channel. It displays all the episodes within this channel. New episodes can easily be created. The view to add a new episode is opened by pressing the menu button and clicking on *Add*. For the creation of a new episode one need to enter the episode name and the path of the file. If the path is not correct an error message will be displayed (see figure 3.4). To create the episode successfully the phone number must be set in the settings. If it is not set and the user clicks on the *Add* button, the user is forwarded to the setting view to enter the phone number and then returns back to the view to finally add the episode to the channel. On figure 3.3 this flow from the *Add Episode View* to *Settings* is a dashed line. This is because the view of settings can not be opened from the *Add Episode View* by the user. It is only opened when the phone number is missing at the time of creation.
When a user clicks on an episode in the list of episodes the episode is opened and the information about the episode is displayed: channel, episode and file name. If the file is a music, video or picture file it can be opened directly with the *Open File* button. For all other files a list of programs is displayed which are able to open the file. After selecting the right application the file is displayed.
The rating is opened by pressing the menu button and clicking on the *Rate* item. This opens a list to rate the episode. After selecting the desired rate and confirming it, the rate will be sent to the author of the episode.



File does not exist

Figure 3.4.: Error Message: File does not exist.

3.4.1.2. Functionalities

The GUI has several functionalities: long clicks, menus and dialogs. These are interaction elements which should help for the communication between the user and the application.

Long Click: A long click is, when a user presses an item for a long time until the GUI displays something. As an example this functionality is implemented in the episode list view. If a user clicks for a long time on an episode name, the GUI displays a list of functions: open or remove episode. The episode can also be opened by a short click but if a user clicks too long he has the possibility to open it this way. More important is that a user can remove an episode with a long click on the episode and then select *Remove Episode*.

Menu: Menus are called when the menu button on the phone is pressed. These menus can also be used to add some functionality. In this application the menus are used to return to the previous view or to add new episodes or channels. In the view of the episode it is used for the rating and to view the content of the episode.

Dialog: The dialogs in this GUI are used to get the confirmation from the user if he wants to remove an episode or channel or if he wants to quit the episode.

3.5. Server

The episodes are published on a web server. It is published when it has several ratings and the ratings exceed a threshold. The web server is an Apache web server, on which a small PHP script is running, which receives the file and stores it in a folder. The file of the episode can then be downloaded.

4. Application Implementation

4.1. Requirements

The requirements for the implementation of the application are:

- Java SDK
- Android SDK
- Eclipse
- Android Development Tool (ADT) for Eclipse
- Android NDK
- Android Source Code and compiler
- Stlport library

Java SDK [18], Android SDK [19], Eclipse and the ADT [20] for Eclipse are used to implement a GUI Application. Eclipse [21] is a well known editor for Java. The ADT plug-in for Eclipse makes it much easier to develop an application. Android Source Code [22] includes the cross compiler, which is used to compile ANA and the required bricks such as vlink, PodNet, TCP, etc. The PodNet Interface code is written in C and needs to be accessed by the GUI, which is written in Java. In between there is JNI. The PodNet Interface C library and the JNI are compiled with the Android NDK [2]. In the sources of Android, the „Standard Template Library“ is not provided but it is required in PodNet. However, the stlport project [23] provides the code required for the Android platform.

4.2. Application

4.2.1. GUI

In a GUI there are several pages, whereas only one is displayed at the time. In Android such a page is called view. Each view is implemented in a Java class, which inherits from the „Activity“ or the „ListActivity“. The view is the graphic part whereas the activity is the whole code in the Java class. The view can be built in two different ways: The most common way is to implement the view in an XML file and the second

one is to implement it dynamically in the activity. The advantages and disadvantages of these two methods are described in section 2.5.1.2.
All the activities must be declared in the AndroidManifest.xml (see section F.1). Only this way Android has the knowledge of all the views. An activity is called active when the view is displayed. When an activity gets active, the *oncreate* function in the activity is called, which must be implemented in each activity. In this function it must be defined how the view should look like.
An activity is more then just the definition of the view. It also contains all the functionalities which are included in the view such as a click on a button. Therefore, for each button, an *OnClickListener* is defined, which is called when the user presses the button. The reaction to this click is implemented in this function such as to ask the user if he really wants to delete the episode or to show the next view. For the first case there is a dialog displayed (see section 3.4.1.2). For the second case there are *intents*.
Intents are used to get from one view to the next view. An intent sets the current activity inactive and stores the variables before the next one gets active. To return to a previous activity, the active activity needs to be closed. Using an intent, to return to the previous activity is not recommended because the previous and new activity would be the same but the environment variables would be stored multiple times. After some time the stack would get too big and the application would crash.
Each Android mobile phone has a menu button. If this button is pressed the menu which is defined for this view appears. This functionality is also implemented in an activity. The *onCreateOptionsMenu* function is called in this case. The menu displays items on which a user can click. The click makes that the *onOptionsItemSelected* function is called. For the simplicity there should not be too many options in the menu. Otherwise there is not enough space for all of them and a *More* button must be displayed. This is not very user-friendly.
The Java class for a view can inherit from „Activity“ or „ListActivity“. „ListActivity“ displays the content in a list. By default the input for the list is a string array. In this application there are two lists: a list of channels and one of the episodes. For example an episode has several attributes like episode name, file name, etc. but to display the list, only the episode name is needed. In order to fill in a list a so called adapter is used. Common adapters do not know which attribute of the episode should be displayed. One way to solve this problem is to get through the list episodes and create a string array which common adapters can handle. A nicer way is to implement an own array adapter. The adapters used in this application are called „ChannelArrayAdapter“ and „EpisodeArrayAdapter“. In these classes the output (channel or episode name) for the list view is implemented.

4.2.2. PodNet Interface

PodNet needs a new interface for C programs. This interface is used that other programs can use PodNet. The interface in PodNet runs its own thread. The thread is

started in podnet.c. The interface uses a Unix socket. Unix sockets are used to communicate between two processes in Unix. Messages are passed from one process to another one. The message passing is bidirectional this means that via the same socket both sides can send and receive messages.
When the tread starts the socket gets initiated. At first a sockets is created and second it needs to be bound to a file with the path which is set with the variable *SOCKET_PATH*. This file contains the message data which is passed from one program to the next one. If the file already exists the *bound* command will throw an error. Further both processes need read and write rights of the file otherwise the interface will not work. The *SOCKET_PATH* needs to be set to a folder where both programs have the read and write permissions.
After the correct setup of the interface on PodNet it only needs to listen to commands such as *channel_list* and execute them. This is done with a while loop. The function *recvfrom* waits until a message arrives at the socket. The command is received from the socket and executed in PodNet. The functions from the command line interface can be used for the socket interface with one simple addition. The default functions directly print the content in the shell. For the use of the functions in the socket interface this content needs to be sent back and not to be printed in the shell. Therefore in the functions *channel_list* and *episode_list* (datastore.cpp) the information is written into a string and finally sent back to the sender of the command. A function (*SendSocketMessage*), which returns the information is implemented in podnetMain.cpp.
The second part of the PodNet Interface is a library which a program can use to send a command to PodNet. In podnet_interface.c there is a function for *setupSocket*, *closeSocket* and for each command in PodNet. The *setupSocket* creates a socket and binds it. It is important that the variable *SOCKET_PATH* is identical to the variable in podnetMain.cpp in PodNet otherwise the two programs can not communicate. *closeSocket* is used to close the socket correctly otherwise the binding will not work at the next start of the interface.
After this two parts, an interface between PodNet and a program which uses the interface library exists. For the use of the library (implemented in C) in the application (implemented in Java) JNI is used. JNI makes it possible to call a function in C from Java. JNI is integrated in Android NDK. The C library is found in the *sources/podnet_interface* folder of Android NDK (How to install Android NDK etc. is described in appendix B). To use the library there needs to be the file jni-brick.c where the code for the JNI is.
In JNI the *OnLoad* function must be implemented.

```
JNIEXPORT jint JNICALL JNI_OnLoad(JavaVM * _jvm, void *
reserved) {
  return JNI_VERSION_1_4;
}
```

Here is an example of the implementation of a function from the PodNet interface. „jstring“ is a string for java and it is the return value. Further *Java_ana_brick_Podnet-*

NDK_listChannels(...) defines that the function *listChannels*() in Java is found in the class „PodnetNDK" which is stored in the folder */src/ana/brick* in the Android project. The function *listChannels*() in the second line calls the function in the C library.

```
/*
* Class:  ana_brick_PodnetNDK
* Method:  listChannels
* Signature:  (Ljava/lang/String;)I
*/
JNIEXPORT jstring JNICALL Java_ana_brick_PodnetNDK_list
  Channels(JNIEnv *env, jobject o) {
  const char* str = listChannels();
  return (*env)->NewStringUTF(env, str);
}
```

The JNI code and the C library are compiled in NDK. The output is a library which needs to be included in the project of the application in the *lib* folder. How to compile the library is explained in appendix B.

4.2.3. Rating System

In the application there is a rating system. The rating is done by sending SMS between the evaluator and the author of the episode (see section 3.3.3). For the rating a first SMS is sent to the author. The „SmsManager" class in Android handles the sending of the message. With the definition of a „PendingIntent" the sender is informed when the message is sent and delivered or when the sending or delivery failed. Sending an SMS needs the „SEND_SMS" permission, which is set in the AndroidManifest.xml.
Similar to the „SEND_SMS" the application needs also the „RECEIVE_SMS" permission, because the SMS from the evaluator is received at the author's mobile phone and is uses by the application. The receiving permission only gives the permission to receive but in order to process the incoming SMS the application needs a „BroadcastReceiver". The *onReceive* function of the „BroadcastReceiver" is called when an SMS arrives. In order to actually receive messages the „BroadcastReceiver" has to be declared in the AndroidManifest.xml.
The rating is finally implemented in the „SMSReceiver.java" class. The variable *MINIMUM_NUMBER_OF_RATINGS* defines how many ratings are at least needed that the file gets published on the server. Not only the number of ratings is important, the rating level is also important. The average rating level must be „Good" or higher for a publication in the Internet. The rating level can be set with the *AVERAGE_RATE_LEVEL*. Only if these two thresholds are exceeded the file of the episode will be published on the server.

4.3. Server

For the communication between the server and the mobile device there is code on the server and the mobile phone. The code to send the file and the code on the server to receive the file. The code in the application to send the file is in the „UploadFile“ class. The application opens a post request in HTTP to the server to upload the file. The post request sends the file to the PHP page (index.php) on the server. Index.php gets the file and stores it in the „podnet“ folder. For this action the server must have writing rights in this folder. By default it is not allowed to store files on a server which are received from the network. In appendix D it is explained how to set up the server.
Until the end of this master thesis, the application on the Nexus One has only one network connection at the time. Which means that the server must be in the same network as the Nexus One. This problem can be solved by implementing the Multi Switch Interface (see section 6.1). With the usage of the MIS the mobile device can have a connection to the server which is in the Internet to upload the files of the episodes and a WiFi ad-hoc connection to mobile devices to exchange the episodes.

5. Verification

For the test cases, which are described in the next section, there are a Nexus One, a HTC Hero and a Laptop with Ubuntu used. At the beginning of these tests there are ANA, PodNet and the required bricks installed on both devices. Additionally there is a shell script on the Nexus One called nexus_podnet.sh which starts and configures ANA, PodNet and the required bricks and a similar script called hero_podnet.sh for the HTC Hero and pc_podnet.sh used for the setup on the Ubuntu.
In the next section there are tests to validate the application. The application passes all these tests.

5.1. Test Cases

5.1.1. ANA and PodNet

Test Case In this test it should be verified if ANA and PodNet are running and if two PodNets on different devices communicate with each other.

Test Setup For the test of PodNet there is the command line interface of PotNet used.

1. Both devices (Nexus One and Laptop) should be connected to the same network (start the minmex in a shell and then execute the two scripts nexus_podnet.sh and pc_podnet.sh) in a second shell
2. Create on both devices the „football" channel (*channel_add football football*).
3. Get the channel number of the „football" channel on the Ubuntu (*channel_list*), it is a three digit number
4. Add a new episode in the „football" channel on Ubuntu (*episode_add <channel number> Episode*1 *<phone number> <file path>*)
5. Get the channel number of the „football" channel on the Nexus One (*channel_list*)
6. List all episodes of the „football" on the Nexus One (*episode_list <channel number>*)

Observed Result In the adb shell the output contains these three lines:

```
podnet:  NOTIFICATION - event stable(VAL) neighbor=XYZ
  linkQ=100 context=MAC timestamp=0
PodNet:  Opening connection to XYZ with result 0 and
  timestamp=0
podnet:  recv message ...ntf(NUM)
```

If these lines lines appear periodically, the PodNets on the Nexus One and Ubuntu are connected and the episode should be exchanged.
For the verification if the episode is really transmitted, the output of the last command on the Nexus One (*episode_list* ...) should return the episode information.

Possible Errors

- There can be the problem that the command line interface is not running then the commands are unknown to the shell → check if the command line interface is compiled in PodNet (*/ana−core/devel/C/bricks/dtn/podnet/podnet.c*)
- The command line interface is compiled but the commands are not accepted → start the minmex in one shell and execute the script in a second shell to use the command line interface.
- It can be that the connection between the two devices is up and the episodes are exchanged but after several seconds or minutes the connection breaks. The problem occurs on the Android phone. The connection between the PodNet and the IMF brick breaks and also the connection to the other ANA nodes. The reason why this problem occurs is not determined yet. Because this problem only occurs on the Android phone and not between the ANA nodes on two Ubuntu devices, maybe Android has an influence on PodNet. Perhaps Android give less computing time to the application process or to the different threads of the application.

5.1.2. Communication between PodnetApp and PodNet

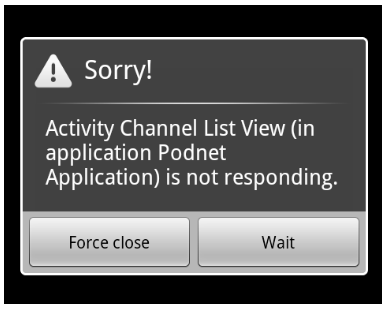
Test Case The PodnetApp uses PodNet and ANA. The communication between the application and PodNet is done via the podnet-interface library, which is integrated in the application and implemented in PodNet. This test is about the communication between the application and PodNet.

Test Setup At the beginning PodNet and ANA are running and the application is compiled and copied to the Nexus One.

1. Start the application (click on the application)
2. Click on *Channel List View*

Observed Results The channel list view can be empty or contains some channel names.

Possible Errors The channel list does not appear and after some seconds there is an error message (see figure 5.1) displayed. This means that there is no connection to PodNet. The problem can be that *SOCKET_PATH* is not the same in the podnet-interface library and in PodNet.



Sorry!

Activity Channel List View (in application Podnet Application) is not responding.

Figure 5.1.: Error Message: Communication to PodNet is broken

5.1.3. Add Channel

Test Case The test case is to add a new channel in the Application.

Test Setup ANA, PodNet and the required bricks are installed on the Nexus One and running. Further the PodnetApp is also installed on the Nexus One.

1. Start the application
2. Click on *Channel List View*
3. Click on the *Menu* button on the phone
4. Select *AddChannel*
5. Enter a channel name which does not contain any space, backslash, quote and number sign characters
6. Click *Add*

Observed Results At the end of the setup the channel should be displayed on the channel list otherwise there is an error.

Possible Errors

- Channel list is not displayed and one get an error message → see section 5.1.2
- The menu in the channel list view is not displayed or there is no action when *AddChannel* is selected → see section 5.1.7

5.1.4. Add Episode

Test Case The test creates a new episode in an existing channel.

Test Setup ANA, PodNet and the required bricks are installed on the Nexus One and running. Further the PodnetApp is also installed on the Nexus One and the channel „football" is already created in PodNet.

1. Start the application
2. Click on *Channel List View*
3. Click on *football*
4. Click on the *Menu* button on the phone
5. Select *AddEpisode*
6. Enter an episode name which does not contain any space, backslash, quote and number sign characters
7. Add the file path to attach a file to the episode
8. Click *Add*

Observed Results If all the inputs are correct the episode is created and displayed on the episode list.

Possible Errors

- Channel list is not displayed and one get an error message → see section 5.1.2
- The menu in the episode list view is not displayed or there is no action when *AddEpisode* is selected → see section 5.1.7
- After pressing *Add*, the settings view is called → the phone number of the user is missing, which is used to create properly an episode, save the settings to return to the view and add the episode
- After pressing *Add* an error message is displayed (see figure 3.4) that the file does not exist → check if the file really exists and if the path is entered correctly

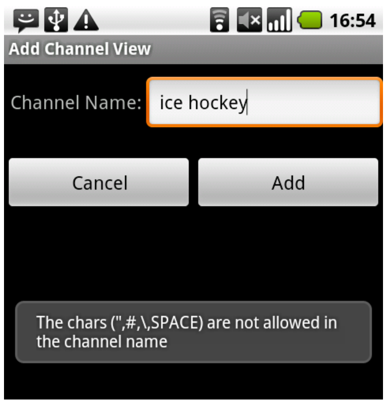
5.1.5. Channel Name or Episode Name Validation

Test Case For channel names and episode names not all characters are allowed. Space, backslash, quote and number sign characters are not allowed. This test checks if there is an error message displayed when a channel name or episode name contains one of these characters.

Test Setup ANA, PodNet and the required bricks are installed on the Nexus One and running. Further the PodnetApp is also installed on the Nexus One. The test is done for the channel name because the validation of the episode name is done in the same function.

1. Start the application
2. Click on *Channel List View*
3. Click on the *Menu* button on the phone
4. Select *AddChannel*
5. Enter a channel name which contains one of the following characters: space, backslash, quote and number sign
6. Click *Add*

Observed Results As long as one of the characters (space, backslash, quote and number sign) is in the channel name, the channel should not be created and the error message (see figure 5.2) should be displayed. The test for the episode name can



ice hockey

The chars (",#,\,SPACE) are not allowed in the channel name

Figure 5.2.: Error Message: Usage of forbidden Characters

be done like in section 5.1.4 but enter this time the forbidden characters. The error messages should be the similar.

Possible Errors

- Channel list is not displayed and one get an error message $\rightarrow$ see section 5.1.2
- The menu in the channel list view is not displayed or there is no action when *Add Channel* is selected $\rightarrow$ see section 5.1.7
- The channel name contains a forbidden character but the channel or episode is created $\rightarrow$ the validation function *validateString* in the class „Helper“ is not correctly implemented

5.1.6. Rating and Server

Test Case A user rates an episode and if the rating exceeds a number of ratings and the average rating level of the episode, the file should be published on a server.

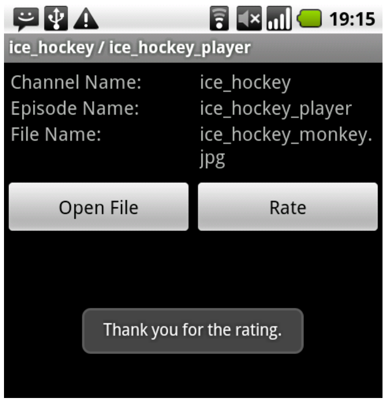
Test Setup Two Android phone which run ANA, PodNet and the PodnetApp. Further there is a web server in the same network.

On the first Android Phone:

1. Start PodNetApp
2. Click on *Settings*
3. Set the IP address of the server or the domain name
4. Save the settings
5. Create an episode in the *football* channel

On the second Android Phone:

1. Start PodNetApp
2. Open the *football* channel or create the *football* channel
3. Open the episode which was created on the first phone
4. Select *Rate*
5. Select a rate which is better or equal than „Good“
6. Agree to the question to send the rating
7. A message, which indicates that the rating is sent, is displayed (see figure 5.3)


19:15

ice_hockey / ice_hockey_player

Channel Name: ice_hockey
Episode Name: ice_hockey_player
File Name: ice_hockey_monkey.jpg

Thank you for the rating.

Figure 5.3.: Rating is sent to the Author of the Episode

Observed Results Three SMS are sent: two form the phone which rates the episode and one from the phone on which the episode was created. After these SMS arrived, the episode should be published on the web site. Therefore the page needs to be opened with a web browser and the file can be downloaded from the page.

Possible Errors

- Channel list is not displayed but an error message is $\rightarrow$ see section 5.1.2
- The menu in the channel list view is not displayed or there is no action when *AddChannel* is selected $\rightarrow$ see section 5.1.7
- The episode is not transmitted to the second phone $\rightarrow$ see section 5.1.1
- After clicking on the *Rate* button nothing happens $\rightarrow$ the dialog is not correctly implemented
- After selecting the rate level there is no question if the rating should be sent $\rightarrow$ the dialog with the question is not correctly implemented
- The message that the rating is sent does not appear $\rightarrow$ the message is not implemented
- The first SMS is not sent $\rightarrow$ the application does not have the sending permission of an SMS
- The second SMS is not sent $\rightarrow$ the application does not have the receiving permission of an SMS
- The file is not on the server $\rightarrow$ the average rating level is lower than „Good“, the IP address is wrong, the server is not correctly implemented (see section D) or there is no Internet connection

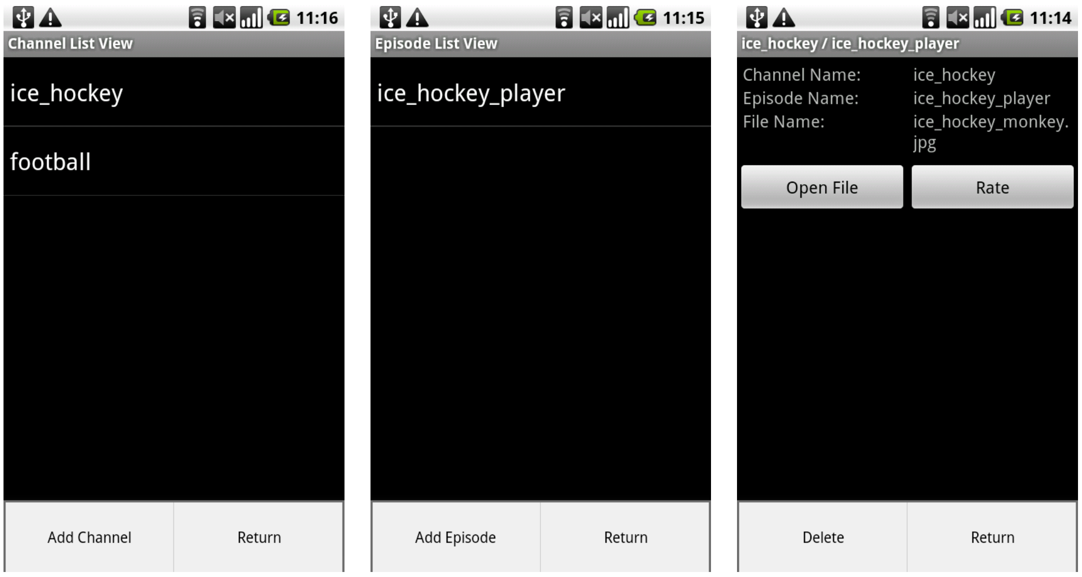
5.1.7. Menu

Test Case Testing the menus of the *Channel List View*, *Episode List View* and the *Episode View*.

Test Setup ANA, PodNet and the required bricks are installed on the Nexus One and running. Further the PodnetApp is also installed on the Nexus One and a channel is added. The test is the same for all these views. Therefore it is only described for the *Channel List View*.

1. Start PodNetApp
2. Click on *ChannelList*
3. Press the *Menu* button on the mobile phone
4. Select an item

Observed Results The menu should be displayed. In figure 5.4 there are the views of all the three views. In case of pressing *Return* it should return to the previous view.



Channel Name:	ice_hockey
Episode Name:	ice_hockey_player
File Name:	ice_hockey_monkey.jpg

(a) Channel List View Menu (b) Episode List View Menu (c) Episode View Menu

Figure 5.4.: Menu of Views

Possible Errors

- The menu is not displayed after the button is pressed → the *onCreateOptions-Menu* is not correctly implemented
- Nothing happens after selecting an item → the *onOptionsItemSelected* is not correctly implemented

5.1.8. Long Clicks

Test Case Long clicks are used to add some more functionalities to the *Channel List View* and the *Episode List View* such as to remove the episode or channel.

Test Setup ANA, PodNet and the required bricks are installed on the Nexus One and running. Further the PodnetApp is also installed on the Nexus One and a channel and an episode are added. The test is the same for both views. Therefore it is only described for the *Channel List View*.

1. Start PodNetApp
2. Click on *ChannelList*
3. Long click (about one second) on a channel name
4. A pop-up select an action in the pop-up list

Observed Results The list (see figure 5.5) is shown and an item can be selected such as to remove the channel or to open it.

Possible Errors

- Channel list is not displayed but an error message is → see section 5.1.2
- The action list is not displayed after the long click → the *onItemLongClickListener* is not implemented or declared in the *onCreate* function.
- After selecting an action, nothing happens → the *onClickListener* of the „DialogInterface“ class is not implemented.

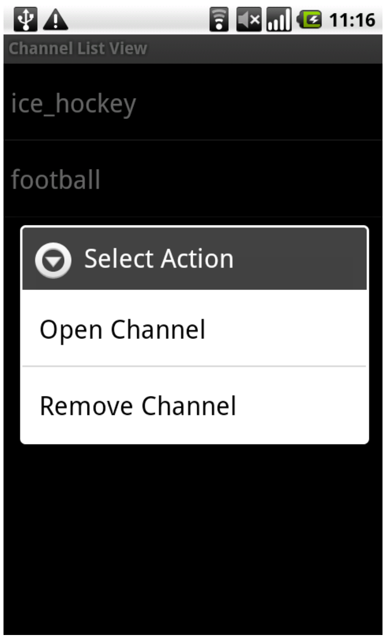


Figure 5.5.: Action List after a Long Click on the Channel Name

6. Towards the Multi Interface Switch

Today's mobile devices can be connected to the Internet in two ways: 3G or WiFi in infrastructure mode. But the WiFi card can also be configured to communicate via WiFi in ad-hoc mode. The device is connected only to one of those three networks at the time. The switching between 3G and WiFi in infrastructure mode is done by Android if it detects an open WiFi network or a network to which it has the key. In our application the communication between two mobile devices to share episodes is done via a WiFi ad-hoc connection. The communication between the mobile device to the web server should be a 3G or a WiFi in infrastructure mode connection. A switch, which could switch automatically from one network connection to another and could store the previous settings, to get back to the previous network later, would give the application the possibility to be connected to all the networks. The application hands over a packet to send it to the Internet or the next mobile device and the switch forwards the packet when it is connected to this specific network.

6.1. Multi Interface Switch

The Multi Interface Switch (MIS) [25] must be implemented in ANA as a brick. This brick can then be used by other bricks and programs such as an application (see figure 6.1). The protocol stack is not set up by the MIS. MIS sets up networks con-

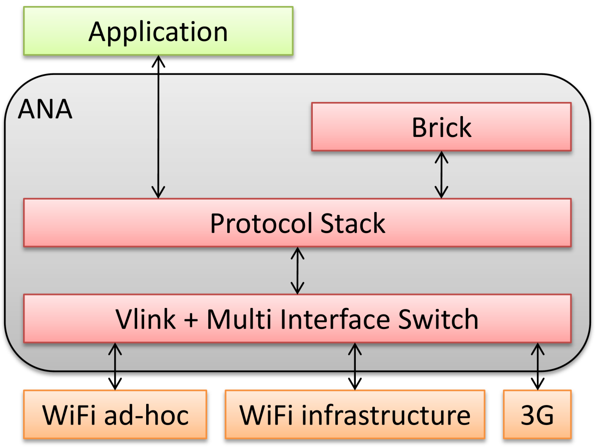


Figure 6.1.: Multi Interface Switch: Protocol Stack

nections, switches between them and forwards the data packets. The best way to implement the MIS is to include the MIS in the existing Vlink. Vlink has a connection to the Ethernet interface.

6.1.1. Implementation

For the implementation of the MIS, the MIS should store all the settings and the active status of the network. This is necessary, otherwise the network interface sets up a new connection every time the MIS is connected to the network and the mobile device might get a new IP address and the communication between the devices needs to be established again.
The MIS also uses a buffer for each network connection (3G, WiFi in infrastructure mode and WiFi in ad-hoc mode). A data packet must be stored in the buffer when the MIS is connected to a different network than the data packet must be delivered. The packet is sent when the MIS is connected to the designated network.
In TCP each data packet generates an ACK which is sent back to the source of the data packet. When a TCP packet is sent a timer for this packet is set. If the ACK arrives before the timer triggers everything is fine. If it arrives late and the timer triggers the packet can be sent again or TCP informs the creator brick of the packet that the connection is broken. Because MIS stores the packets until it is connected to the correct network, it delays the packet which enlarges the Round Trip Time [1] (RTT). The RTT is used to set the timeout for a timer.
For a good performance it is important to have an optimal scheduling between the different networks otherwise the bandwidth is not used optimal and a lot of bandwidth is unused.

6.1.2. Network Connections on Android

The setup for a network connection can be dependent on the mobile device. It is different for the HTC Hero and the Nexus One.

HTC Hero For the HTC Hero there are configuration files tiwlan0.ini and wpa_supplicant.conf for the setup of a network connection. The WiFi ad-hoc connection can be setup by disabling the network interface, changing these two configuration files and then starting the network interface again. If the setup of the network connection is done programmatically the configuration files must be copied to the application which sets up the network connection.

Nexus One Setup For the Nexus One there are no configuration files. For the network connection the ifconfig and netcfg unit shell commands can be used. To set up an ad-hoc connection the iwconfig Unix shell command is needed. By default there

[1] Round Trip Time: The time which passes until the ACK returns

is no library which provides this command in Android but there is the $iwconfig$ library in the Android Tether application [26] which can handle this Unix shell commands. The commands to set up an ad-hoc WiFi connection are:

```
netcfg eth0 down
iwconfig eth0 mode ad-hoc
iwconfig eth0 essid NETWORKNAME
ifconfig eth0 IP netmask NETMASK
netcfg eth0 up
```

The setup on the HTC Hero only worked manually but not programmatically. Whereas the setup on the Nexus One works properly manually. For the MIS it is better to use the Nexus One than then the HTC Hero because HTC Hero could not be setup programmatically.

7. Summary

7.1. Summary

ANA and some bricks such as vlink, eth-vl, PodNet, etc. are ported to run on Android. The Android cross compiler and the stlport library were needed to compile ANA and the bricks for Android. Further a new ROM image was installed that provides root access on the Android phone to run ANA because ANA needs root rights.
The PodnetApp is designed and implemented for Android to use PodNet and ANA. Additionally the application is equipped with a rating system which allows to rate episodes and the possibility to publish episodes with a high rating on a web server. For the communication between the application and PodNet the PodNet Interface needed to be implemented as a library which is integrated in a JNI and the interface on PodNet.
The application works only if the mobile devices and the server are in the same network. The implementation of the Multi Interface Switch would allow the application to have a connection to the Internet server to publish the episodes and a connection to all the mobile devices to share episodes with them.

7.2. Conclusion

The cross compiler uses the libraries from the Android source. Libraries which are not used in Android need to be added manually. To compile PodNet the STL library is used which is not in the Android Source. The stlport library provides the STL for Android.
The implementation of the switch depends on the kernel of the Android phone. The kernel dependents on mobile device producers. The kernel of the HTC Hero is different to the kernel in the Nexus One. Modifications in the kernel have an impact on the setup of a network interface. For the Nexus One there is only an additional library used to setup a WiFi ad-hoc connection programmatically. In the HTC Hero there are files which need to be modified and loaded to setup a WiFi connection programmatically. Finally it is easier to setup a WiFi connection in the Nexus One than in the HTC Hero. The application works stable on ANA and PodNet. PodnetApp supports to create channels and episodes and allows to open the files from the episodes. The rating of the episodes works as well as the publishing of the episode files on the server. The application provides a lot of validations such as file path, episode and channel names, etc. Episodes are exchanged via a WiFi in ad-hoc mode connection. Finally

PodnetApp passes all the verification tests.

7.3. Future Work

For future work the Multi Interface Switch needs to be implemented which can be used by all bricks of ANA. This gives the possibility to communicate to devices in different networks without changing the network settings manually. In the application this switch can be used to setup an Internet connection (3G or WiFi in infrastructure mode) to publish the episodes and to setup a WiFi in ad-hoc mode connection to exchange episodes with other mobile devices.
As described in the summery section (see section 7.1) the communication between PodNet and the IMF brick breaks after a few seconds or minutes. This only happens on a Android device and not on a laptop with Ubuntu running ANA and PodNet. This problem causes that PodNet on the Android phone can not exchange episodes with other devices anymore. Further investigations on this problem are needed that the PodnetApp can run properly.

A. How to run ANA and PodNet on Android

This tutorial explains how to run ANA and PodNet on the Nexus One Android phone.

1. In ANA Core Documentation [27] in section 1.1 it is described how to download ANA
2. Source of a Linux kernel must be compiled (tutorial is found here [28])
3. Android Source is used to cross compile ANA for Android (source and tutorial can be found here [22])
4. Stlport library needs to be installed to compile PodNet for Android (tutorial can be found here [23])
5. The path of the compilers and libraries in $/ana-core/devel/config.txt$ and in other Makefiles are needed to be adapted to your environment.
6. Execute the command *make android* to compile ANA for Android
7. Install a new ROM Image of the Android phone to get root access (for the Nexus one this page can help: [29])
8. The Android SDK needs to be installed (tutorial can be found here [30])
9. The files in the $/ana-core/devel/bin$ and $/ana-core/devel/so$ must be copied to sdcard of the Android phone. This can be done by including the SD card as an external drive or using the *adb push* command which can be executed in the $/ANDROID\_SDK/tools$ folder.
10. Installing the busybox on the Android which gives the user more commands such as copy (the tutorial and the link to the busybox is found here: [31])
11. Open a shell on the Android phone ($/ANDROID\_SDK/tools/adb\ shell$)
12. Create a the folders *bin* and *bricks* at $/data/local/ANA$
13. Define the base folder of ANA: $export\ ANA\_BASE\_DIR = /data/local/ANA$ on the phone

14. Copy $minmex\ vlconfig$ and $mxconfig$ to the bin folder, the bricks to the $bricks$ folder and the nexus_podnet.sh and podnet_start.sh scripts to the base folder of ANA

15. Start the minmex $/data/local/ANA/bin/minmex/$

16. Open a second shell and start the nexus_podnet.sh script ($sh\ nexus\_podnet.sh$)

17. PodNet and ANA are running

B. PodNet Interface

For the PodNet Interface there is the Android SDK and Android NDK needed to be installed.

1. Install Android SDK [30] and Android NDK [2]
2. Copy the folders in $AndroidNDK/apps$ to the $apps$ folder of the Android NDK and the $AndroidNDK/sources$ to the $sources$ folder of the Android NDK
3. Go to the the root folder of the Android NDK and enter the command $make\ APP = podnet-interface$ to compile the PodNet Interface library
4. Copy the files of $/apps/podnet-interface/project/libs/armeabi/libpodnet--interface.so$ to the $/libs/armeabi$ folder of PodnetApp

C. How to run the Application on Android

For the Application to run there are Android SDK and Java and Eclipse and the ADT plug-in for Eclipse needed to be installed.

1. Install Android SDK, Java, Eclipse and the ADT (link and tutorial can be found here [30])
2. Import the PodnetApp in Eclipse
3. Connect the Android phone with the USB cable to the laptop
4. Run PodnetApp in Eclipse

PodnetApp is automatically started on the Android phone. To use the application ANA and PodNet needs to be started first otherwise the application crashes with a click on the *ChannelListView*. See appendix A

D. Run Web Server

The operating system on which the web server will run is Ubuntu and the web server is from apache.

1. Install the apache web server ($sudo\ apt-get\ install\ apache2$), PHP ($sudo\ apt-get\ install\ php5$) and libapache2-mod-php5 ($sudo\ apt-get\ install\ libapache2-mod-php5$)

2. Open the file $default$ in the folder $/etc/apache2/sites-available$

3. Add the following code in the $<VirtualHost * : 80>$ this gives the server the rights to create files in the podnet folder

```
<Directory /var/www/podnet>
  Options Indexes FollowSymLinks MultiViews
  AllowOverride All
  Order allow,deny
  allow from all
</Directory>
```

4. Create the folder $podnet$ in $/var/www/$

5. Create the file index.php in $/var/www$ with the code: (the original code [32] is used and adapted for this needs)

```
<?php
$target_path = "podnet/";
$target_path = $target_path . basename( $_FILES['-
  uploadedfile']['name']);
if(move_uploaded_file($_FILES['uploadedfile']['tmp-
  _name'], $target_path)) {
  echo "The file ". basename( $_FILES['uploaded-
    file'] ['name']). " has been uploaded";
} else {
  echo "There was an error uploading the file,
    please try again!";
}
?>
```

6. Restart the web server ($/etc/init.d/apache2\ restart$)

7. Open the page $127.0.0.1/index.php$ it should display: “There was an error uploading the file, please try again!“ Otherwise PHP is not correctly installed.

E. Folder structure of the Android Application

Project Path: $/ana-core/devel/Android/apps/PodnetApp$

```
- AndroidManifest.xml
  (declaration of Permissions, Activities)
+ bin
  - ... (contains .class files)
- default.properties
+ gen
  - ... (auto generated file)
+ libs
  + armeabi (external libraries)
    - libana-app.so
    - libpodnet-interface.so
+ res
  + drawable
    - icon.png (application logo)
  + layout (view layouts)
    - layout-addchannel.xml
    - layout-addepisode.xml
    - layout-settings.xml
    - layout-viewepisode.xml
    - layout-main.xml
  + menu (layout for menus)
    - channellist.xml
    - episodelist.xml
    - episode.xml
  + values (strings, styles, ... in english)
    - strings.xml
  + values-de (stings, styles, ... in german)
    - strings.xml
+ src
  + ana
    + brick
    - DataHandler.java
    - JavaBrick.java (Java class for ANA-API)
    - PodnetNDK.java (Java class for Podnet-Interface)
  + ch/ethz/tik/ana (main application folder)
    + arrayadapter (array adapters for channel and episode)
      - ChanelArrayAdapter.java
      - EpisodeArrayAdapter.java
    - FileUpload.java (upload episode file to server)
    - Helper.java (validation of strings, copy of files etc.)
- PodnetInterface.java (create channel and episode lists of information from PodNet)
+ properties
   - PropertyHandler.java (read and write properties)
+ sms
   - SMSReceiver.java (receive SMS, save rating)
+ type
   - Channel.java
   - Episode.java
   - Rating.java
+ view (application views)
   - AddChannelView.java (validate and create channel)
   - AddEpisodeView.java (validate and create episode)
   - ChannelListView.java (list channels)
   - EpisodeListView.java (list episodes)
   - EpisodeView.java (list attributes of episode, open episode, send rating)
   - MainView.java (starting view of application)
   - SettingsView.java (set and save settings)
```

F. Code

F.1. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ch.ethz.tik.ana"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity
            android:name="ch.ethz.tik.ana.view.MainView"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity
            android:name="ch.ethz.tik.ana.view.ChannelListView"
            android:label="@string/view_channel_list">
        </activity>
        <activity
            android:name="ch.ethz.tik.ana.view.EpisodeListView"
            android:label="@string/view_episode_list">
        </activity>
        <activity
            android:name="ch.ethz.tik.ana.view.EpisodeView">
        </activity>
        <activity
            android:label="@string/view_channel_add"
            android:name="ch.ethz.tik.ana.view.AddChannelView">
        </activity>
        <activity
            android:name="ch.ethz.tik.ana.view.AddEpisodeView"
            android:label="@string/view_episode_add">
            </activity>
        <activity
            android:label="@string/view_settings"
            android:name="ch.ethz.tik.ana.view.SettingsView">
        </activity>
        <receiver android:name="ch.ethz.tik.ana.sms.SMSReceiver">
            <intent-filter>
```

```
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
    </application>

    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

G. Time Schedule

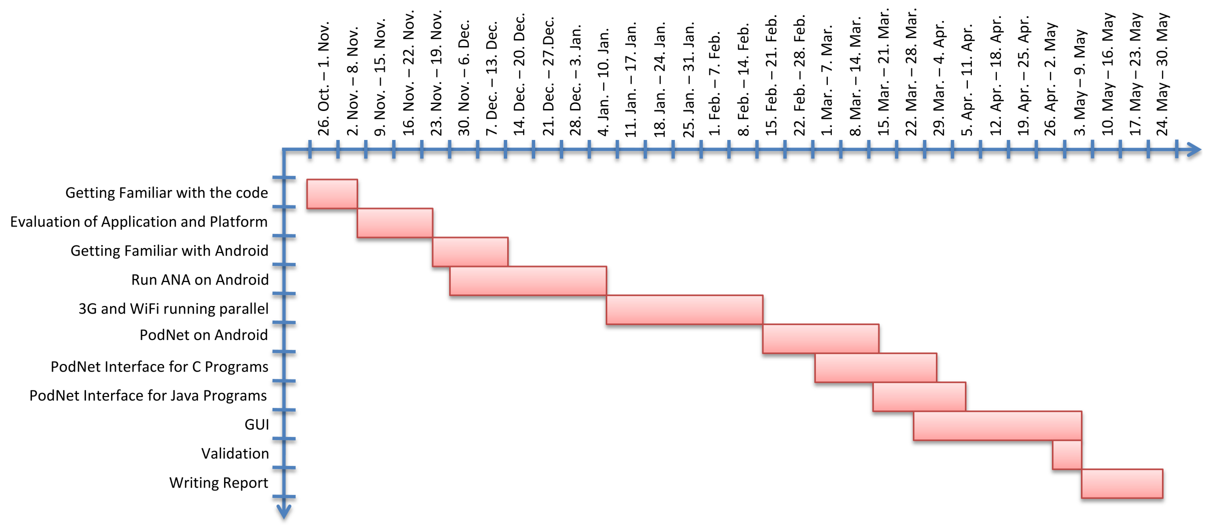


Figure G.1.: Time Schedule

Bibliography

[1] ANA: http://www.ana-project.org/, 11. May 2010

[2] Android NDK: http://developer.android.com/sdk/ndk/index.html, 6. May 2010

[3] UMTS: http://www.etsi.org/WebSite/Technologies/UMTS.aspx, 11. May 2010

[4] WCDMA: http://www.etsi.org/WebSite/Technologies/wcdma.aspx, 11. May 2010

[5] C. Smith and D. Collins, *3G wireless networks*. McGraw-Hill communications, pub-MCGRAW-HILL:adr: McGraw-Hill, second ed., 2007.

[6] GSM: http://www.etsi.org/WebSite/Technologies/gsm.aspx, 11. May 2010

[7] GPRS: http://www.etsi.org/WebSite/Technologies/gprs.aspx, 11. May 2010

[8] EDGE: http://www.etsi.org/WebSite/technologies/edge.aspx, 11. May 2010

[9] ETSI: http://www.etsi.org/WebSite/homepage.aspx, 11. May 2010

[10] 3GPP Release 1999: http://www.3gpp.org/article/release-1999, 11. May 2010

[11] 3GPP: http://www.3gpp.org/, 11. May 2010

[12] C.-L. Tsao and R. Sivakumar, „On effectively exploiting multiple wireless interfaces in mobile hosts“, in *CoNEXT 09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, (New York, NY, USA), pp. 337-348, ACM, 2009.

[13] Open Handset Alliance: http://www.openhandsetalliance.com/, 7. May 2010

[14] Apache License: http://www.opensource.org/licenses/apache2.0.php, 7. May 2010

[15] Bluetooth: http://www.bluetooth.com/English/Pages/default.aspx, 11. May 2010

[16] WiFi: http://grouper.ieee.org/groups/802/11/, 11. May 2010

[17] Data throughput of WiFi and Bluetooth: http://focus.ti.com/pdfs/vf/bband/coexistence.pdf, 11. May 2010

[18] Java SDK: http://java.sun.com/javase/downloads/index.jsp, 11. May 2010

[19] Android SDK: http://developer.android.com/sdk/index.html, 11. May 2010

[20] ADT for Eclipse: http://developer.android.com/sdk/eclipse-adt.html, 11. May 2010

[21] Eclipse: http://www.eclipse.org/, 7. May 2010

[22] Android source code: http://source.android.com/source/download.html, 7. May 2010

[23] stlport library:
http://tungchingkai.blogspot.com/2009/10/android-ndk-add-stlport-and-compile-c.html, 9. March 2010

[24] Cross Compiler Tutorial:
http://android-dls.com/wiki/index.php?title=Compiling_for_Android, 7. May 2010

[25] MultiNet: http://research.microsoft.com/en-us/um/redmond/projects/virtualwifi/multinet_infocom.pdf, 14. May 2010

[26] Tethering Code:
http://code.google.com/p/android-wifi-tether/, 14. May 2010

[27] ANA Core Documentation: http://www.ana-project.org/deliverables/2008/ana-d1.11-final.pdf, 23. Mai 2010

[28] How to compile source of Linux kernel: http://www.howtoforge.com/kernel_compilation_ubuntu, 27. November 2010

[29] Tutorial get root access on Nexus One: http://forum.xda-developers.com/showthread.php?t=612858, 9. March 2010

[30] Installing instructions for developing an Android Application: http://developer.android.com/sdk/installing.html, 23. Mai 2010

[31] Busybox: http://benno.id.au/blog/2007/11/14/android-busybox, 9. March 2010

[32] Code File Upload:
http://getablogger.blogspot.com/2008/01/android-how-to-post-file-to-php-server.html, 10. May 2010