

Abstract

The main objective of this work is to analyse the retransmission mechanisms of both RLP (Radio Link Protocol) and RLC (Radio Link Control) and to investigate their influence on the IP layer when accessing the Internet. RLP is used in CDMA2000 (Code Division Multiple Access) and RLC in GPRS/GSM (General Packet Radio Service/ Global System for Mobile Telecommunication).

The primary function of RLP and RLC is to reduce the over-the-air frame error rate to a tolerable level. Since both protocols provide an in-sequence delivery of data to upper layers, retransmissions of lost frames or frames received in error cause delayed delivery of correctly received frames and therefore a delay on upper layers. This delay distribution, the lowered bitrate and the minimized drop rate due to the RLP and RLC retransmission mechanisms are analytically derived in this work.

To calculate these characteristics of the air interface, we developed an application with a graphical user interface, where the required values such as roundtrip time, frame error rate, desired bitrate, etc. can be set for CDMA2000 and GPRS/GSM.

To illustrate the behaviour of IP over CDMA2000 and GPRS/GSM, we built up an emulation network including client mobile station, emulator and proxy server, which is connected with the Internet. So the air interface for both mobile network architectures can be emulated by simply pressing a button in our application.

1	<i>Project Management</i>	4
1.1	Introduction.....	4
1.2	Main Work.....	4
1.3	Time Schedule (Target/Actual).....	5
1.4	Phase Diagram / Conceptual Formulation.....	6
2	<i>General Introduction</i>	8
2.1	Introduction.....	8
2.2	The mobile radio channel.....	8
2.3	Transmission protocols	8
2.4	Problem.....	8
2.5	CDMA2000.....	8
2.6	GPRS.....	10
2.7	UMTS.....	11
3	<i>Experimental Setup</i>	12
3.1	Introduction.....	12
3.2	Network.....	12
3.3	Client 1	12
3.3.1	Introduction.....	12
3.3.2	Operating System.....	12
3.4	Client 2	13
3.4.1	Introduction.....	13
3.4.2	Operating System.....	13
3.4.3	Extension.....	13
3.5	Basestation	13
3.5.1	Introduction.....	13
3.6	Emulator.....	13
3.6.1	Introduction.....	13
3.6.2	Operating System.....	13
3.6.3	Routing.....	13
3.6.4	NistNet.....	14
3.7	Server	15
3.7.1	Introduction.....	15
3.7.2	Operating System.....	15
3.7.3	Routing.....	16
3.7.4	Apache running as Proxy-Server	16
3.7.5	Ethereal Network Analyzer.....	17
3.7.6	Webswift.....	17
4	<i>Radio Link Protocol (RLP)</i>	18
4.1	Introduction.....	18
4.2	How does RLP operate ?.....	19
4.3	Effective bit rate for IP over RLP	21
4.4	RLP retransmission process as a decision tree.....	24
4.5	IP datagram loss rate over RLP.....	27
4.6	IP datagram delay over RLP	29
4.7	Traffic Channels.....	33
4.8	Summary	35
4.8.1	IP Bit Rate.....	35
4.8.2	IP Loss Rate	35
4.8.3	IP Delay and Standard Deviation.....	35

5	<i>Radio Link Control (RLC)</i>	36
5.1	Introduction.....	36
5.2	Segmentation.....	36
5.3	Brief description of RLC data transmission.....	37
5.4	RLC/MAC block structure.....	37
5.5	Procedures and Parameters.....	38
5.6	Assumptions.....	40
5.7	Number of RLC blocks per IP datagram.....	41
5.8	Effective bit rate for IP over RLC.....	42
5.9	IP datagram loss rate over RLC.....	45
5.10	IP datagram delay over RLC.....	48
5.11	Summary.....	55
5.11.1	IP Bit Rate.....	55
5.11.2	IP Loss Rate.....	55
5.11.3	IP datagram delay over RLC.....	55
6	<i>Software Engineering</i>	56
6.1	Introduction.....	56
6.2	Development Environment.....	56
6.3	Shell Script.....	56
6.4	Graphical User Interface (GUI).....	57
6.4.1	Appliance.....	57
6.4.2	Parameters for CDMA2000.....	58
6.4.3	Parameters for GPRS/GSM.....	58
6.5	Class Diagram.....	58
6.6	Calling NistNet and Perl from Java.....	59
7	<i>Closing Words</i>	61
8	<i>References</i>	62
9	<i>Register of Illustrations</i>	63
10	<i>Acronyms</i>	64
A	<i>Java Files</i>	65
B	<i>Matlab Files</i>	80

1 Project Management

1.1 Introduction

This chapter shall contain an exact description of this work and in order to keep track of our project, we created a time schedule, which shows project progress and deviations between actual and target time. Every part of this work shall be described as precise as possible with the help of a phase diagram.

1.2 Main Work

The goal of this work is to have a network with a tool, which allows to emulate the radio link of both CDMA2000 and GPRS/GSM. This work consists of five main parts, which are briefly described below.

- ❑ Draw up project documentation in English.
- ❑ Rebuild existing network from last work.
- ❑ Analyse two mobile data network architectures CDMA2000 and GPRS/GSM and their behaviour in packet oriented data transport over TCP/IP. Analyse their radio link protocols RLP and RLC.
- ❑ Develop an application with a graphical user interface based on the network emulating tool NistNet. It shall calculate the parameters for NistNet due to the RLP and RLC retransmission characteristics.
- ❑ Present work and show results. At the end of the work a placard has to be designed which shows our work in brief.

Please refer to the subchapters Time Schedule and Phase Diagram / Conceptual Formulation for detailed description of our work.

1.3 Time Schedule (Target/Actual)

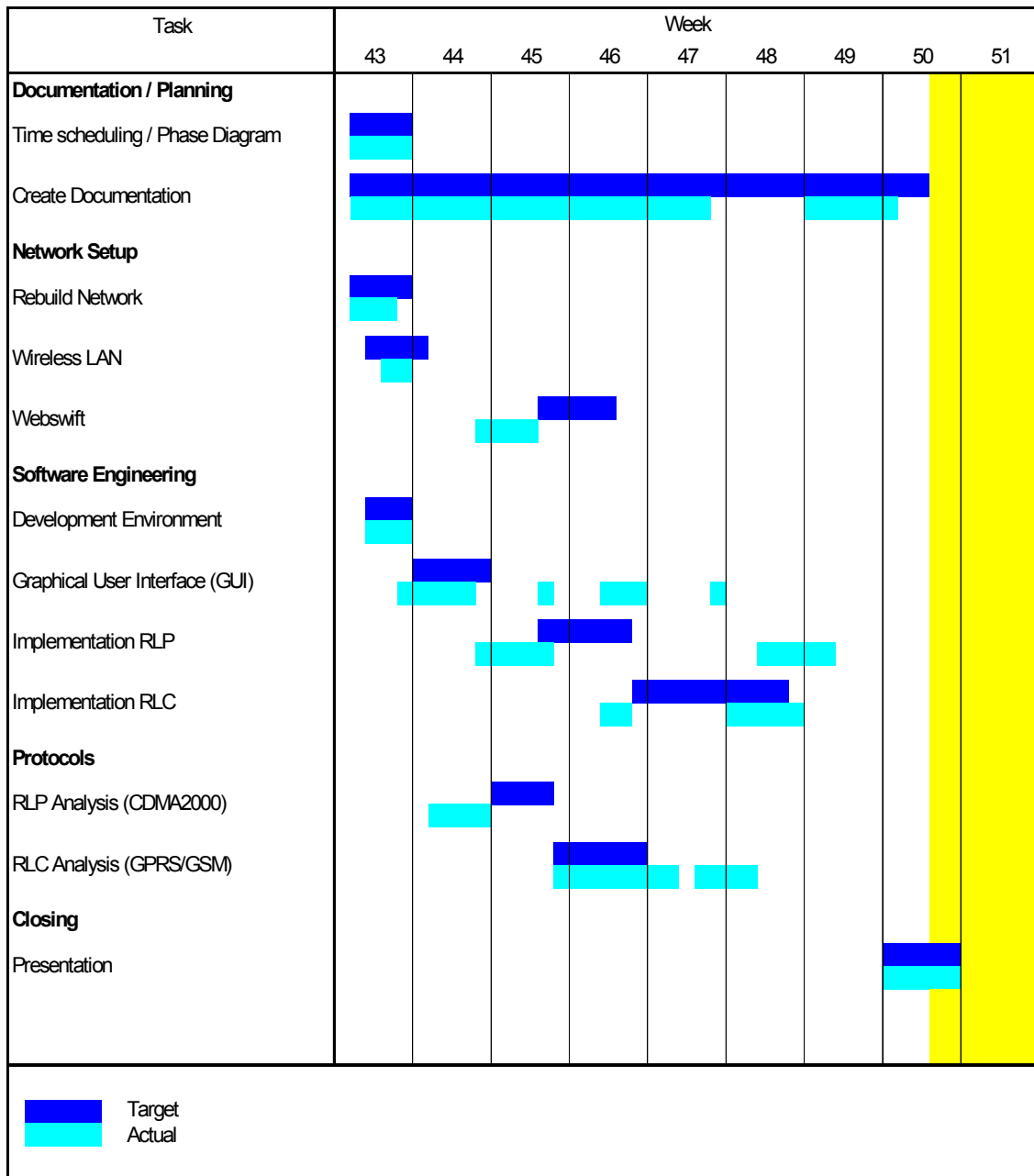


Figure 1: Time Schedule

1.4 Phase Diagram / Conceptual Formulation

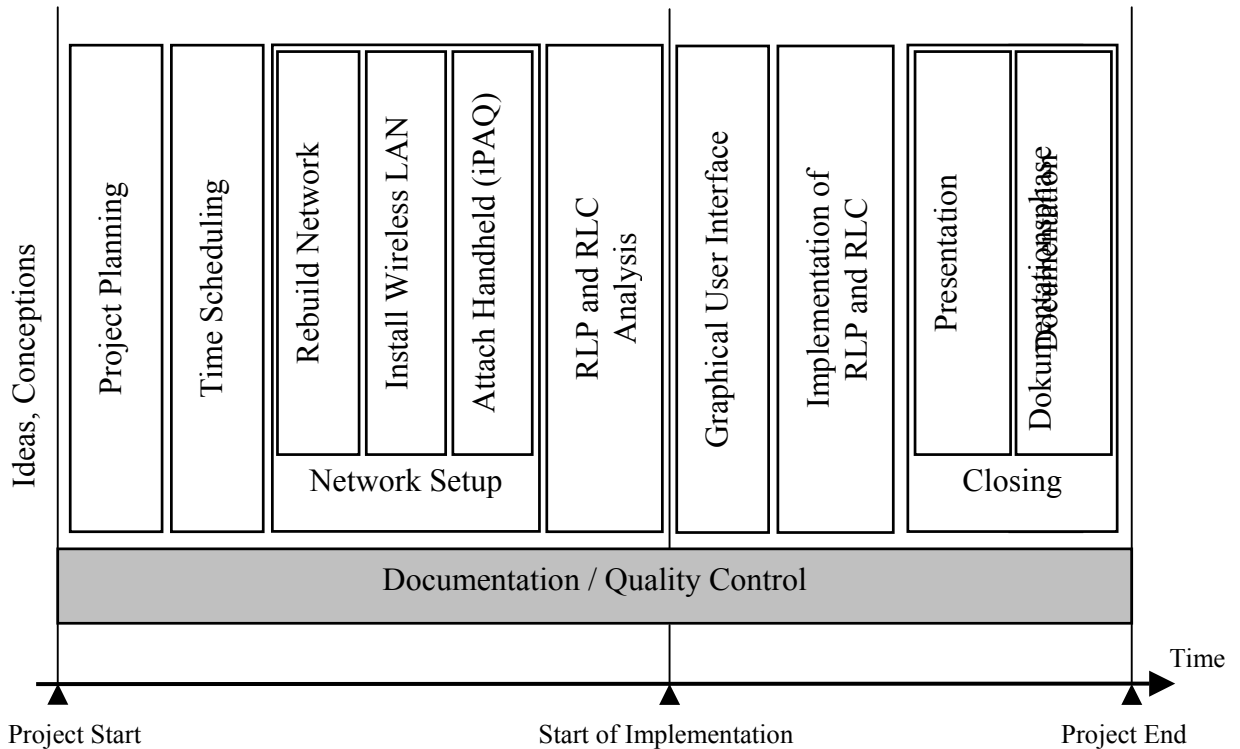


Figure 2: Phase Diagram

Documentation / Quality Control

The documentation shall contain an introduction to the problem, time management, the procedure itself as well as theoretical/analytical foundations and results. It shall be built up preferably parallel to the remaining work.

Quality Control not only means that we clearly define the course of events and targets but also the continuous checkup between actual and target. In case of huge differences, reasons and corrective measures shall be documented.

Project Planning / Time Scheduling

This part consists of getting an overall view of the whole project. The project shall be split up in several parts, a phase diagram shall explain and describe the work of every part. The time flow shall be demonstrated as exact as possible with help of a time schedule.

Network Setup

The practical part of this work consists of rebuilding our existing network according to the subscription in the chapter after next.

During this time section we need to build up a system, which later on allows to emulate CDMA2000 and GPRS/GSM. We illustrate the radio link between the mobile client and the base station of CDMA2000 and GPRS/GSM with a wireless LAN. So the user sees a wireless connection, which can be modified with our network emulator.

Rebuild Network

The network setup from our last project shall be rebuilt. Instead of emulating Internet conditions, we directly attach our network to the Internet. So a first computer emulates the radio link and forwards the data packets to a second computer, which itself has access to the Internet. It works as a proxy server (Apache or Webswift) and shall be able to compress data.

Install Wireless LAN

This part of the project deals with the setup and configuration of the wireless LAN. A bridge, which is connected with our network provides wireless access to the network with a notebook or a handheld device. A wireless PC card has to be configured and insert into our mobile client.

Attach Handheld (iPAQ)

As a further step, we think of extending our network by attaching an iPAQ instead of using a notebook.

RLP and RLC Analysis

We at this point need to study CDMA2000 and GPRS/GSM and their behaviour in packet oriented data transmission over TCP/IP. To get the effective bandwidth, delay time and loss rate on IP layer we need to analyse the subjacent protocols RLP and RLC, which are responsible for delaying IP datagrams. So we need to work out the effects RLP and RLC has on upper layers.

Graphical User Interface

The user of our application shall have the possibility to enter values (frame error rate, bit rate, MTU, roundtrip time etc.) through a graphical user interface either for CDMA2000 or GPRS/GSM. The application then calculates the respective characteristics for the network emulator NistNet and starts it.

Implementation of RLP and RLC

To calculate the characteristics for our network emulator, we first need to integrate the behaviour of RLP and RLC in our application. So this part consists of developing those algorithms in Matlab and then implementing in Java.

Closing

Documentation

Even though we aim at writing the documentation parallel to the work, we plan some time at the end to update and revise our project documentation.

Presentation

Another part of this section is the creation of a placard at the end of the project, which represents our work in brief.

2 General Introduction

2.1 Introduction

Cellular mobile radio – the magic technology that enables everyone to communicate anywhere with anybody – has created an entire industry in mobile telecommunications, an industry that is rapidly growing and that has become a backbone for business success and efficiency and a part of modern lifestyles all over the world.

2.2 The mobile radio channel

As a user of mobile phones, the deficiencies of mobile radio channels are well known. They sometimes lead to the interruption of a call. A mobile radio channel is just not a continuous medium such as a cable. It is afflicted with interferences and the mechanisms of the multipath propagation, which mostly do not allow a continuous signal, are another problem. The discontinuity of the mobile radio channels is a new aspect when accessing remote data. This leads to burst errors and to aborted connections, occasionally.

2.3 Transmission protocols

In the world of data transmission, protocols are used to cope with the contrarities of a link among other things. A protocol provides lots of standardized operations to manage the data transmission and other net activities between computers.

TCP/IP has become one of the most popular data communication protocols for transmissions over wired links. TCP/IP is the protocol stack of the Internet and almost every computer in the world is able to communicate with this stack. Due to this fact it is preferable to utilize TCP/IP also in mobile data transmission. If this idea does not operate good enough, the solution should be compatible with the consisting TCP/IP protocol stack. Because of the many developed software using the TCP/IP stack, this compatibility is necessary so that these applications can be integrated in the mobile environment. But do not forget, TCP/IP was developed for the use over continuous medias.

2.4 Problem

A number of design incompatibilities between wireless protocols and the Internet have come up as the Internet connectivity reaches out to the mobile users of cellular systems. TCP/IP has been developed under the assumption that packet losses are mainly caused by network congestion, so TCP/IP contains congestion avoidance algorithms, which therefore cause rate reduction. In a radio link over TCP/IP, misinterpretation of packet losses over radio links as congestion problems lead to significant throughput lowering. This is because TCP/IP sets its internal timers among other things due to the roundtrip time, which can be quite mercurial in wireless systems. So TCP/IP cannot cope with the mercurial roundtrip time and therefore as the case may be aborts connections instead of caring for its integrity.

Lower level protocols provide different error control methods such as ARQ (automatic request for retransmissions) to improve reliability. Link layer ARQ error control can reduce frame losses and improve the overall performance as specified in the standards of RLP and RLC, but these protocols cause an enormous packet delay on the upper layers.

2.5 CDMA2000

CDMA2000 is a third generation (3G) wireless system, which is based on the CDMA ONE system. Unlike some 3G standards, it is an evolution of an existing wireless standard. It supports 3G services as defined by the International Telecommunications Union ITU for IMT-2000. CDMA2000 is both an air interface and a core network solution. This system delivers high-bandwidth data and voice services to users of mobile equipment. The figure below shows the infrastructure of a CDMA2000 wireless network.

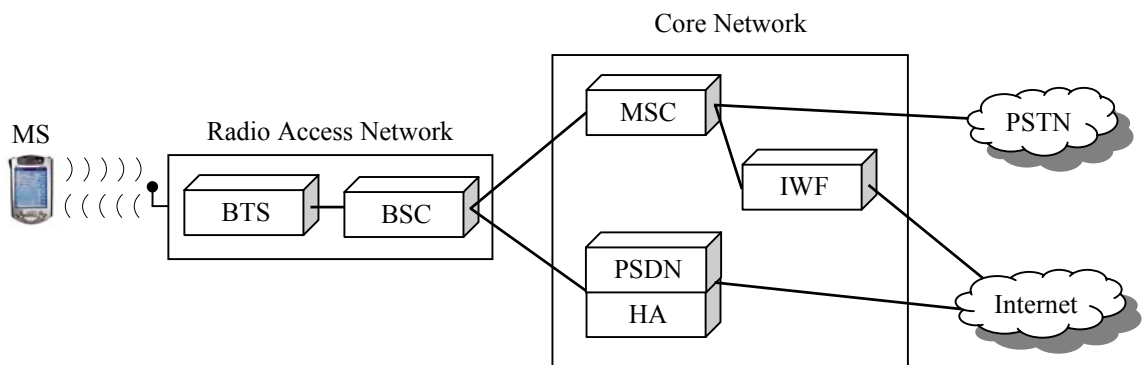


Figure 1: CDMA2000 System

One of the most important concepts to any cellular telephone system is that of “multiple access”, meaning that multiple, simultaneous users can be supported. In other words, a large number of users share a common pool of radio channels and any user can gain access to any channel (each user is not always assigned to the same channel).

A channel can be thought of as merely a portion of the limited radio resource, which is temporary allocated for a specific purpose, such as someone's phone call. A multiple access method is a definition of how the radio spectrum is divided into channels and how channels are allocated to the many users of the system. In the next figure we show the protocol stack of CDMA2000 connected with the Internet.

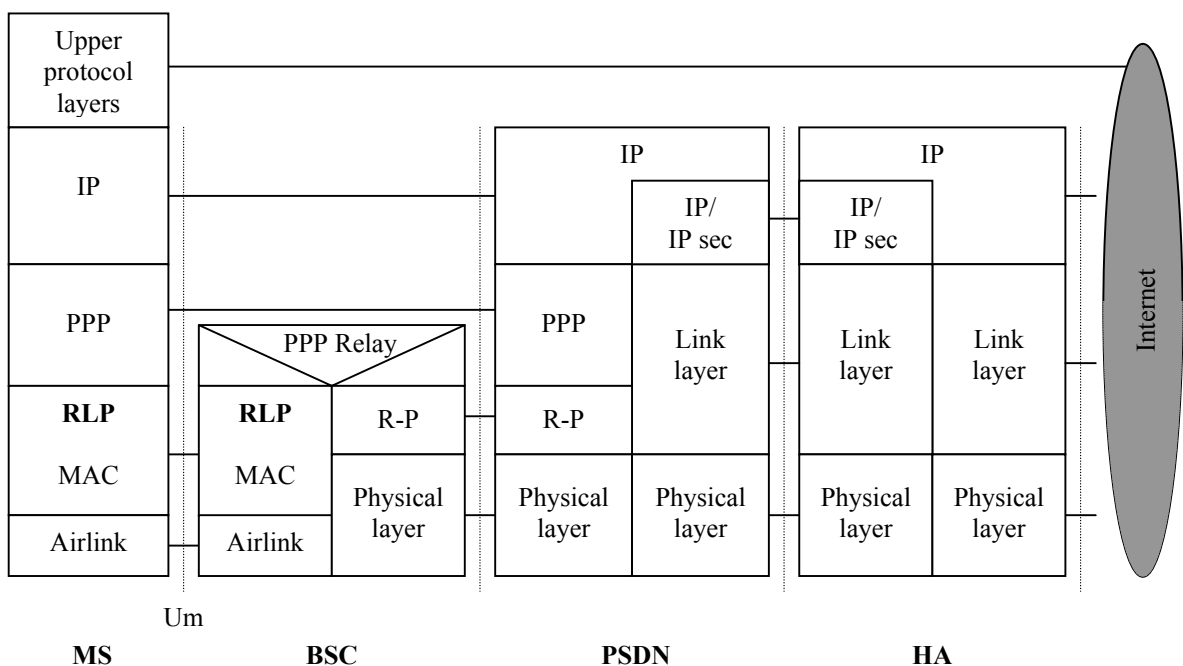


Figure 2: CDMA 2000 Protocol Stack

2.6 GPRS

The General Packet Radio Service is an addition to the GSM system that makes the use of packet switched data service possible.

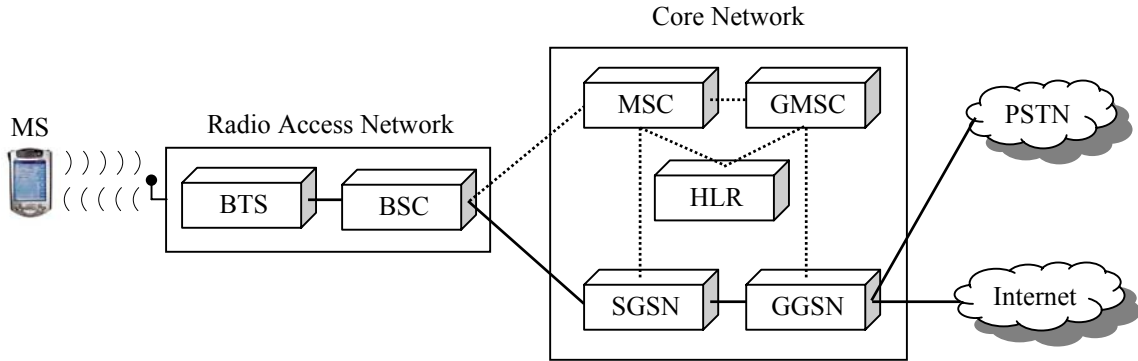


Figure 3: GPRS System

The radio resources will be reserved only for the duration of the actual data transfer instead of the duration of a connection as in the GSM circuit switched data. This will enable volume based charging which is more suitable for data applications not producing a constant stream of data but which are bursty in nature. The operators can serve more users with the same resources because they are used more efficiently. Four new channel coding schemes are introduced which are optimized for packet data transfer. This together with the possibility to use several channels (time slots) simultaneously enables considerably higher bit rates than in the traditional GSM data. The connection establishment time in GPRS is much shorter than in the circuit switched data, which enables instantaneous access to the Internet.

A new GPRS protocol stack is introduced into the mobile station and into the network. In the next figure we show the protocol stack of GPRS connected with the Internet. The network provides packet data services over the radio to the MS through the air interface, it can also be seen in this figure (Um interface).

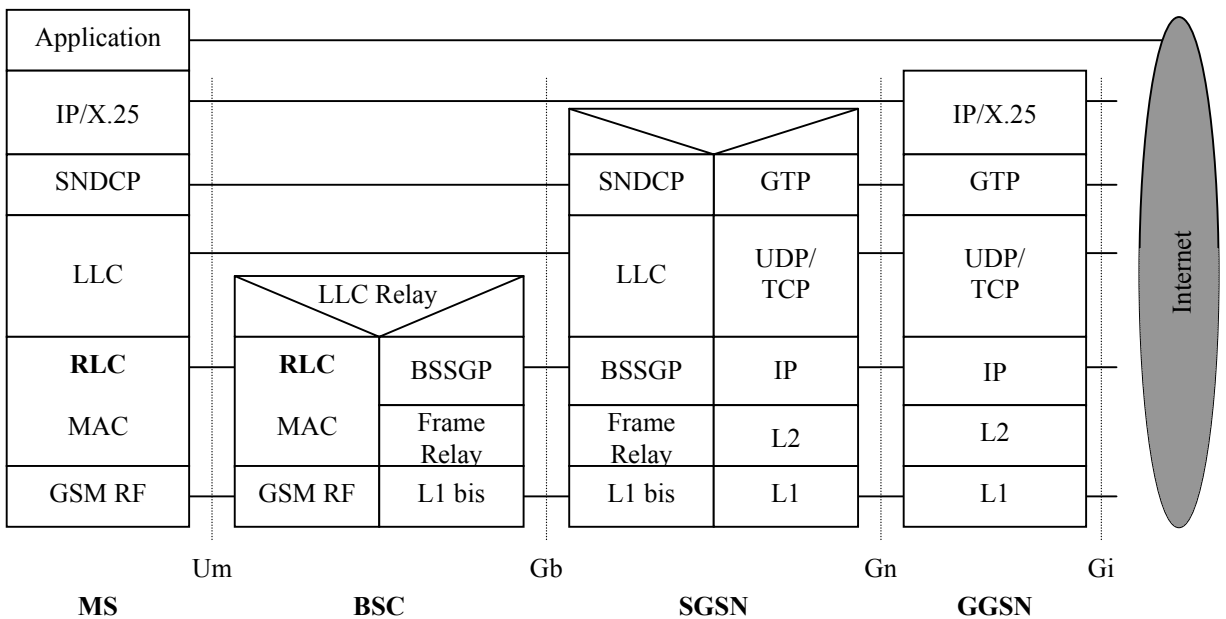


Figure 4: GPRS Protocol Stack

2.7 UMTS

The universal mobile telecommunication system (UMTS) is a third generation (3G) wireless system that delivers high-bandwidth data and voice services to mobile users. UMTS evolved from global systems for mobile communications (GSM). UMTS has a new air interface based on W-CDMA and a core network based on general-packet radio service (GPRS). The figure below shows the infrastructure of a UMTS wireless network.

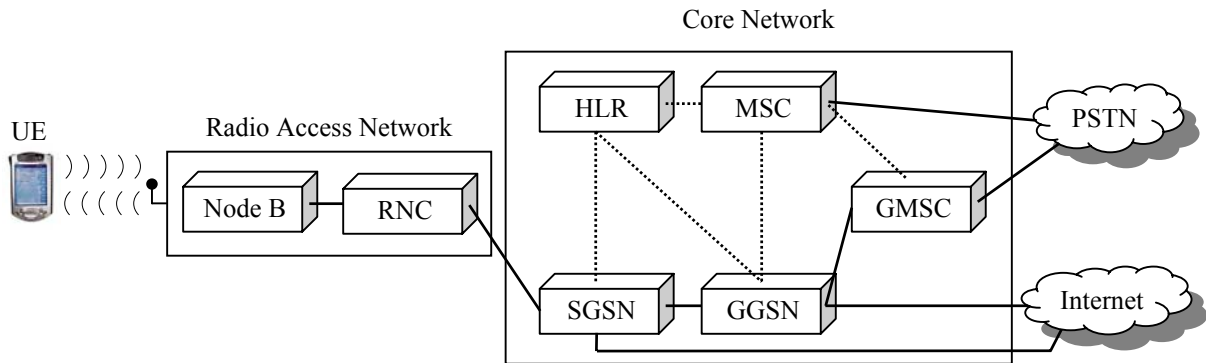


Figure 5: UMTS System

As mentioned above, UMTS differs from GSM mostly in the new principles of the air interface transmission (W-CDMA instead of time division multiple access TDMA / frequency division multiple access FDMA). Therefore, a new radio access network called UTRAN has to be introduced with UMTS.

The UMTS standard can be seen as an extension of existing networks. Two new network elements are introduced in UTRAN, RNC and Node B. UTRAN is subdivided into individual radio network systems (RNSs), where each RNS is controlled by an RNC. The RNC is connected to a set of Node B elements each of which can serve one or several cells.

Existing network elements, such as MSC, SGSN and HLR, can be extended to adopt the UMTS requirements, but RNC, Node B, and the user equipment (UE) must be completely new designs. In the next figure we show the protocol stack of the UMTS architecture.

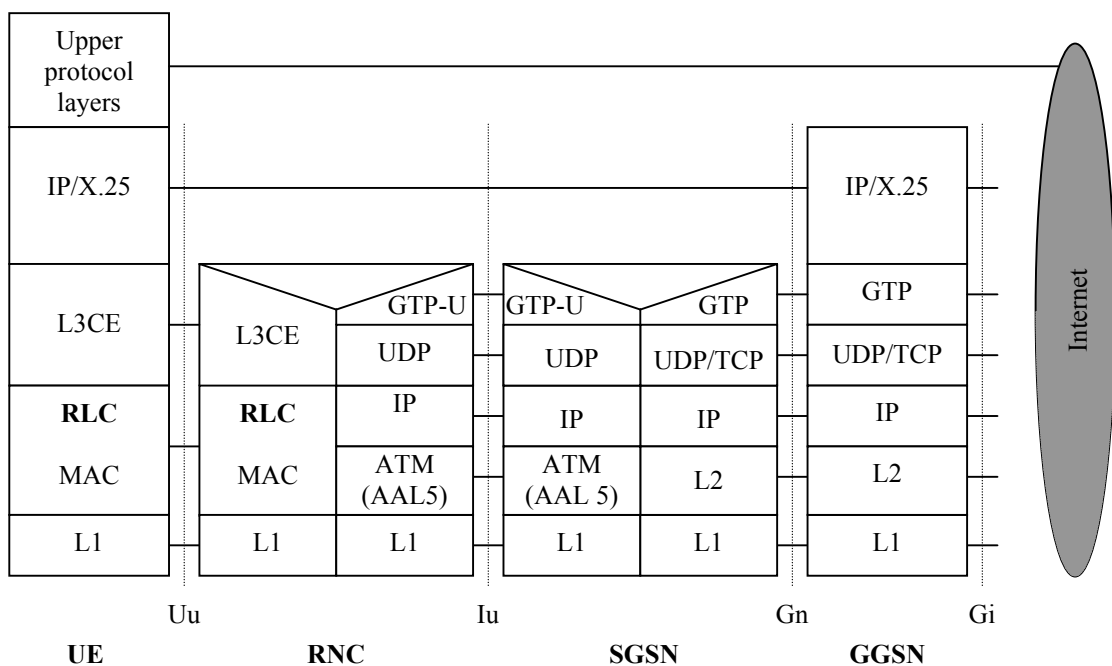


Figure 6: UMTS Protocol Stack

3 Experimental Setup

3.1 Introduction

In this chapter we describe the setup of the network with all its components and all the software we installed. We also wrote down the most important settings such as routing information and version of OS for each computer.

3.2 Network

Principle: Separate transport connection used for fixed and wireless link. End-to-End connectivity support via a proxy-server placed at the end of our network. So we have one TCP connection between the mobile client and the proxy-server and a second connection between the proxy-server and a webserver in the Internet.

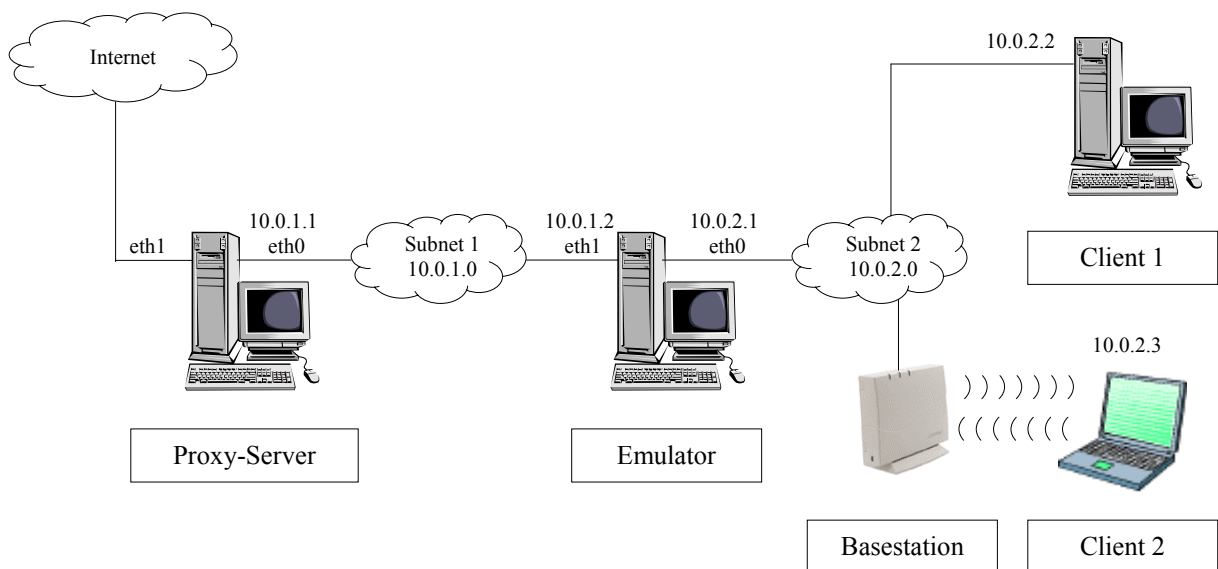


Figure 3: Network Structure

3.3 Client 1

3.3.1 Introduction

As a first client, we use a PC, which of course must not be connected with the Internet. If we want to browse on the web with our client, or strictly speaking, we want to have a TCP connection to a webserver in the Internet, a first TCP connection is going to be established between client and server. The proxy opens a second TCP connection to the respective host in the Internet and sends the request. The host answers and sends its packets (response) to the server, which forwards them to the client.

3.3.2 Operating System

An Intel Pentium II PC, on which Microsoft Windows NT version 4 is installed, runs as a client and is connected with the emulator.

3.4 Client 2

3.4.1 Introduction

As a second client, we currently use a Toshiba notebook with a wireless PC card WL 110 from Compaq. The wireless link between client 2 and the emulator shall illustrate the radio link of the CDMA2000 and GPRS/GSM network architecture.

3.4.2 Operating System

The notebook runs as a client on Microsoft Windows 98 and is connected through the air interface with the wireless bridge.

3.4.3 Extension

Later on our notebook is going to be replaced by a handheld device from Compaq (iPAQ). To realise this exchange we only need to put the same wireless PC card into our handheld.

3.5 Basestation

3.5.1 Introduction

The basestation, a wired to wireless bridge WL410 from Compaq, can either be used to connect wireless cells to one another or to a wired LAN. This bridge does just what the name implies in that it bridges the wired network with the wireless one. Each bridge has an antenna that is used to communicate with the wireless PC card. This card will be able to communicate with the bridge within a varying radius of the bridge. How fast and how far away from the bridge the wireless card will operate is dependent upon many factors including the physical construction of the building, antenna placement, and outside interferences.

3.6 Emulator

3.6.1 Introduction

The emulator has to reproduce the lossy path (radio link of CDMA2000 and GPRS/GSM) between the clients and the proxy-server. To emulate this link, we use the network emulation packet NistNet, which we describe later on.

3.6.2 Operating System

Linux Redhat 7.1 with kernel version 2.2.16 runs on the emulator.

3.6.3 Routing

To make this machine routing the incoming packets, we had to type the following two commands:

```
$ cd /proc/sys/net/ipv4/conf/all  
$ echo "1" > forwarding
```

We also need to modify the routing table and afterwards it should look like this:

<i>Dest:</i>	<i>Router:</i>	<i>Genmask:</i>	<i>Interface:</i>
10.0.1.0	*	255.255.255.0	eth1
10.0.2.0	*	255.255.255.0	eth0
127.0.0.0	*	255.0.0.0	lo
default	10.0.1.1	0.0.0.0	eth1

3.6.4 NistNet

NistNet is a network emulation package that runs on Linux. NistNet allows a single Linux PC set up as a router to emulate various network conditions. Some information about NistNet:

The Nist Network Emulation Tool (NistNet) is a general-purpose tool for emulating performance dynamics in IP networks. The tool is designed to allow controlled, reproducible experiments with network performance sensitive/adaptive applications and control protocols in a simple laboratory setting. By operating at the IP level, NistNet can emulate the critical end-to-end performance characteristics imposed by various wide area network situations (e.g., congestion loss, varying round trip time).

NistNet is implemented as a kernel module extension to the Linux operating system and an X Window system-based user interface application. In use, the tool allows an inexpensive PC-based router to emulate numerous complex performance scenarios, including: tunable packet delay distributions, congestion and background loss, bandwidth limitation, and packet reordering/duplication. The X interface allows the user to select and monitor specific traffic streams passing through the router and to apply selected performance “effects” to the IP packets of the stream. In addition to the interactive interface, NistNet can be driven by traces produced from measurements of actual network conditions.

You need to load the NistNet module before starting the tool:

```
$ /home/pinoccio/NistEmulator/nistnet  
$ ./Load.Nistnet
```

Note that when using our application, you do not need to load the module manually. We created a shell script, which loads the module, starts the application and unloads the module from the kernel when closing the application. Description of this shell script and of the application follows in chapter Software Engineering. In Nistnet, the following values can be set (there are some more which we do not use):

Delay [ms]	:	Mean of delay (one way delay)
Delsigma [ms]	:	Standard deviation of delay
Bit Rate [Byte/s]	:	Maximum allowed bit rate
Drop [%]	:	Percentage of dropped packets

To control the emulator, two tools are included in the NistNet package. There is a GUI version and a command line interface. The command line interface takes the following arguments:

```
Turn emulator on:      cnistnet -u  
Turn emulator off:    cnistnet -d  
Add new entry:        cnistnet -a asource adestination --delay adelay astd  
                      --drop adrop --bandwidth abandwidth  
Read list of entries:  cnistnet -R  
Remove entry:         cnistnet -r asource adestination  
See statistics for entry: cnistnet -s asource adestination
```

Since we can set delay and bit rate in Nistnet, it is not obvious how Nistnet handles with the delay time. Setting a bit rate also causes a delay in the system, so we do not know whether Nistnet adds those two delays or just takes the larger one. To work this out, we do the following experiment:

With `ping -s 1300 10.0.2.2` we send packets of 1308 Byte (8 Byte ICMP-Header) to the client:

<i>Delay (ms)</i>	<i>Bit Rate (Byte/s)</i>	<i>AVG(delay) theoretically (ms)</i>	<i>AVG(delay) measured (ms)</i>
0	100'000	$1308/100'000=13.08$	10.3
0	10'000	$1308/10'000=130.8$	130.4
200	10'000	$1308/10'000=130.8$	399.4
200	5'000	$1308/5000=261.6$	400.1 (2·200)
200	3'000	$1308/3000=436.0$ (>400!)	439.7

At 5000Byte/s we should have a delay of approximately 260ms. Since the oneway delay is set to 200ms, our measured delay is 400ms. Decreasing the bit rate to 3000Byte/s results in a theoretical delay of 436ms, which is larger than the set delay time. We now clearly see that Nistnet does not add the two delays but always takes the larger one.

Another problem to solve is the handling of the standard deviation of Nistnet. We do not know whether the standard deviation, which can be set either relates to the set delay or to the delay, which occurs due to the set bit rate or to the total delay. To work this out, we do the following experiment:

With `ping -s 1300 10.0.2.2` we send packets of 1308 Byte (8 Byte ICMP-Header) to the Client:

<i>Delay (ms)</i>	<i>Bit Rate (Byte/s)</i>	<i>Delsigma(ms)</i>	<i>AVG(delay)/MDEV(delay) measured (ms)</i>
0	100'000	50	41.4/42.6
0	10'000	50	132.4/13.1
200	10'000	50	394.0/59.4
200	5'000	50	418/63.5
200	3'000	50	456.3/34.8

As we see the standard deviation we set (Delsigma) does have impact of the total delay and not only of the set delay.

3.7 Server

3.7.1 Introduction

Proxies are a sort of server, usually designed for the use of an internal network to forward traffic to the Internet and back to the internal machine. A proxy normally caches Internet data. It does this by accepting requests for objects that people want to download and handling their request in their place. In other words, if a person wants to download a web page, they ask the proxy server to get the page for them. The proxy then connects to the remote server and requests the page. It then transparently streams the data through itself to the client machine, but at the same time keeps a copy. The next time someone wants that page, the server simply reads it off disk, transferring the data to the client machine almost immediately. This makes web browsing much faster. It's possible to turn off this mechanism so that the proxy server does not save a copy of the searched web pages on his disk. For our experimental setup we turned off this mechanism, so our proxy works only as an agent between the Internet and "wireless" part of our network and does not cache any data.

3.7.2 Operating System

On the machine, which has to run as a proxy server we installed Linux Redhat 7.1 with the kernel version 2.4.13 and set it up as a workstation. For this installation it was important to know which types of hardware the machine is using (screen, keyboard, graphic-card). Other important

information we had to type in during the installation-process were the partitions from where the Linux Operating System can boot. We choose the following partition settings:

<i>Mount Point:</i>	<i>Device:</i>	<i>Requested:</i>	<i>Actual:</i>	<i>Type:</i>
/boot	hda1	24 MB	24 MB	Linux native
/	hda5	3300 MB	3300 MB	Linux native
	hda6	128 MB	128 MB	Linux swap

3.7.3 Routing

We also need to modify the routing table and afterwards it should look like this:

<i>Dest:</i>	<i>Router:</i>	<i>Genmask:</i>	<i>Interface:</i>
10.0.1.0	*	255.255.255.0	eth0
10.0.2.0	10.0.1.2	255.255.255.0	eth0
152.96.24.0	*	255.255.252.0	eth1
127.0.0.0	*	255.0.0.0	lo
0.0.0.0	152.96.24.53	0.0.0.0	eth1

Router 152.96.24.53 refers to the default gateway of the HSR network.

3.7.4 Apache running as Proxy-Server

We decided to install the Apache httpd server version 1.3.19. We installed the following three packages from the original Redhat CD's to get the Apache httpd server.

- ❑ apache-1.3.19-25.i386.rpm
- ❑ apache-devel-1.3.19-25.i386.rpm
- ❑ apache-manual-1.3.19-25.i386.rpm

To start/stop the Apache server, we type the commands:

```
$ /etc/rc.d/init.d/httpd start  
$ /etc/rc.d/init.d/httpd stop
```

To let Apache know that it has to run as a proxy, we had to do some more work:

To make Apache load an unloaded module, you need to uncomment the following two lines in the httpd configuration file (httpd.conf):

```
#LoadModule proxy_module      modules/libproxy.so  
#AddModule mod_proxy.c
```

An other thing we had to do as we want to run our Apache server as a proxy was to uncomment all the proxy server directives in the httpd.conf file:

```
# Proxy Server directives. Uncomment the following lines to  
# enable the proxy server:  
#  
#<IfModule mod_proxy.c>  
ProxyRequests On  
#  
<Directory proxy: *>  
    Order deny, allow  
    Deny from all  
    Allow from all  
</Directory>  
#
```

Now we only had to restart the Apache server and automatically it was running as a proxy.

To restart the Apache server, we type the command:

```
$ /etc/rc.d/init.d/httpd restart
```

After this work it was possible to browse in the web with the client through our proxy. By default the proxy watches only on port 80, which is the port HTTP.

3.7.5 Ethereal Network Analyzer

Ethereal is a free network protocol analyzer for both Linux and Windows. It allows you to examine data from a live network or from a capture file on disk. You can interactively browse the capture data, viewing summary and detail information for each packet. Ethereal has several powerful features, including a rich display filter language and the ability to view the reconstructed stream of a TCP session. To install this tool we downloaded version 2.0.10 from www.ethereal.com.

3.7.6 Webswift

Webswift is a compressing web proxy server. Webswift includes compression of images, text and HTML. Webswift can be installed at ISPs or at web sites, wherever there is a bottleneck. It alleviates that bottleneck by compressing documents before transmission. By using industry standard compression formats, users' web browsers can uncompress the documents without any special software. This compression has two positive effects. First, the compressed document can reach the user quicker, which increases user satisfaction. Second, the compressed document requires less bandwidth to transfer, which means companies do not need to buy as much expensive bandwidth.

Webswift is fully standards compliant, and will work with any web server on the Internet. It will automatically detect if the web browser can support compression. Supported browsers include Netscape 4, MSIE 4 and higher, Opera 4.0 and higher, and Lynx 2.8 and higher. Webswift uses lossless compression for text-based documents. Thus (unlike some other HTML compression schemes), the web browser receives the document exactly as it was stored on the origin web server. Image compression is based on the image type, GIF images are transformed on the fly to PNG images. PNG, which stands for Portable Network Graphics, is a new and more efficient image encoding format, that is an average of 10% smaller than GIF on the same image. In addition, Webswift can optionally perform lossy compression on JPEG images, greatly reducing their size while slightly reducing image quality. Overall compression ratios achieved range anywhere from 20% to over 50%, depending on the type of content transferred.

To run Webswift instead of the apache proxy server, you need to change the port at the mobile client from 80 to 3127 and to start the server with the following commands:

```
$ /usr/local/webswift  
$ ./compress_server
```

4 Radio Link Protocol (RLP)

4.1 Introduction

In the IS-707 standard on data transmission the RLP is defined as a NAK-based selective repeat ARQ (automatic request for retransmissions) scheme. Its purpose is to achieve an acceptable packet error performance for the IP datagrams that will be transferred to higher layers. RLP uses the CDMA2000 20ms frame structure. Therefore, the RLP frame size at a user data rate of 9600bps is 192bit (24Byte) which requires that an IP datagram, which usually consists of several hundred bytes, is split up into many RLP frames.

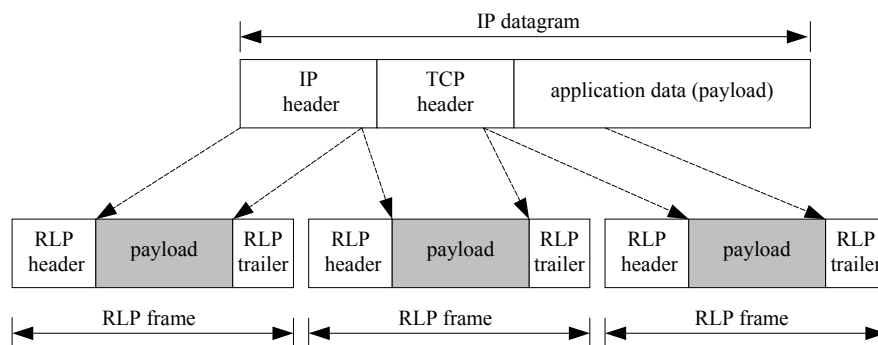


Figure 4: IP Datagram split up into RLP frames

$$RLPframeSize = \frac{UserDataRate}{20ms} = \frac{9600bps}{20ms} = 24Byte$$

Since RLP needs some space (3 Bytes at 9600bps, for CRC), the payload decreases to 21Byte, which is 87.5% of the whole RLP frame. To get the payload of an RLP frame, we always decrease the frame length, which depends on the user data rate, by 12.5%.

By assuming that an IP datagram consists of 1500Byte (MTU) it has to be split up into $N_1=72$ RLP frames:

$$N_1 = \frac{MTU}{UserDataRate \cdot 20ms \cdot 0.875} = \frac{1500Byte}{9600bps \cdot 20ms \cdot 0.875} = 71.4$$

We can now calculate the bit rate for IP over RLP if we assume that the frame error rate is zero (FER=0).

$$BitRate = \frac{MTU}{N_1 \cdot 20ms} = \frac{1500Byte}{72 \cdot 20ms} = 1040Byte/s$$

Furthermore, it can be assumed that the FER (frame error rate) ranges between 1% and 5% on average. Depending on the situation, FER can rise up to 20% temporarily.

If a RLP frame gets lost TCP/IP can only recover by resending the entire packet. Now we can compute the IP datagram success probability without RLP:

$$p_{IPsuccess} = (1 - FER)^{N_1} = (1 - 0.01)^{72} = 0.485 \text{ (quite bad)}$$

As we see a selective link level retransmission process is clearly needed to minimize the IP datagram loss probability.

4.2 How does RLP operate ?

RLP uses NAK (not acknowledged) frames to indicate/retransmit lost data frames. So, the receiver does not acknowledge correct RLP data frames. In case of data frame losses, RLP performs data frame recovery through limited retransmission attempts requested by NAK frames. Therefore, RLP is not a reliable protocol. Further error recovery and end to end reliability can be handled by the error and flow control mechanisms defined by TCP/IP.

RLP frames are numbered sequentially and RLP uses a transmitting sequence number (SEQ) variable $V(S)$ to identify new data frames. It also maintains two receiving variables $V(R)$ and $V(N)$ to reconstruct the received data frames. $V(R)$ is the SEQ of the next data frame expected to be received whereas $V(N)$ is the SEQ of the next data frame needed for sequential delivery.

The RLP receiver provides a storage buffer for resequencing out of sequence data frames. When all frames up to a certain SEQ are received in order of sequence, the RLP layer passes them to its upper layer.

RLP maintains a NAK retransmission timer for each data frame requested in a NAK frame. It is implemented as a frame counter, which is incremented by one for each valid RLP data frame. It expires when reaches a value greater than twice the estimated RLP frame roundtrip time plus a guard interval.

There are three retransmission rounds for a lost RLP data frame. The first round is starting when the receiving RLP notices that a data frame is missing (SEQ of received frame $> V(R)$). The second and third round for this requested frame are activated when the NAK retransmission timer expires at the first and second time. The number of NAK frames sent in each round is the retransmission round number. Because each received NAK frame causes a data frame retransmission, multiple copies of a data frame may be retransmitted when the sender receives a multiple NAK for the requested data frame. If the NAK retransmission timer expires three times for the same data frame, frame error recovery is given up for that requested data frame.

The RLP transmission procedure can briefly be described by the following rules:

- ❑ If $SEQ < V(N)$ or if the frame is already stored in the resequencing buffer, discard the frame.
- ❑ If $SEQ = V(N)$, update $V(N)$ to the next oldest missing frame sequence number. Pass received frames up to $V(N)-1$ to the upper layer.
- ❑ If $V(N) < SEQ < V(R)$, store frame SEQ in the resequencing buffer if it is missing.
- ❑ If $SEQ = V(R) = V(N)$, pass all received frames up to $V(R)$ to the upper layer.
- ❑ If $SEQ = V(R) \neq V(N)$ or $SEQ > V(R)$, increment $V(R)$ and store frame SEQ in the resequencing buffer.
- ❑ For all cases, send NAK's of missing frames if their retransmission timers are not yet set or expired.

The next figure shows the RLP frame retransmission mechanisms in a timeline.

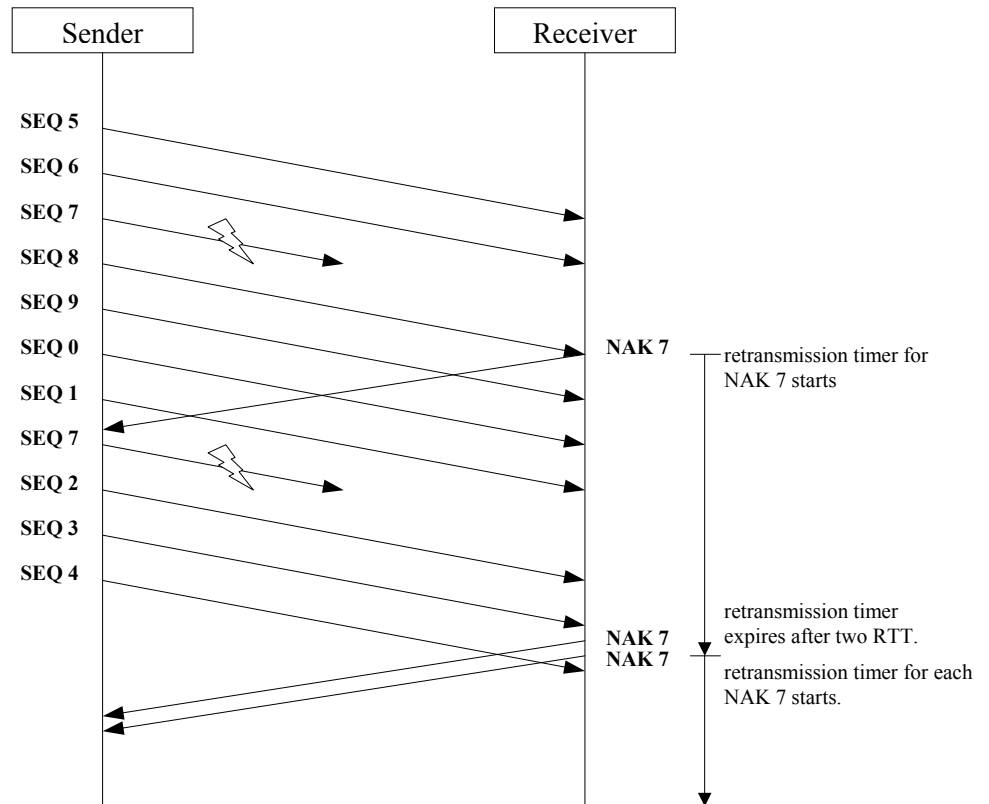


Figure 5 : RLP retransmission process in a timeline

4.3 Effective bit rate for IP over RLP

In this part we present an analytical approach to calculate the average number of RLP data frames needed for an IP packet over RLP. The presented approach has the advantage that it results in a simple formula.

Let N_1 be the number of RLP frames for a given IP packet. As we have seen before, N_1 depends on the user data rate and the MTU of the link. Taking the previous example (9600bps, MTU=1500B), it takes $N_1=72$ RLP data frames to transmit the entire IP packet when no frame is lost.

In this chapter, we think of the RLP retransmission algorithm in the following fashion, which is not the way RLP works, but for calculating the average number of RLP frames for an IP packet, this point of view is accurate and simplifies the calculation.

First all RLP frames get sent. This is the first round. We now assume that no NAK frames are lost (this is very close to true since the probability that a frame and its NAK are lost is on the order of p^2 and $p < 5\%$). Now all NAKs of the first round are evaluated and all frames that did not make it in the first round are resent. This is the second round (first retransmission round) and the number of frames sent in the second round is a random variable, which we call N_2 . Note that this point of view assumes that after we sent the last frame of the IP packet, we know which frames did not make it in this round. This is a correct assumption for the frames at the beginning of the IP packet, but an incorrect one for the last few frames, for which the NAK might still be in the network. While this effect is very important for the IP packet delay, for calculating the average number of frames for the purpose of finding the effective bit rate of the RLP link, this point of view does not change the result. After the second round, we evaluate all the NAKs of the second round and send the frames, which did not make it as the third round. According to RLP, in the third round, we send every frame twice. The number of sent frames in the third round is also a random variable and we call it N_3 . Again, after the third round, we evaluate the NAKs and send the frames that did not make it again, this time we send every frame three times. The number of sent frames in the fourth round is also a random variable and we call it N_4 . At this point, RLP gives up.

Let p be the probability that a frame is lost (Frame error rate, FER) and q the probability that a frame made it ($1-p$). Then the distribution of N_2 (frames sent in the second round = frames lost in the first round) is a Binomial distribution with parameters N_1 and p . Hence its mean is simply $N_1 \cdot p$. The distribution of N_3 (frames sent in the third round = 2·frames lost in the second round) is also a binomial distribution with parameters N_2 and p . Note that N_2 itself is a random variable. The distribution of N_4 (frames sent in the fourth round = 3·frames lost in the third round) is also a binomial distribution with parameters N_3 and p . Note that N_3 itself is a random variable.

We are now interested in the average total number of frames transmitted:

$$\begin{aligned} N &= E[N_1 + N_2 + N_3 + N_4] = N_1 + E[N_2 + N_3 + N_4] = N_1 + E[N_2] + E[N_3] + E[N_4] = \\ N &= N_1 + N_1 \cdot p + E[N_2 \cdot p] + E[N_3 \cdot p] = N_1 + N_1 \cdot p + 2 \cdot N_1 \cdot p^2 + 3 \cdot E[N_2 \cdot p] \cdot p = \\ N &= N_1 + N_1 \cdot p + 2 \cdot N_1 \cdot p^2 + 3 \cdot 2 \cdot N_1 \cdot p^3 = N_1 \cdot (1 + p + 2 \cdot p^2 + 6 \cdot p^3) \end{aligned}$$

So:

$$N = N_1 \cdot (1 + p + 2 \cdot p^2 + 6 \cdot p^3)$$

This formula shows the effect RLP has. While it reduces the IP packet loss to nearly zero (next chapter), it achieves this by increasing the number of frames needed to transmit a given IP packet. The increase is $(1+p+2 \cdot p^2+6 \cdot p^3)$, as expected a function of the frame loss rate p . Please refer to the following table, which shows the increase of frames depending on p .

p (frame error rate, FER)	frames to transmit
0	N
0.01	1.01·N
0.02	1.02·N
0.05	1.06·N
0.10	1.13·N
0.15	1.22·N

Figure 6: Increase of data frames

For p=0.01 the increase is small, about 1%, since the linear term dominates. For p=0.05 it is already 6%, since the higher order terms are taking effect and for p=0.10 the increase is 13% and it gets worse as we go on.

Since RLP increases the average number of frames necessary to transmit for a given IP packet it decreases the average throughput by the same factor of $(1+p+2\cdot p^2+6\cdot p^3)$. Hence,

$$BitRate_{eff} = \frac{BitRate}{1 + p + 2p^2 + 6p^3} = \frac{MTU}{N_1 \cdot 20ms} \cdot \frac{1}{1 + p + 2p^2 + 6p^3} = 1031 \text{ Byte/s}$$

p (FER)	User Data Rate [bps]	MTU [Byte]	N	Bit Rate (p=0) [Byte/s]	Bit Rate (after RLP) [Byte/s]
0.01	9600	1500	73	1040	1031
0.05	9600	1500	76	1040	987
0.10	9600	1500	81	1040	925
0.01	57600	1500	13	6250	6187
0.05	57600	1500	13	6250	5920
0.10	57600	1500	14	6250	5551

Figure 7: Decrease of Bit Rate

For example, at 9600 bps for a 1500 Byte IP frame with 1% frame error rate FER, the number of frames needed for the IP packet moves from 72 to 73 frames which reduces the throughput from 1040 Byte/s to 1031 Byte/s, both trivial amounts. For a frame error rate of 5%, the results are 76 frames and 987 Byte/s. For 10% the results are 81 frames and 925 Byte/s.

The procedure to calculate the average total number of frames transmitted for an IP packet can also be presented in a decision tree. It shall help to understand our simplified RLP retransmission algorithm.

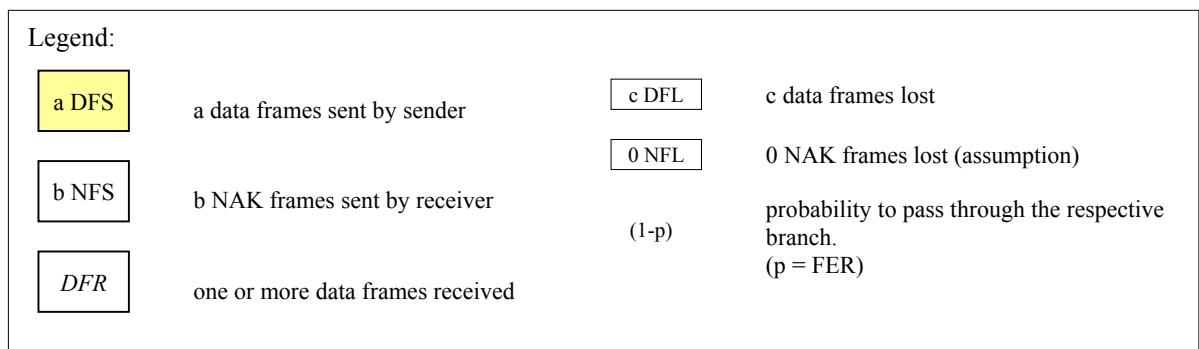


Figure 8: Legend for tree

Number of delivered data frames:

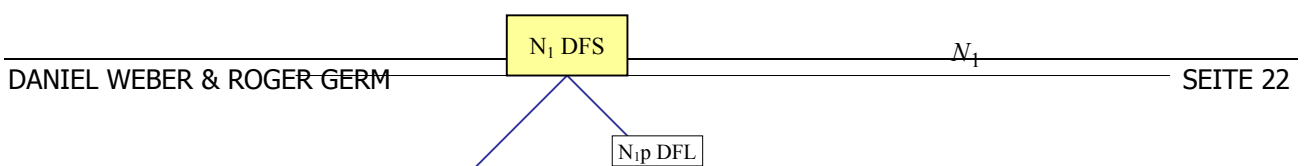


Figure 9: Simplified RLP retransmission process

Summing up the number of delivered data frames on the right leads us to the following formula, which we already know:

$$N = N_1 \cdot (1 + p + 2 \cdot p^2 + 6 \cdot p^3)$$

4.4 RLP retransmission process as a decision tree

The RLP three-round data frame retransmission process is often diagrammed as a flow chart. As we need to compute the probability that an IP datagram arrives after RLP retransmission process, we first need to compute the probability that an RLP frame arrives. So we decided to illustrate the retransmission process as a decision tree. (In order to show this complex tree, we split it up, therefore our decision tree consists of three parts: starting tree, tree 2, tree 3.)
 To compute the one way delay time on IP layer and its standard deviation we use an other method which we describe later on.

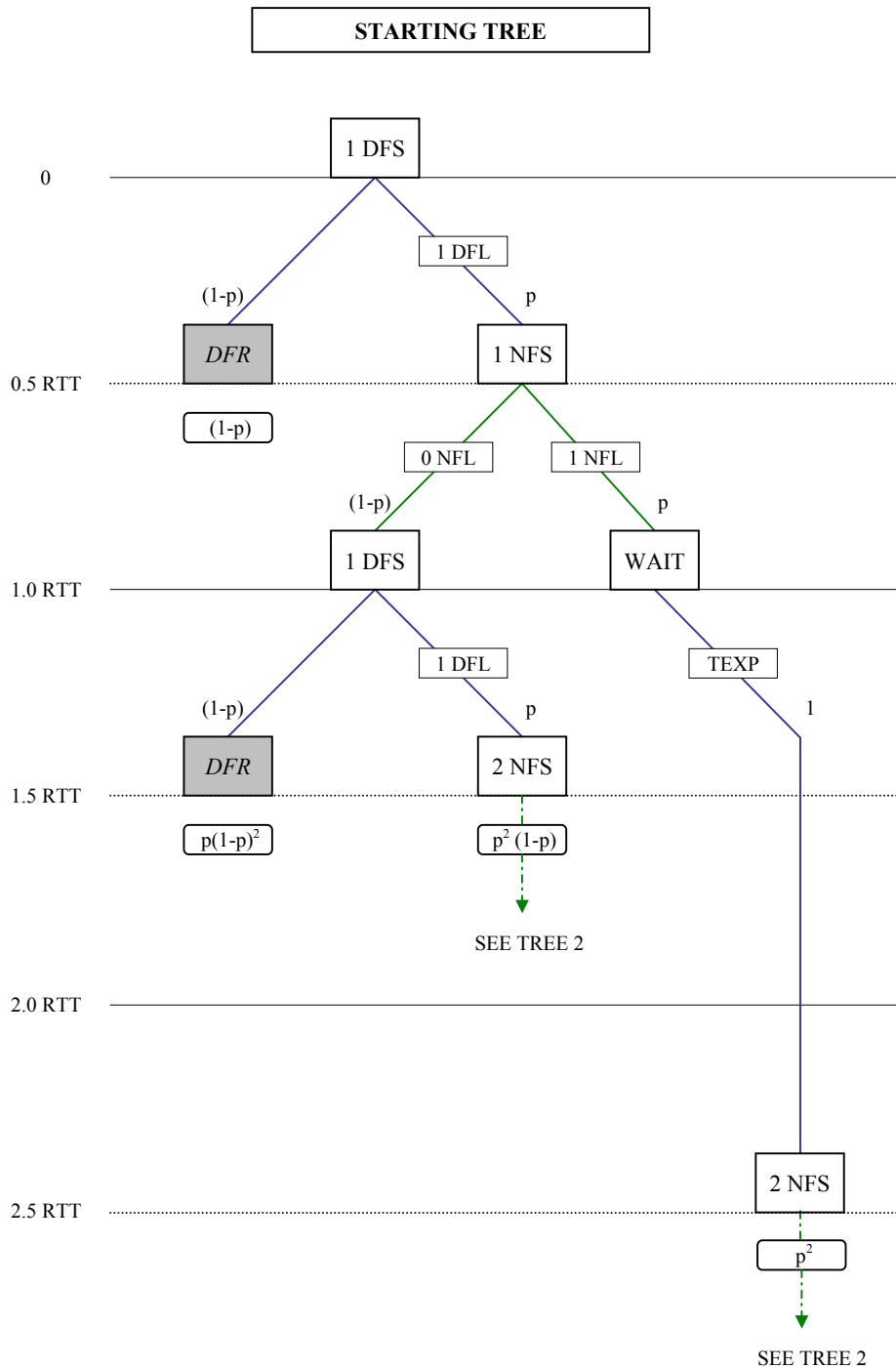


Figure 10: Starting tree (RLP retransmission process)

Legend:			
a DFS	a data frames sent by sender	c DFL	c data frames lost
b NFS	b NAK frames sent by receiver	d NFL	d NAK frames lost
DFR	one or more data frames received	TEXP	timer expires after 2·RTT
NDFR	no data frames received	(1-p)	probability to pass through the respective branch. (p = frame loss probability = FER)
		(1-p)	partial probability for the respective path

Figure 11: Legend

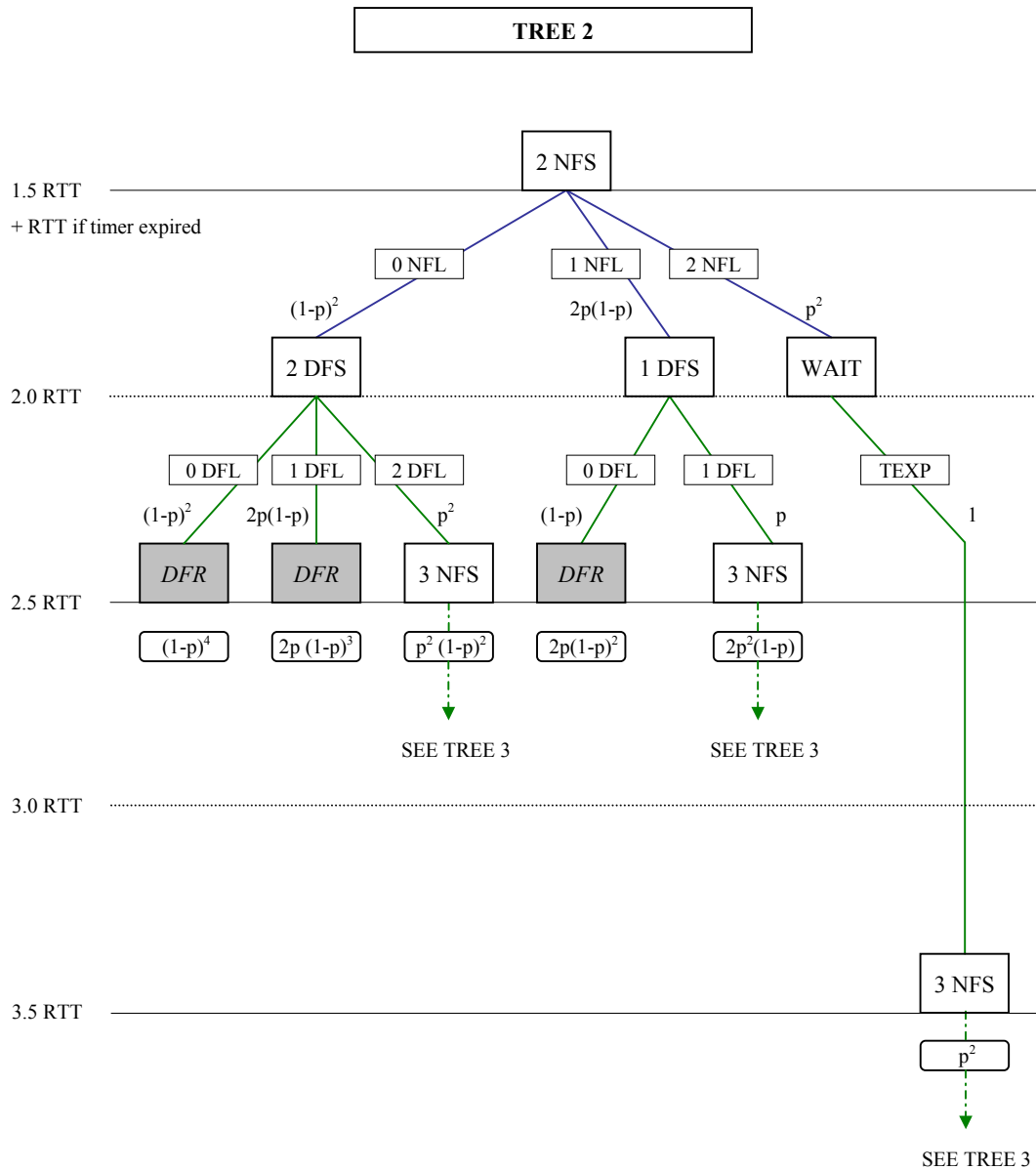


Figure 12: Tree 2 (RLP retransmission process)

4.5 IP datagram loss rate over RLP

Now the frame arrival probability p_{arr} after RLP retransmission process can be computed at a certain RTT. We only need to sum up the partial probabilities of the branches, which end with *DFR* (Data Frame Received) at the required time. (In the following terms we use p for the FER because we used p in our trees):

$$p_{arr}(0.5RTT) = 1 - p$$

$$p_{arr}(1.5RTT) = p(1 - p)^2$$

$$p_{arr}(2.5RTT) = [p^2(1 - p)] \cdot [(1 - p)^4 + 2p(1 - p)^3 + 2p(1 - p)^2]$$

$$p_{arr}(3.5RTT) = p^2 \cdot [(1 - p)^4 + 2p(1 - p)^3 + 2p(1 - p)^2] + [p^2(1 - p)] \cdot [p^2(1 - p)^2 + 2p^2(1 - p)] \cdot [(1 - p)^6 + 3p(1 - p)^5 + 3p^2(1 - p)^4 + 3p(1 - p)^4 + 6p^2(1 - p)^3 + 3p^2(1 - p)^2]$$

$$p_{arr}(4.5RTT) = [(p^2(1 - p)) \cdot p^2 + p^2 \cdot (p^2(1 - p)^2 + 2p^2(1 - p))] \cdot [(1 - p)^6 + 3p(1 - p)^5 + 3p^2(1 - p)^4 + 3p(1 - p)^4 + 6p^2(1 - p)^3 + 3p^2(1 - p)^2]$$

$$p_{arr}(5.5RTT) = p^2 \cdot p^2 \cdot [(1 - p)^6 + 3p(1 - p)^5 + 3p^2(1 - p)^4 + 3p(1 - p)^4 + 6p^2(1 - p)^3 + 3p^2(1 - p)^2]$$

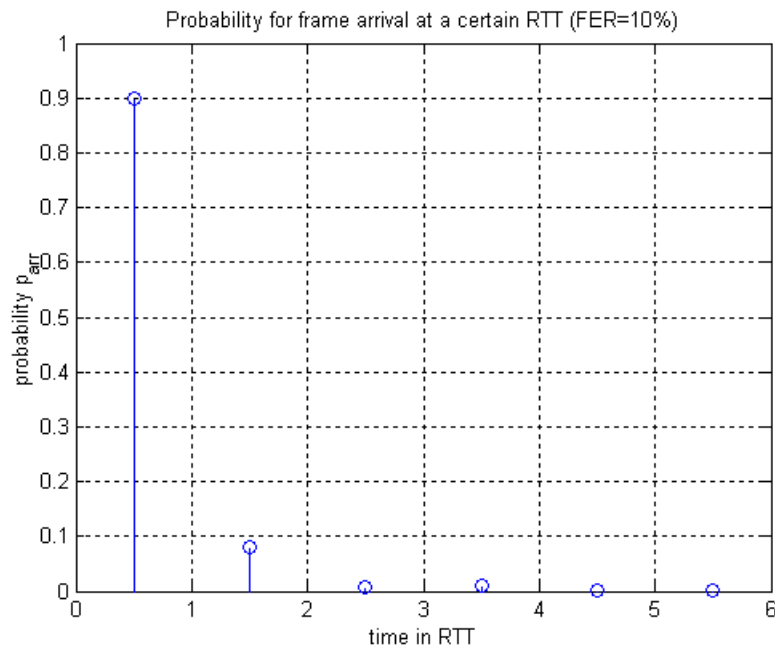


Figure 14: Frame arrival rate at a certain RTT

As we see in the upper graphic the probability that a frame needs to be retransmitted twice or even three times is almost zero.

Now we can compute the frame loss probability p_F after RLP retransmission process by subtracting the sum of all p_{arr} from one:

$$p_F = 1 - \sum p_{arr}$$

$$p_F = 1 - (1 - 64p^7 + 192p^8 - 240p^9 + 160p^{10} - 60p^{11} + 12p^{12} - p^{13})$$

$$p_F = 64p^7 - 192p^8 + 240p^9 - 160p^{10} + 60p^{11} - 12p^{12} + p^{13}$$

Lets say that one IP datagram consists of N_1 RLP data frames. So we are able to compute the IP datagram loss probability p_{lossIP} due to p and N_1 .

$$p_{lossIP} = 1 - (1 - p_F)^{N_1}$$

$$p_{lossIP} = 1 - \left[1 - (64p^7 - 192p^8 + 240p^9 - 160p^{10} + 60p^{11} - 12p^{12} + p^{13}) \right]^{N_1}$$

To get a numerical value for the drop rate for IP over RLP, please run Matlab file rlpLoss.m.

4.6 IP datagram delay over RLP

In this chapter, we discuss an analytical approach to derive the IP packet delay probability mass function and the standard deviation for IP over RLP. As we mentioned in the last subchapter the probability that a frame needs to be retransmitted twice or even three times is almost zero. Therefore the basic simplifying assumption is, that if a first frame gets lost no other ones are lost in the same transaction. In other words:

- a NAK frame never gets lost and
- a data frame sent due to a received NAK frame also never gets lost.

This is a reasonable assumption for small loss rates say $p < 5\%$ since the probability of such a loss is in the order of p^2 , which is very small.

Under this assumption, there are two independent delays, described through the number of additionally required times lots, n_1 and n_2 , that result in the overall additionally required time slots n . Consider the following figure that shows three consecutive IP datagrams:

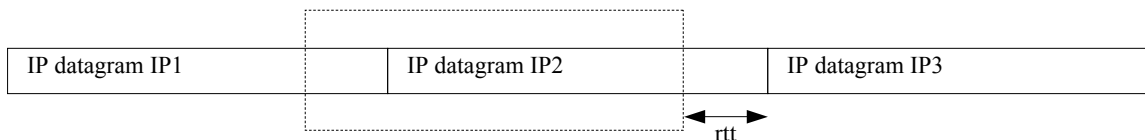


Figure 15: Three consecutive IP datagrams

The larger dashed window, which is of the same size as the IP datagrams, indicates the RLP frames which when lost, have a simple influence on the delay of the middle IP datagram IP2. Note that this includes frames from the previous IP datagram IP1. When a frame out of this window is lost, then it takes exactly one time slot more to transmit the middle IP datagram. The reason for this is, that under the above assumption, the transmitter knows about the lost frame after the first frame of the IP datagram IP3 has been transmitted, but before the last frame of the IP datagram IP3 has been transmitted. Note that it takes rtt (round trip time in number of frames) frames until the transmitter knows that the frame was lost.

If a frame out of the last rtt frames of IP datagram IP2 is lost, then the additional time it takes to send the middle IP datagram depends on the position of the lost frame. If it is very close to the end, then it adds more time than when it is very close to the beginning.

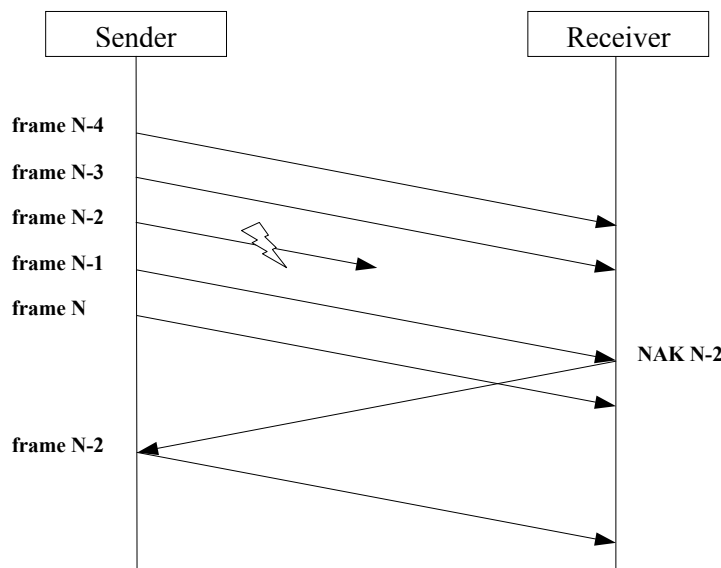


Figure 16: Retransmission, which needs more than one time slot

So a first difficulty occurs because one frame retransmission not always requires only one additional time slot. As seen, this phenomenon occurs at the end of the transmission of N frames. Supposed we have sent the second last frame (N-1) and a NAK frame for the prior frame arrives at the sender, we clearly see that we need more than one additional time slot to retransmit one frame.

This situation can be drawn as follows for $rtt = 4$, where the retransmitted frames are marked with R and the X's indicate don't cares, in the sense that it does not matter if they were lost or not.

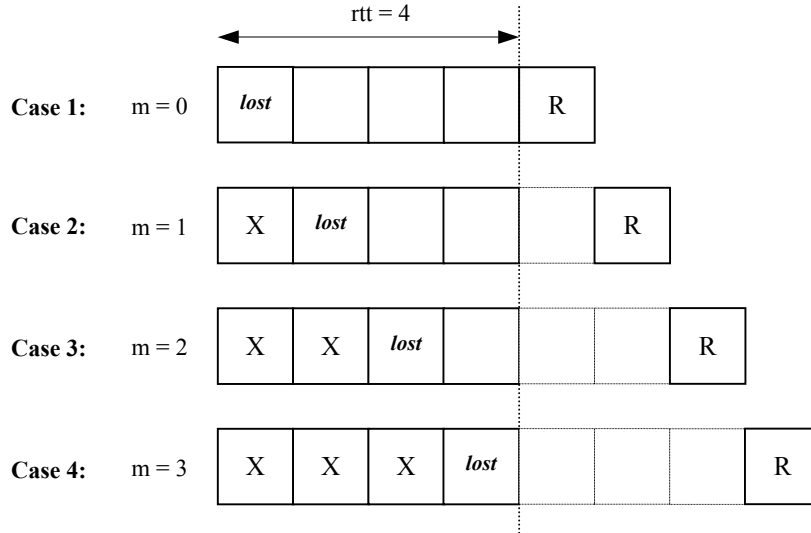


Figure 17: Frame loss at the end causes more than one additional time slot

We can now calculate the probability for each case:

$$P(case1) = \binom{m}{0} \cdot p^0 \cdot q^{m-0} \cdot p \cdot q^{rtt-1-m}$$

$$P(case2) = \left[\binom{m}{0} \cdot p^0 \cdot q^{m-0} + \binom{m}{1} \cdot p^1 \cdot q^{m-1} \right] \cdot p \cdot q^{rtt-1-m}$$

$$P(case3) = \left[\binom{m}{0} \cdot p^0 \cdot q^{m-0} + \binom{m}{1} \cdot p^1 \cdot q^{m-1} + \binom{m}{2} \cdot p^2 \cdot q^{m-2} \right] \cdot p \cdot q^{rtt-1-m}$$

$$P(case4) = \left[\binom{m}{0} \cdot p^0 \cdot q^{m-0} + \binom{m}{1} \cdot p^1 \cdot q^{m-1} + \binom{m}{2} \cdot p^2 \cdot q^{m-2} + \binom{m}{3} \cdot p^3 \cdot q^{m-3} \right] \cdot p \cdot q^{rtt-1-m}$$

As mentioned above, the number of added time slots is proportional to how late the frame is in the datagram. As seen before for each of the above four cases, there is a particular probability that it happens and hence we can calculate the probability mass function for the random variable n_2 describing this effect.

So the problem is now broken down into calculating the pmf of a first random variable n_1 , which is simply a binomial distribution (see next figure) and the pmf of a second random variable n_2 , which is describing the end of datagram effect. Since these two variables are independent and their sum is the total number of retransmitted frames n , the pmf of the total number of retransmitted frames n is simply the convolution of the two pmf's.

Calculating the binomial pmf of the delay of all the frames, which result in only one additional time slot when lost results in the pmf below. Note that Matlab does not start with index 0 for the first element of an array and matrix, so we always add the offset of 1 to the respective index. When

mapping these functions to Java, we need to remove this offset since Java starts with index 0 for the first element of an array or matrice.

$$pmf1(k+1) = \binom{N}{k} \cdot p^k \cdot q^{N-k} \quad k = 0 \dots N$$

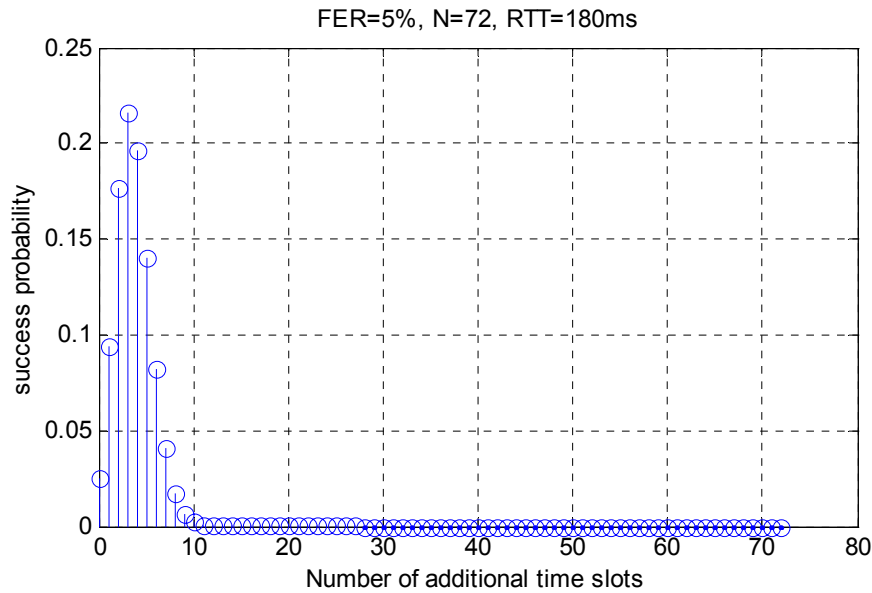


Figure 18: Number of additional time slots to transfer 72 RLP frames (one IP datagram)

Now we calculate the pmf of all the frames, which are at the end of an IP datagram.

$$pmf2(1) = q^{rtt}$$

$$pmf2(m+2) = p \cdot q^{rtt-1-m} \cdot \sum_{k=0}^m \binom{m}{k} \cdot p^k \cdot q^{m-k} \quad m = 0 \dots rtt - 1$$

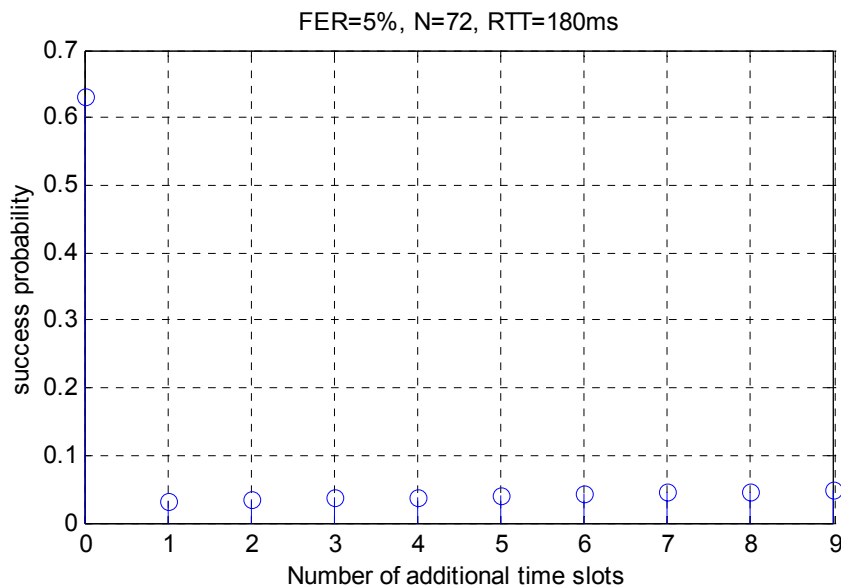


Figure 19: End of the datagram

Now we have the two pmf's. The next step is to convolve the two pmf's and to add the offset of 72 frames plus the forward channel delay of $(rtt-3)/2$. This results in the following pmf:

$$pmf(n + N + \frac{rtt-3}{2}) = (pmf1 * pmf2)(n + N + \frac{rtt-3}{2}) = \sum_j pmf1(j) \cdot pmf2(j + 1 - n) \quad n = 0 \dots N$$

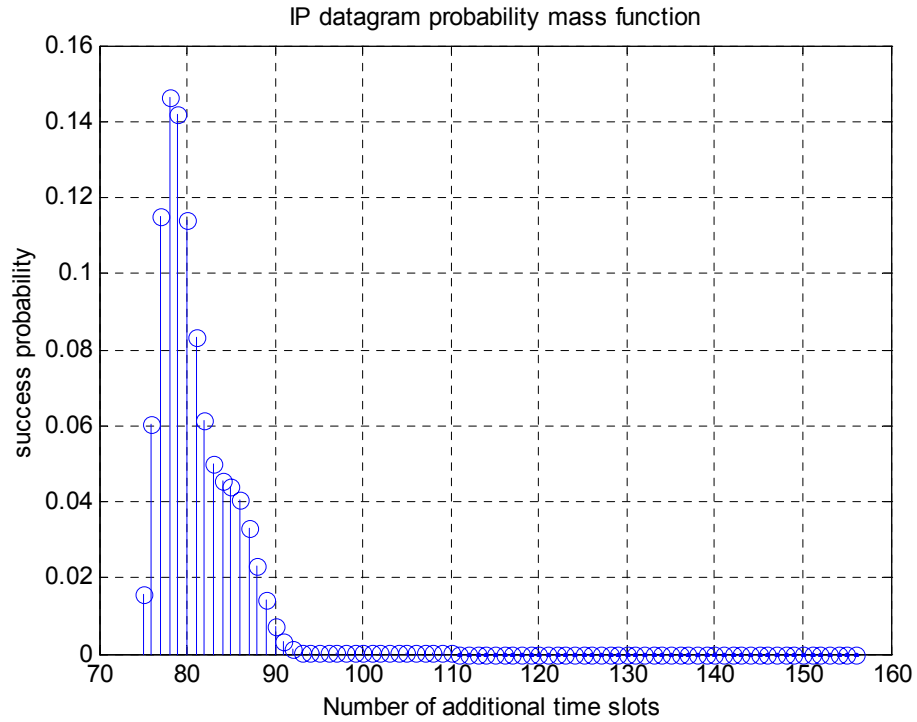


Figure 20: IP datagram PMF

To get a numerical value for the delay and the standard deviation for IP over RLP, please run Matlab file rlpDelay.m.

4.7 Traffic Channels

CDMA2000 provides means of transmitting data over two types of traffic channels, which are the fundamental channel (FCH) and the supplemental channel (SCH). The fundamental channel has a fixed low bandwidth, which is either 9.6kbps or 14.4kbps. The bandwidth of the supplemental channel is a multiple of that and could be as high as 32 times of the FCH bandwidth. The FCH is always assigned before the data transmission begins whereas the SCH is assigned on per needed basis. When SCH is being used we say that the transmission is in burst. There are two types of SCH assignments: finite and infinite, which will be referred to as finite burst and infinite burst. Infinite burst means that SCH can be used for transmitting data until a release command is issued. Finite burst mode of operation allows the use of the SCH when necessary (buffer to full).

The following perl script includes this behaviour in our calculation.

```
#####
#!/usr/bin/perl -w
# ARGV[0] = high_watermark_in_bytes          ARGV[7] = low_delay_in_ms
# ARGV[1] = low_watermark_in_bytes          ARGV[8] = low_std_in_ms
# ARGV[2] = eff_low_speed_in_bytes/s        ARGV[9] = low_droprate_in_%
# ARGV[3] = eff_high_speed_in_bytes/s       ARGV[10]= high_delay_in_ms
# ARGV[4] = source IP address               ARGV[11]= high_std_in_ms
# ARGV[5] = destination IP address          ARGV[12]= high_droprate_in_%
# ARGV[6] = mtu
#
# Example:
# adaptive 8000 4000 1020 10495 10.0.1.1 10.0.2.2 1500 1471 28 4e-7 145 12 5e-8
#####

# set up the constants and thresholds
#####
$sch_delay_on    = 1.5 ; # time in seconds to bring up supplemental channel sch
$sch_delay_off   = 0.02; # time in seconds to bring down sch

$MTU              = $ARGV[6];
$high_threshold  = int($ARGV[0]/$MTU+0.5); # high threshold in number of IP datagrams
$low_threshold    = int($ARGV[1]/$MTU+0.5); # low threshold in number of IP datagrams

#low speed in bytes/s
#####
$low_speed        = "cnistnet -a $ARGV[4] $ARGV[5] --delay $ARGV[7] $ARGV[8] --drop $ARGV[9]
                    --bandwidth $ARGV[2]";
$low_speed_uplink= "cnistnet -a $ARGV[5] $ARGV[4] --delay $ARGV[7] $ARGV[8] --drop $ARGV[9]
                    --bandwidth $ARGV[2]";

#high speed in bytes/s
#####
$high_speed       = "cnistnet -a $ARGV[4] $ARGV[5] --delay $ARGV[10] $ARGV[11]
                    --drop $ARGV[12] --bandwidth $ARGV[3]";
$system_state     = "low_on";

# reset the emulator and start it in low_speed for up and downlink
#####
system("cnistnet -d");
system("cnistnet -u");
system("$low_speed");
system("$low_speed_uplink");

# the adaption is only done in the downlink from the network to the client
#####
for(;;)
{
    # read the status of this connection
    #####
    open(F,"cnistnet -s $ARGV[4] $ARGV[5] |");
    @file=<F>;
    close(F);

    # parse output of the status line to find number of packets in buffer
    #####
    undef @entries;
    @elements=split(/ /,$file[2]);
    foreach $element (@elements)
    {
        if($element ne "")
        {
            push(@entries,$element);
        }
    }
}
```

```
}
$ysize=$entries[8];

# decision logic for switching from low_speed to high_speed and back
#=====
if ( ($ysize >= $high_threshold) && ($system_state eq "low_on") )
{
    $system_state="high_on";

    # since the addition of a SCH takes some time
    # we fork a process which waits that long before switching to higher speed
    #=====
    $pid = fork;
    if ($pid == 0)
    {
        select(undef,undef,undef,$sch_delay_on);
        system("$high_speed");
        exit;
    }
}
else
{
    if ( ($ysize <= $low_threshold) && ($system_state eq "high_on") )
    {
        $system_state = "low_on";

        # since the deletion of a SCH takes some time
        # we fork a process which waits that long before switching to lower speed
        #=====
        $pid = fork;
        if ($pid == 0)
        {
            select(undef,undef,undef,$sch_delay_off);
            system("$low_speed");
            exit;
        }
    }
}
}
```

4.8 Summary

4.8.1 IP Bit Rate

RLP reduces the IP datagram loss rate to nearly zero, achieving this by increasing the number of frames needed to transmit a given IP datagram. This means that RLP decreases the average bit rate by the same factor it increases the number of frames:

$$BitRate_{eff} = \frac{BitRate}{1 + p + 2p^2 + 6p^3} = \frac{MTU}{N_1 \cdot 20ms} \cdot \frac{1}{1 + p + 2p^2 + 6p^3}$$

4.8.2 IP Loss Rate

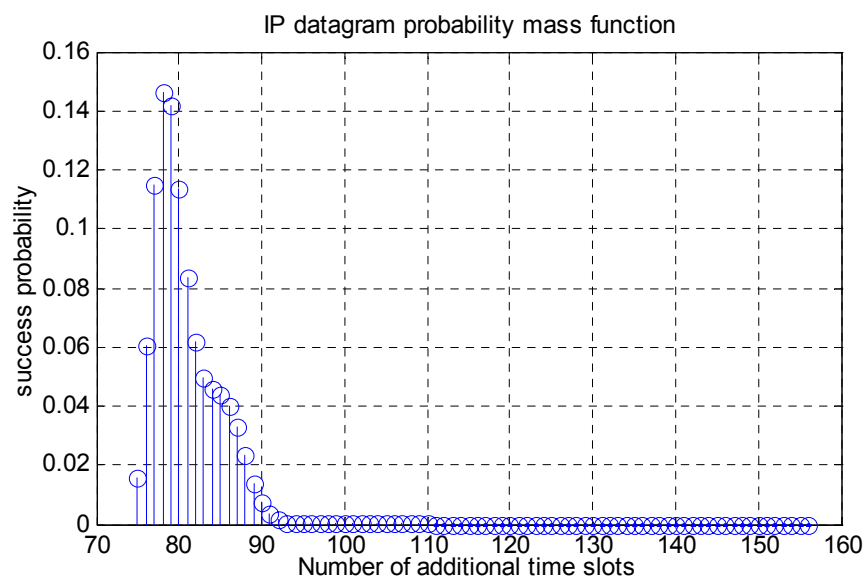
RLP minimizes the IP datagram loss rate by its frame retransmission process:

$$p_{lossIP} = 1 - \left[1 - (64p^7 - 192p^8 + 240p^9 - 160p^{10} + 60p^{11} - 12p^{12} + p^{13}) \right]^{N_1}$$

To get a numerical value for the drop rate for IP over RLP, please run Matlab file rlpLoss.m

4.8.3 IP Delay and Standard Deviation

The calculation of the delay and the standard deviation does not result in a simple formula, so we draw up the pmf graphically:



To get a numerical value for the delay and the standard deviation for IP over RLP, please run Matlab file rlpDelay.m

5 Radio Link Control (RLC)

5.1 Introduction

The over-the-air communication between the mobile station (MS) and the GPRS network is defined by the physical layer and the data link layer functionalities. The physical layer functions involve modulation/demodulation, channel coding/decoding, etc. The data link layer consists of two sublayers, which are logical link control (LLC) and radio link control / media access control (RLC/MAC). The LLC layer operates between the MS and the SGSN and provides a logical link between them. Packet data units (PDU) from higher layer are segmented into variable size LLC frames. An ARQ mechanism is provided at the LLC layer to retransmit erroneous LLC frames.

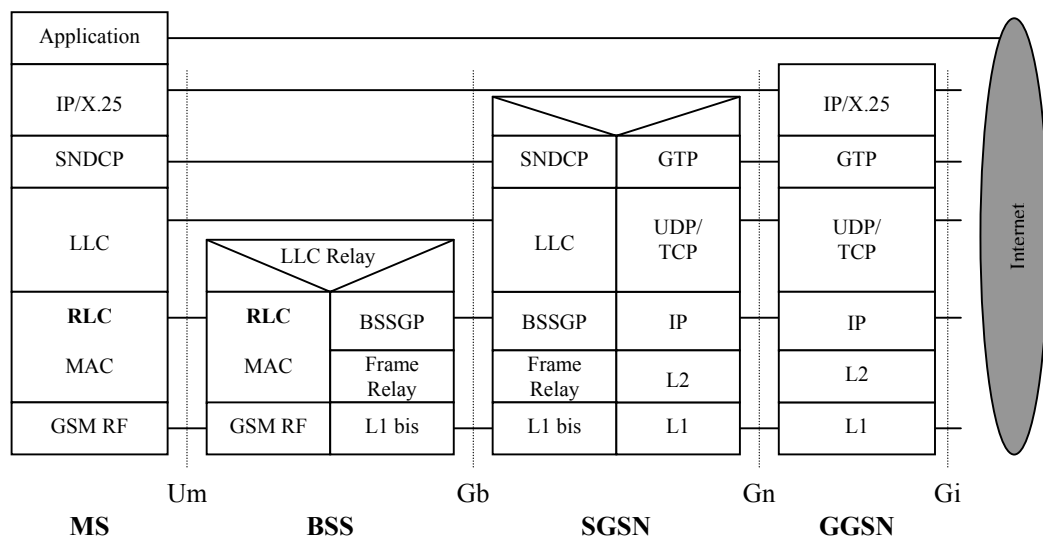


Figure 21: GPRS Protocol Stack

The RLC/MAC layer is primarily responsible for the efficient sharing of common radio resources by several mobile users. So it is in charge of adapting the packet-oriented transmission to the GSM physical layer. The uplink and downlink packet transfer procedures are handled independently by the RLC/MAC layer. The performance of GPRS networks strongly depends on the RLC/MAC layer operation and its efficiency and flexibility to manage the available radio resources. The RLC/MAC peers are at the MS and the BSS.

The RLC function provides for the selective retransmission of erroneous or lost RLC data blocks. These blocks can be transferred over the radio interface (Um) in two different modes, which are acknowledged mode and unacknowledged mode. RLC acknowledged mode operation uses retransmission of RLC data blocks to achieve high reliability. The transmitting side numbers the RLC data blocks via the block sequence number (BSN), which is used for retransmission and reassembly. The receiving side sends packet ACK/NACK messages to request retransmission of RLC data blocks. RLC unacknowledged mode operation does not utilize retransmission of RLC data blocks. The BSN in the RLC data block header is used to number the RLC data blocks for reassembly. The receiving side sends ACK/NACK messages to convey the necessary other control signaling.

As we are interested in the RLC retransmission process, we only take account of the RLC acknowledged mode.

5.2 Segmentation

In order to protect LLC peer to peer communication, PDU's coming from higher layers are segmented into variable size LLC frames and cyclic redundancy check (CRC) bits are added to every LLC frame for error detection. Each LLC frame is passed to the RLC/MAC layer, where it is

further segmented into RLC/MAC blocks of fixed size depending on the channel coding scheme (CS) used at the physical layer.

For our calculations we do not include the influence of LLC, since LLC layer user data retransmission is not necessary (even degrades the system performance) for TCP/IP traffic over GPRS/GSM. For detail please refer to [6].

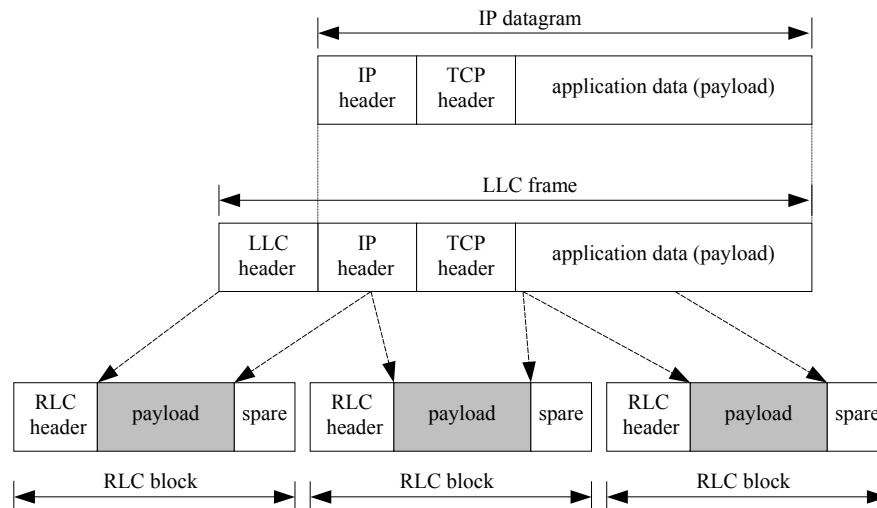


Figure 22: IP datagram split up into RLC blocks

5.3 Brief description of RLC data transmission

The transfer of RLC data blocks is controlled by a selective ARQ mechanism coupled with the modulo-128 numbering of the RLC data blocks within one temporary block flow (TBF). The sending side (either mobile client or network) transmits blocks within a window of 64 blocks (WS) and the receiving side periodically sends ACK/NACK messages.

Note that the uplink and downlink data transfer are handled differently. The network side sends ACK/NACK messages when needed, but the mobile client cannot deliver an ACK/NACK message without being requested from the network to do so. Therefore, the network side periodically sends piggyback a poll message in a RLC data block, which invokes the client to immediately deliver an ACK/NACK message to the network side.

Every such message acknowledges all correctly received RLC data blocks up to an indicated block sequence number (BSN), thus moving the beginning of the sending window on the sending side. Additionally, a bitmap that starts at the same RLC data block is used to selectively request lost RLC data blocks for retransmission. The sending side then retransmits the lost or erroneous RLC data blocks, eventually resulting in further sliding the sending window.

Note that the receiver acknowledges WS RLC data blocks at the same time. For this, it transmits an ACK/NACK message to the transmitter. This message contains the starting sequence number (SSN) and the RBB. The RBB contains the receive states of the last WS blocks. After the transmitter has received the ACK/NACK message, it knows which blocks need to be retransmitted.

5.4 RLC/MAC block structure

An RLC/MAC block containing an RLC data block may be encoded using one of the four available channel coding schemes. The size of the RLC data block for each of the channel coding schemes is shown beneath. The values are related to a single time slot. If a user is able to transmit over several packet data channels (PDCH) in parallel, the data rates have to be multiplied by the given number of time slots.

Channel Coding Scheme	RLC data block size	Number of spare bits	RLC data block size without spare bits	RLC payload [Bit]
-----------------------	---------------------	----------------------	--	-------------------

	[Bit]	[Bit]	[Bit]	
CS-1	176	0	176	160
CS-2	263	7	256	240
CS-3	307	3	304	288
CS-4	423	7	416	400

Figure 23: RLC data block size

The four coding schemes have been defined with different degrees of data protection with the purpose of adapting the transmission of the RLC/MAC data blocks to the different channel quality conditions. The more data protection is applied by the schemes the more redundancy is included. Therefore, more number of errors can be corrected. On the other hand, the more coding is applied the less quantity of data bits can be picked up per data block, which results in a lower throughput.

Note that different RLC/MAC block structures are defined for data transfers and for control message transfers. The RLC data block consists of an RLC header, an RLC data unit and some spare bits.

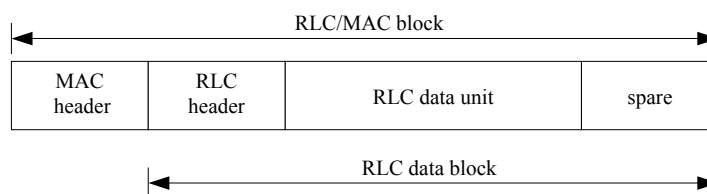


Figure 24: Block structure for data transfer

The RLC/MAC control block consists of a control message contents field and in the downlink direction an optional control header.

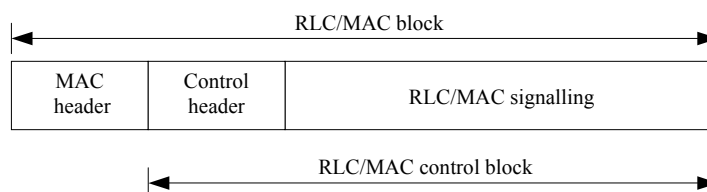


Figure 25: Block structure for control block

5.5 Procedures and Parameters

SNS	Sequence number space	<ul style="list-style-type: none"> Is set to 128 for GPRS
WS	Window size	<ul style="list-style-type: none"> Is set to 64 for GPRS
BSN	Block sequence number	<ul style="list-style-type: none"> Each RLC data block contains a BSN field. At the time an RLC data block is designated for transmission, BSN is set equal to V(S).
BSN'	Block sequence number'	<ul style="list-style-type: none"> BSN of most recently received RLC data block.
RBSN	Reduced block sequence number	<ul style="list-style-type: none"> Sequence number of RLC/MAC control block. At the time an RLC/MAC control block is designated for transmission, RBSN is set equal to V(CS).
RBB	Receive block bitmap	<ul style="list-style-type: none"> Part of ACK/NACK BSN in RBB = (SSN - bit position in bitmap) mod SNS BSN in RBB is valid if $V(A) \leq BSN < V(S)$
SSN	Starting sequence number	<ul style="list-style-type: none"> Part of ACK/NACK SSN=V(R)
ACK/NACK	Packet ACK/NACK message	<ul style="list-style-type: none"> ACK/NACK is sent by the RLC receiver and received by the RLC transmitter. Contains SSN and RBB.



V(S)	Send state variable	<ul style="list-style-type: none"> ▪ Denotes sequence number of the next RLC data block to be transmitted. ▪ Shall be set to 0 at beginning. ▪ Incremented by one after transmission of RLC data block with $BSN=V(S)$. ▪ $(V(S)-V(A) \bmod SNS) \leq WS$ ▪ $V(S)=[0 \dots SNS-1]$
V(CS)	Control send state variable	<ul style="list-style-type: none"> ▪ Denotes sequence number of next RLC/MAC control block to be transmitted for control action. ▪ $V(CS)=[0,1]$
V(A)	Acknowledge state variable	<ul style="list-style-type: none"> ▪ Contains the BSN value of the oldest RLC data block that has not been positively acknowledged. ▪ Shall be set to 0 at beginning. ▪ Shall be updated from the values received from RBB of ACK/NACK ▪ $(V(S)-V(A) \bmod SNS) \leq WS$ ▪ $V(A)=[0 \dots SNS-1]$
V(B)	Acknowledge state array	<ul style="list-style-type: none"> ▪ Array of SNS indicating the acknowledgement status of WS previous RLC data blocks. ▪ Shall be updated from the values received from RBB of ACK/NACK ▪ $V(B)[i]=[NACKED,PEND_ACK,ACKED,INVALID]$ NACKED: data block not acknowledged PEND_ACK: after transmitting data block ACKED: data block acknowledged INVALID: outside the active transmit window
V(R)	Receive state variable	<ul style="list-style-type: none"> ▪ Denotes BSN which has a value one higher than the highest BSN yet received. ▪ Represents end of receiver window. ▪ Shall be set to 0 at beginning. ▪ $V(R)=[0 \dots SNS-1]$ ▪ Shall be set to $BSN'+1 \bmod SNS$, provided RLC data block was error free and $V(R) \leq BSN' < V(Q)+WS$
V(Q)	Receive window state variable	<ul style="list-style-type: none"> ▪ Denotes lowest BSN not yet received. ▪ Represents start of receiver window. ▪ Shall be set to 0 at beginning. ▪ $V(Q)=[0 \dots SNS-1]$ ▪ Shall be updated when receiver gets data block whose $BSN=V(Q)$. $V(Q)$ shall be set to the BSN in receive window that has not been yet received and which minimizes $(BSN-V(R)) \bmod SNS$ or it shall be set to $V(R)$ if all data blocks in the receive window have been received properly.
V(N)	Receive state array	<ul style="list-style-type: none"> ▪ Array of SNS indicating the receive status of WS previous RLC data blocks. ▪ $V(N)[i]=[RECEIVED,INVALID]$ RECEIVED: block received with $(V(Q) \leq BSN < V(R))$ INVALID: element outside the active window

5.6 Assumptions

For the calculations further on we use the following assumptions.

- ❑ The block error rate (BER) can be set to a value between 1% and 20%, but it is constant over the transmission of IP datagrams.
- ❑ To get the RLC block size and payload we use the values, which result when using channel coding scheme CS-1.
- ❑ We do not include the effect of the LLC layer since LLC does not help in improving the throughput. For our calculations we assume that one LLC block is 1520 Bytes of length, so the whole IP datagram can be packed into a single LLC frame.
- ❑ We only take account of the RLC acknowledged mode.
- ❑ M denotes the maximum number of retransmission rounds of an RLC data block.
- ❑ The number of used packet data channels (PDCH) is set to 1.
- ❑ The ACK/NACK message is delivered due to an RLC data block with Pollbit=1.
- ❑ RLC blocks with Pollbit=1 and the following ACK/NACK message never get lost.

5.7 Number of RLC blocks per IP datagram

Since we use the values from channel coding scheme CS-1, the RLC block has the following structure.

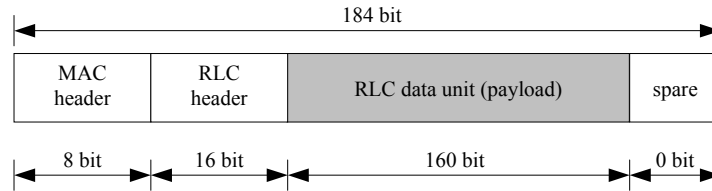


Figure 26: RLC block structure for CS-1

By assuming that an IP datagram consists of 1500 bytes (MTU) it has to be split up into $N_1=76$ RLC data blocks:

$$N_1 = \frac{MTU + LLCheader}{RLCpayload} = \frac{1500Byte + 20Byte}{160bit / 8} = 76$$

We can now calculate the bit rate for IP over RLC if we assume that the block error rate is zero (BER=0).

$$BitRate = chosenBitRate \cdot \frac{RLCpayload}{RLCframeSize} = 9600bps \cdot \frac{160bit}{184bit} = 1043Byte/s$$

This reduced bit rate occurs because of the RLC header and trailer transmissions. But this value is still not the effective bit rate on the link since we need to transmit more than only N_1 RLC data blocks per IP datagram because of the lossy radio link (BER).

5.8 Effective bit rate for IP over RLC

In this subchapter we present an analytical approach to calculate the average number of RLC data blocks needed for an IP packet over RLC. The presented approach has the advantage that it results in a simple formula.

Let N_1 be the number of RLC blocks for a given IP datagram. As we have seen before, N_1 depends on the MTU of the link and the channel coding scheme. Taking the previous example (MTU=1500Byte, CS-1), it takes $N_1=76$ RLC data blocks to transmit the entire IP datagram when no block is lost.

In this subchapter, we think of the RLC retransmission algorithm in the following fashion, which is not the way RLC works, but for calculating the average number of RLC blocks for an IP packet, this point of view is accurate and simplifies the calculation.

First all RLC blocks get sent. This is the first round. We now assume that no ACK/NACK messages are lost. Now the ACK/NACK message of the first round are evaluated and all blocks that did not make it in the first round are retransmitted. This is the second transmission round (first retransmission round) and the number of blocks sent in the second round is a random variable, which we call N_2 . Note that this point of view assumes that after we sent the last block of the IP datagram, we know which blocks did not make it in this round. This is a correct assumption for the blocks at the beginning of the IP datagram, but an incorrect one for the last few blocks, for which the ACK/NACK message might still be in the network. While this effect is very important for the IP datagram delay, for calculating the average number of blocks for the purpose of finding the effective bit rate of the RLC link, this point of view does not change the result.

After the second round, we evaluate the ACK/NACK message of the second round and send the blocks, which did not make it at the third round. The number of sent blocks in the third round is also a random variable and we call it N_3 . Again, after the third round, we evaluate the ACK/NACK message and send the blocks that did not make it again. The number of sent blocks in the fourth round is also a random variable and we call it N_4 . We assume that the maximum number of retransmissions for an RLC block is set to $M=3$, so RLC now gives up.

Let p be the probability that a block is lost (Block error rate, BER) and q the probability that a block made it ($1-p$). Then the distribution of N_2 (blocks sent in the second round = blocks lost in the first round) is a binomial distribution with parameters N_1 and p . Hence its mean is simply $N_1 \cdot p$. The distribution of N_3 (blocks sent in the third round = blocks lost in the second round) is also a binomial distribution with parameters N_2 and p . Note that N_2 itself is a random variable. The distribution of N_4 (blocks sent in the fourth round = blocks lost in the third round) is also a binomial distribution with parameters N_3 and p . Note that N_3 itself is a random variable.

We are now interested in N , which denotes the average total number of blocks transmitted:

$$\begin{aligned} N &= E[N_1 + N_2 + N_3 + N_4] = N_1 + E[N_2 + N_3 + N_4] = N_1 + E[N_2] + E[N_3] + E[N_4] = \\ N &= N_1 + N_1 \cdot p + E[N_2 \cdot p] + E[N_3 \cdot p] = N_1 + N_1 \cdot p + N_1 \cdot p^2 + E[N_2 \cdot p] \cdot p = \\ N &= N_1 + N_1 \cdot p + N_1 \cdot p^2 + N_1 \cdot p^3 = N_1 \cdot (1 + p + p^2 + p^3) \end{aligned}$$

So:

$$N(p, M = 3) = N_1 \cdot (1 + p + p^2 + p^3)$$

Following the approach from above, we can calculate the increase of RLC blocks depending on the maximum number of retransmissions M for an RLC block. Then the general form of N is:

$$N(p, M) = N_1 \cdot \sum_{i=0}^M p^i \quad \text{where } p \text{ denotes the block error rate BER}$$

This formula shows the effect RLC has. While it reduces the IP packet loss to nearly zero (next chapter), it achieves this by increasing the number of blocks needed to transmit a given IP datagram, which is given by the following formula:

$$inc(p, M) = \frac{N(p, M)}{N_1} = \sum_{i=0}^M p^i$$

The increase is, as expected a function of the block error rate BER and the number of retransmission rounds k. Please refer to the following graph, which shows the increase of blocks depending on BER and k.

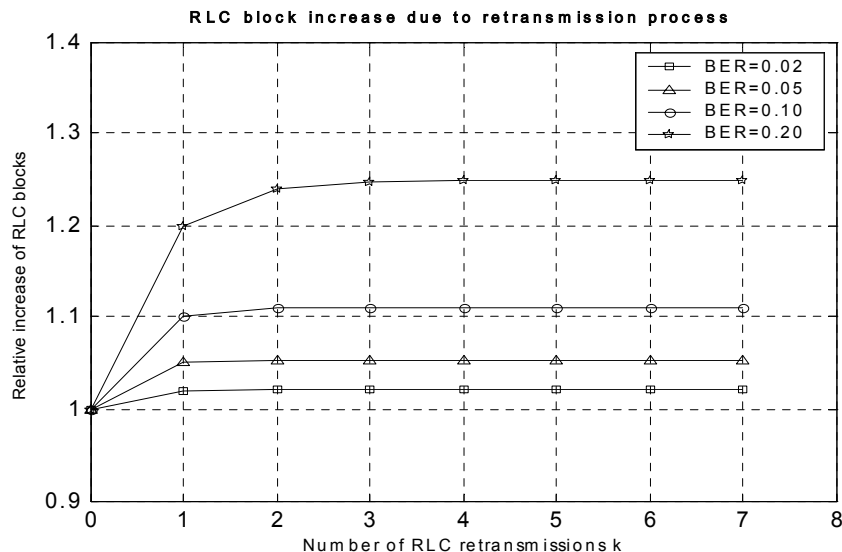


Figure 27: Increase of data blocks

For BER=0.02 the increase is small, about 2%, since the linear term dominates. For BER=0.05 it is already 5%, since the higher order terms are taking effect and for BER=0.10 the increase is 11% and it gets worse as we go on.

Now we are able to calculate the effective bit rate on the RLC link. Since RLC increases the number of transmitted blocks by $inc(p,k)$, it decreases the bit rate by the same factor.

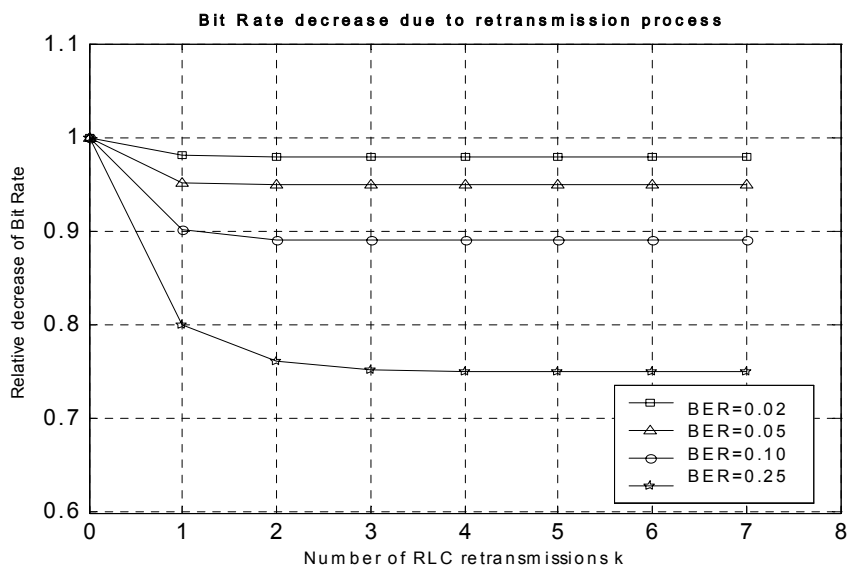


Figure 28: Decrease of bit rate

$$BitRate_{eff} = BitRate \cdot inc(p, M)^{-1} = BitRate \cdot \left(\sum_{i=0}^M p^i \right)^{-1}$$

Note that we set the maximum number of retransmissions for an RLC block to M. Not setting this threshold leads to the following formula:

$$N = N_1 \cdot (1 + p + p^2 + p^3 + \dots + p^\infty)$$

$$N = N_1 \cdot \sum_{k=0}^{\infty} p^k$$

$$N = N_1 \cdot \frac{1}{1 - p}$$

Therefore, the increase of RLC data blocks is given by:

$$inc(p, M = \infty) = \frac{1}{1 - p}$$

Hence:

$$BitRate_{eff} = BitRate \cdot inc(p, M = \infty)^{-1} = BitRate \cdot (1 - p)$$

5.9 IP datagram loss rate over RLC

The main objective of this chapter is to define analytically an expression for the IP datagram loss rate over the radio link with RLC. At first we need to understand Bayes' Theorem, which is a result that allows new information to be used to update the conditional probability of an event:

$$P(A \cap B) = P(A) \cdot P(B|A) = P(B) \cdot P(A|B) \quad \text{where}$$

$P(A)$ = probability that event A occurs

$P(B)$ = probability that event B occurs

$P(A|B)$ = probability that event A occurs given that event B has already occurred

$P(B|A)$ = probability that event B occurs given that event A has already occurred

N_1 denotes the IP datagram size in RLC blocks, BER the block error rate and k the number of RLC block retransmissions, where $k=0$ refers to the original RLC block transmission and $k=1$ to the first retransmission (second transmission). We now proceed to calculate the probability that all N_1 RLC blocks of an IP datagram are received correctly at the original transmission. It is equal to:

$$P(k = 0) = (1 - BER)^{N_1}$$

With the help of Bayes' Theorem we can calculate $P(k=1)$. $P(k=1)$ is the probability that the IP datagram does not arrive after the original transmission of the N_1 data blocks but arrives correctly after the first retransmission. So,

$$P(k = 1) = P(A) \cdot P(B|A) = (1 - (1 - BER)^{N_1}) \cdot (1 - BER)^{BER \cdot N_1} \quad \text{where}$$

$P(A)$ refers to the probability that the IP datagram is not complete after the original transmission and $P(B|A)$ to the probability that the IP datagram is complete after the first retransmission given that it was not complete after the original transmission.

$P(k=2)$ is the probability that the IP datagram does not arrive after the first retransmission of the lost data blocks but arrives correctly after the second retransmission. So,

$$P(k = 2) = P(C) \cdot P(D|C) = (1 - (1 - BER)^{N_1}) \cdot (1 - (1 - BER)^{BER \cdot N_1}) \cdot (1 - BER)^{BER^2 \cdot N_1} \quad \text{where}$$

$P(C)$ refers to the probability that the IP datagram is not complete after the first retransmission and $P(D|C)$ to the probability that the IP datagram is complete after the second retransmission given that it was not complete after the first retransmission. Following the approach from above, the general form of the probability mass function $P(k)$ is:

$$P(k = 0) = (1 - BER)^{N_1}$$

$$P(k > 0) = (1 - BER)^{BER^k \cdot N_1} \cdot \prod_{j=0}^{k-1} (1 - (1 - BER)^{BER^j \cdot N_1}) \quad k = 1, 2, \dots, M$$

where M denotes the maximum number of retransmission rounds for an RLC data block. Combining $P(k=0)$ and $P(k>0)$ in one PMF $P(k)$ can be achieved by using the signum function: Therefore,

$$P(k) = (1 - BER)^{BER^k \cdot N_1} \cdot \left(\prod_{j=0}^{k-1} 1 - (1 - BER)^{BER^j \cdot N_1} \right)^{\text{sign}(k)} \quad k = 0, 1, 2, \dots, M$$

We can now calculate the total probability that an IP datagram is successfully delivered. Summing up the probabilities that all blocks arrived correctly up to a certain number of retransmission rounds M is equal to:

$$p_{arr} = \sum_{k=0}^M P(k) = \sum_{k=0}^M (1 - BER)^{BER^k \cdot N_1} \cdot \left(\prod_{j=0}^{k-1} 1 - (1 - BER)^{BER^j \cdot N_1} \right)^{sign(k)}$$

Hence

$$p_{lossIP} = 1 - p_{arr} = 1 - \sum_{k=0}^M (1 - BER)^{BER^k \cdot N_1} \cdot \left(\prod_{j=0}^{k-1} 1 - (1 - BER)^{BER^j \cdot N_1} \right)^{sign(k)}$$

To illustrate this approach graphically, we again draw up a decision tree.

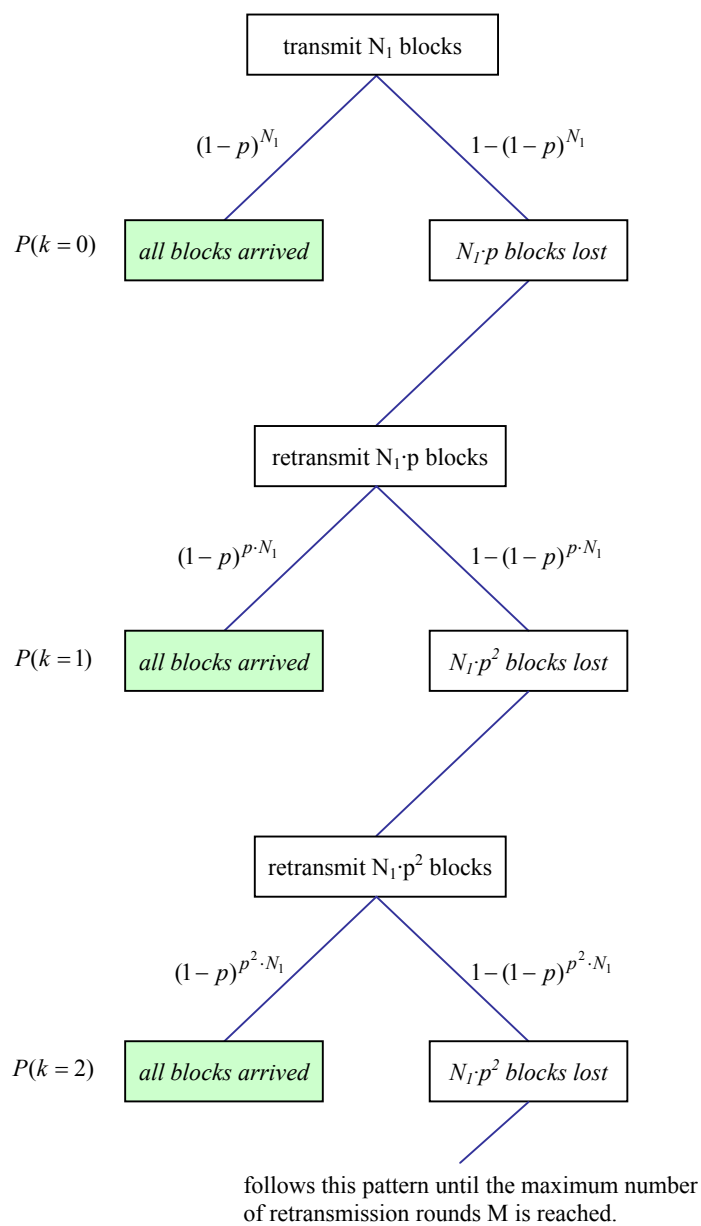


Figure 29: IP datagram success probability tree

The following figure shows the probability mass function $P(k)$ and the cumulative distribution function depending on M and BER for an IP datagram size of $N_1=76$ RLC data blocks.

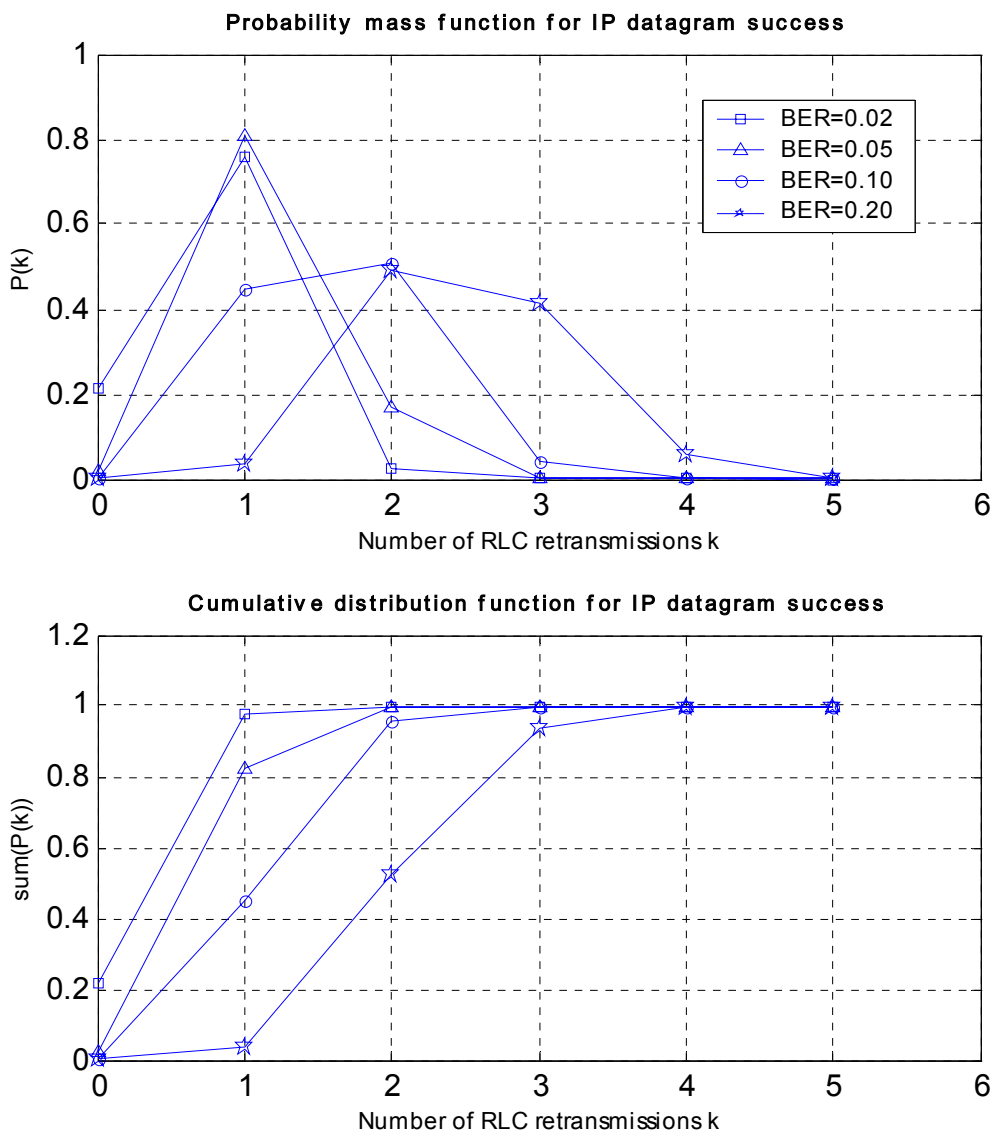


Figure 30: PMF, CDF for IP datagram success

To get a numerical value for drop rate for IP over RLC, please run Matlab file rlcLoss.m.

5.10 IP datagram delay over RLC

In this chapter we present an analytical approach to calculate the probability mass function (PMF) for the IP datagram delay time over RLC. The basic simplifying assumption is that an RLC data block can be retransmitted only twice (M=2) and that the RLC data block with Pollbit set and its ACK/NACK message never get lost.

Let n_1 be the number of RLC data blocks for a given IP datagram, p the probability that a block is lost (BER) and q the probability that a block made it ($1-p$).

First all RLC blocks get sent. This is the original transmission round. Now all ACK/NACK messages of this original round are evaluated and all blocks that did not make it in this original round are resent. This is the first retransmission round and the number of blocks sent in this round is a random variable, which we call N_2 . The distribution of N_2 is a binomial distribution with parameters n_1 and p . Now we can calculate the PMF for N_2 . The distribution of N_3 , which denotes the distribution of blocks sent in the second retransmission round depends on the distribution of N_2 . Hence,

$$P(N_2 = n_2) = \binom{n_1}{n_2} \cdot p^{n_2} \cdot q^{n_1-n_2} \quad n_2 = 0,1,\dots,n_1$$

$$P(N_3 = n_3) = \binom{N_2}{n_3} \cdot p^{n_3} \cdot q^{N_2-n_3} \quad n_3 = 0,1,\dots,n_1$$

We are now interested in the resulting PMF for random variable N , which denotes the distribution of the total number of blocks for the whole IP datagram.

$$P(N = n_1 + n_2 + n_3) = P(N_3 = n_3 | N_2 = n_2) \cdot P(N_2 = n_2)$$

$$P(N = n_1 + n_2 + n_3) = \binom{n_2}{n_3} \cdot p^{n_3} \cdot q^{n_2-n_3} \cdot \binom{n_1}{n_2} \cdot p^{n_2} \cdot q^{n_1-n_2} \quad n_2 = 0,1,\dots,n_1 \text{ and } n_3 = 0,1,\dots,n_2$$

This PMF is correct for the blocks at the beginning of the IP datagram, but it is not correct for those very close at the end of the IP datagram, since a block loss can only be detected when a ACK/NACK message arrives. So a single block loss at the end can cause more than only one additional time slot to retransmit it. This effect depends on the poll interval and on the roundtrip time of the blocks.

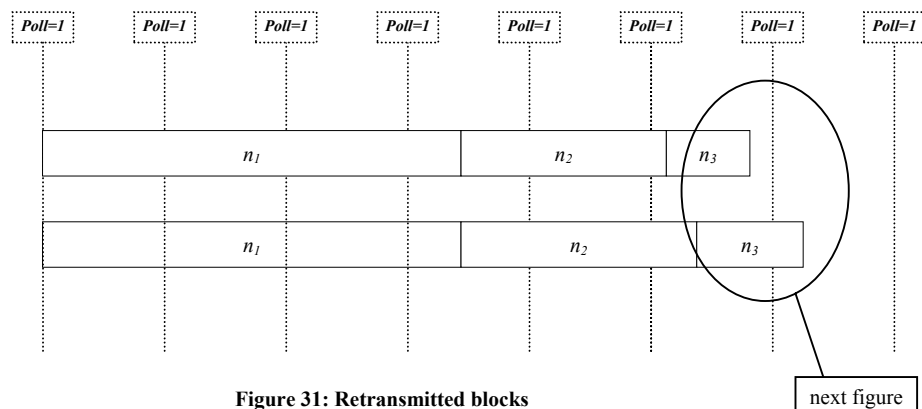


Figure 31: Retransmitted blocks

To simplify matters we assume that a block loss at the end of n_1 never causes more than one additional time slot even though this could be possible. Therefore, all blocks, including the retransmitted ones, are consecutive.

Now the end of the IP datagram shall be analysed. The hatched rectangle illustrates the last block to transmit whereas the squared rectangle indicates a block loss. The time between sending the last block and receiving the acknowledgement or retransmitting the lost blocks consists of two parts. A first delay is the RTT, which does not depend on the position of the last block, a second delay is variable and depends on the position of the last block. Note that the transmitter has a delay of one block when it receives an ACK/NACK message before it can retransmit the lost blocks.

To get the total distribution for this effect, we have to calculate the distribution for the blocks in the last poll interval for each case, add the variable offset, sum up all distributions and divide it by the number of cases. For details please refer to matlab file rlcDelay.m.

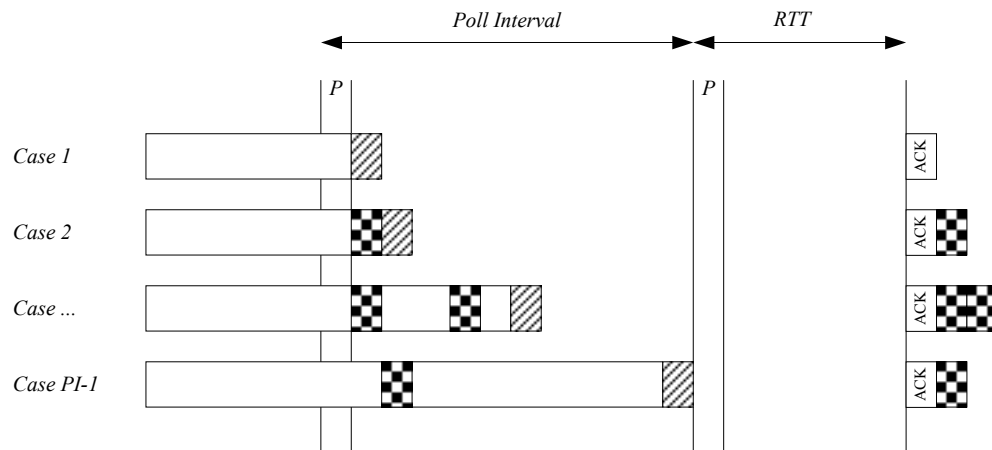


Figure 32: Block loss in last poll interval

There is another effect we did not yet include in our calculation. We not only take account of the blocks in the last poll interval but also of those in the second last poll interval.

The hatched rectangle again illustrates the last block to transmit. If losing a block (squared rectangle) in the second last poll interval, we need to wait RTT blocks until we know that it has been lost. So a block loss results in waiting a variable time, depending on the position of the last block. If the last block is more than RTT blocks away from the block with pollbit=1 set, we only wait one additional time slot.

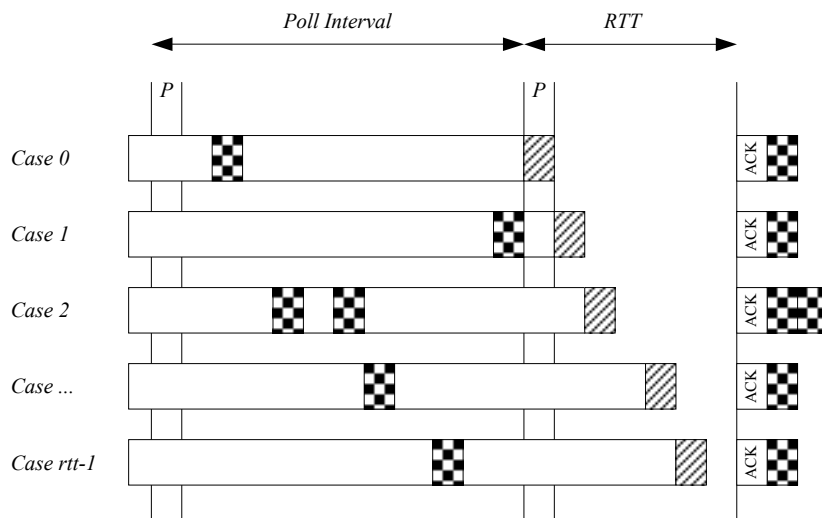


Figure 33: Block loss in second last poll interval

The following four figures refer to the ideas and calculations from above. The first picture shows the PMF for the number of additional time slots needed to transmit all blocks. Note that this picture refers to the idea that one block loss only causes one additional time slot. This is not correct and therefore needs to be extended by the effects at the end of an IP datagram, which are responsible for additional delays.

The end of the IP datagram is described by the next two figures. The one top right shows the PMF for block losses in the last poll interval, the one bottom left illustrates the PMF for block losses in the second last poll interval.

Convolving the first three PMF's and adding the offset of $n_1=68$ blocks and the channel delay of $(rtt-2)/2$ results in the total PMF. So the last figure shows the effective number of blocks to transmit for a given IP datagram with $n_1=68$ blocks. Both the average delay and the standard deviation can now easily be derived from the total PMF.

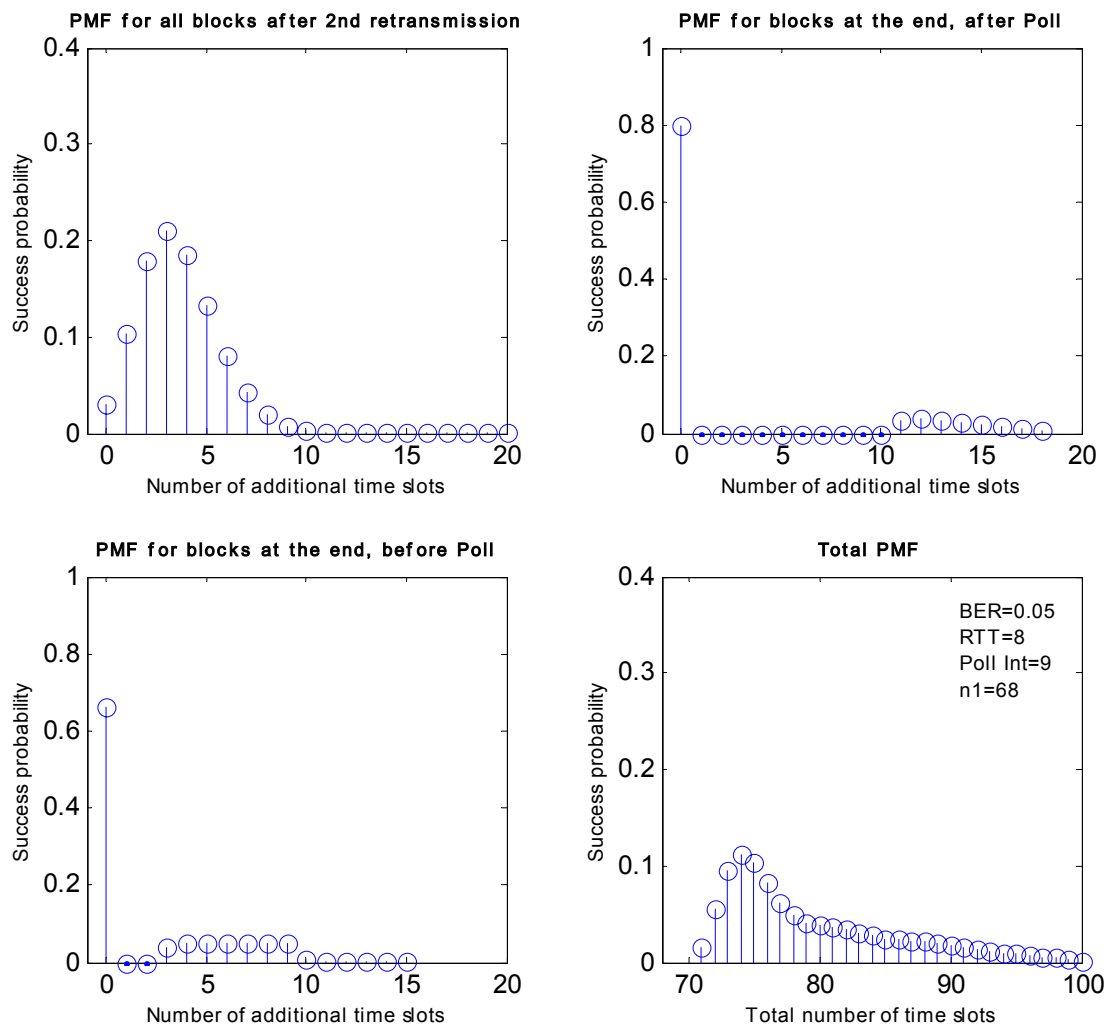


Figure 34: PMF's for IP datagram

The next four figures present the total PMF when increasing the block error rate. If BER rises, more blocks have to be retransmitted, which means that it takes more time slots until an IP datagram is transmitted completely. The PMF therefore gets flattened but wider.

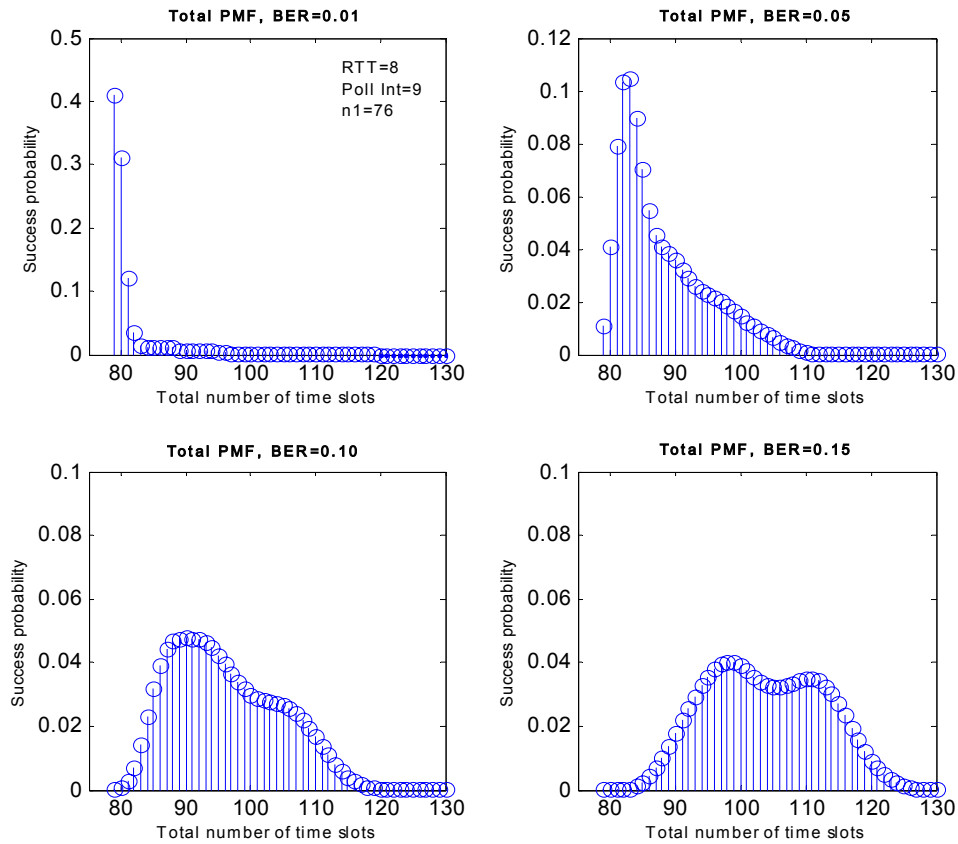


Figure 35: PMF for increasing BER

So with a rising BER, both the average delay and the standard deviation of an IP datagram increase.

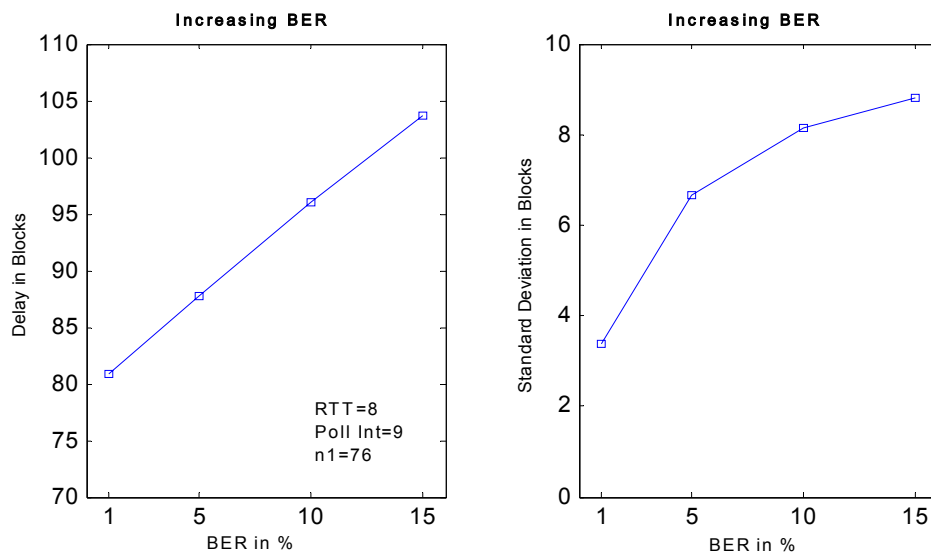


Figure 36: Increasing BER

The next four figures present the total PMF when increasing the roundtrip time. We can clearly see that the average delay rises the higher the RTT is.

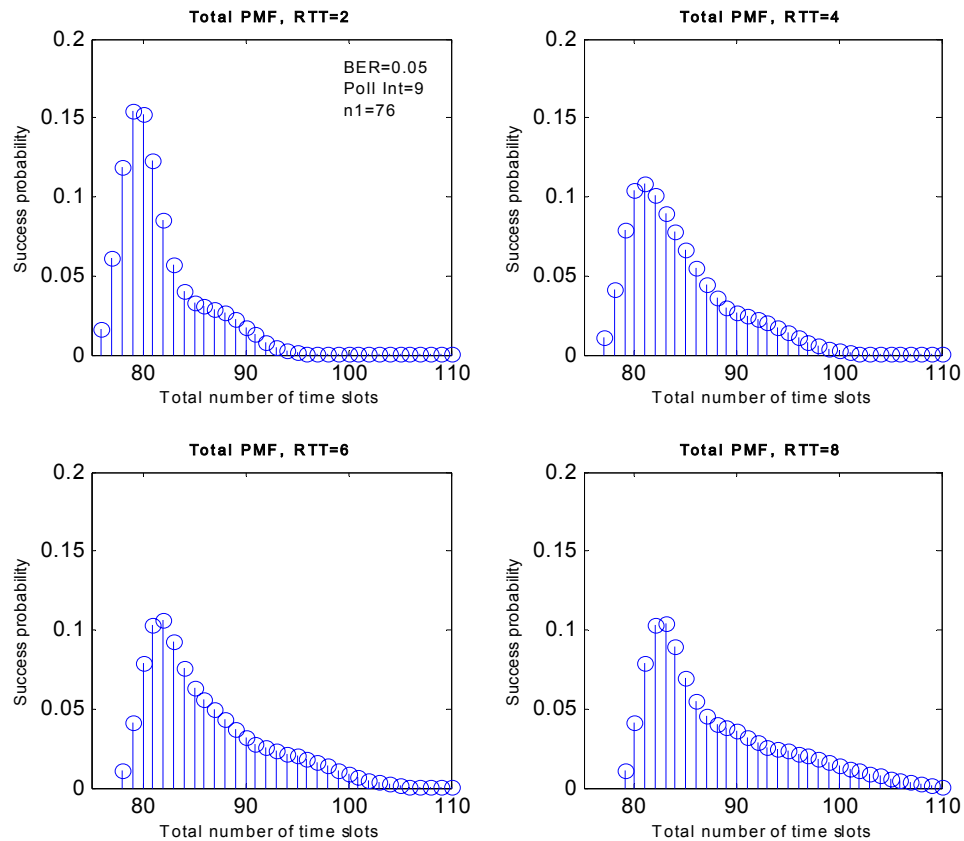


Figure 37: PMF for increasing RTT

If the system has a higher RTT, the average delay of the IP datagram rises at least by the additional channel delay, which is $(rtt-2)/2$. If the two graphs below are studied, we see that the delay rises by more than only the channel delay ($delay_{RTT=2}=0$, $delay_{RTT=4}=1$).

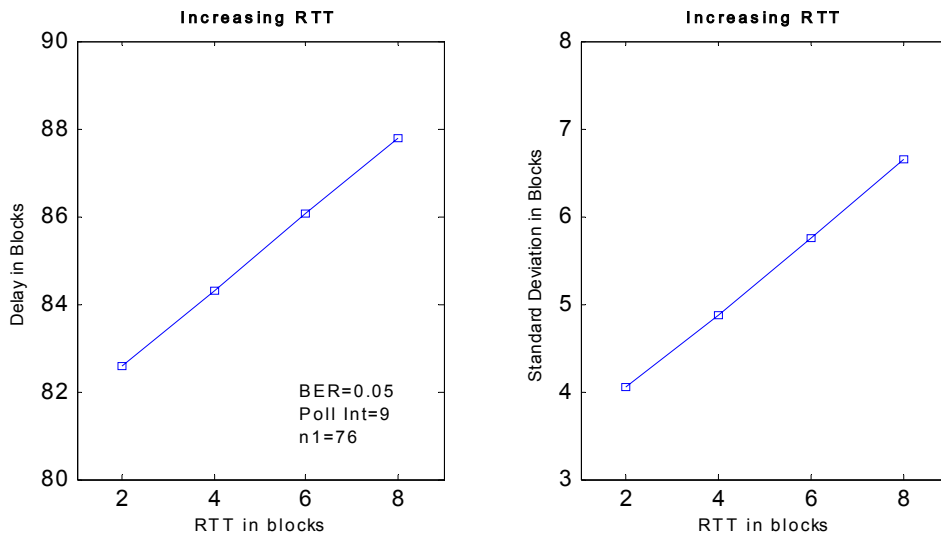


Figure 38: Increasing RTT

If the poll interval is too long, the transmitted blocks are not acknowledged frequently enough at the end of the IP datagram, which causes an additional delay.

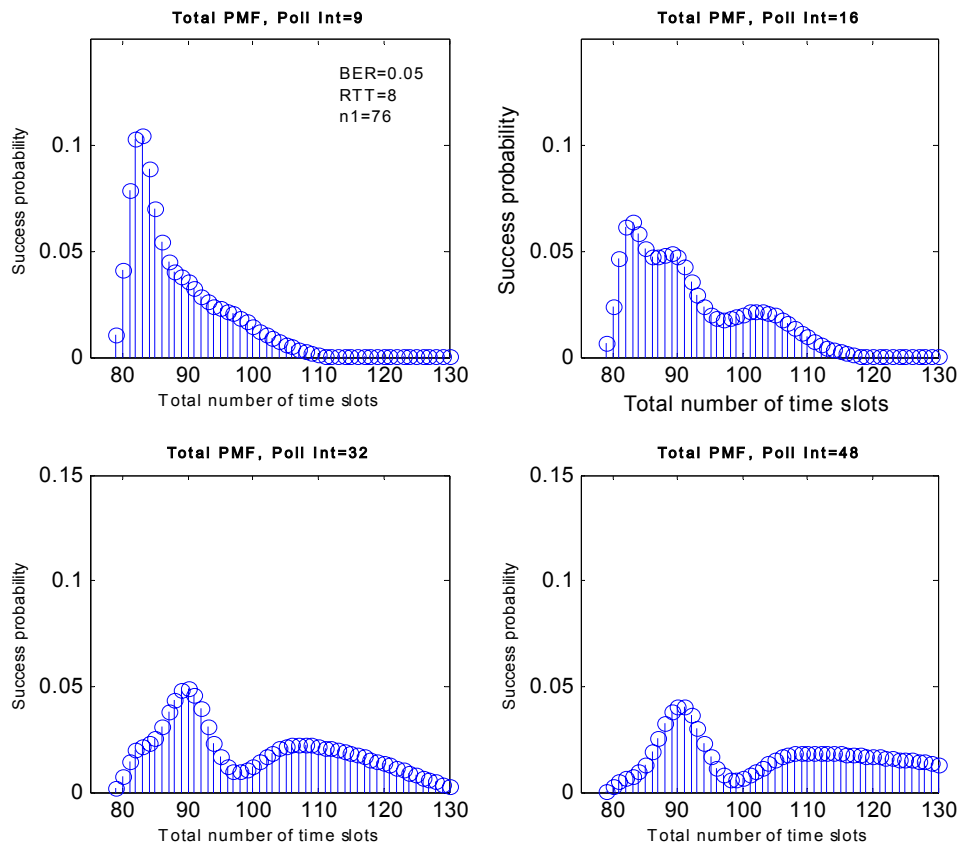
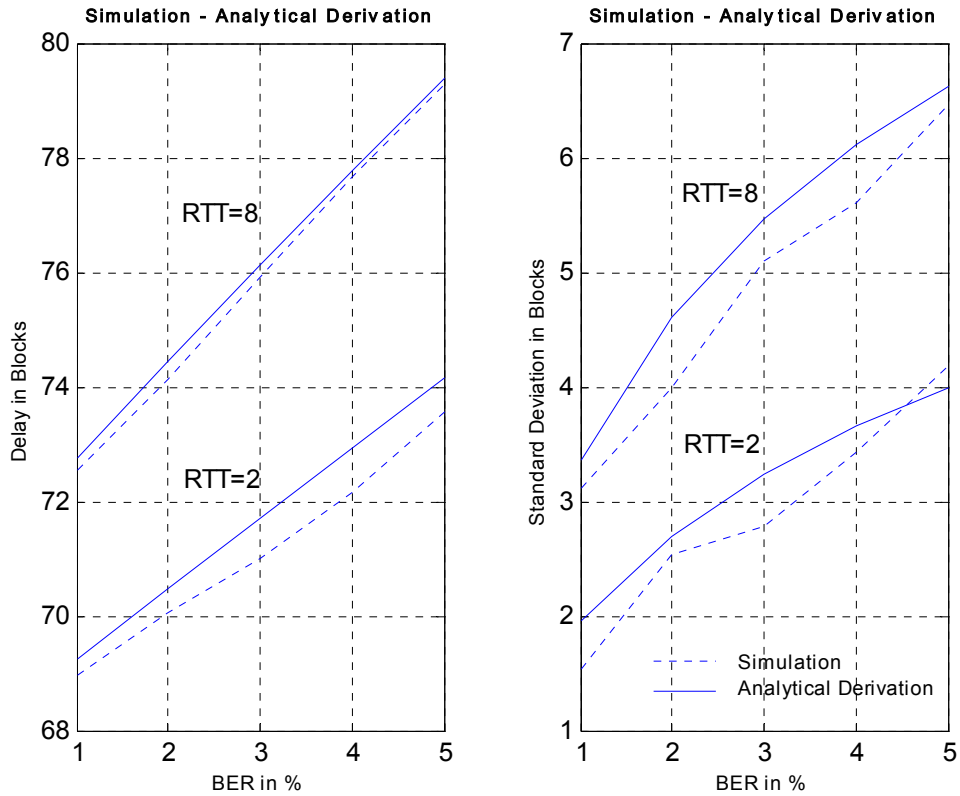


Figure 39: PMF for increasing poll intervall

To get a numerical value for the delay and the standard deviation for IP over RLC, please run Matlab file rlcDelay.m.

The next figure illustrates the comparison between our analytical derivation and the simulation model, which has been developed by A.Rüegg. The simulation model is valid for GPRS working in acknowledged mode. It is based on the standard ETSI TS 101 349 GSM 04.60V8.3.1, for details see [5,11].



5.11 Summary

5.11.1 IP Bit Rate

RLC reduces the IP datagram loss rate to nearly zero, achieving this by increasing the number of blocks needed to transmit a given IP datagram. This means that RLC decreases the average bit rate by the same factor it increases the number of blocks:

$$BitRate_{eff} = BitRate \cdot \left(\sum_{i=0}^M p^i \right)^{-1}$$

To get a numerical value for effective bit rate for IP over RLC, please run Matlab file rlcBitrate.m.

5.11.2 IP Loss Rate

RLC minimizes the IP datagram loss rate by its selective block retransmission process:

$$p_{lossIP} = 1 - p_{arr} = 1 - \sum_{k=0}^M (1 - BER)^{BER^k \cdot N_1} \cdot \left(\prod_{j=0}^{k-1} 1 - (1 - BER)^{BER^j \cdot N_1} \right)^{sign(k)}$$

To get a numerical value for drop rate for IP over RLC, please run Matlab file rlcLoss.m.

5.11.3 IP datagram delay over RLC

To get a numerical value for the delay and the standard deviation for IP over RLC, please run Matlab file rlcDelay.m.

6 Software Engineering

6.1 Introduction

In this chapter we give some information about the development environment and how we built up the application. We also explain how the communication between our application and the network emulator takes place. A brief description of the GUI and its buttons shall help to get along with our application.

6.2 Development Environment

Since we attended a Java course last term at school, we developed the graphical user interface in Java. We decided to develop the mathematical algorithms at first in Matlab because Java does not support mathematical functions such as `product()`, `sum()`, `nchoosek()` and `convolution()`. We then, in a second step, mapped those algorithms to Java. To do so, we installed JBuilder 5 on our workstations.

6.3 Shell Script

A shell script collects commands you have to type in a shell. Our script first loads the Nistnet module then sets standard, class and program path and executes our application. At the end it removes the Nistnet module from the kernel. Note that if you want to run our application on another computer, you need to modify the paths. Our shell script starts by doubleclicking the icon on the desktop.

```
#!/bin/bash

# local auxiliary variables
# STDPATH refers to the root directory of the jdk
#=====

STDPATH="/home/pinoccio/jbuilder5/jdk1.3"
CLASSPATHADDITIONAL="."
PROGIPATH="/root/jbproject/DipApi/classes"

# with (...) we create a subshell (child-process) in order not to
# change the actual set path.
# parent-process waits until child-process is terminated
#=====

( cd /home/pinoccio/NistEmulator/nistnet ; ./Load.Nistnet )

# if the path of the java virtual machine (jvm) is set in PATH,
# we can execute CalculatorMain. If not, we first have to add
# the correct path of the jvm.
#=====

if which java 2> /dev/null > /dev/null; then
    cd $PROGIPATH
    java dipapi.CalculatorMain $*
else
    echo 'Java Virtual Machine konnte nicht gefunden werden!'
    echo "Setze Standardpfad ($STDPATH)"
    export PATH=$STDPATH/bin:$PATH
    cd $PROGIPATH
    java dipapi.CalculatorMain -classpath "$STDPATH:$CLASSPATHADDITIONAL" $*
fi

# removes nistnet module
#=====

rmmod nistnet 2> /dev/null
```

6.4 Graphical User Interface (GUI)

6.4.1 Appliance

Doubleclicking the “Network Emulator for CDMA2000 and GPRS/GSM”-Icon on the desktop starts shell script, which itself makes our graphical user interface appear. The following figure shows the GUI we created.

First you have to decide which network architecture you want to emulate. After that you need to choose values for the listed parameters. Pressing the “Calculate”-Button calculates the settings for the network emulator due to our analytical derivations in the previous chapters. You now have to set the IP addresses, if you do not want to use the preset routes. Note that the emulator emulates the link from source to destination and from destination to source with the same parameters only for the GPRS/GSM mode. In CDMA2000, the parameters for up- and downlink are different, for details refer to next paragraph or to chapter RLP.

Pressing the “Start Nistnet”-Button starts the Nistnet network emulator with the calculated values as arguments. Now the label of the “Start Nistnet”-Button changes to “Stop-Nistnet”. Pressing the button now stops the emulator, removes the set route and the label changes to “Start Nistnet”.

To quit the application you can either press the “Quit”-Button or the cross on the top right, both actions stop the emulator, remove the module and quit the application.

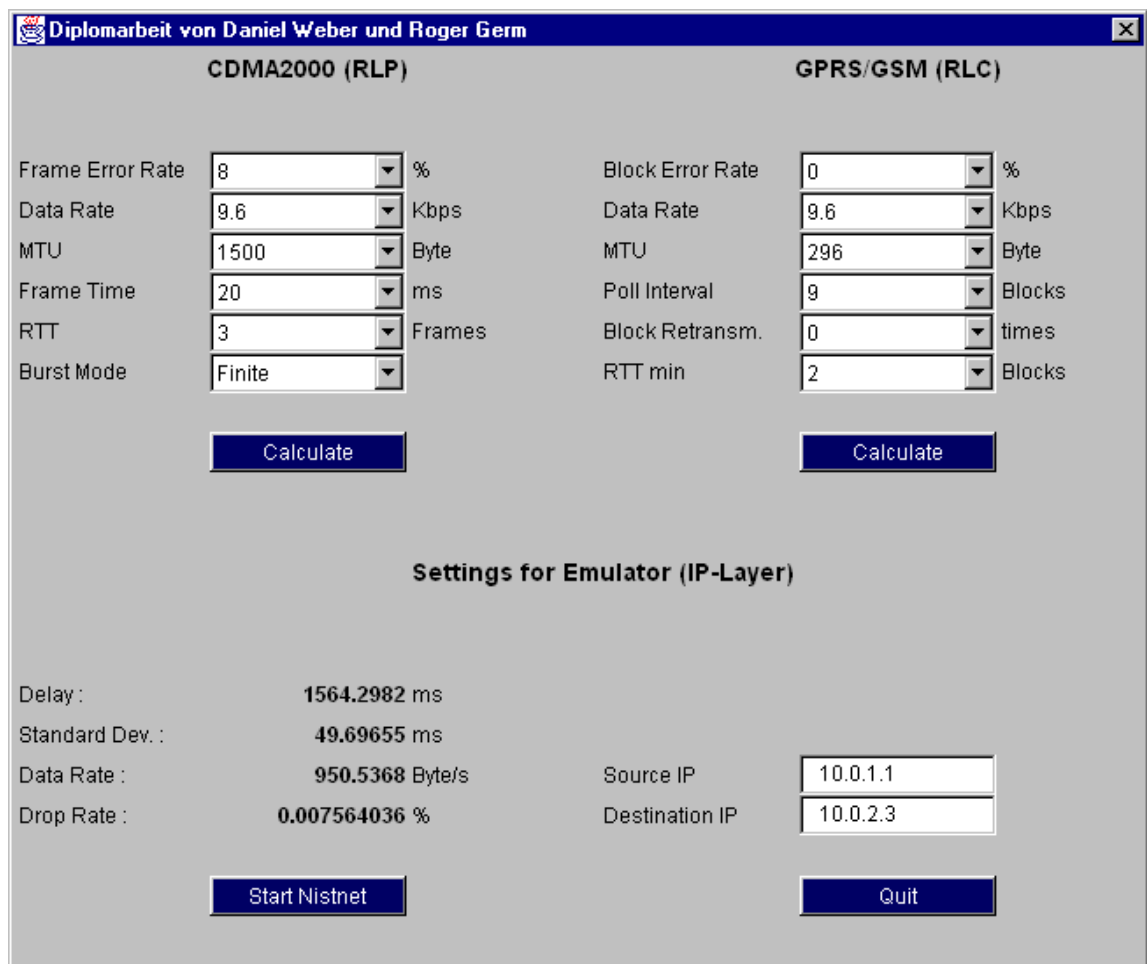


Figure 40: GUI

6.4.2 Parameters for CDMA2000

Frame Error Rate [0, 1, 2, 3, 5, 8, 10, 15, 25] %

Denotes the frame error rate on the radio link, which can rise up to 25% temporarily.

Data Rate [9.6, 19.2, 28.8, 57.6, 105.6, 153.6, 201.6, 316.8] Kbps

Denotes the desired user data rate on the radio link. Has to be a multiple of 9.6kbps.

MTU [296, 576, 1006, 1449, 1500, 1600] Byte

Denotes the maximum transmission unit, which is the total length of an IP datagram.

Frame Time [20] ms

Denotes the length of a frame in time. Since CDMA2000 uses the 20ms frame structure, this value is always 20ms.

RTT [3, 5, 7, 9] Frames

Denotes the roundtrip time of a frame on the radio link. Has to be an odd integer because of the RLP retransmission process.

Burst Mode [Finite, Infinite]

Denotes the burst mode. Finite means that the supplemental channel (SCH) is assigned only when needed, which depends on the buffer size. Infinite means that SCH is assigned when the transmission starts.

6.4.3 Parameters for GPRS/GSM

Block Error Rate [0, 1, 2, 3, 5, 8, 10, 15, 25] %

Denotes the block error rate on the radio link, which can rise up to 25% temporarily.

Data Rate [9.6, 19.2, 28.8, 57.6, 105.6, 153.6, 201.6, 316.8] Kbps

Denotes the user data rate on the radio link.

MTU [296, 576, 1006, 1500, 1600] Byte

Denotes the maximum transmission unit, which is the total length of an IP datagram.

Poll Intervall [9, 16, 32] Blocks

Denotes the poll interval in number of blocks, which is a message for the peer entity to send an ACK/NACK message.

Block Retransmission [0, 1, 2, 3, 4, 5, 6, 7] times

Denotes the maximum number of retransmission rounds for an RLC data block.

RTT min [2, 4, 6, 8] Blocks

Denotes the minimum roundtrip time of a block on the radio link. Has to be an even integer because of the RLC retransmission process.

6.5 Class Diagram

As you see, our user interface follows the model-view-controller design pattern. Please refer to appendix A and B for Java and Matlab source code. The model functions `calculatedcdma()` and `calculategsm()` call the methods `setChanged()` and `notifyObserver()`, which cause that in our view the function `update()` will be called. The function `update()` receives the model as parameter. It then updates the view with the actual data of the model.

To react on an input from a user, such as clicking a button, the respective component needs to have a controller. The controller has a persistent reference to the model. Therefore it can exert influence on the model data through `calculatedcdma()` for example.

So now the circle is closed: The controller modifies the data in the model by calling `calculatedcdma()` for instance, which triggers the `update()` method in the view, whereby it is guaranteed that the graphical display in the view is up to date.

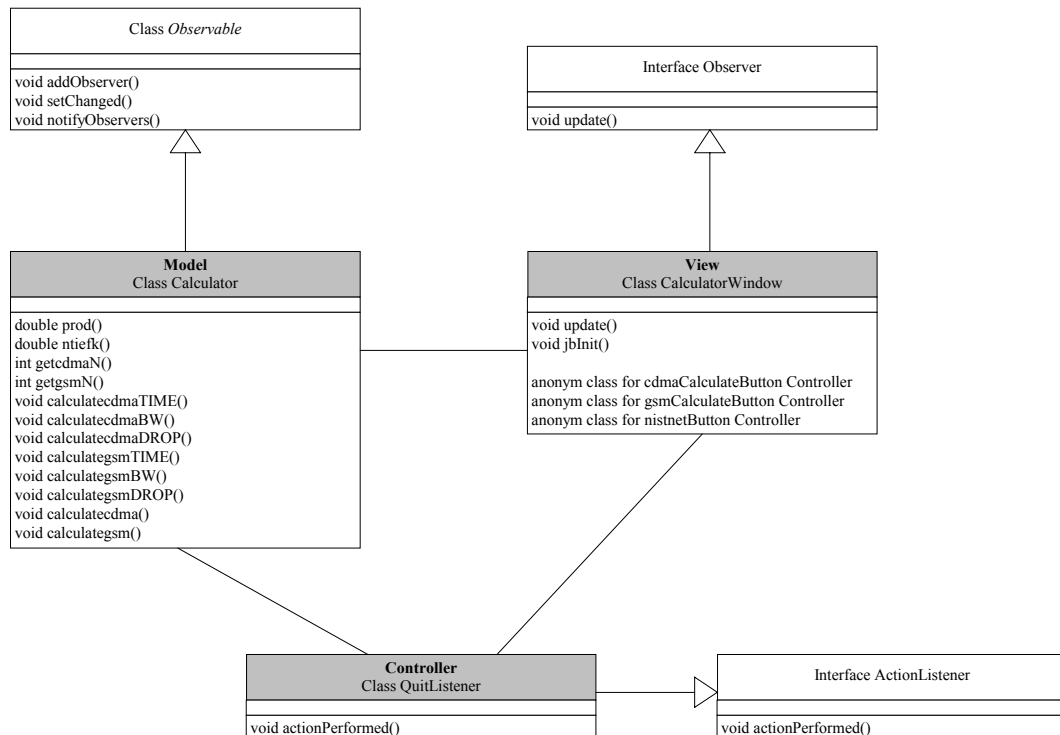


Figure 41: Class Diagram

6.6 Calling NistNet and Perl from Java

A specialty in our application is the call of NistNet and Perl from Java. Java offers two classes, `Process` and `Runtime`, which include functions to handle other processes.

Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running.

The class `Process` provides methods for performing input from the process, performing output to the process, waiting for the process to complete, checking the exit status of the process, and destroying (killing) the process. We do not need a handler for the process NistNet because by removing the module from the kernel, the application is automatically stopped. In the process Perl, we need a handler because of the endless for-loop in our perl script. So the termination of this application has to be done by our program. In the next few lines we describe how we implemented the call of NistNet and Perl in Java.

First we instance two objects, Perl as instance of the class `Process` which conduces as a handler of the process Perl and `RT` as instance of the class `Runtime` which allows to interface with the environment.

```

Process Perl = null;
Runtime RT = null;

```

The current runtime can be obtained from the `getRuntime()` method. We assigned this runtime to the Runtime object `RT`.

```
RT=Runtime.getRuntime();
```

The `Runtime.exec()` method creates a native process and returns an instance of a subclass of `Process` that can be used to control the process and obtain information about it. As mentioned above in case of `NistNet`, we do not need these handlers. But in the case of `CDMA2000` mode, when we start the perl script, we need a handler to stop this subprocess later on. So we assigned this handler to the `Process` object `Perl`. The strings as arguments of this functions are the same you have to type in a shell if you want to start the `NistNet` by the shell command.

```
RT.exec("cnistnet -u -a"+" "+destIP+" "+sourceIP+" "+"--delay"+" "+outDELAY+" "+outSTD+" "+"--drop"+" "+outDROP+" "+"--bandwidth"+" "+outBWL);
```

```
RT.exec("cnistnet -u -a"+" "+sourceIP+" "+destIP+" "+"--delay"+" "+outDELAY+" "+outSTD+" "+"--drop"+" "+outDROP+" "+"--bandwidth"+" "+outBWL);
```

```
Perl=RT.exec("perl adaptive.pl 8000 4000"+" "+outBWLslow+" "+outBWL+" "+sourceIP+" "+destIP+" "+cdmaMTU+" "+outDELAYslow+" "+outSTDslow+" "+outDROPSlow+" "+outDELAY+" "+outSTD+" "+outDROP);
```

As mentioned above with a handler we have the possibility to terminate the subprocess represented by the `Process` object `Perl`.

```
Perl.destroy();
```

7 Closing Words

This work has given us an insight into the world of mobile communication. It was really diversified since we had to set up a network, do analytical derivations and develop an application with a graphical user interface.

Rebuilding the network went on quite smooth because we could revert to the gained experience during last work, where the network setup was an important part. The analysis of RLP and RLC was the main part of this work. Even though we had some problems with the theory of probabilities at the beginning, this task was very interesting and instructive. We do not regret implementing our algorithms in Java even though it does not offer many mathematical functions. A big advantage of Java is that it provides a considerable class library for the development of graphical user interfaces and that it is platform-independent. Moreover, we appreciated the simpleness of calling other programs with Java.

Another quite challenging but also time-consuming task was the creation of this project documentation in a foreign language. Therefore, we apologise for all the spelling mistakes we made in this document.

With Prof. Dr. Guido M. Schuster we had an extremely helpful and qualified tutor. We would like to express a big thank you to him at this point for supporting us during the last eight weeks. We do wish him all the best for the future. We also thank André Rüegg for providing his simulations and Peter Roffler for arranging and updating all the hardware we used.

Rapperswil, 10. December 2001

Roger Germ

Daniel Weber

8 References

- [1] J.M.Harris; M.Airy: „*Analytical Model for Radio Link Protocol for IS-95 CDMA Systems*“; Motorola
- [2] Bernhard Walke: „*Mobilfunknetze und ihre Protokolle 1*“; Teubner Verlag, 2001
- [3] M.Meyer: „*TCP Performance over GPRS*“; Ericsson Eurolab Germany, 1999
- [4] Q.Pang: „*Performance Evaluation of Retransmission Mechanisms in GPRS Networks*“; Dept. of Electrical and Computer Engineering, University of British Columbia, Vancouver
- [5] ETSI TS 101 349 V8.3.1: „*GPRS, MS-BSS, RLC/MAC*“; www.etsi.org, 2000
- [6] C.Demetrescu: „*LLC-MAC Analysis of GPRS in GSM*“; Bell Labs Technical Journal, 1999
- [7] Guido Schuster: „*Effective bit rate for IP over RLP*“; Hochschule Rapperswil HSR, 2000
- [8] Guido Schuster: „*IP packet delay probability mass function for IP over RLP*“; Hochschule Rapperswil HSR, 2000
- [9] Behcet Sarikaya: „*Packet Mode in Wireless Networks*“; University of Aizer, 2000
- [10] D.Weber; R.Germ: „*IP over CDMA*“; Hochschule Rapperswil HSR, 2000
- [11] A.Rüegg: „*Simulation of the RLC retransmission functions in GSM with GPRS*“; Hochschule Rapperswil HSR, 2000
- [12] F.Khafizov; M.Yavuz: „*TCP over CDMA2000 Networks*“; PILC Working Group, 2001

9 Register of Illustrations

Figure 1: CDMA2000 System	9
Figure 2: CDMA 2000 Protocol Stack.....	9
Figure 3: Network Structure	12
Figure 4: IP Datagram split up into RLP frames.....	18
Figure 5 : RLP retransmission process in a timeline	20
Figure 6: Increase of data frames.....	22
Figure 7: Decrease of Bit Rate.....	22
Figure 8: Legend for tree	22
Figure 9: Simplified RLP retransmission process.....	23
Figure 10: Starting tree (RLP retransmission process)	24
Figure 11: Legend.....	25
Figure 12: Tree 2 (RLP retransmission process)	25
Figure 13: Tree 3 (RLP retransmission process)	26
Figure 14: Frame arrival rate at a certain RTT	27
Figure 15: Three consecutive IP datagrams	29
Figure 16: Retransmission, which needs more than one time slot.....	29
Figure 17: Frame loss at the end causes more than one additional time slot	30
Figure 18: Number of additional time slots to transfer 72 RLP frames (one IP datagram)	31
Figure 19: End of the datagram	31
Figure 20: IP datagram PMF.....	32
Figure 21: GPRS Protocol Stack.....	36
Figure 22: IP datagram split up into RLC blocks	37
Figure 23: RLC data block size	38
Figure 24: Block structure for data transfer	38
Figure 25: Block structure for control block.....	38
Figure 26: RLC block structure for CS-1	41
Figure 27: Increase of data blocks	43
Figure 28: Decrease of bit rate.....	43
Figure 29: IP datagram success probability tree	46
Figure 30: PMF, CDF for IP datagram success	47
Figure 31: Retransmitted blocks	48
Figure 32: Block loss in last poll interval	49
Figure 33: Block loss in second last poll interval	49
Figure 34: PMF's for IP datagram.....	50
Figure 35: PMF for increasing BER	51
Figure 36: Increasing BER.....	51
Figure 37: PMF for increasing RTT	52
Figure 38: Increasing RTT.....	52
Figure 39: PMF for increasing poll intervall	53
Figure 40: GUI.....	57
Figure 41: Class Diagram	59



10 Acronyms

ARQ	:	Automatic Repeat Request
ATM	:	Asynchronous Transfer Mode
BSC	:	Base Station Controller
BSS	:	Base Station Subsystem
BSSGP	:	Base Station Subsystem GPRS Protocol
BTS	:	Base Transceiver Station
CDMA	:	Code Division Multiple Access
FDMA	:	Frequency Division Multiple Access
GGSN	:	Gateway GPRS Support Node
GMSC	:	Gateway Mobile-Services Switching Centre
GPRS	:	General Packer Radio Service
GSM	:	Global System for Mobile Communication
GTP	:	GPRS Tunnelling Protocol
HA	:	Home Agent
HLR	:	Home Location Register
HTTP	:	Hypertext Transfer Protocol
IMT	:	International Mobile Telecommunication
IP	:	Internet Protocol
ITU	:	International Telecommunication Union
IWF	:	Interworking Function
LAN	:	Local Area Network
LLC	:	Logical Link Control
MAC	:	Media Access Control
MS	:	Mobile Station
MSC	:	Mobile Switching System
OS	:	Operating System
PSDN	:	Public Switched Data Network
PPP	:	Point to Point Protocol
PSTN	:	Public Switched Telephony Network
RAN	:	Radio Access Network
RLC	:	Radio Link Control
RLP	:	Radio Link Protocol
RNC	:	Radio Network Controller
SGSN	:	Serving GPRS Support Node
SNDGP	:	Subnetwork Dependent Convergence Protocol
TDMA	:	Time Division Multiple Access
TCP	:	Transport Control Protocol
UDP	:	User Datagram Protocol
UE	:	User Equipment
UMTS	:	Universal Mobile Telecommunication System

A Java Files

A.1 CalculatorMain.java

```
package dipapi;

//=====
//This class includes the main function from the application which generates a view, a
//model and an observer.
//=====

public class CalculatorMain {

    private Calculator theCalculator;
    private CalculatorWindow theCalculatorWindow;

    public CalculatorMain ()
    {
        theCalculator = new Calculator();
        theCalculatorWindow = new CalculatorWindow(theCalculator);
        theCalculator.addObserver(theCalculatorWindow);
    }

    public static void main(String[] args)
    {
        CalculatorMain application = new CalculatorMain();
    }
}
```

A.2 Calculator.java

```
package dipapi;

//=====
//The class Calculator represents the model of the model-view-controller-architecture.
//It is responsible for the calculation of all parameters for NistNet.
//=====

import java.util.*;

public class Calculator extends Observable{

    private float outDELAY, outSTD, outBWL, outDROP;
    private float outDELAYslow, outSTDslow, outBWLslow, outDROPSlow;

    public Calculator()
    {
        outDELAY=0;
        outSTD=0;
        outBWL=0;
        outDROP=0;
        outDELAYslow=0;
        outSTDslow=0;
        outBWLslow=0;
        outDROPSlow=0;
    }

    public float getoutDELAY(){return outDELAY;}
    public float getoutSTD(){return outSTD;}
    public float getoutBWL(){return outBWL;}
    public float getoutDROP(){return outDROP;}
    public float getoutDELAYslow(){return outDELAY;}
    public float getoutSTDslow(){return outSTD;}
    public float getoutBWLslow(){return outBWL;}
    public float getoutDROPSlow(){return outDROP;}

    //=====
    //The following functions prod(), ntiefk(), getcdman() and getgsmN() are only auxiliary
    //functions.
    //=====

    //=====
    //This function calculates the product of all numbers between n and m.
    //=====

    public double prod(double n, double m)
    {
        double mult1=n;
        double mult2=n+1;
    }
}
```

```
double result=0;
double k=n;

if(m<n || m<0 || m==1)
{
    result=1;
}
else
{
    while(k<m)
    {
        result=mult1*mult2;
        mult1=result;
        mult2=mult2+1;
        k++;
    }
}
return(result);
}

//=====
//This function returns the number of combinations of dN objects taken k at a time.
//=====

public double ntiefk(double dN, int k)
{
    double y;

    if(k==0)
    {
        y=1;
    }
    else
    {
        y=prod(1,dN)/(prod(1,k)*prod(1,(dN-k)));
    }
    return(y);
}

//=====
//This two functions calculate the number of frames resp. blocks of an IP Datagram for
//an RLP or an RLC transmittance.
//=====

int getcdmaN(double acdmaMTU, double acdmaFTL, double acdmaBWL)
{
    double frameSize=acdmaBWL*acdmaFTL/8*0.875;
    int N =0;

    if((acdmaMTU/frameSize)-(int)(acdmaMTU/frameSize)==0)
    {
        N=(int)(acdmaMTU/frameSize);
    }
    else
    {
        N=(int)(acdmaMTU/frameSize+1);
    }
    return(N);
}

int getgsmN(double agsmMTU)
{
    double payload=160/8;
    int N =0;

    if(((agsmMTU+20)/payload)-(int)((agsmMTU+20)/payload)==0)
    {
        N=(int)((agsmMTU+20)/payload);
    }
    else
    {
        N=(int)((agsmMTU+20)/payload+1);
    }
    return(N);
}

//=====
//The following three functions calculate the four parameters for NistNet when it works
//in the CDMA2000-mode.
//=====
//=====
//This function calculates the Delay and the standard Deviation for IP Datagrams.
//=====
```

```
public void calculatecdmaTIME(double acdmaFER, double acdmaRTT, double acdmaBWL,
                             double acdmaMTU, double acdmaFTL)
{
    double q, ave, forwardchanneldelay, std, delay;
    double dp=acdmaFER/100;
    double dRTT=acdmaRTT;
    int dN=0;

    dN=getcdmaN(acdmaMTU,acdmaFTL,acdmaBWL);

    double P1[]=new double [dN+1];
    double P2[]=new double [(int)dRTT+1];

    q=1-dp;

    for(int k=0;k<P1.length;k++)
    {
        P1[k]=ntiefk(dN,k)*Math.pow(dp,k)*Math.pow(q,(dN-k));
    }

    P2[0]=Math.pow(q,dRTT);

    for(int m=0;m<P2.length-1;m++)
    {
        double temp=0;

        for(int k=0;k<m+1;k++)
        {
            temp=temp+ntiefk(m,k)*Math.pow(dp,k)*Math.pow(q,(m-k));
        }
        P2[m+1]=temp*dp*Math.pow(q,(dRTT-1-m));
    }

    int P1size=P1.length;
    int P2size=P2.length;
    int hmax=P1size+P2size-1;
    double P3[]=new double[hmax];

    //Convolution

    for(int h=1;h<hmax+1;h++)
    {
        P3[h-1]=0;
        int jmin=Math.max(1,h+1-P2size);
        int jmax=Math.min(h,P1size);

        for(int j=jmin;j<jmax+1;j++)
        {
            P3[h-1]=P3[h-1]+(P1[j-1]*P2[h-1+1-j]);
        }
    }

    forwardchanneldelay=(dRTT-3)/2;

    int nsize=P3.length;
    double n[]=new double[nsize];

    for(int s=0;s<nsize;s++)
    {
        n[s]=dN+s+forwardchanneldelay;
    }

    ave=0;

    for(int u=0;u<n.length;u++)
    {
        ave=ave+(n[u]*P3[u]);
    }

    outDELAY=(float)(ave*acdmaFTL);
    double stdsum=0;
    std=0;

    for(int t=0;t<n.length;t++)
    {
        stdsum=stdsum+(Math.pow(n[t]-ave,2)*P3[t]);
    }

    double wurzelexp=0.5;
    std=Math.pow(stdsum,wurzelexp);

    outSTD=(float)(std*acdmaFTL);
}

//=====
//This function calculates the effective Data Rate for IP Datagrams.
//=====
```

```
public void calculatecdmaBWL(double acdmaBWL, double acdmaFER, double acdmaFTL,
                             double acdmaMTU)
{
    double fer=acdmaFER/100;
    int N=0;

    N=getcdmaN(acdmaMTU,acdmaFTL,acdmaBWL);

    double bw=acdmaMTU/(N*acdmaFTL/1000);

    outBWL=(float)(bw/(1+fer+2*Math.pow(fer,2)+6*Math.pow(fer,3)));
}

//=====
//This function calculates the Drop Rate for IP Datagrams.
//=====

public void calculatecdmaDROP(double acdmaFER, double acdmaMTU, double acdmaFTL,
                              double acdmaBWL)
{
    double p=acdmaFER/100;
    int N =0;

    N=getcdmaN(acdmaMTU,acdmaFTL,acdmaBWL);

    outDROP=100*(float)(1-Math.pow((1-(64*Math.pow(p,7)-192*Math.pow(p,8)+240*Math.pow(p,9)-
    160*Math.pow(p,10)+60*Math.pow(p,11)-12*Math.pow(p,12)+Math.pow(p,13))),N));
}

//=====
//The following three functions calculate the four parameters for NistNet when it works
//in the GPRS/GSM-mode.
//=====

//=====
//This function calculates the Delay and the Standard Deviation for IP Datagrams.
//=====

public void calculategsmTIME(double agsmBER, double agsmBR, double agsmMTU,
                             double agsmACK, double agsmRET, double agsmRTT)
{
    int n1=getgsmN(agsmMTU);
    double p=agsmBER/100;
    double q=1-p;
    int Poll=(int)agsmACK;
    int rtt=(int)agsmRTT;
    double channeldelay=(rtt-2)/2;
    int rlcpayload=160;
    int blocksize=184;
    double bitrate=agsmBR*1000/8*rlcpayload/blocksize;
    double frametime=(rlcpayload/8)/bitrate*1000;

    double PMF1ret[]=new double [n1+1];
    double temp[][]=new double [n1+1][3*n1+1];
    double PMF2ret[]=new double [3*n1+1];

    for(int a=0;a<PMF1ret.length;a++)
    {
        PMF1ret[a]=ntiefk(n1,a)*Math.pow(p,a)*Math.pow(q,(n1-a));
    }

    for(int b=0;b<PMF1ret.length;b++)
    {
        for(int c=0;c<b+1;c++)
        {
            temp[b][b+c]=ntiefk(b,c)*Math.pow(p,c)*Math.pow(q,(b-c))*PMF1ret[b];
        }
    }

    for(int d=0;d<PMF2ret.length;d++)
    {
        PMF2ret[d]=0;
    }

    for(int e=0;e<3*n1+1;e++)
    {
        for(int f=0;f<n1+1;f++)
        {
            PMF2ret[e]=PMF2ret[e]+temp[f][e];
        }
    }

    double PMFafterPoll[]= new double[Poll+rtt+2];
    double temp2[][]= new double[Poll-1][Poll+rtt+2];
```

```
for(int as=0;as<rtt+Poll+1;as++)
{
    for(int us=0;us<Poll-1;us++)
    {
        temp2[us][as]=0;
    }
}

for(int block=1;block<Poll;block++)
{
    for(int g=0;g<block+1;g++)
    {
        if(g==0)
        {
            temp2[block-1][g]=ntiefk(block,g)*Math.pow(p,g)*Math.pow(q,(block-g));
        }
        else
        {
            temp2[block-1][rtt+1+Poll-block+g]=ntiefk(block,g)*Math.pow(p,g)*
            Math.pow(q,(block-g));
        }
    }
}

for(int h=0;h<PMFafterPoll.length;h++)
{
    PMFafterPoll[h]=0;
}

for(int i=0;i<PMFafterPoll.length;i++)
{
    for(int j=0;j<Poll-1;j++)
    {
        PMFafterPoll[i]=PMFafterPoll[i]+temp2[j][i];
    }
    PMFafterPoll[i]=PMFafterPoll[i]/(Poll-1);
}

double PMFbeforePoll[]= new double[rtt+Poll];
double temp3[][]= new double[rtt-1][rtt+Poll];

for(int es=0;es<rtt+Poll;es++)
{
    for(int is=0;is<rtt-1;is++)
    {
        temp3[is][es]=0;
    }
}

for(int k=1;k<rtt;k++)
{
    for(int m=0;m<Poll;m++)
    {
        if(m==0)
        {
            temp3[k-1][m]=ntiefk(Poll-1,m)*Math.pow(p,m)*Math.pow(q,(Poll-1-m));
        }
        else
        {
            temp3[k-1][rtt-k+m+1]=ntiefk(Poll-1,m)*Math.pow(p,m)*Math.pow(q,(Poll-1-m));
        }
    }
}

for(int x=0;x<PMFbeforePoll.length;x++)
{
    PMFbeforePoll[x]=0;
}

for(int o=0;o<PMFbeforePoll.length;o++)
{
    for(int r=0;r<rtt-1;r++)
    {
        PMFbeforePoll[o]=PMFbeforePoll[o]+temp3[r][o];
    }
    PMFbeforePoll[o]=PMFbeforePoll[o]/(rtt-1);
}
//Convolution of PMFafterPoll and PMFbeforePoll results in ConvAfterBefore

int Pasize=PMFafterPoll.length;
int Pbsize=PMFbeforePoll.length;
int CABmax=Pasize+Pbsize-1;
double ConvAfterBefore []=new double[CABmax];

for(int s=1;s<CABmax+1;s++)
{
    ConvAfterBefore[s-1]=0;
    int tmin=Math.max(1,s+1-Pbsize);
```

```
    int tmax=Math.min(s, Pasize);

    for(int t=tmin;t<tmax+1;t++)
    {
        ConvAfterBefore[s-1]=ConvAfterBefore[s-1]+(PMFafterPoll[t-1]*
            PMFbeforePoll[s-1+1-t]);
    }
}

//Convolution of ConvAfterBefore and PMF2ret results in PMFtotal

int CABsize=ConvAfterBefore.length;
int P2size=PMF2ret.length;
int totalmax=CABsize+P2size-1;
double PMFtotal []=new double[totalmax];

for(int u=1;u<totalmax+1;u++)
{
    PMFtotal[u-1]=0;
    int vmin=Math.max(1,u+1-P2size);
    int vmax=Math.min(u,CABsize);

    for(int v=vmin;v<vmax+1;v++)
    {
        PMFtotal[u-1]=PMFtotal[u-1]+(ConvAfterBefore[v-1]*PMF2ret[u-1+1-v]);
    }
}

int nsize=PMFtotal.length;
double n[]=new double[nsize];

for(int w=0;w<nsize;w++)
{
    n[w]=n1+w+channeldelay;
}

double ave=0;

for(int y=0;y<n.length;y++)
{
    ave=ave+(n[y]*PMFtotal[y]);
}

outDELAY=(float)(ave*frametime);

double stdsum=0;
double std=0;

for(int z=0;z<n.length;z++)
{
    stdsum=stdsum+(Math.pow(n[z]-ave,2)*PMFtotal[z]);
}
double wurzelexp=0.5;

std=Math.pow(stdsum,wurzelexp);

outSTD=(float)(std*frametime);
}

//=====
//This function calculates the effective Data Rate for IP Datagrams.
//=====

public void calculategsmBWL(double agsmBR, double agsmRET, double agsmBER)
{
    double bitrate=0;
    double p=agsmBER/100;
    int rlcpayload=160;
    int blocksize=184;

    bitrate=agsmBR*1000/8*rlcpayload/blocksize;

    double b=1;
    for(int u=1;u<agsmRET+1;u++)
    {
        b=b+Math.pow(p,u);
    }

    outBWL=(float)(bitrate/b);
}

//=====
//This function calculates the Drop Rate for IP Datagrams.
//=====

public void calculategsmDROP(double agsmMTU, double agsmBER, double agsmRET)
```

```

    {
        int N=getgsmN(agsmMTU);
        double p=agsmBER/100;
        int k=(int)agsmRET;

        double P[]=new double [k+1];

        P[0]=Math.pow((1-p),N);

        for(int m=1;m<P.length;m++)
        {
            double b[]=new double [m];
            double a=Math.pow((1-p),N*Math.pow(p,m));
            double c=1;

            for(int r=0;r<b.length;r++)
            {
                b[r]=(1-Math.pow((1-p),(N*Math.pow(p,r))));
            }

            for(int h=0;h<b.length;h++)
            {
                c=c*b[h];
            }
            P[m]=a*c;
        }

        double x=0;

        for(int u=0;u<P.length;u++)
        {
            x=x+P[u];
        }

        outDROP=(float)(100*(1-x));
    }

//=====
//The following two functions are needed by the GUI if the user want to calculate the
//four parameters for NistNet.
//=====

//=====
//This function calculates the parameters for NistNet when it works in the CDMA2000-mode.
//First it calls the tree functions to calculate the parameters for the fundamental
//channel (9.6 Kbps) and after this it calls the tree functions to calculate the parameters
//for the higher Data Rate (supplemental channel).
//=====

    public void calculatecdma(double acdmaFER, double acdmaBWL, double acdmaMTU,
                             double acdmaFTL, double acdmaRTT)
    {
        calculatecdmaTIME(acdmaFER, acdmaRTT, 9.6, acdmaMTU, acdmaFTL);
        calculatecdmaBWL(9.6, acdmaFER, acdmaFTL, acdmaMTU);
        calculatecdmaDROP(acdmaFER, acdmaMTU, acdmaFTL, 9.6);

        outDELAYSlow=outDELAY;
        outSTDslow=outSTD;
        outBWLslow=outBWL;
        outDROPSlow=outDROP;

        calculatecdmaTIME(acdmaFER, acdmaRTT, acdmaBWL, acdmaMTU, acdmaFTL);
        calculatecdmaBWL(acdmaBWL, acdmaFER, acdmaFTL, acdmaMTU);
        calculatecdmaDROP(acdmaFER, acdmaMTU, acdmaFTL, acdmaBWL);

        setChanged();
        notifyObservers();
    }

//=====
//This function calls the tree functions to calculate the four parameters for NistNet
//for the GPRS/GSM-mode
//=====

    public void calculategsm(double agsmBER, double agsmBR, double agsmMTU,
                             double agsmACK, double agsmRET, double agsmRTT)
    {
        calculategsmTIME(agsmBER, agsmBR, agsmMTU, agsmACK, agsmRET, agsmRTT);
        calculategsmBWL(agsmBR, agsmRET, agsmBER);
        calculategsmDROP(agsmMTU, agsmBER, agsmRET);

        setChanged();
        notifyObservers();
    }

```

```
}
```

A.3 CalculatorWindow.java

```
package dipapi;

//=====
//This class represents the view of the model-view-controller architecture and is
//responsible for displaying the graphical elements such as Buttons and Textfields.
//Another job of this class is to start the NistNet application and set the routes
//with its parameters. When the "Stop-NistNet" button is pressed, the routes will be
//removed.
//=====

import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Observer;
import java.util.Observable;
import java.io.*;

public class CalculatorWindow extends Frame implements Observer{

    private double cdmaFER, cdmaBWL, cdmaMTU, cdmaFTL, cdmaRTT;
    private double gsmBER, gsmBR, gsmMTU, gsmACK, gsmRET, gsmRTT;
    private float outDELAY, outSTD, outBWL, outDROP;
    private float outDELAYSslow, outSTDslow, outBWLslow, outDROPSlow;
    private String destIP = null;
    private String sourceIP = null;
    private String BM = null;
    private int Zustand;
    private int Mode;
    private int Modeold ;
    private Process Perl = null;

    static Runtime RT = null;

    public Calculator model;

    BorderLayout BorderLayout1 = new BorderLayout();
    Panel titlePn = new Panel();
    GridLayout GridLayout1 = new GridLayout();
    Panel dataPn = new Panel();
    Panel nistnetPn = new Panel();
    Label cdmaLb = new Label();
    Label gsmLb = new Label();
    GridLayout GridLayout3 = new GridLayout();
    BorderLayout BorderLayout2 = new BorderLayout();
    Button quitBt = new Button();
    Button nistnetBt = new Button();
    Label label1 = new Label();
    Label label2 = new Label();
    Label label3 = new Label();
    Panel calculatePn = new Panel();
    Panel outputPn = new Panel();
    Panel inputPn = new Panel();
    GridLayout GridLayout2 = new GridLayout();
    Button cdmaCalculateBt = new Button();
    Button gsmCalculateBt = new Button();
    Label label4 = new Label();
    Label label5 = new Label();
    Label label6 = new Label();
    GridLayout GridLayout4 = new GridLayout();
    Label settingsLb = new Label();
    Panel outputDataPn = new Panel();
    GridLayout GridLayout5 = new GridLayout();
    Label outDELAYLb = new Label();
    Label label8 = new Label();
    Label parameter5Lb = new Label();
    Label destIPLb = new Label();
    Label outDROPValueLb = new Label();
    Label sourceIPLb = new Label();
    Label outBWLb = new Label();
    Label parameter6Lb = new Label();
    Label outSTDLb = new Label();
    Label label17 = new Label();
    Label label19 = new Label();
    Label outDROPLb = new Label();
    Label outBWUnitLb = new Label();
    Label parameter6UnitLb = new Label();
    Label outSTDUnitLb = new Label();
    Label outDELAYUnitLb = new Label();
    Label label21 = new Label();
    Label label22 = new Label();
    Label label114 = new Label();
```

```
Label label115 = new Label();
Label outDROPUnitLb = new Label();
Label label117 = new Label();
Label outBWValueLb = new Label();
Label parameter6ValueLb = new Label();
Label outSTDValueLb = new Label();
Label outDELAYValueLb = new Label();
Label label24 = new Label();
Label parameter5ValueLb = new Label();
GridLayout gridLayout6 = new GridLayout();
Panel cdmaDataPn = new Panel();
Panel gsmDataPn = new Panel();
GridLayout gridLayout7 = new GridLayout();
GridLayout gridLayout8 = new GridLayout();
Label cdmaFERLb = new Label();
Label cdmaBWUnitLb = new Label();
Label cdmaRTTUnitLb = new Label();
Label cdmaBWLb = new Label();
Label cdmaRTTLb = new Label();
Label cdmaFTTUnitLb = new Label();
Label cdmaFTLb = new Label();
Label cdmaMTUUnitLb = new Label();
Label cdmaFERUnitLb = new Label();
Label cdmaMTULb = new Label();
Label label30 = new Label();
Label gsmFERLb = new Label();
Label gsmFERUnitLb = new Label();
Label gsmBWLb = new Label();
Label gsmBWUnitLb = new Label();
Label gsmMTULb = new Label();
Label gsmMTUUnitLb = new Label();
Label gsmFTLb = new Label();
Label gsmFTUnitLb = new Label();
Label gsmRTTLb = new Label();
Label gsmRTTUnitLb = new Label();
Label label31 = new Label();
Label label32 = new Label();
Label label33 = new Label();
Label label34 = new Label();
Label label35 = new Label();
Label label36 = new Label();
Label label37 = new Label();
Label label38 = new Label();
Label label39 = new Label();
TextField destIPTx = new TextField();
TextField sourceIPTx = new TextField();
Choice cdmaFERTx = new Choice();
Choice cdmaBWTx = new Choice();
Choice cdmaMTUTx = new Choice();
Choice cdmaFTTx = new Choice();
Choice cdmaRTTTx = new Choice();
Choice gsmFERTx = new Choice();
Choice gsmBWTx = new Choice();
Choice gsmMTUTx = new Choice();
Choice gsmACKTx = new Choice();
Choice gsmRETTx = new Choice();
Label label7 = new Label();
Choice gsmRTTTx = new Choice();
Label label9 = new Label();
Label cdmaBMLb = new Label();
Choice cdmaBMTx = new Choice();

public CalculatorWindow(Calculator themodel)
{
    cdmaFER=0;
    cdmaBWL=0;
    cdmaMTU=0;
    cdmaFTL=0;
    cdmaRTT=0;
    gsmBER=0;
    gsmBR=0;
    gsmMTU=0;
    gsmACK=0;
    gsmRET=0;
    Zustand=0;
    Mode=0;
    Modeold=0;

    this.model=themodel;

    try
    {
        jbInit();
        pack();
        setVisible(true);
        setBounds(50,50,650,650);
    }
    catch(Exception e)
```

```
{
    e.printStackTrace();
}

private void jbInit() throws Exception
{
    this.setBackground(Color.lightGray);
    this.setTitle("Diplomarbeit von Daniel Weber und Roger Germ");
    this.setResizable(false);
    this.setLayout(borderLayout1);
    titlePn.setLayout(gridLayout1);
    gridLayout1.setColumns(2);
    gridLayout1.setRows(2);
    gridLayout1.setVgap(10);
    dataPn.setLayout(borderLayout2);
    nistnetPn.setLayout(gridLayout3);
    titlePn.setBackground(Color.lightGray);
    titlePn.setForeground(Color.red);
    cdmaLb.setBackground(Color.lightGray);
    cdmaLb.setFont(new java.awt.Font("Dialog", 1, 14));
    cdmaLb.setForeground(Color.black);
    cdmaLb.setAlignment(1);
    cdmaLb.setText("CDMA2000 (RLP)");
    gsmLb.setBackground(Color.lightGray);
    gsmLb.setFont(new java.awt.Font("Dialog", 1, 14));
    gsmLb.setForeground(Color.black);
    gsmLb.setAlignment(1);
    gsmLb.setText("GPRS/GSM (RLC)");
    gridLayout3.setColumns(6);
    gridLayout3.setRows(2);
    quitBt.setBackground(new java.awt.Color(0, 0, 100));
    quitBt.setForeground(Color.white);
    quitBt.setLabel("Quit");
    nistnetBt.setBackground(new java.awt.Color(0, 0, 100));
    nistnetBt.setForeground(Color.white);
    nistnetBt.setLabel("Start Nistnet");
    calculatePn.setLayout(gridLayout2);
    gridLayout2.setColumns(6);
    cdmaCalculateBt.setBackground(new java.awt.Color(0, 0, 100));
    cdmaCalculateBt.setForeground(Color.white);
    cdmaCalculateBt.setLabel("Calculate");
    gsmCalculateBt.setForeground(Color.white);
    label17.setText("RTT min");
    label19.setText(" Blocks");
    cdmaBMLb.setText(" Burst Mode");

    cdmaFERTx.add("0");
    cdmaFERTx.add("1");
    cdmaFERTx.add("2");
    cdmaFERTx.add("3");
    cdmaFERTx.add("5");
    cdmaFERTx.add("8");
    cdmaFERTx.add("10");
    cdmaFERTx.add("15");
    cdmaFERTx.add("25");

    cdmaBWTx.add("9.6");
    cdmaBWTx.add("19.2");
    cdmaBWTx.add("28.8");
    cdmaBWTx.add("57.6");
    cdmaBWTx.add("105.6");
    cdmaBWTx.add("153.6");
    cdmaBWTx.add("201.6");
    cdmaBWTx.add("316.8");

    cdmaMTUTx.add("296");
    cdmaMTUTx.add("576");
    cdmaMTUTx.add("1006");
    cdmaMTUTx.add("1449");
    cdmaMTUTx.add("1500");
    cdmaMTUTx.add("1600");

    cdmaFTTx.add("20");

    cdmaRTTTx.add("3");
    cdmaRTTTx.add("5");
    cdmaRTTTx.add("7");
    cdmaRTTTx.add("9");

    cdmaBMTx.add("Finite");
    cdmaBMTx.add("Infinite");

    gsmFERTx.add("0");
    gsmFERTx.add("1");
    gsmFERTx.add("2");
    gsmFERTx.add("3");
    gsmFERTx.add("5");
}
```

```
gsmFERTx.add("8");
gsmFERTx.add("10");
gsmFERTx.add("15");
gsmFERTx.add("25");

gsmBWTx.add("9.6");
gsmBWTx.add("19.2");
gsmBWTx.add("28.8");
gsmBWTx.add("57.6");
gsmBWTx.add("105.6");
gsmBWTx.add("153.6");
gsmBWTx.add("201.6");
gsmBWTx.add("316.8");

gsmMTUTx.add("296");
gsmMTUTx.add("576");
gsmMTUTx.add("1006");
gsmMTUTx.add("1500");
gsmMTUTx.add("1600");

gsmACKTx.add("9");
gsmACKTx.add("16");
gsmACKTx.add("32");

gsmRETTx.add("0");
gsmRETTx.add("1");
gsmRETTx.add("2");
gsmRETTx.add("3");
gsmRETTx.add("4");
gsmRETTx.add("5");
gsmRETTx.add("6");
gsmRETTx.add("7");

gsmRTTTx.add("2");
gsmRTTTx.add("4");
gsmRTTTx.add("6");
gsmRTTTx.add("8");

gsmCalculateBt.setBackground(new java.awt.Color(0, 0, 100));
gsmCalculateBt.setLabel("Calculate");
outputPn.setLayout(gridLayout4);
gridLayout4.setColumns(2);
gridLayout4.setRows(2);
settingsLb.setFont(new java.awt.Font("Dialog", 1, 14));
settingsLb.setAlignment(1);
settingsLb.setText("Settings for Emulator (IP-Layer)");
outputDataPn.setLayout(gridLayout5);
gridLayout5.setColumns(6);
gridLayout5.setRows(5);
outDELAYLb.setText(" Delay :");
destIPLb.setText("Destination IP");
outDROPValueLb.setBackground(Color.lightGray);
outDROPValueLb.setFont(new java.awt.Font("Dialog", 1, 12));
outDROPValueLb.setAlignment(2);
sourceIPLb.setText("Source IP");
outBWLb.setText(" Data Rate :");
outSTDLb.setText(" Standard Dev. :");
outBWUnitLb.setText(" Byte/s");
outSTDUnitLb.setText(" ms");
outDELAYUnitLb.setText(" ms");
label22.setText(" ");
outDROPUnitLb.setText(" %");
outBWValueLb.setBackground(Color.lightGray);
outBWValueLb.setFont(new java.awt.Font("Dialog", 1, 12));
outBWValueLb.setAlignment(2);
parameter6ValueLb.setBackground(Color.lightGray);
parameter6ValueLb.setFont(new java.awt.Font("Dialog", 1, 12));
parameter6ValueLb.setAlignment(2);
outSTDValueLb.setBackground(Color.lightGray);
outSTDValueLb.setFont(new java.awt.Font("Dialog", 1, 12));
outSTDValueLb.setAlignment(2);
outDELAYValueLb.setBackground(Color.lightGray);
outDELAYValueLb.setFont(new java.awt.Font("Dialog", 1, 12));
outDELAYValueLb.setAlignment(2);

outDELAYValueLb.setText("");
outSTDValueLb.setText("");
outBWValueLb.setText("");
outDROPValueLb.setText("");

parameter5ValueLb.setBackground(Color.lightGray);
parameter5ValueLb.setFont(new java.awt.Font("Dialog", 1, 12));
parameter5ValueLb.setAlignment(2);
inputPn.setLayout(gridLayout6);
gridLayout6.setColumns(2);
cdmaDataPn.setLayout(gridLayout7);
gsmDataPn.setLayout(gridLayout8);
gridLayout7.setColumns(3);
```

```
gridLayout7.setRows(7);
gridLayout8.setColumns(3);
gridLayout8.setRows(7);
cdmaFERLb.setText(" Frame Error Rate");
cdmaBWUnitLb.setText(" Kbps");
cdmaRTTUnitLb.setText(" Frames");
cdmaBWLb.setText(" Data Rate");
cdmaRTTLb.setText(" RTT");
cdmaFTUnitLb.setText(" ms");
cdmaFTLb.setText(" Frame Time");
cdmaMTUUnitLb.setText(" Byte");
cdmaFERUnitLb.setText(" %");
cdmaMTULb.setText(" MTU");
gsmFERLb.setText("Block Error Rate");
gsmFERUnitLb.setText(" %");
gsmBWLb.setText("Data Rate");
gsmBWUnitLb.setText(" Kbps");
gsmMTULb.setText("MTU");
gsmMTUUnitLb.setText(" Byte");
gsmFTLb.setText("Poll Interval");
gsmFTUnitLb.setText(" Blocks");
gsmRTTLb.setText("Block Retransm.");
gsmRTTUnitLb.setText(" times");
label31.setBackground(Color.lightGray);
label32.setBackground(Color.lightGray);
label32.setForeground(Color.gray);
destIPTx.setText("10.0.2.3");
sourceIPTx.setText("10.0.1.1");
outDROPLb.setText(" Drop Rate :");
this.add(titlePn, BorderLayout.NORTH);
titlePn.add(cdmaLb, null);
titlePn.add(gsmLb, null);
titlePn.add(label32, null);
titlePn.add(label31, null);
this.add(nistnetPn, BorderLayout.SOUTH);
nistnetPn.add(label11, null);
nistnetPn.add(nistnetBt, null);
nistnetPn.add(label13, null);
nistnetPn.add(label33, null);
nistnetPn.add(quitBt, null);
nistnetPn.add(label2, null);
nistnetPn.add(label34, null);
nistnetPn.add(label35, null);
nistnetPn.add(label36, null);
nistnetPn.add(label37, null);
nistnetPn.add(label38, null);
nistnetPn.add(label39, null);
this.add(dataPn, BorderLayout.CENTER);
dataPn.add(calculatePn, BorderLayout.CENTER);
calculatePn.add(label14, null);
calculatePn.add(cdmaCalculateBt, null);
calculatePn.add(label15, null);
calculatePn.add(label30, null);
calculatePn.add(gsmCalculateBt, null);
calculatePn.add(label6, null);
dataPn.add(outputPn, BorderLayout.SOUTH);
outputPn.add(settingsLb, null);
outputPn.add(outputDataPn, null);
outputDataPn.add(outDELAYLb, null);
outputDataPn.add(outDELAYValueLb, null);
outputDataPn.add(outDELAYUnitLb, null);
outputDataPn.add(parameter5Lb, null);
outputDataPn.add(parameter5ValueLb, null);
outputDataPn.add(label22, null);
outputDataPn.add(outSTDLb, null);
outputDataPn.add(outSTDValueLb, null);
outputDataPn.add(outSTDUnitLb, null);
outputDataPn.add(parameter6Lb, null);
outputDataPn.add(parameter6ValueLb, null);
outputDataPn.add(parameter6UnitLb, null);
outputDataPn.add(outBWLb, null);
outputDataPn.add(outBWValueLb, null);
outputDataPn.add(outBWUnitLb, null);
outputDataPn.add(sourceIPLb, null);
outputDataPn.add(sourceIPTx, null);
outputDataPn.add(label117, null);
outputDataPn.add(outDROPLb, null);
outputDataPn.add(outDROPValueLb, null);
outputDataPn.add(outDROPUnitLb, null);
outputDataPn.add(destIPLb, null);
outputDataPn.add(destIPTx, null);
outputDataPn.add(label19, null);
outputDataPn.add(label115, null);
outputDataPn.add(label114, null);
outputDataPn.add(label17, null);
outputDataPn.add(label18, null);
outputDataPn.add(label24, null);
outputDataPn.add(label21, null);
```

```
dataPn.add(inputPn, BorderLayout.NORTH);
inputPn.add(cdmaDataPn, null);
cdmaDataPn.add(cdmaFERLb, null);
cdmaDataPn.add(cdmaFERTx, null);
cdmaDataPn.add(cdmaFERUnitLb, null);
cdmaDataPn.add(cdmaBWLb, null);
cdmaDataPn.add(cdmaBWTx, null);
cdmaDataPn.add(cdmaBWUnitLb, null);
cdmaDataPn.add(cdmaMTULb, null);
cdmaDataPn.add(cdmaMTUTx, null);
cdmaDataPn.add(cdmaMTUUnitLb, null);
cdmaDataPn.add(cdmaFTLb, null);
cdmaDataPn.add(cdmaFTTx, null);
cdmaDataPn.add(cdmaFTUnitLb, null);
cdmaDataPn.add(cdmaRTTLb, null);
cdmaDataPn.add(cdmaRTTTx, null);
cdmaDataPn.add(cdmaRTTUnitLb, null);
cdmaDataPn.add(cdmaBMLb, null);
cdmaDataPn.add(cdmaBMTx, null);
inputPn.add(gsmDataPn, null);
gsmDataPn.add(gsmFERLb, null);
gsmDataPn.add(gsmFERTx, null);
gsmDataPn.add(gsmFERUnitLb, null);
gsmDataPn.add(gsmBWLb, null);
gsmDataPn.add(gsmBWTx, null);
gsmDataPn.add(gsmBWUnitLb, null);
gsmDataPn.add(gsmMTULb, null);
gsmDataPn.add(gsmMTUTx, null);
gsmDataPn.add(gsmMTUUnitLb, null);
gsmDataPn.add(gsmFTLb, null);
gsmDataPn.add(gsmACKTx, null);
gsmDataPn.add(gsmFTUnitLb, null);
gsmDataPn.add(gsmRTTLb, null);
gsmDataPn.add(gsmRETTx, null);
gsmDataPn.add(gsmRTTUnitLb, null);
gsmDataPn.add(label17, null);
gsmDataPn.add(gsmRTTTx, null);
gsmDataPn.add(label19, null);

RT=Runtime.getRuntime();

cdmaCalculateBt.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            cdmaFER=Double.parseDouble(cdmaFERTx.getSelectedItem());
            cdmaBWL=Double.parseDouble(cdmaBWTx.getSelectedItem());
            cdmaMTU=Double.parseDouble(cdmaMTUTx.getSelectedItem());
            cdmaFTL=Double.parseDouble(cdmaFTTx.getSelectedItem());
            cdmaRTT=Double.parseDouble(cdmaRTTTx.getSelectedItem());
        } catch (NumberFormatException nfe){}
        model.calculatecdma(cdmaFER, cdmaBWL, cdmaMTU, cdmaFTL, cdmaRTT);

        if(BM.equals("Finite"))
        {
            Mode=0;
        }
        else
        {
            Mode=1;
        }
    }
});

gsmCalculateBt.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            gsmBER=Double.parseDouble(gsmFERTx.getSelectedItem());
            gsmBR=Double.parseDouble(gsmBWTx.getSelectedItem());
            gsmMTU=Double.parseDouble(gsmMTUTx.getSelectedItem());
            gsmACK=Double.parseDouble(gsmACKTx.getSelectedItem());
            gsmRET=Double.parseDouble(gsmRETTx.getSelectedItem());
            gsmRTT=Double.parseDouble(gsmRTTTx.getSelectedItem());
        } catch (NumberFormatException nfe){}

        model.calculategsm(gsmBER, gsmBR, gsmMTU, gsmACK, gsmRET, gsmRTT);
        Mode=1;
    }
});

nistnetBt.addActionListener(new ActionListener()
```

```
{
    public void actionPerformed(ActionEvent e)
    {
        destIP=destIPTx.getText();
        sourceIP=sourceIPTx.getText();
        outDELAY=((Calculator)model).getoutDELAY();
        outSTD=((Calculator)model).getoutSTD();
        outBWL=((Calculator)model).getoutBWL();
        outDROP=((Calculator)model).getoutDROP();
        outDELAYSLOW=((Calculator)model).getoutDELAYSLOW();
        outSTDLOW=((Calculator)model).getoutSTDLOW();
        outBWLLOW=((Calculator)model).getoutBWLLOW();
        outDROPSLOW=((Calculator)model).getoutDROPSLOW();

        if(Zustand==0)
        {
            if(Mode==1)
            {
                try
                {
                    RT.exec("cnistnet -u -a" + " " +destIP+ " " +sourceIP+ " "+"--delay"+" "
                        +outDELAY+ " " +outSTD+ " " + "--drop" + " " +outDROP+ " " + "--bandwidth" + " " +outBWL);

                    RT.exec("cnistnet -u -a" + " " +sourceIP+ " " +destIP+ " "+"--delay"+" "
                        +outDELAY+ " " +outSTD+ " " + "--drop" + " " +outDROP+ " " + "--bandwidth" + " " +outBWL);
                }
                catch(java.io.IOException ef)
                {
                    System.out.println("Fehler beim Starten von NistNet");
                }
            }
            else
            {
                try
                {
                    Perl=RT.exec("perl adaptive.pl 8000 4000" + " " +outBWLLOW+ " " +outBWL+ " "
                        +sourceIP+ " " +destIP+ " " +cdmaMTU+ " " +outDELAYSLOW+ " " +outSTDLOW+ " "
                        +outDROPSLOW+ " " +outDELAY+ " " +outSTD+ " " +outDROP);
                }
                catch(java.io.IOException ef)
                {
                    System.out.println("Fehler beim Starten von NistNet");
                }
            }
            nistnetBt.setLabel("Stop Nistnet");
            Zustand=1;
            Modeold=Mode;
        }
        else
        {
            if(Modeold==1)
            {
                try
                {
                    RT.exec("cnistnet -r" + " " +destIP+ " " +sourceIP);
                    RT.exec("cnistnet -r" + " " +sourceIP+ " " +destIP);
                }
                catch(java.io.IOException ef)
                {
                    System.out.println("Fehler beim update");
                }
            }
            else
            {
                try
                {
                    Perl.destroy();
                    RT.exec("cnistnet -r" + " " +destIP+ " " +sourceIP);
                    RT.exec("cnistnet -r" + " " +sourceIP+ " " +destIP);
                }
                catch(java.io.IOException ef)
                {
                    System.out.println("Fehler beim update");
                }
            }
            nistnetBt.setLabel("Start Nistnet");
            outDELAYValueLb.setText("");
            outSTDValueLb.setText("");
            outBWLValueLb.setText("");
            outDROPValueLb.setText("");
            Zustand=0;
        }
    }
};

quitBt.addActionListener(new QuitListener());
```

```
        addWindowListener(new CalculatorWindowListener());
    }

    public void update (Observable model, Object argument)
    {
        outDELAYValueLb.setText(""+((Calculator)model).getoutDELAY());
        outSTDValueLb.setText(""+((Calculator)model).getoutSTD());
        outBWValueLb.setText(""+((Calculator)model).getoutBWL());
        outDROPValueLb.setText(""+((Calculator)model).getoutDROP());
    }
}
```

A.4 CalculatorWindowListener.java

```
package dipapi;

//=====
//This class represents the Listener of the window, which is responsible for actions
//such as pressing the cross on the top right. By closing the application, NistNet is
//stopped and the module is removed from the kernel.
//=====

import java.awt.event.*;

public class CalculatorWindowListener implements WindowListener{

    public void windowOpened(WindowEvent e){}
    public void windowClosing(WindowEvent wEvent){

        try
        {
            Runtime.getRuntime().exec("cnistnet -d");
            Runtime.getRuntime().exec("rmmod nistnet");
            wEvent.getWindow().dispose();
            System.exit(0);
        }
        catch(java.io.IOException ef)
        {
            System.out.println("Fehler beim verlassen");
        }
    }

    public void windowClosed(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}
}
}
```

A.5 QuitListener.java

```
package dipapi;

//=====
//This class represents the Listener of the "Quit"-button which is responsible for
//stopping the application and closing the GUI. By pressing the "Quit"-button, NistNet
//is stopped and the module is removed from the kernel.
//=====

import java.awt.event.*;

public class QuitListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {

        try
        {
            Runtime.getRuntime().exec("cnistnet -d");
            Runtime.getRuntime().exec("rmmod nistnet");
            System.exit(0);
        }
        catch(java.io.IOException ef)
        {
            System.out.println("Fehler beim verlassen");
        }
    }
}
}
```

B Matlab Files

B.1 rlpLoss.m

```
%=====
% Calculates the drop rate for IP datagrams over RLP
%=====

N=72; % Number of frames per IP datagram
time=[0.5 1.5 2.5 3.5 4.5 5.5]; % Time vector in RTT
p=0.10; % p=FER (frame error rate)

% Frame loss probability after after a certain RTT
%=====
p_05=1-p;
p_15=p*(1-p)^2;
p_25=((1-p)^4+2*p*(1-p)^3+2*p*(1-p)^2)*(p^2*(1-p));
p_35=((1-p)^4+2*p*(1-p)^3+2*p*(1-p)^2)*p^2+(p^2*(1-p))*(p^2*(1-p)^2+2*p^2*(1-p))*((1-p)^6+3*p*(1-p)^5+3*p^2*(1-p)^4+3*p*(1-p)^4+6*p^2*(1-p)^3+3*p^2*(1-p)^2);
p_45=((1-p)^6+3*p*(1-p)^5+3*p^2*(1-p)^4+3*p*(1-p)^4+6*p^2*(1-p)^3+3*p^2*(1-p)^2)*p^2*(p^2*(1-p))+p^2*((1-p)^6+3*p*(1-p)^5+3*p^2*(1-p)^4+3*p*(1-p)^4+6*p^2*(1-p)^3+3*p^2*(1-p)^2)*(p^2*(1-p)^2+2*p^2*(1-p));
p_55=((1-p)^6+3*p*(1-p)^5+3*p^2*(1-p)^4+3*p*(1-p)^4+6*p^2*(1-p)^3+3*p^2*(1-p)^2)*p^4;

% Total frame loss probability
%=====
p_tot=p_05+p_15+p_25+p_35+p_45+p_55;
pF=1-(p_tot);

% IP datagram drop rate
%=====
plossIP=1-(1-pF)^N

% Plots PMF
%=====
p_vector=[p_05 p_15 p_25 p_35 p_45 p_55];
stem(time,p_vector);
title('Probability for frame arrival at a certain RTT');
axis([0 6 0 1]);
ylabel('probability p_a_r_r');
xlabel('time in RTT');
```

B.2 rlpDelay.m

```
%=====
% Calculates Delay and Standard Deviation for IP Datagrams over RLP
%=====

N=72; % Number of RLP frames per IP datagram
p=0.01; % Frame error rate FER
q=1-p; % Frame arrival rate
rtt=3; % Roundtrip time in frames

% Calculates PMF for all frames which result in one time slot when lost
%=====
P1=bi(N,p);

% Calculates PMF for all frames at the end of IP datagram
%=====
P2=ending(rtt,p);

% Calculates total PMF
%=====
P3=conv(P1,P2);

% Calculates total delay and standard deviation and plot PMF
%=====
n=N+0:N+length(P3)-1;
forwardchanneldelay=(rtt-3)/2;
n=n+forwardchanneldelay;

stem(n,P3);
grid on
ave=sum(n.*P3)
std=sqrt(sum((n-ave).^2.*P3))

% Auxiliary functions
%=====
function P=bi(N,p)
n=0:N;
```

```

q=1-p;
P=zeros(1,N+1);
for k=0:N
    P(k+1)=ntiefk(N,k)*p^k*q^(N-k);
end

function P=ending(rtt,p)
q=1-p;
P(1)=q^rtt;
for m=0:rtt-1
    temp=0;
    for k=0:m
        temp=temp+ntiefk(m,k)*p^k*q^(m-k);
    end
    P(m+2)=temp*p*q^(rtt-1-m);
end

function y=ntiefk(n,k);
if(k==0) y=1;
else
y=prod(1:n)/(prod(1:k)*prod(1:(n-k)));
end
    
```

B.3 rlcBitrate.m

```

%=====
% Calculates the effective bitrate for IP datagrams over RLC
%=====

br=9.6;           % this is the bitrate in the gui
pl=160;          % RLC payload
fs=176;          % RLC frame size
M=3;             % Maximum number of retransmission rounds
BER=0.1;         % Block Error Rate

% Reduced bitrate due to RLC header transmission
%=====
bitrate=br*1000/8*pl/fs
b=1;
for i=1:M
    b=b+BER^i;
end

% Effective bitrate due to block retransmissions
%=====
eff_bitrate=bitrate/b
    
```

B.4 rlcLoss.m

```

%=====
% Calculates the drop rate for IP datagrams over RLC
%=====

N=76;            % Number of RLC blocks per IP datagram
BER=0.1;         % Block Error Rate
M=6;            % Maximum number of block retransmissions

% Arrival rate for IP after original transmission
%=====
P(1)=(1-BER)^N;

% Arrival rate after each retransmission up to M (PMF)
%=====
for k=2:(M+1)
    a=(1-BER)^(N*BER^(k-1));
    for j=0:k-2
        b(j+1)=(1-(1-BER)^(N*BER^j));
    end
    P(k)=prod(b);
    P(k)=a*P(k);
    clear b;
end

% Total arrival rate after M retransmissions
%=====
x=0;
for i=1:(M+1)
    x=x+P(i);
end
    
```

```
% IP datagram drop rate
%=====
1-x

% Plots PMF and CDF
%=====
index=0:length(P)-1;
subplot(2,1,1);
plot(index,P);
grid on;
axis([0 M+1 0 1]);
title('Probability mass function for IP datagram success');
xlabel('Number of RLC retransmissions');
ylabel('P(k)');
hold on

y(1)=P(1);
for r=2:k
    y(r)=y(r-1)+P(r);
end

subplot(2,1,2);
plot(index,y);
grid on;
axis([0 M+1 0 1.2]);
title('Cumulative distribution function for IP datagram success');
xlabel('Number of RLC retransmissions');
ylabel('sum(P(k))');
hold on
```

B.5 rlcDelay.m

```
%=====
% Calculates Delay and Standard Deviation for IP Datagrams over RLC
%=====

n1=76; % Number of RLC blocks per IP datagram
p=0.02; % Block error rate BER
q=1-p; % Block arrival rate
Poll=9; % Poll intervall in blocks
rtt=2; % Roundtrip time in blocks
channeldelay=(rtt-2)/2; % Channel delay in blocks

% Calculates PMF for first and second retransmission
% for all RLC blocks of an IP datagram
%=====
for n2=0:n1
    PMF1ret(n2+1)=ntiefk(n1,n2)*p^n2*q^(n1-n2);
end

PMF2ret=zeros(n2+1,3*n1+1);
for n2=0:n1
    for n3=0:n2
        PMF2ret(n2+1,n2+n3+1)=ntiefk(n2,n3)*p^n3*q^(n2-n3)*PMF1ret(n2+1);
    end
end
PMF2ret=sum(PMF2ret);

% Calculates PMF for RLC blocks at the end of IP
% datagram, which are after second last Poll -> block
% loss causes more than one additional time slot.
%=====
PMFafterPoll=zeros(Poll-1,Poll+rtt+1);
for block=1:Poll-1
    for k=0:block
        if k==0
            PMFafterPoll(block,k+1)=ntiefk(block,k)*p^k*q^(block-k);
        else
            PMFafterPoll(block,rtt+1+Poll-block+k+1)=ntiefk(block,k)*p^k*q^(block-k);
        end
    end
end
PMFafterPoll=sum(PMFafterPoll)./(Poll-1);

% Calculates PMF for RLC blocks at the end of IP
% datagram, before second last Poll -> block loss
% causes more than one additional time slot.
%=====
PMFbeforePoll=zeros(rtt-1,rtt+Poll+1);
for j=1:rtt-1
    for w=0:Poll-1
        if w==0
            PMFbeforePoll(j,w+1)=ntiefk(Poll-1,w)*p^w*q^(Poll-1-w);
        end
    end
end
```

```
        else
            PMFbeforePoll(j,rtt-j+1+w+1)=ntiefk(Poll-1,w)*p^w*q^(Poll-1-w);
        end
    end
end
if rtt>2
    PMFbeforePoll=sum(PMFbeforePoll)./(rtt-1);
end

% Calculates PMF for whole transmission of IP datagram
%=====
total=conv(conv(PMFafterPoll,PMFbeforePoll),PMF2ret);
n=n1+0:n1+length(total)-1;
n=n+channeldelay;

% Calculates total delay and standard deviation
% and plots the PMF's
%=====
avel=sum(n.*total)
std=sqrt(sum((n-avel).^2.*total))

index1=0:length(PMF2ret)-1;
subplot(2,2,1)
stem(index1,PMF2ret);
title('PMF for all blocks after 2nd retransmission');
xlabel('Number of additional time slots');
ylabel('Success probability');

index2=0:length(PMFafterPoll)-1;
subplot(2,2,2)
stem(index2,PMFafterPoll);
title('PMF for blocks at the end, after Poll');
xlabel('Number of additional time slots');
ylabel('Success probability');

index3=0:length(PMFbeforePoll)-1;
subplot(2,2,3)
stem(index3,PMFbeforePoll);
title('PMF for blocks at the end, before Poll');
xlabel('Number of additional time slots');
ylabel('Success probability');

subplot(2,2,4)
stem(n,total);
title('Total PMF');
xlabel('Total number of time slots');
ylabel('Success probability');
```