# Modelling and Evaluation of a Bluetooth Data Logger in the Presence of Interference Sources

MAGNUS KARLSSON

KTH Microelectronics
and Information Technology

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LCN 2005-08

**Royal Institute of Technology** **12th April 2005**
Master's report

# Modelling and Evaluation of a Bluetooth Data Logger in the Presence of Interference Sources

Magnus Karlsson

Examiner: Prof. Gerald Q. Maguire Jr.
Royal Institute of Technology, Stockholm
Department of Microelectronics and Information Technology

Supervisor: Carl-Axel Ohlsson
Industrial Development Centre, Olofström

# Forword

During the 10th century the Danish Viking king Harald Blåtand ruled in Scandinavia. He is remembered for his excellent communication skills which lead to the uniting of Norway and Denmark. When the idea of a technology that would unite data- and telecommunication was born it was named Bluetooth after Harald Blåtand[1].

At first Bluetooth was intended as a wire replacement for computers, headset, and cellular telephones. However, the technology has also turned out to be appropriate for industrial applications. The Industrial Development Centre in Olofström has developed a measurement logger which collects information from a number of sensors (e.g. temperature, pressure, etc.). The information from these is sent wirelessly with the Bluetooth technology to, for instance a computer that handles them in an appropriate way.

Between the computer and logger there could be sources of interference that have negative effects on the effective data rate, i.e. how many measurement values can be transmitted during a certain amount of time. The measurement values can be sent immediately by the logger or stored in the logger's buffer and then send in larger groups, experiments has shown that the size of these groups will affect the data transfer rate - especially if there is interferences.

In this master's thesis the Bluetooth Logger's effective data transfer rate was examined though a number of computer simulations and experiments with or without buffering and with different kinds logger settings and with interferences. The results of these measurments were used to define a model for the Bluetooth Logger's data communication and to construct a simulation program that developers can use in a flexible way of evaluating the operation of a measurement system based on this Bluetooth logger

[1]Blåtand is Danish for Bluetooth

# Förord

På 900-talet regerade den danska vikingakungen Harald Blåtand. Han blev känd för att genom sin goda förmåga att föra dialog lyckades förena Norge och Danmark. När idén föddes om en trådlös teknologi som skulle förena data- och telekommunikations branschen fick det namnet Bluetooth[2] efter just Harald Blåtand.

Bluetooth var initialt avsett som kabelersättare för datorer, headset och mobiltelefoner. Med tiden har det emellertid visat sig att teknologin lämpar sig väl för industriella tillämpningar. Industriellt Utvecklingscentrum i Olofström har utvecklat en mätvärdeslogger som samlar information från ett antal mätare (exv. temperatur, tryck, etc.). Informationen från dessa skickas sedan trådlöst med Bluetooth teknologin till exv. en dator där de tas om hand på lämpligt sätt.

Mellan datorn och loggern kan det finnas störningar som har negativa effekter på den effektiva dataöverföringshastigheten, dvs. på antalet mätvärden som kan skickas över på en viss tid. Om mätvärdena skickas direkt de kommer in i loggern, eller om de mellanlagras och skickas i skurar och i så fall hur stora skurarna skall har genom experiment visat sig ha direkt att påverka datahastigheten. Speciellt då det förekommer störningar.

I det här examensarbetet kommer den effektiva datahastigheten att undersökas med en rad datorsimuleringar och experiment med eller utan mellanlagringar och med eller utan störningar. Resultatet från dessa simuleringar och experiment ska ligga till grund för en modell som beskriver dataöverföringen för Bluetooth mätvärdesloggern samt konstruktionen av ett simuleringsprogram som utvecklare kan använda för att konstruera sina trådlösa mätsystem.

[2]Bluetooth betyder på svenska Blåtand

## Abstract

Industrial Development Centre (IUC) in Olofström inc. has constructed a measurement value logger which can sample values from eight channels, buffer them and then send them wireless with the Bluetooth technology to e.g. a computer.

In this thesis the data transfer rate, i.e. the number of values per second has been studied for different logger settings and when there are interferences in the Bluetooth traffic. How Bluetooth is affected by interferences has been studied with a number of experiments performed at IUC's RF-damped Faraday's cage.

The thesis provides a model for this Bluetooth logger. The model extends the existing simulation system NS2-UCBT with a "logger protocol". NS2-UCBT was also extended to better support modeling of losses (due to Bluetooth channel impairments) and for the model of these losses to be based on experiments. The resulting simulation program allows developers to construct and evaluate a measurement system utilizing such a Bluetooth logger. Using the simulation model, the data rate measured in samples per second was examined for the logger. The simulations shows that optimizing the logger's configuration will improve that data rate considerable.

This report contains: a summary of the problem and earlier research, an explanation of the simulation system and the simulation program, comparisons between simulations and experiments, some conclusions, and proposes future work in this area.

Keywords: **Bluetooth**, **simulations**, **NS2-UCBT**, **Logger**, **buffer** and **interferences**.

## Sammanfattning

Industriellt Utvecklings Centrum (IUC) i Olofström AB har tagit fram en mätvärdeslogger som kan sampla värden från åtta kanaler, mellanlagra dem och skicka dem till exv. en dator trådlöst med Bluetooth teknologin.

I det här examensarbetet har överföringshastigheten, dvs antal mätvärden per sekund studerats för olika inställningar på loggern och då det förekommer störningar i Bluetooth trafiken. Hur Bluetooth påverkas av störningar har undersökts genom en rad experiment i IUC's RF-dämpade skärmlabb.

Arbetet har lett fram till en modell för mätvärdesloggern och ett simuleringsprogram som gör det möjligt för utvecklare att konstruera och utvärdera sina mätsystem med mätvärdes logger. Modellen använder det befintliga NS2-UCBT simuleringssystemet utvidgat med ett "logger protokoll", mer utvecklad förlusthantering än NS2-UCBT i grundutförandet erbjuder och flexibel förlusthantering baserad på experiment. Med simuleringsmodellen undersöktes loggerns datahastighet i sampel per sekund. Simuleringarna visa att genom att förbättra loggerns konfiguration kan avsevärt högre datahastighet nås.

Den här rapporten innehåller: en sammanfattning av problemställningen och tidigare forskning, en beskrivning av simulationssystemet och simulationsprogrammet, jämförelser mellan simuleringar och experiment, en del slutsatser, och förslag på framtida arbete i området.

Nyckelord: **Bluetooth**, **simulering**, **NS2-UCBT**, **logger**, **buffer** och **störningar**.

## Acknowledgments

I would like to thank **Carl-Axel Ohlsson** at Industrial Development Centre in Olofström who has given me the opportunity to perform this master's thesis. He has also been my supervisor at IUC and has spent a lot of time (which I think is his free time) to discuss with me and help me in different ways during the work. He has also given me the opportunity to decide for my self over the thesis and which direction it should take.

I also would like to thank **Prof. Gerald Q. Maguire Jr.**, Department of Microelectronics and Information Technology, Royal Institute of Technology, Kista, Stockholm for his quick answers and great patience. A more dedicated examiner is hard to imagine.

I also would like to thank **Gösta Strandberg** at Industrial Development Centre in Olofström who has helped me obtain the necessary equipment and has answered my questions, thus making it possible for me to perform experiments at the IUC's Faraday cage.

Finally I would like to thank my girlfriend **Wictoria** who has allowed me to use her apartment to work from and who has listned to me talking about my master's thesis even though she has no interest or knowledge about the subject.

Thank's all of you!

# Tack till

Jag skulle vilja tacka **Carl-Axel Ohlsson** på Industriellt Utvecklings Centrum i Olofström som har givit mig möjligheten att utföra det här examensarbetet. Han har också varit min handledare på IUC och lagt ner mycket (vilket jag tror av sin fritid) på att diskutera med mig och hjälpa mig på olika sätt under arbetets gång. Han har även givit mig stora möjligheter att själv styra över mitt arbete och bestämma vilken riktning examensarbetet skall ta.

Jag vill även tacka **Prof. Gerald Q. Maguire Jr.** vid institutionen för mikroelektronik och informationsteknologi på Kungliga Tekniska Högskolan i Kista, Stockholm för hans snabba svar och hans stora tålamod. En mer hängiven examinator har jag svårt satt föreställa mig.

Jag vill även tacka **Gösta Strandberg** på Industriellt Utvecklings Centrum i Olofström som har hjälpt mig med att skaffa fram utrustning och svara på frågor som gjort det möjligt för mig att utföra mina experiment i IUCs skärmlabb.

Slutligen vill jag tacka min flickvän **Wictoria** som har låtit mig använda hennes lägenhet att sitta och arbete från och som har lyssnat på mig när jag pratat om examensarbetet fast än hon varken är intresserad eller insatt i vad det har handlat om.

Ett hjärtligt tack till Er alla!

# Contents

# List of Figures

# List of Tables

# 1 Introduction

To state the effective data transfer rate between two Bluetooth devices based on existing theoretical models is difficult today. How can one guarantee that a computer with a Bluetooth interface connected to one or several Bluetooth equipped logging devices actually receives all necessary information from the logging devices in a bounded time? If for instance some device needs to resend a packet because of some interference in the transmission, the effective data transfer rate will decrease. Thus one cannot rely on that the maximum rate stated by the Bluetooth protocol actually being achieved; only that it will **not** be *exceeded*.

This master's thesis will focus on an investigation of methods /models for effective and fast data transfer between a computer with a Bluetooth interface and Bluetooth equipped logging devices (with focus on the BlueCenter/IUC DL141E). The main goal is to optimize the logger settings for **different measurment settings**.

The BlueCenter/IUC DL141E has eight single or four differential input channels and can work as an ordinary logger (see 2.4). Data can be transferred after manual data collection or continuous/intermittent logging of values can be collected automatically. The experimental and theoretical investigations will focus on the BlueCenter/IUC DL141E, but most of the results will be applicable to any Bluetooth data source device. Details of the BlueCenter/IUC logger can be found in section 2.4.

The aspects that should be considered when considering Bluetooth data transfer rate optimization and qualification in this thesis are:

- **Environment :** where are the equipment located? A damp environment for example and will increase the number of packets that needs to be resend. While lot of sheet metal walls will increase the range of the network because they will bounce on the sheet metal and it will also shield other areas. Resending of packets will reduce the effective transfer rate. Are there any sources of interference? Examples of interferences could be: other wireless devices, welding equipment, micro wave ovens and heaters, wireless LANs (WLANs) etc. Other Bluetooth networks will also interfere. For instance if a logger is connected to a computer in a workshop there could be interference from Bluetooth equipped cellular telephone. Interferences has been studied in this thesis by performing experiments in IUC's faradays cage.

- **The logger buffer :** the size of buffer in the logging device could effect the effective data transfer rate in many aspects. Should the buffer be filled before its values are collected or should a value be sent immediately? Different approaches should probably be used depending on the size of the buffer and how many channels are being filled. It is also crucial to decide upon the average maximum delay from when a value is put in the buffer until it has reached the Bluetooth connected computer. The buffer's effect on the data transfer rate and measured output under different buffer strategies is the main topic of this thesis.

- **Packet type :** the Bluetooth protocol [1] supports a number of different types of packets. Which of these supply the most effective data transfer rate in a given environment? If a large packet is received properly the data rate will most certainly be higher than if a small packet is received properly. On the other hand, loss of a large packet will cost more than loss of a small packet with respect to data transfer rate.

- **Network size :** how will the effective data rate be effected as the piconet grows in size? What different approaches can be used when constructing larger networks? In a piconet with one master connected to several slaves the scheduler decides which slave gets to transmit next. Because the master does not know if the slaves have something to send, the choice of scheduling algorithm will have a great effect on effective data transfer rate at different scenarios. There are additional aspects when considering large network sizes; however, most of them are out of the scope for this thesis.

Theory, experiments, and simulations were used to define and validate a model of Bluetooth Logger's data transfer rate in various settings (i.e., under various conditions). See figure 1. The experiments were

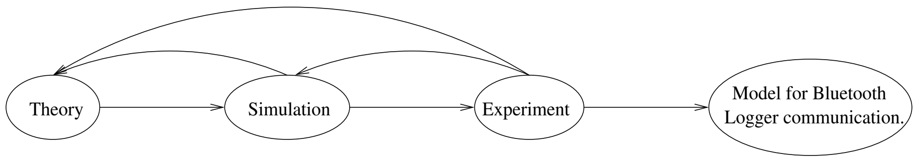


Figure 1: The thesis workflow

facilitated by the availability of first class measurement equipment for both data traffic and RF signals in a Faraday cage (i.e., a RF screened measurement chamber (see section 3.1). Thus the measurements should have a high degree of reproducibility and traceability. For more information about the measurements see section 3. The simulations were performed by augmenting the existing Bluetooth model for the simulation system NS2 [2] with the UCBT [3] extension (see section 4.1). An example of how the model is constructed could be the following workflow:

1. A simulation is performed where one Bluetooth equipped computer sends some data to another Bluetooth equipped computer (see figure 2). This simulation use the underlying Bluetooth data transfer model.

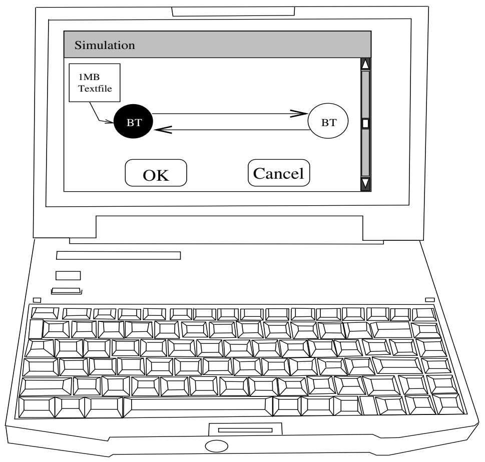


Figure 2: Simulation example

2. An experiment is performed that resembles the simulation, i.e. a computer with a Bluetooth radio sends some data to another Bluetooth equipped computer (see fig 3). The experiments are performed in a Faraday cage (see section 3.1).
3. The results from the simulations are compared with the results from the experiments.
4. The simulation model is adjusted (or experiment changed if necessary) until the simulation and experiment provide the same result.

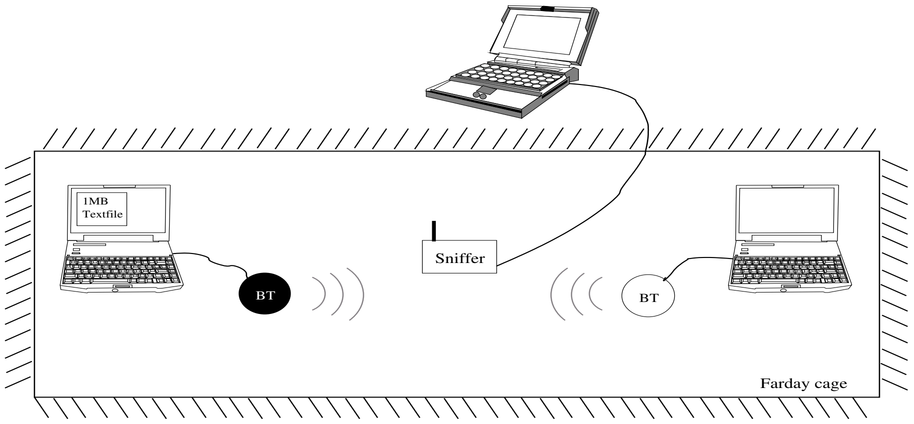


Figure 3: Experiment example

5. Add the parameters determined from this simulation to the model.

Once the experiment and the simulation provide the same result (if that is possible) one can add a noise source and redo steps 1-5 with the new scenario, etc.

As the work proceeds more and more parameters will be verified and added to the model. To allow flexible simulations for a common user a program (written in java) that uses the model to simulate the communication of the logger. This program can be used to compare different logger settings to see which performs the best. It could also be used to examine if a desired system configuration could actually be performed with the desired properties, i.e. meet the delay bounds. Finally some of the DL141E's properties could be changed to examine if a different logger construction would perform better (see section 6).

The work in this thesis is the foundation of a project to characterize both the logger and Bluetooth data traffic in general. The work will hopefully be continued via other projects, master's thesis, etc. at IUC Olofström for many years (see section 7.2). The theory described in this report (mostly in section 2) covers more than the current model. Thus the theory should form a basis for further work on this subject.

To summarize, this thesis provides:

1. A background to the thesis and the subject (see section 2). This background contains several references to articles, etc in the subject and would be the first thing that one should read if one were about to continue with some of the topics for this thesis.
2. A simulation model that is an extension of NS2-UCBT (see section 4). The model uses results from experiments to handle interference in traffic in a flexible way (see section 5.2.2).
3. A user friendly program (see section 5) that allows a developer to perform simulations where different parameter settings could be compared in different environments, i.e. with different interferences. These simulations can help the developer to optimize the measurement system and to see if a desired system could be accomplished.
4. Results from a number of experiments (see section 6) with settings as the ones possible with the Dl141E and with settings other than those that are inside the frames of the DL141E.
5. Conclusions from the experiments and several proposals on further works related to this subject (see section 7).

# 2 Background

This section gives a overview of topics that are of relevant for this master's thesis. First there is a short introduction to Bluetooth$^{TM}$. The *Bluetooth Overview* is followed by a section, *Buffer* which gives some background as to why adding a buffer to a measurement logger is crucial for achieving high transfer speeds and reliability. In the section *Data Throughput* different aspects concerning Bluetooth data transfer rate are described. The section *BlueCenter/IUC's DL141E* will describe the measurement value logger DL141E.

## 2.1 Bluetooth Overview

This section presents a brief overview of Bluetooth, embracing those parts that are of special interest for this master's thesis. For readers with further interest there are a number of more thorough Bluetooth descriptions; [1, 4, 5].

Bluetooth is a wireless technology designed to replace short cables. Bluetooth would typically be used to replace cables between a cellular telephone and headset, a cellular telephone and a computer, or between a computer and mouse/keyboard. The Bluetooth trademark belongs to the Bluetooth Special Interest Group (SIG) [1] who defined the Bluetooth protocol stack. The basic protocols in the Bluetooth protocol stack are: *Baseband Protocol* which is responsible for framing, flow control, timeout mechanisms, and medium access control, *Link Manager Protocol (LMP)* which manages the link state and the power control, and the *Logical Link Control and Adaptation Protocol (L2CAP)* which is a data link level protocol which operates in both connection oriented and connectionless modes and is responsible for packet multiplexing, segmentation, and reassembly. All of the above utilize a radio based physical layer.

Bluetooth operates in the 2.4 GHz Industrial, Scientific and Medical (ISM) band. There are a lot of other wireless devices that also use this ISM band, these can lead to interference. To reduce the effects of interference Bluetooth divides the 2.400 - 2.4839GHz spectra into 79 channels[3] and uses a fast frequency hopping strategy with 1 600 hops/s (625$\mu$s/hop) between these channels.

Bluetooth offers two data link layer transmission services:

- **Asynchronous Connectionless (ACL)** service for data transmissions. This service utilizes an automatic repeat request (ARQ) algorithm in which packets are retransmitted until a positive acknowledgment is received by the sender. This will ensure that the packet has not been lost during transmission.
- **Synchronous Connection Oriented (SCO)** service for real-time applications, especially voice. This is a symmetric point-to-point service in which the master transmits during reserved slots and the selected slave responds in the following slot.

ACL supports six types of packets which are 1, 3, or 5 slots long (see table 1). Note that all packets are followed by a reply packet thus the effective number of slots are 2, 4, or 6. Some modes support Forward Error Correction (FEC).

Bluetooth networks are built in two ways:

1. A master connects from one to seven slaves, forming a so-called *piconet* (see figure 4).
2. A Bluetooth device acting as slave in one net and master in another can bridge packets from one piconet to another. These Bluetooth devices form a so-called *scatternet* (se figure 5).

[3]The number of channels which can be used varies by country.

| Packet type | FEC encoded | Number of occupied slots | bytes of data |
|---|---|---|---|
| DM1 | YES | 1 | 17 |
| DH1 | NO | 1 | 27 |
| DM3 | YES | 3 | 121 |
| DH3 | NO | 3 | 183 |
| DM5 | YES | 5 | 224 |
| DH5 | NO | 5 | 339 |

Table 1: Bluetooth ACL packet types

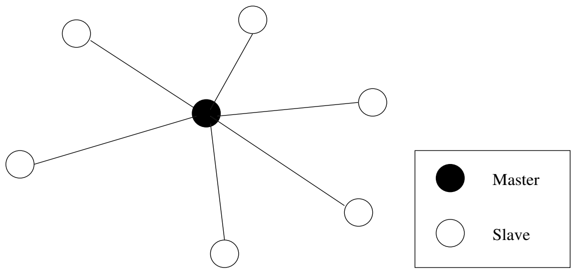


Figure 4: Piconet example

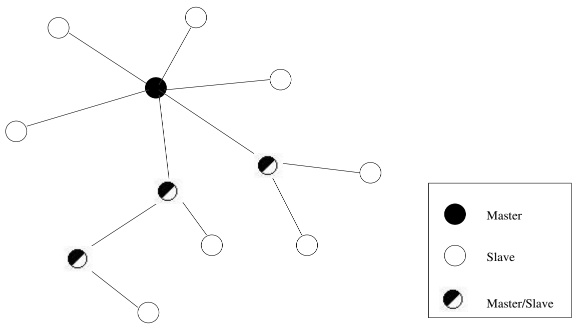


Figure 5: Scatternet example

Initially Bluetooth was intended to operate only within a in 10m radius. However, there was some interest in using Bluetooth for longer distances. This lead to the classification of devices into three transmit power classes (see table 2).

| Class | Max. output power | Range |
|---|---|---|
| 1 | 100mW (20dBm) | 100m+ |
| 2 | 2.5mW (4dBm) | 10m+ |
| 3 | 1mW (0dBm) | 1m+ |

Table 2: Transmit Power Classes

## 2.2 Buffering

The buffer in the logger together with the effective data transfer rate must be sufficient for the time resolution of the measured signal. Consider the example in figure 6 in which a sinusoid signal should be transferred with a DL141E (see section 2.4). The frequency of the signal is omitted in this example.

There are three approaches for transmitting the data that will be considered in this thesis:

1. **immediate:** values are sent immediately as soon as they arrives. The instanious sample rate can **not** exceed the maximum transfer rate of the logger.
2. ***k*-buffered:** values are stored in the logger's buffer and sent in groups. The parameter $k$ indicates how many samples should be stored before transmission.
3. **full-buffered:** the buffer is filled before the values are transmitted, i.e. *k-buffered* with $k$ set to the buffer size.

Consider the signal in figure 6. Imagine that the signal is of such a frequency that the available Bluetooth transfer rate only allows ten samples per period. Imagine further that there is interference in the channel, then the effective data rate will decrease hence the received signal will not represent the sample signal. In figure 7 there are four examples of transferred signals. First there is a interference free environment where all ten values could be received properly. In the next three interference occurs as a percent of packet error rate (PER), e.g. a packet error rate of 50% results in five transmitted sample values for this example. Note that values do not disappear as a result of interference, but the sample rate that has to be decreased due to the lower transfer rate.

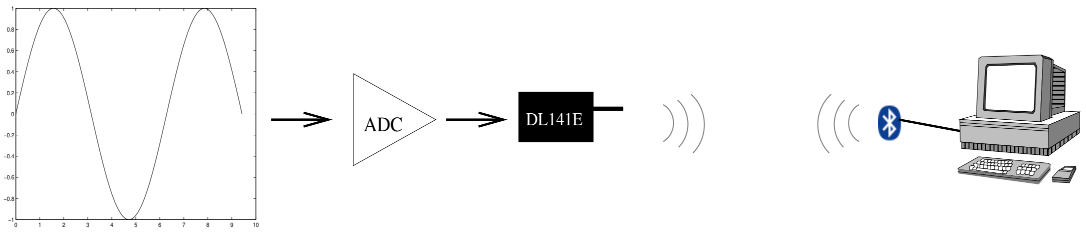


Figure 6: Signal transfer example

As shown in the example in figure 7 it is possible that a signal with a certain frequency could not be measured at a high PER. However, if it would be possible to store samples in the logger's buffer, then the samples could be sent as shown in figure 8, where a 50-samplebuffer and a 100-samplebuffer is used. Note that the buffer will affect the delay, i.e. the time from which a sample enters the buffer until it is received by the computer. This approach may not be applied on continuous signals (which is mostly the case) because the buffer could overflow or as is the case with the DL141E (see section 2.4) since the logger does not support sampling and sending samples at the same time. To avoid this problem one could use two DL141E where they alternate so that one samples when the other sends and vice versa (see figure 9). The *immediate*, *k-sampled-buffered* and *fully-buffered* approach together with the *Alternately sample/transmit approach* are the main topic for this thesis. The simulations and experiments with these approaches will be performed in different environments and with different parameters that affect the effective data transfer rate (see section 2.3).

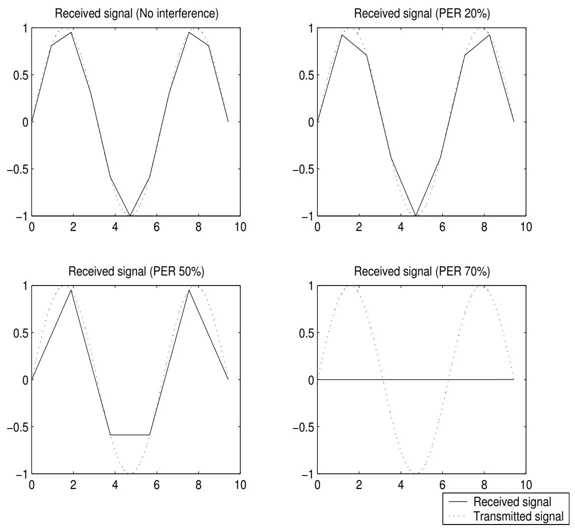


Figure 7: Transfered signals example

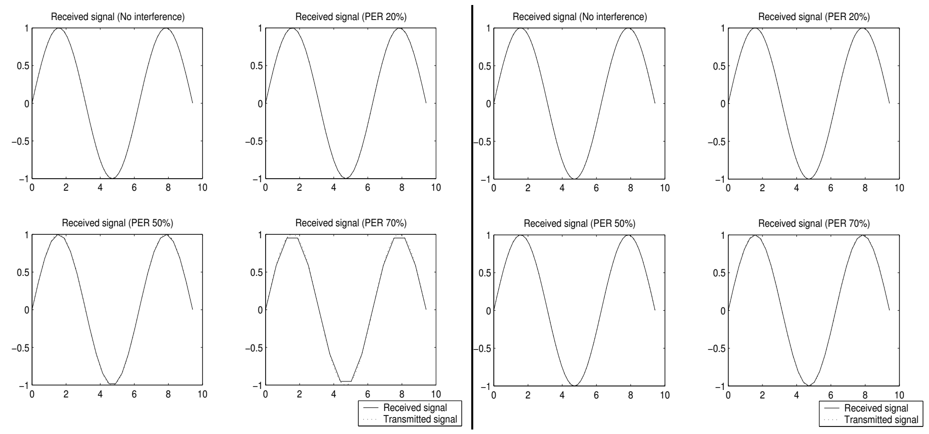


Figure 8: Transfered signals example with the 50- and 100-buffer approach

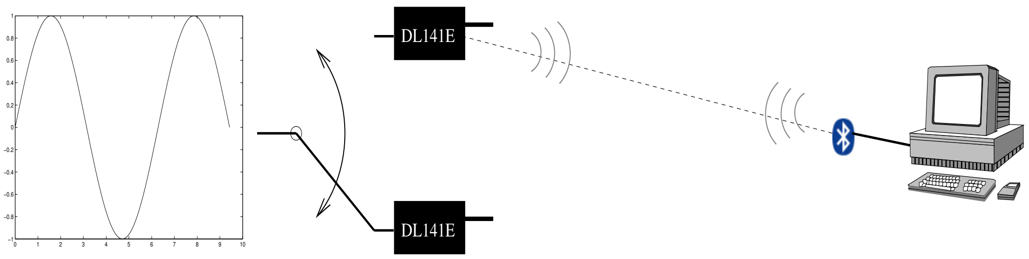


Figure 9: Alternately sample/transmit approach

## 2.3 Data Throughput

There are a number of factors that affect the data transfer rate in a Bluetooth network. In this section some of these factors are described through the following sub-sections: *Packet Size*, *Interferences*, and *Network scaling*.

### 2.3.1 Packet Size

The Bluetooth specification [1] supports six different ACL packets, see table 1 in section 2.1. In [6] the net bit rate for each of the six packet types is determined for an error-free environment. These results are shown in table 3.

| Packet type | net bit rate [kbit/s] |
|---|---|
| DM1 | 108.8 |
| DM3 | 387.2 |
| DM5 | 477.9 |
| DH1 | 172.8 |
| DH3 | 585.6 |
| DH5 | 723.2 |

Table 3: Bluetooth packets net bit rate

To see why the net bit range differs between the different packets consider the following example: A 1500-byte message can be sent using four 339-byte DH5 packets and a 183-byte DH3 packet (as $1500 < 339 \cdot 4 + 183 \cdot 1$). This transmission occupies 23 slots ($4 \cdot 5 + 1 \cdot 3$) plus five slots for the acknowledgment of the five packets. If one instead used DH3 or DH1 packets the message would take 36 and 112 slots respectively. The results shown in table 3 will be particularly significant to this thesis as they provides an upper bound to throughput.

In [7] the authors have extended the work in [6] for packets a noisy environment. The authors have derived analytical expressions for each packet type. They show as expected that at high SNR[4] the packets reach their maximum throughput value, i.e. those stated in table 3. However, at low SNR using different packets will achieve maximum throughput. This implies that the choice of packet type should be adaptively selected. At high SNR *long uncoded* packets are preferable and at low SNR *shorter* packets protected by Hamming code are preferable.

In [8] the authors investigated three methods for improving Bluetooth point-to-point performance. The first strategy was *custom FEC coding* using the AUX1 packet [1] to transport BCH[5] codes that are more powerful than the Hamming codes used by the DM$x$ packets. The second strategy was *distributed detection* where the packet is broadcasted to a group of two or more receivers. These two strategies are not very important for this thesis as the logger has insufficient computational power to compute BCH encodings nor does the receiver of the logger's data have multiple receivers. However the third strategy involves *adaptive rate control* which dynamicy changes the packet type based on the channel SNR and it always sends BCH coded packets which maximizes throughput. The SNR is determined by using past knowledge to predict future SNR. Results show a sharp improvement in the throughput when using adaptive BCH codes for SNR below 22dB.

[4]Signal to Noise Ratio

[5]Bose, Chaudhuri, Hocquenghem: a family of cyclic code for error detection and correction

The authors also suggests a *fully adaptive* method that chooses the packet type among the six packets (see table 1) that provides the highest throughput. They show nice results from this approach. Changing the packet type adaptively does not violate the Bluetooth specification [1], however many of today's applications define the packet type at setup and thus it cannot utilize this method. For instance work the DL141E operates in this way (see section 2.4). This would make the *fully adaptive* method difficult to implement with an actual Bluetooth system.

An other example of building on [7] is [9], here the authors use *turbo codes* which are a powerful FEC coding. Their approach uses the AUX packet, for which ARQ is disabled (see section 2.1) and the error correction is performed off-chip, in a DSP or computer. This also does not violate the Bluetooth specification [1] as the coding occurs above the Bluetooth protocol layers. The use of *turbo codes* improves the throughput at low SNR, but the relative small packet size of Bluetooth prevents the full potential of *turbo cods* from beeing realized.

The authors of [10] also build on [6] and suggest an adaptive coding scheme based on a *hybrid FEC/ARQ* scheme. When the SNR is high no FEC is used, but at low SNR a FEC is applied to the data. The channels are monitored with a link status monitor (LSM). The *hybrid FEC/ARQ* has been used as a base for [11] where an adaptive method that chooses the right packet type depending on the BER[6] is derived. The method uses an error counter and a effective bandwidth calculator to choose the right packet type and FEC.

This section has described how the data throughput depends on the choice of packet type and SNR. These results will be of importance later when describing the simulations and experiments.

### 2.3.2 Interference

Because Bluetooth devices utilize the ISM band (see section 2.1) they could be exposed to many sources of interference. This will lead to retransmissions of packets which will result in lower effective data transfer rate. Interference could be of importance for this thesis as it affects whether the buffering (here in the DL141E) should be different depending on the conditions of the transmission channel. Buffering issues are further discussed in section 2.2

Examples of common sources of interference could be: welding equipment, microwave ovens, etc. In [12] the author devotes an entire master's thesis to investigating a microwave oven's interference with Bluetooth data transfer performance. Microwave ovens are interesting because they produce high power interference in the ISM band. The result shows that different microwave ovens behave differently i.e. each oven must be treated as an individual interference source. This probably applies to all sources of interference of this kind. Two similar welding systems would most certainly behave in different ways regarding their interferences with Bluetooth network. The same welding set might even interfere differently depending on who is using it and what they are doing. The characterisation of interference will be a compromize between flexibility and reliability.

A major source of interference in Bluetooth networks are wireless LANs (WLAN), such as IEEE802.11b. There has already been a lot of research in the area. The reason for the large interest is that WLAN and Bluetooth are most likely to appear in the same environment. Bluetooth uses fast frequency hopping (see section 2.1) as strategy to avoid interference in the ISM band. Because of this WLAN is more affected by Bluetooth then vice versa. IEEE found this problem of such a significance that they formed a *Coexistent Task Group*[13]. This is such a complex area that it could cover more that one master thesis itself, thus it will be out of the scope for this thesis.

[6]Bit Error Rate

There are a lot of published articles concerning Bluetooth and WLAN interference. One of the most thorough and widely cited is [14], where the authors draw the following conclusions:

- Power control has small benefits. Even if a WLAN's power is increased to over fifty times the power of Bluetooth, WLAN packet loss were not sufficient reduced because Bluetooth is a dynamic source of interference which does not listen to the channel before transmitting. For Bluetooth on the other hand, low WLAN power may help avoid interference since Bluetooth's power is concentrated in a narrow frequency band while the 802.11 WLAN power is spread over a large band.
- Using longer packets in Bluetooth may reduce the interference to WLAN, since the Bluetooth transmitter does not hop in the middle of a packet transmission.
- Bluetooth voice represents the worst interference source for Bluetooth. Because Bluetooth does not back off if there is someone else already transmitting in the current channel.
- The errors in Bluetooth caused by interference from WLAN are often too many for the error correcting block codes in Bluetooth to overcome, so the packet has to be retransmitted.

These four conclusions lead to the main conclusion of the article which is that a coexistence mechanism between Bluetooth and WLAN **should** be developed. Two examples of articles proposing such interference mitigation techniques are:

- the OverLap Avoidance (OLA) scheme [15].
- a backoff strategy (BIAS) for Bluetooth [16]. This strategy avoids transmission of during WLAN transmission.[7]

Another major area which has been the subject of many articles is interference between Bluetooth networks. This is an important area since different Bluetooth networks are likely to exist in the same area. An example of this could be a small company which has a Bluetooth equipped computer connected to a DL141E logger. In this company there are two fork-lifts operating. Each forklift driver needs to have their hands free and thus uses a Bluetooth wireless headset to communicate with their cellular telephone. As they need to leave the fork-lift occasionally and do not want cables between their telephone and headset, they use Bluetooth equipped telephones and a Bluetooth headset. There are now have three Bluetooth networks present. Depending on each fork-lift's location these networks may or may not affect each other.

In this thesis a point-to-point connection is considedred. Everything else in the environment e.g. other Bluetooth nets and WLAN are considered as interferences. To simulate interference from e.g. a Bluetooth or WLAN device one would have to look at the spectra of the device and then add this to the model as described in section 5.2.2. The articles [17, 18, 19, 20, 21] all considers several Bluetooth networks close to each other. However, they mostly treats aggregated throughput[8] which is out of the bounds for this thesis.

This section has introduced issues concerning interference in Bluetooth networks. Interference from some devices, such as microwave ovens and welding equipment, are hard to predict and derive an analytical expression for. WLAN vs. Bluetooth interference there has been a lot of research. However, most of the researches focus on Bluetooth interference to a WLAN. The reason for this is probably that Bluetooth interferes with WLAN more than WLAN interfere with Bluetooth. Bluetooth piconets interference with each other is an important area that has been the subject of many articles. WLAN interferences will be out of the scope for this master's thesis and building an experiment with several piconets will be difficult with the available resources.

[7]The interpretation of US FCC part 15 regulations by G Q Maguire Jr. is that since Bluetooth is a tertiary user of the 2.4GHz ISM band it **must** back off (or be considered an illegal interference source).

[8]Total throughput of the collocated piconets

### 2.3.3 Network Scaling

When Bluetooth networks become large there are many issues that need to be considered. Many questions arise which has lead to a lot of research in the area. However, in this master's thesis it will not be possible to perform measurements on larger networks, thus this thesis will focus simply on point-to-point connections. However some of the issues concerning larger Bluetooth networks will be described in the following example.

Imagine that one would like to build a system with 30 DL141E loggers. There are four main approaches to construct such a system:

1. **The point-to-point approach :** each DL141E is connected via its own point-to-point connection to the computer (see figure 10). The computer then needs to have 30 separate Bluetooth radios connected via some bus(es).

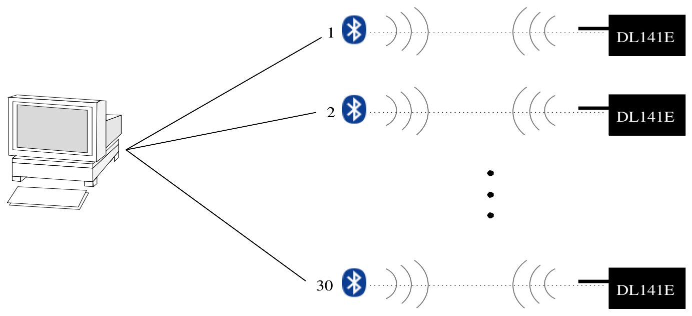


Figure 10: The point-to-point approach

2. **The point-to-multipoint approach :** this approach uses (four) Bluetooth radios that are connected to the computer (via up to four buses) (see figure 11). If each computer connected Bluetooth radio acts as a master and forms its own piconet (see section 2.1) where radios 1-3 can each have up to seven slaves, then the aggregated throughput is approximately four times that of a single piconet.
3. **The parked slaves approach :** the Bluetooth protocol [1] supports a *park state* for a slave in a piconet. The *park state* is a state where the slave does **not** participate in the piconet channel, but it remains synchronized with the channel. By switching between active and parked state the piconet could be made arbitrary large (see figure 12). However the aggregated throughput is at most that of one Bluetooth piconet.
4. **The scatternet approach :** in this approach all the DL141E form a scatternet. The loggers use each other to forward their information to the computer (see figure 13). However the aggregated throughput is at most that of one Bluetooth piconet.

At first glance, the point-to-point approach (1) seems like a strange approach for constructing a network. However, this is the approach that BlueCenter/IUC will use in their networks. At least for the initial construction of such networks. This is one of the reasons why this thesis focuses on point-to-point connections. One should keep in mind that the cost of the computer connected Bluetooth radio is neglected compared to the cost of the DL141E. The benefits of using the point-to-point approach are:

- **Low complexity.** Setting up a point-to-point connection is less complex than setting up point-to-multipoint connections.

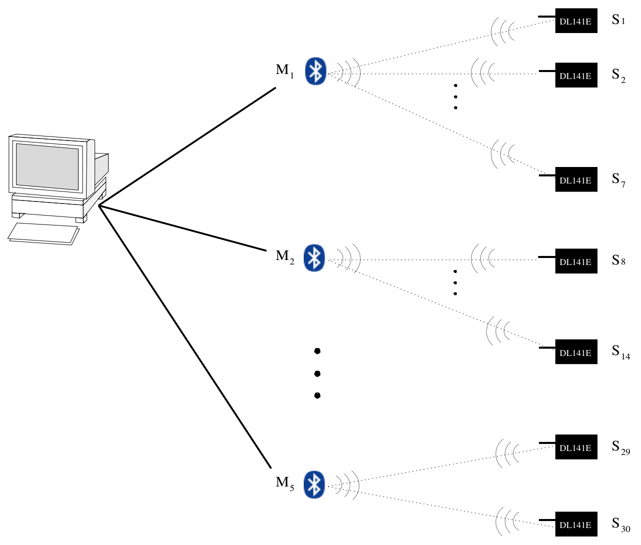


Figure 11: The point-to-multipoint approach

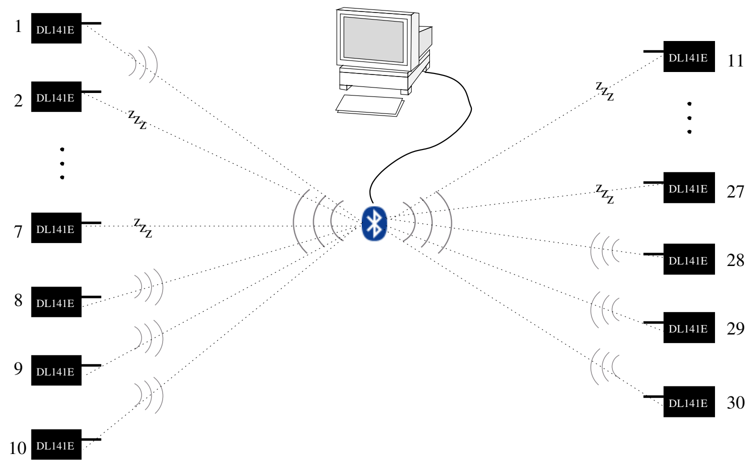


Figure 12: The parked slaves approach

- **Facilitates construction of long distance networks.** If the DL141E are located far away from each other it is possible to use cables to spread the computer's 30 Bluetooth radios allowing the network to cover a larger area.
- **Data rate gainings.** The throughput could be up to that of 30 independent Bluetooth links.

In the point-to-multipoint approach the computer only needs five radios (for this example). Where each computer connected Bluetooth radio is a master of a piconet. The way in which the master schedules packet transmission to slaves (i.e. DL141E) or polls them are decided by the master's *scheduler*[9]. Scheduling

[9]The computer in the point-to-point approach will also need a scheduler for its busses, but this will be ignored in this example, as they are assumed to be independent piconets and the bus bandwidth is assumed to be much greater than the aggregated Bluetooth bandwidth

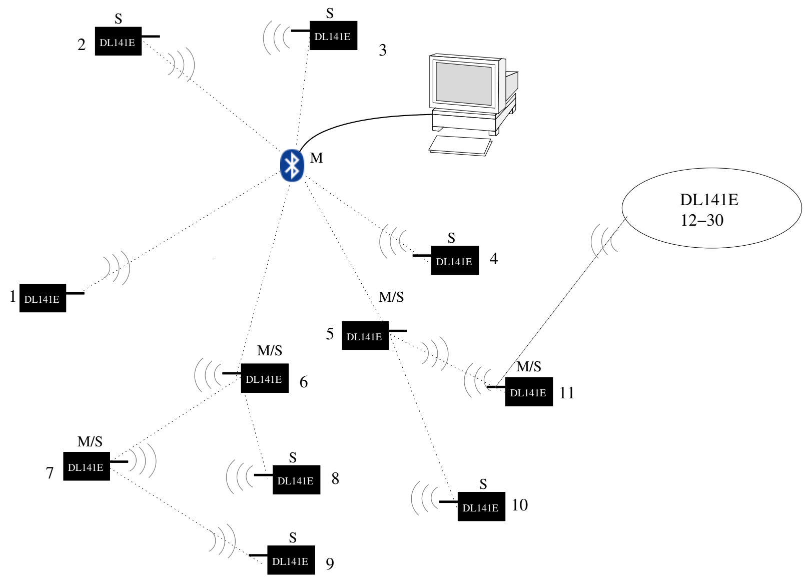


Figure 13: The scatternet approach

algorithms is a classic problem. Which algorithm to use is **not** specified by the Bluetooth specification [1]. The properties of the scheduler will determine the system's performance, especially in a highly loaded piconet with different traffic demands. Thus there has been a lot of research in this area. The main effect upon the transfer rate is **when** the master polls slaves that have **no** data to send. Most of the issues concerning scheduling algorithms are out of the scope for this thesis, however Björn Jonsson in his master's thesis [22] investigates both classical Round Robin algorithms like (PRR, ERR, LRR and WRR, see table 4) and more recently developed algorithms.

The investigations in [22] resulted in a new algorithm which provides:

- Fairness among the nodes.
- QoS (Quality of Service) reservations.
- Utilize the effective bandwidth efficiently.
- Low complexity.

These algorithms were tested and compared with other scheduling algorithms by simulations, using the network simulator NS2 (see section 4.1).

In this area, some of the most frequently cited research papers are:

1. [23] which investigates the classical Round Robin algorithms and proposes a new algorithm: the Fair Exhaustive Polling algorithm (FEP).
2. [24] which investigates PRR, ERR, and LPM (see table 4) and proposes a new Limited and Weighted Round Robin algorithm (LWRR).
3. [25] proposes a Predictive Fair Poller algorithm (PFP) and compares it to the PRR (see table 4) and the FEP [23].

After [22], others have continued to examine scheduling policies, see the articles listed below:

| | |
|---|---|
| PRR (Pure Round Robin) | Polls one slave one after another in a cycle. All slaves get equal chance to transfer. |
| ERR (Exhaustive Round Robin) | Round Robin where each slave gets to send its entire buffer before next slave gets to send. This could lead to starvation, i.e. the master gets stuck on one slave (but only with unbounded buffers). |
| LRR (Limited Round Robin) | Round Robin where each slave gets to send a limited number of its packets. |
| WRR (Weighted Round Robin) | Round Robin where each slave is assigned different weights determine how many packets each slave may transfer at each poll. |

Table 4: Classical scheduling algorithms

1. [26] is one of the most recently published articles. Here a number of polling schemes are evaluated and compared.
2. [27] models and compares the two scheduling algorithms ERR and LRR (see table 4)
3. [28] proposes a scheduling algorithm Bluetooth Interference Aware Scheduling(BIAS) guarantees QoS while reducing the impact of interference.
4. [29] proposes a mathematical treatment for performance evaluation of scheduling algorithms with a focus on PRR, ERR, and LRR (see table 4).
5. [30] proposes a Adaptive Share Polling (ASP) algorithm that is constructed for networks that consist of sources sending short data packets at a constant rate.

The advantage of the third approach for constructing larger networks (the parked slave approach) is that the computer only needs one Bluetooth radio. This would be useful if one would like a flexible system where the devices that collect the measurement values are a laptop or PDA. This approach places higher demands on the scheduler, which now needs to handle parking and unparking devices. In [31] and [32] the data throughput in a parked slave piconet is investigated. These investigations show that using several radios instead of one performs better than the parked slaves approach. The reason is that **until** they start to interfere with each other they are parallel channels and hence the aggregate throughput is higher.

The fourth and last approach, the scattenet approach is the most complex. The Bluetooth standard describes scatternets, but says nothing about how a scatternet should be build. In particular the Bluetooth specification does not specify any routing abilities of the master/slave nodes in the scatternet. Another example of difficulties with scatternets occurs if a logger is disconnected, i.e. fails or runs out of battery power (see figure 14). How will the nodes previous connected via this node reconnect to the rest of the net? Thus forming a scatternet demands an advanced scheduler and routing. This has lead to a large research area: scatternet-scheduling. However this is very much out of the scope for this thesis. Interested readers are referred to a recent article, which describes a scatternet scheduling algorithm called BTSpin [33]. In this article the author divides the scatternet schedulers into the groups: centralized, distributed, single-hop, multi-hop, static and dynamic. The article also contains a table with references to other articles divided into the above mentioned groups.

## 2.4 BlueCenter/IUC's DL141E

The experiments in this thesis focus mainly on the DL141E logger. Figure 15 shows this logger connected to a PDA. The logger is a Bluetooth industrial wireless logger and sensor converter. The logical structure of the DL141E is shown in figure 16. The logger card was developed by BlueCenter IUC. The microprocessor

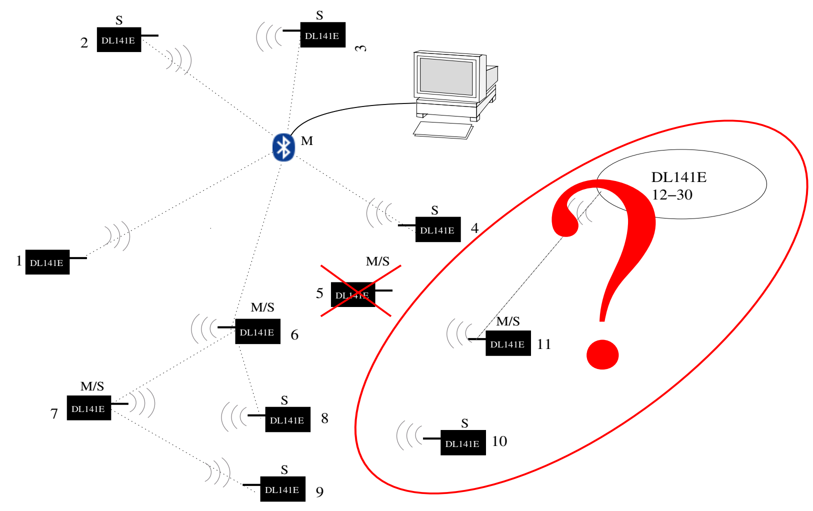


Figure 14: Disconnected scatternet master/slave

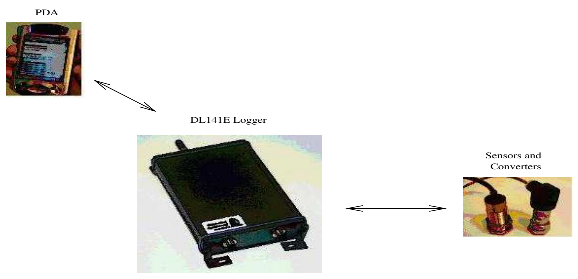


Figure 15: DL141E with PDA

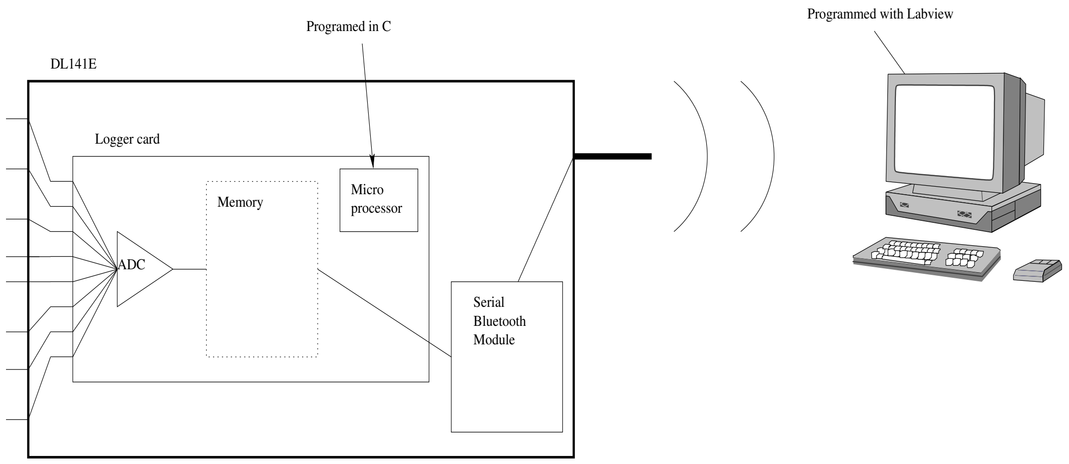


Figure 16: The DL141E structure

controls the ADC and the memory. The Bluetooth module was developed by connectBlue [34] and is connected with the logger card via an RS-232 serial interface. The micro processor is programmed in the programming language C. The device receiving data from the DL141E logger does need not be a laptop or fixed computer, but could also be a PDA. The usual interface software utilizes Labview which makes it easy to arrange the data in the way the user desires, e.g. in a Microsoft Excel spreadsheet or as a plot. However, this thesis has used its own program for communicating with the logger. This program is written in Java and has been developed during this thesis. This program allows one to program the logger, receive values, and calculate bit rate and samples per second. Because of the benefits of Java vs. Labview as the basis of this program, this approach could be used as a complement or if further tests turns out well, it might be a replacement for the Labview based program.

The DL141E has the following technical specifications:

- **Inputs :** 8 single or 4 differentials. Additional options are available, e.g. 4 single and 2 differentials, etc.
- **Sample rate :** variable up to 30 kHz spread among the active channels.
- **Resolution :** 14 bits.
- **Memory :** 32 000 samples/channel, up to 256 000 total samples.
- **Data collection :** manual (by programming or external trigger) or automatic.
- **Bluetooth Packet type :** the DL141E uses the DM5 packet type to transmit all of its data. Thus the upper bound to data throughput is 477.9 kbit/s for the Bluetooth link as per(see table 3).
- **Connections:** a serial RS232 interface connects the logger card and the Bluetooth module. The current default baud rate is 115200 bps, but this rate will probably be increased later (when the logger has been more thoroughly tested).

The logger supports point-to-multipoint connections, but this has not yet been fully implemented and tested.Since one of the important issues concerning point-to-multipoint is the choice of scheduler, the developers at BlueCenter IUC would like to investigate which scheduling algorithm to use (see section 7.2) The memory is single ported, i.e. if one is collecting data samples from a sensor it is not possible to send data via the Bluetooth link at the same time and vice versa. Today only prototypes of the DL141E exist, but there are a number of these prototypes being tested at different companies.

The logger supports two modes for transmitting data. It is possible to send the data as text separated with a space. This text could be read by a computer application written in Labview or simply captured, e.g. by Microsoft's HyperTerminal. The space between each measured value and representing the values as text instead of binary means that a lot more data must be sent across the Bluetooth link. Since the size of each value represented as text could be as large as 56 bits, this mode consumes a lot of bandwidth. However, the advantage of this mode is that the developers can see the data values using existing terminal programs and do not need to use any special software. There is also a *raw data* mode were each measured value is send as a 16 bit value. Thus only two bits are wasted for each sample. Even this inefficiency is unnecessary, but was probably convenient during the initial programming of the logger. In later versions of the logger software each sample will probably use only 14 bits per value. However, the *raw data mode* mode is not supported by the Labview interface. Tests has been made in raw data mode with the Java program developed for this thesis. If the Java program for reading the logger is developed further, it will certainly use only raw data mode as the better link efficiency supports higher sampling rates. More information about text mode, raw data mode, and the negative effects on baud rate can be found in section 6

| Command | Action |
|---|---|
| [a,$f$ | Sets the sampling frequency to $f$ Hz. |
| [b,$n$ | Sets the number of samples that should be collected during each reading interval to $n$. |
| [h,$t$ | Sets the waiting time between sampling periods to $t$ s. |
| [d | Sample and send data. |
| [g | returns the logger's stat us. |

Table 5: DL141E commands

| | |
|---|---|
| Baud rate | 115200 |
| Data bits | 8 |
| Parity | None |
| Stop bits | 1 |
| Flow Control | RTS/CTS |

Table 6: DL141E RS232 interface Settings

The DL141E can be programmed using AT commands. The most important commands are shown in table 5. The default settings for the DL141E RS232 interface are shown in table 6

BlueCenter/IUC also has a Bluetooth based industrial wireless sensor or I/O converter called TR112E. The TR112E transceiver has one input sensor or I/O signal that could be transmitted to a PC or PDA via a Bluetooth radio, a PLC[10] or other control system with Bluetooth radio, or a TR112E receiver with an output channel. The TR112E is not as advanced as the DL141E, the TR112E only works as a cable replacement.

[10]Programmable Logic Controller

# 3 Experiment

This section describes the equipments and settings involved in the experiments. The experiments in this master's thesis where performed at IUC in Olofström. The results from the experiments were used to construct the model described in 4.

## 3.1 BlueCenter/IUC's Measurement Equipment

The main apparatus used for performing accurate measurements is a Faraday cage. The Faraday cage is a RF screened off measurement cage located at the lab in IUC Olofström. The cage is top modern and was built by personal at IUC and prevents RF signals from the outside getting in and reduces the reflections of signals inside. It is 20dB better than other Faraday cages on the market. Here one can perform measurements without interference from unknown sources. The advantage of using a Faraday cage is that one can control the measurement, e.g. if one wishes to add a specific noise source to a Bluetooth system one can be sure this is the only noise that effects the system.

Two useful devices during the experiments are CATC's (Computer Acces Technology) Merlin$^{T}M$ which is a Bluetooth network sniffer and Rohde&Schwarz's CMU200 which is multiprotocol tester for mobile radio networks, including Bluetooth. These devices are described below in sections 3.1.1 and 3.1.2.

### 3.1.1 Merlin

CATC's Merlin is a non-intrusive Bluetooth traffic sniffer (see figure 17). It records all the traffic and enables the user to analyze all of the traffic later [34]. Thus one can analyze BASEBAND, LMP, L2CAP,

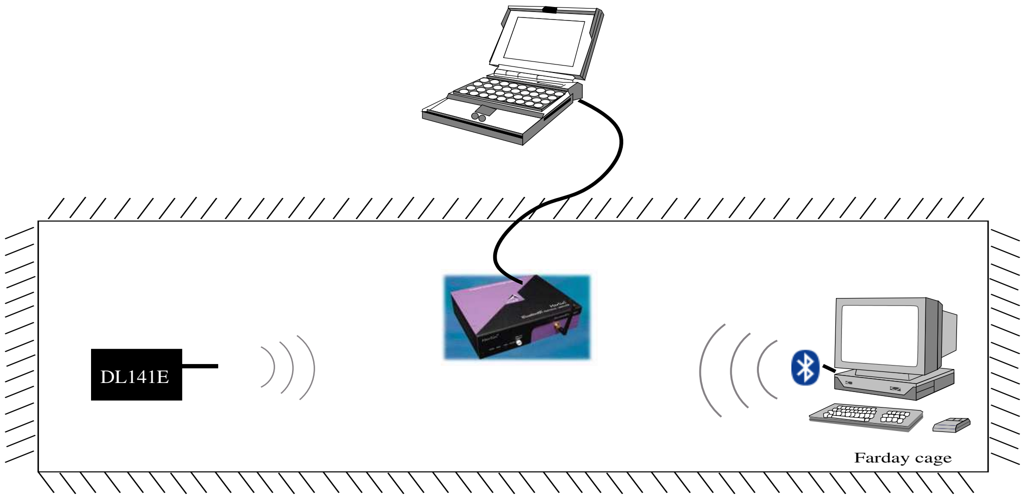


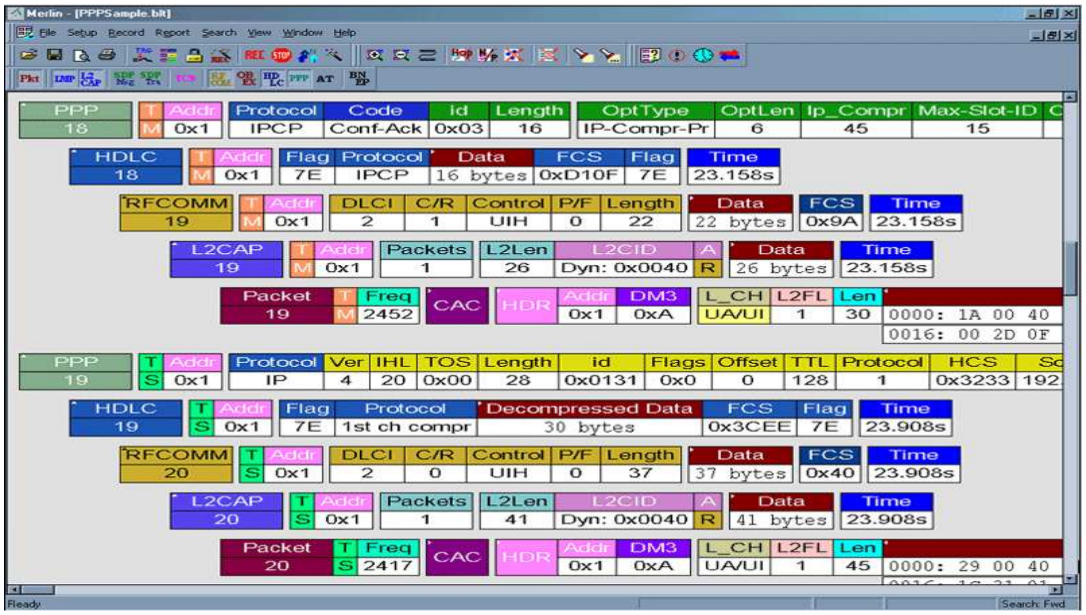
Figure 17: Merlin sniffer example

SDP, RFCOMM, TCS, HDLC, PPP, OBEX, BNEP, and AT levels of the Bluetooth 1.1 protocol stack [1]. It supports both point-to-point or point-to-multipoint piconets. It does not participate in the Bluetooth network, as it is a purely passive listener. The recording memory is 128MB which, depending on the test conditions is enough memory to record about 25 minutes of traffic. To preserve recording memory a hardware filter enables the device to focus on a interesting (i.e. relevant) events.

The analysis of the recordings are performed on a portable host or a desktop PC connected to Merlin via its USB port. The captured traffic is displayed in colour graphics using the vendor's software which run under Microsoft's Windows 98/ME/NT/2000/XP. Figure 18 shows an example of captured Bluetooth traffic recording.

Merlin is easy to use. For example when recording there is a wizard that helps the user to configure Merlin to detect nearby Bluetooth devices and then record traffic in this piconet. The software is free and could be used with or without the Bluetooth radio and traffic capture box. This enables recording in the lab and analysis at the office or home. It is also possible to analyze a previous experiment while another experiment is being performed.

Merlin provides a table for each of its captures a *traffic summary* in HTML format. This includes among other things the number of good packets send via each Bluetooth channel. This table has been very useful during the experiments. An example of a *Merlin traffic summary* can be seen in appendix B.


Merlin - [PPPSample.blt]
File Setup Record Report Search View Window Help
Pkt LMP L2CAP SDP Msg SDP Trs RF COMM OBEX HDLC PPP AT BNEP
Ready
Search: Fwd

| PPP | T | Addr | Protocol | Code | id | Length | OptType | OptLen | Ip_Compr | Max-Slot-ID | C |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 18 | M | 0x1 | IPCP | Conf-Ack | 0x03 | 16 | IP-Compr-Pr | 6 | 45 | 15 | |

| HDLC | T | Addr | Flag | Protocol | Data | FCS | Flag | Time |
|---|---|---|---|---|---|---|---|---|
| 18 | M | 0x1 | 7E | IPCP | 16 bytes | 0xD10F | 7E | 23.158s |

| RFCOMM | T | Addr | DLCI | C/R | Control | P/F | Length | Data | FCS | Time |
|---|---|---|---|---|---|---|---|---|---|---|
| 19 | M | 0x1 | 2 | 1 | UIH | 0 | 22 | 22 bytes | 0x9A | 23.158s |

| L2CAP | T | Addr | Packets | L2Len | L2CID | A | Data | Time |
|---|---|---|---|---|---|---|---|---|
| 19 | M | 0x1 | 1 | 26 | Dyn: 0x0040 | R | 26 bytes | 23.158s |

| Packet | T | Freq | CAC | HDR | Addr | DM3 | L_CH | L2FL | Len | |
|---|---|---|---|---|---|---|---|---|---|---|
| 19 | M | 2452 | | | 0x1 | 0xA | UA/UI | 1 | 30 | 0000: 1A 00 40 |
| | | | | | | | | | | 0016: 00 2D 0F |

| PPP | T | Addr | Protocol | Ver | IHL | TOS | Length | id | Flags | Offset | TTL | Protocol | HCS | Sc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | S | 0x1 | IP | 4 | 20 | 0x00 | 28 | 0x0131 | 0x0 | 0 | 128 | 1 | 0x3233 | 192 |

| HDLC | T | Addr | Flag | Protocol | Decompressed Data | FCS | Flag | Time |
|---|---|---|---|---|---|---|---|---|
| 19 | S | 0x1 | 7E | 1st ch compr | 30 bytes | 0x3CEE | 7E | 23.908s |

| RFCOMM | T | Addr | DLCI | C/R | Control | P/F | Length | Data | FCS | Time |
|---|---|---|---|---|---|---|---|---|---|---|
| 20 | S | 0x1 | 2 | 0 | UIH | 0 | 37 | 37 bytes | 0x40 | 23.908s |

| L2CAP | T | Addr | Packets | L2Len | L2CID | A | Data | Time |
|---|---|---|---|---|---|---|---|---|
| 20 | S | 0x1 | 1 | 41 | Dyn: 0x0040 | R | 41 bytes | 23.908s |

| Packet | T | Freq | CAC | HDR | Addr | DM3 | L_CH | L2FL | Len | |
|---|---|---|---|---|---|---|---|---|---|---|
| 20 | S | 2417 | | | 0x1 | 0xA | UA/UI | 1 | 45 | 0000: 29 00 40 |

Figure 18: Merlin recording example

### 3.1.2 CMU200

Rohde&Schwarz's CMU200 is a multiprotocol universal radio communication tester designed for current and future mobile radio networks. The front panel of the CMU200 is shown in figure 19. According to [35] the CMU200 has high testing speed, makes highly accurate measurements, is modular in design, while providing comprehensive spectrum analysis and fast switching between networks. With CMU200 one can test Bluetooth (version 1.0B and 1.1 [1]) as well as all major mobile radio standards. The CMU200 acts as a slave in a Bluetooth network and is able to query all Bluetooth devices in the network. It can perform transmit and receiver measurements simultaneously. The CMU200 also offer a number of statistical monitoring and measurement functions, an example is the possibility to define individual tolerances for each measured value and to stop a measurement sequence after a certain number of measurements or when a tolerance has been exceeded.

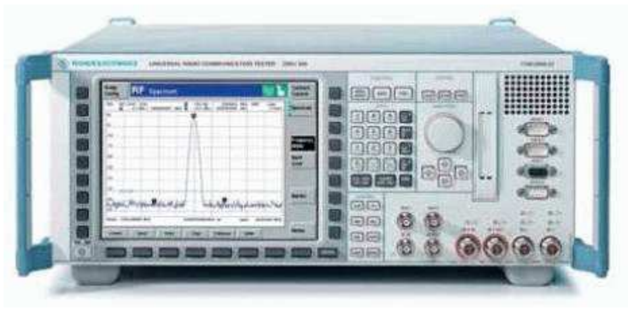


Figure 19: The CMU200 front panel

Three useful measurement parameters are:

- **BER** (Bit Error Rate) Determine the number of bit errors (in percent) that have occurred within the current statistical cycle.
- **BER search function** Find a sensitivity level for a predefine BER level.
- **PER** (Packet Error Rate) Determine the number of packet errors (in percent) that have occurred within the current statistical cycle. (A packet error is a packet with a header which cannot be corrected).

However in the experiments performed in this thesis the CMU200 has been used simply as a signal generator to generate interference on the Bluetooth point-to-point connection.

## 3.2 Experiment settings

The placement of devices in the Faraday's cage shown in figure 20 have been used during all experiments. Different Bluetooth devices and interference sources have been used in different experiments (see section 4). All measurements in figure 20 are from the device's antennas. In most experiments Merlin has been used to sniff the traffic. More information about Merlin can be found in section 3.1.1. The experiments have been monitored outside the cage in a small room with computers, signals generators, etc. In those cases cables were from the outside to the inside of the cage these cables were squeezed between the wall and the shut door. This was considered enough for these experiments.

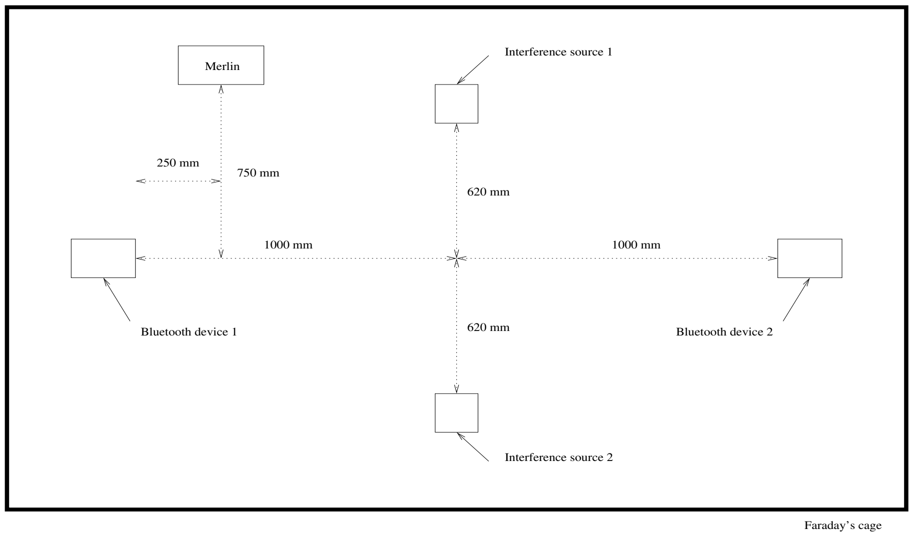


Figure 20: Experiment settings

# 4 The Model

A crucial part for this thesis is the model from whom logger simulations are beeing performed. An overview of the model can be found in figure 21. The Network Simulator 2 (NS2) [2] together with the Bluetooth

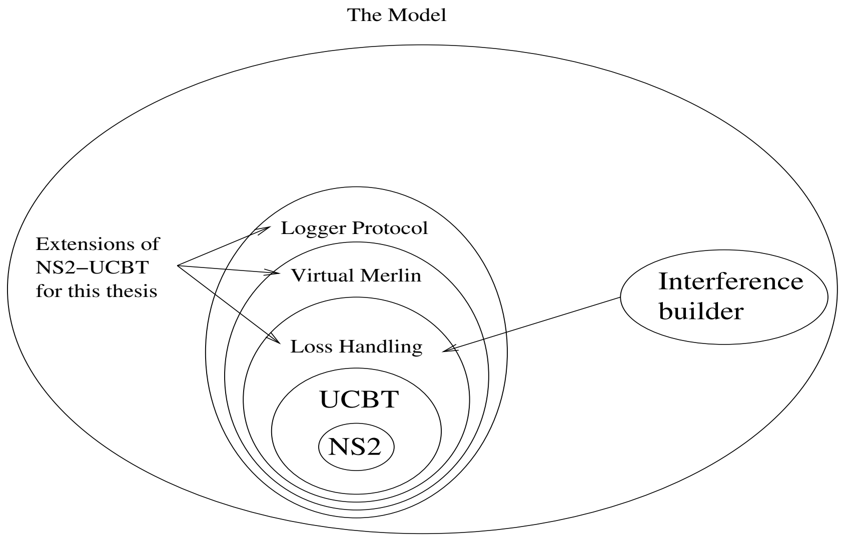


Figure 21: The Model

extension [3] forms the base for the model. NS2-UCBT was extended with three parts that were necessary for this thesis. The three parts are **Loss Handling** (described in section 4.2.1), **Virtual Merlin** (described in section 4.2.2), and the **Logger Protocol** (described in section 4.2.3). The loss handler has an extension itself, the **interference builder** that is a part of the simulation program (see section 5). The interference builder adjusts the packet loss probability in the loss Handler based on experiments. The interference builder are described more thorough in section 5.2.

## 4.1 NS2-UCBT Simulation System

The Bluetooth protocol simulations in the model are performed using Network Simulator 2 (NS2) with a Bluetooth extension from the University at Cincinnati (UCBT). Information about NS2, downloads, installations, manuals, etc. can be found at [2]. Information and downloads of the UCBT extension etc. can be found at [3]. NS2 and the UCBT extension can sometimes be very difficult to work with. The documentation is vague and sometimes the only way of figuring out how things work is to read the source code. However some advices can be found in [36] and [37].

NS2 is a discrete event driven simulator that allows simulations of TCP, routing, and multicast protocols over wired and wireless (local area and satellite) networks. NS2 is written in C++ and OTcl [38] and the simulation specification for a run for NS2 is written by the user in OTcl script. The OTcl script is however handled by the simulation program that has been developed during this thesis, i.e. the user does not need to know anything about NS2. The reason for a more user friendly program is that during this thesis it was shown that NS2-UCBT is very complex to use ([39] has drawn similar conclusions) and getting NS2 to work as wanted can sometimes be difficult. Section 5 explain the simulation program and what can be done with it.

A small NS2-UCBT guide for readers not familiar with NS2-UCBT and to show how NS2-UCBT is used for readers that are familiar with NS2-UCBT an example simulation is included here. The simulation OTcl script is a script generated by the simulation program. Every operation in the script is followed step by step:

First a new simulation object is created:

```
set ns_ [new Simulator]
```

Then the nodes in the simulation are set to be Bluetooth, i.e. the UCBT extension is used.

```
$ns_ node-config -macType Mac/Bluetooth
```

The two specific nodes need to be set up. One DL141E and one computer. The zero in the DL141E node means it is a slave while number one in the Computer node means slave. This is how the setup is normally made when BlueCenter/IUC build their point-to-point connections. It is also the setting that has been used during experiments in this thesis.

```
set DL141E   [$ns_ node 0]
set Computer [$ns_ node 1]
```

The nodes are set to use the Ad hoc On Demand Distance Vector (AODV) protocol which is a routing protocol designed for mobile ad hoc networks. This protocol is never used by the simulation, but needs to be defined in order to get the simulation to work properly.

```
$DL141E   rt AODV
$Computer rt AODV
```

Next the *Lossmode* for the simulation is turned on. This is described more thoroughly in section 4.2.1 and 5.2.2.

```
$Computer LossMod LossFile lossfile.txt off
$DL141E   pos 0 0
$Computer pos 0 0
```

The topology of the simulation is now set up. It is now necessary to set up what happened in the wireless network. All data in NS2 are sent between two agents. The following lines create a Logger agent and connects it to the DL141E node:

```
set log [new Agent/Logger]
$ns_ attach-agent $DL141E $log
```

Often NS2 simulations uses traffic generators, however the Logger protocol generates its own traffic and thus does not need any traffic generator. To receive the data Logger agent a logger sink agent has to be connected to the computer node. Both the *Logger*- and the *Sink* agent are developed specially for this thesis. The agents are described more thoroughly in section 4.2.3.

```
set logSink [new Agent/LoggerSink]
$ns_ attach-agent $Computer $logSink
$ns_ connect $log $logSink
```

Now a finish process is defined. The process finalizes the *lossfile* (see section 4.2.1) and makes sure that NS2 is terminated.

```
proc finish {} {
 Computer LossFile finalize
 exit 0
}
```

At time 0.0 the computer and the DL141E are turned on, and the DL141E opens a connection to the computer.

```
$ns_ at 0.0 "$Computer on"
$ns_ at 0.0 "$DL141E   on"
$ns_ at 0.0 "$DL141E   make-bnep-connection $Computer DM5 DM5"
```

After one second the DL141E starts collecting values and sends its buffer. After 60 seconds the simulation is finished.

```
$ns_ at 1.0  "start"
$ns_ at 60.0 "finish"
```

Finally to start the simulation:

```
$ns_ run
```

This is a short example of how NS2-UCBT is used to simulate the logger. Logger simulations with NS2-UCBT are explained more thoroughly in section 4.2.3.

## 4.2 Extensions to NS2-UCBT

In this thesis three extensions to NS2-UCBT have been made.

1. **Loss Handling** which offers a more flexible handling with Bluetooth packet losses than the original NS2-UCBT.
2. **Virtual Merlin** provides a Merlin similar output for the simulations
3. **Logger Protocol** which is two new NS2 agents for handling Logger communications.

### 4.2.1 Loss Handling

To simulate loss of Bluetooth packets the UCBT extension uses a table driven approach provided by BlueHoc [40]. The table specifies the probability that a specific Bluetooth packet is lost when the Bluetooth devices are located at a specific distance from each other. The table is located in the file *distfer.h*. This table was replaced with a class that allows the user to change the packet lost probability without recompiling NS2-UCBT. The source code for this extension can be found in the files *ferloss.h* and *ferloss.cc*. The *ferloss* object (for Frame Error Rate loss) reads a text file which entry the *distfer* table. The FER versus Distance table must have an entry for each of the 16 different packet types, but could be made arbitary (but must contain at least one entry) with respect to distance. This file is called the *lossfile* and could for instance be written by hand in a text editor or generatated by Matlab [41]. However, when the simulations are performed by the simulation program it uses the interference builder (see section 5.2) to construct the *lossfile* for the desired experiment.

The *lossfile* also contains one or more 16 rows and 79 columns that allows the user to specify the packet loss probability for a specific packet transmitted at a certain channel at a certain time in the simulation. Figure 22 shows how the *lossfile* is constructed. The second field in figure 22 is the loss table related to

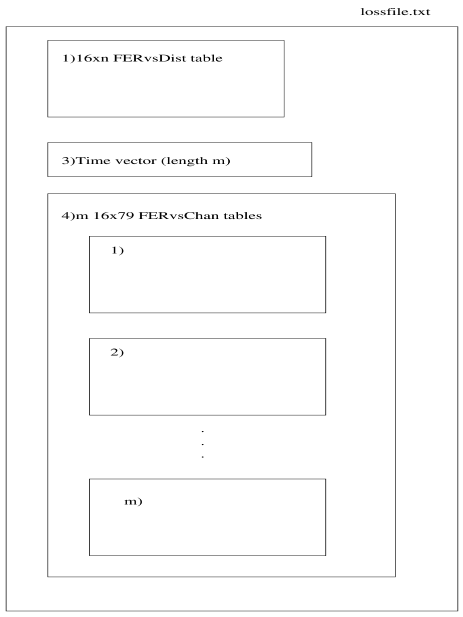


Figure 22: Lossfile construction

the Bluetooth devices distance from each other. The third field is a time vector that tells when to choose next table related to transmition frequency in the fourth field. For example if the time vector has two values 0 and 1, then the FERvsChan filed needs to have two 16x79 FERvsChan tables. The first second of the simulation the first table is used and during the rest of the simulation the second table is used.

To use the lossfile extension in a simulation the following lines needs to be added to the OTcl script.

```
$Computer LossMod LossFile lossfile.txt off
$DL141E   pos 0 0
$Computer pos 0 0
```

The first line, `Computer LossMod` is the same as when using BlueHoc's table drivven approach as lossmode for node `computer`. Then instead of `BlueHoc` (that one should have after `LossMod` if one would like to use BlueHoc's loss mode), the word `LossFile` specifies that the *lossfile* extension should be used. Then `lossfile.txt` specifies the filename of the lossfile and `off` or `on` tells if the *Virtual Merlin* (see section 4.2.2) should be turned on or off. When using the *LossFile* mode it is also necessary to have a finalize command preferably in the finish procedure of the simulation. The finalize procedure are defined as: `Computer LossFile finalize`. This row finalizes the files produced by Virtual Merlin.

### 4.2.2 Virtual Merlin

The purpose with the *Virtual Merlin* is to provide an output similar to the output provided by the sniffer *Merlin*$^{TM}$ described in section 3.1.1. The *Virtual Merlin* is only available together with the *Loss Mode* extension provided in this thesis. Figure 23 shows how Virtual Merlin are connected to the NS2-UCBT simulation system. The Bluetooth nodes are defined by the UCBT extension [3], the losshandler by this

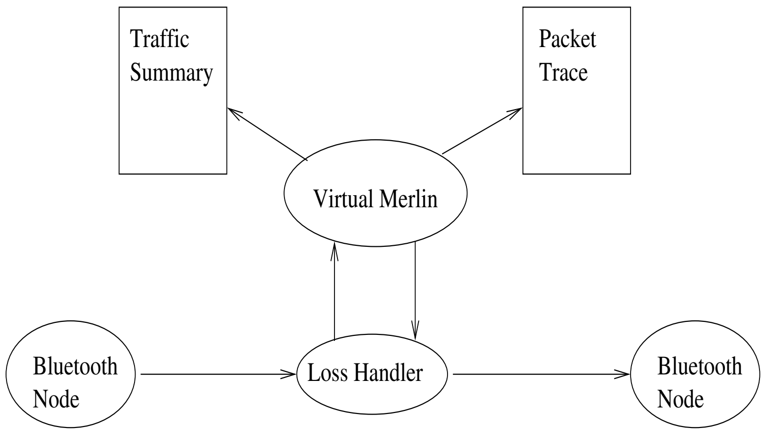


Figure 23: Virtual Merlin's NS2-UCBT connection

thesis as described in section 4.2.1. The Virtual Merlin extension detects if the packet will be lost or delivered and outputs its results into two files:

1. **Traffic Summary:** Is an HTML file similar to the traffic summary provided by Merlin. An example of a *Merlin* traffic summary could be found in appendix B and an example of a *Virtual Merlin* traffic summary could be found in appendix C.1. The Traffic Summaries from *Merlin* and *Virtual Merlin* could be compared visualy or by using the *TrafficSumReader* provided by the *Simulation Program* (see section 5). This feature was used frequently in this thesis when comparing simulations and experiments.

2. **Packet Trace:** Each HTML file displays Bluetooth packets that are being sent formatted in the same way as *Merlin* (but are not as detailed as the actual Merlin output). An example of a Merlin Packet trace is shown in figure 18 and a example of a Virtual Merlin packet trace is in appendix C.2. Each HTML page displays 100 packets. New pages are automatically created as necessary to display all

the packets captured. At the top of each page is a link that points to the previous page and at the bottom of each page is a link to the next page. Note that even a short simulation time will produce a large number of packet trace files.

The *Virtual Merlin* has been a useful tool when the *Interference Builder* (see section 5.2) was constructed. However when running a logger simulation it is better to turn off the *Virtual Merlin* as generating a large number of files slows down the simulation.

### 4.2.3 Logger Protocol

NS-2 allows the user to define their own protocols by extend the code for NS-2 with a new class that describes the *agent* which (in this case) uses the Bluetooth channel to transfer data to the receiver agent, which is called the *sink agent*. For this thesis two new agents has been created.

1. **Logger Agent:** This agent behaves like the DL141E logger. It is also possible to simulate more general loggers, however it might be necessary to change some of the *Logger Agent code*. The code for the Logger Agent is written in C++ and can be found in appendix D.1. The agents are based on the *ping protocol* example in [36]. This example should be studied if one would like to learn how to define a new protocol in NS2. The parameters that can be changed in the *Logger Agent* are:
   - `sampleNumber_` : The number of samples that should be transfered each reading period
   - `sampleSize_` : Size of each sample in bits. Default for the DL141E is $\approx$ 48 bits in *text mode* and 16 in *rawdata mode*.
   - `samplingRate_` : The sampling rate in hertz.
   - `samplingPeriods_` : The periods between sampling in seconds. If this value is set to zero then the *Logger Agent* resumes sampling as soon as the last transmission is finished.
   - `baudRate_` : The baud rate for the serial transmission.
   - `timeout_` : The delay (in seconds) while the Bluetooth Module takes the values received by the serial interface and creates a Bluetooth packet and sends it.
2. **Logger SinkAgent :** This agent behaves like a simple logger receiver program. The *Logger SinkAgent* receives and acknowledges the values send from *Logger Agent*. The code for the *Logger SinkAgent* could be found in appendix D.2.

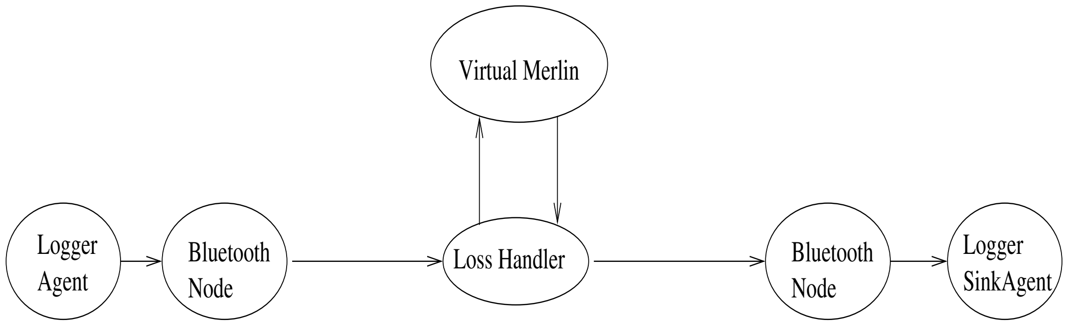


Figure 24: The Logger Protocol's NS2-UCBT connection

Figure 24 shows how the *Logger Agents* are connected to NS2-UCBT with the extensions.

The `sampleNumber_`, `sampleSize_`, `samplingRate_`, `samplingPeriods_` and `baudRate_` are tuneable in the DL141E and also in the *Logger Agent*. While within the DL141E the parameter values are limited (by the specifics of the DL141E - see section 2.4), they could be set to arbitrary values in the simulation Logger Agent. The `timeout_` value could not be set in the DL141E but could be set to any positive value in the *Logger Agent*. This allows the user to simulate properties of other logger that the DL141E. However the important properties of the DL141E that it uses always DM5 packets (even if the data payload is small enough to fit in a smaller packet) and that the Dl141E cannot send and sample at the same time are not changeable in the *Logger Agent*. To support different properties than this the code of the *Logger Agent* will have to be changed.

By studying Merlin traces (see section 3.1.1) the average size of the payload in each DM5 packet sent via the Bluetooth interface from the DL141E was observed to about 86 bytes which indicates a timeout of about 0.006s ($115200/8 \cdot 0.006 \approx 86$) where nine of these bytes are control bytes (e.g. startbit, stopbit, packet size, etc.) added by the serial interface. It was difficult to find out exactly how the packing of data in the Bluetooth module was determined. No one at either ConnectBlue or BlueCenter/IUC could state exactly how this was done. But, as shown later in this section the settings with a timeout on 0.006s which sends packet at constant size after the previous packet has been acknowledged produces the most accurate simulation compared to experiments Merlin traces and throughput measurements.

Table 7 describes the properties of 17 different experiments where the byte rate and sample rate from the DL141E was measured. The DL141E was set to *text mode* which means that each sample had a size of 48 bytes. The purpose with these experiments was to compare them to similar simulations. Figure 25

| Experiment | Sampling rate [in Hz] | Number of Samples |
|---|---|---|
| 1 | 5000 | 50 |
| 2 | 5000 | 100 |
| 3 | 5000 | 250 |
| 4 | 5000 | 500 |
| 5 | 5000 | 1000 |
| 6 | 10000 | 50 |
| 7 | 10000 | 100 |
| 8 | 10000 | 250 |
| 9 | 10000 | 500 |
| 10 | 10000 | 1000 |
| 11 | 25000 | 50 |
| 12 | 25000 | 100 |
| 13 | 25000 | 250 |
| 14 | 25000 | 500 |
| 15 | 25000 | 1000 |
| 16 | 25000 | 2500 |
| 17 | 25000 | 5000 |

Table 7: Logger Simulations and Experiments

shows a comparison between simulations and experiments where the reading interval was set to one second. The x-axis shows the number of the experiment/simulation as stated in table 7. Figure 25 shows both byte rate and sample rate. The reason for showing both is that later in section 6, the DL141E will be examined and optimized based on these simulation results. The focus in this optimization will be on increasing the number of samples per second which can be supported.

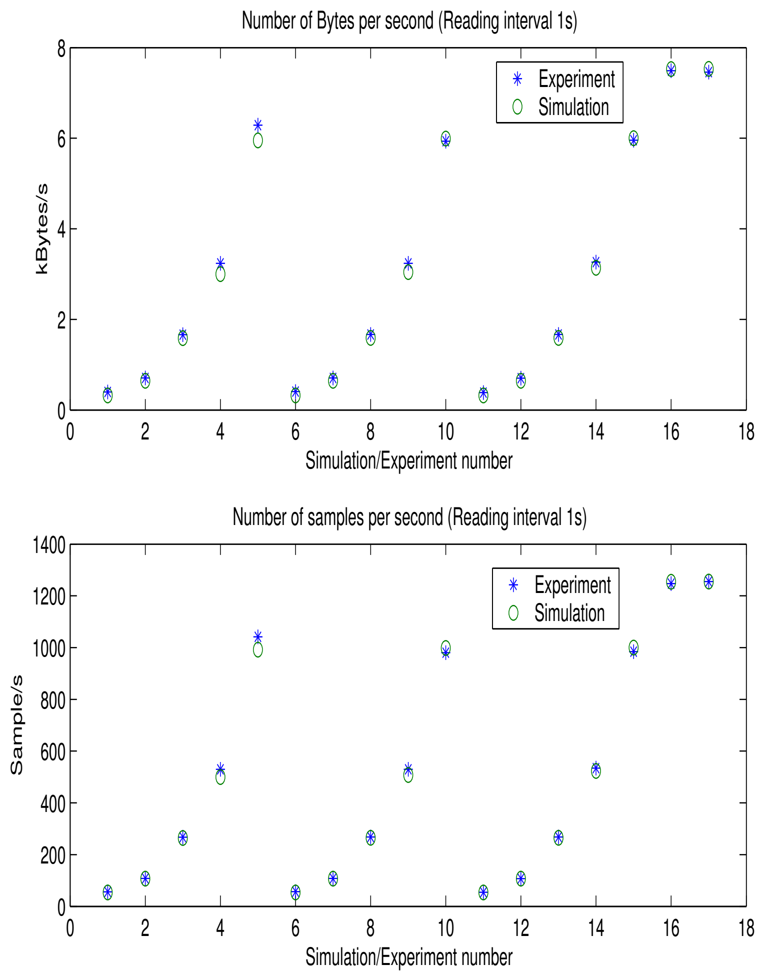


Figure 25: Logger Simulation vs. Experiment with reading interval 1s

Figure 26 shows a comparision between simulations and experiments where the reading interval was set to two seconds. As in figure 25, the x-axis shows the number of the experiment/simulation as stated in table 7.

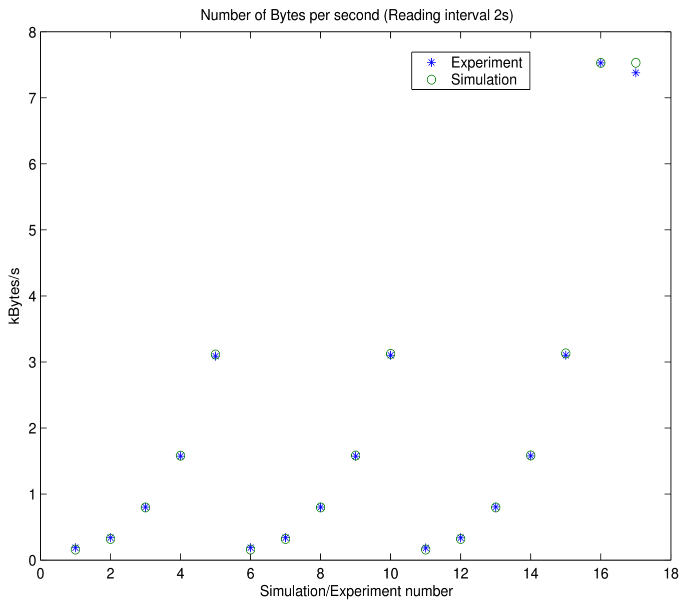


Figure 26: Logger Simulation vs. Experiment with reading interval 2s

Finally in figure 27 a three second interval has been used.

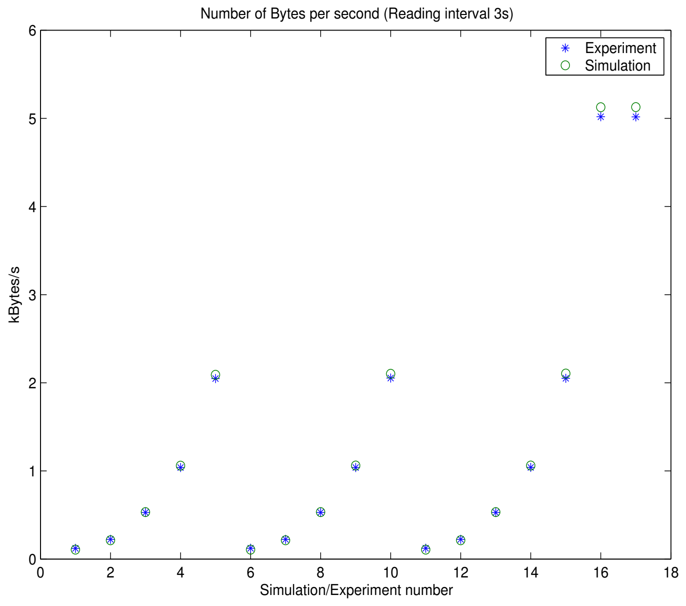


Figure 27: Logger Simulation vs. Experiment with reading interval 3s

The results are overall good with only a few kBytes/s differences between simulation and experiments.

# 5 The Simulation Program

The purpose of the *Simulation Program* is to enable others to easy to use it to evaluate the performance of their DL141E or similar logger in a specific configuration. With this program, the user can:

- Simulate different logger configurations.
- Add interference sources in a flexible way.
- View results from a simulation.
- Program and read a DL141E and plot the received values.
- Save DL141E data captures and simulations.

This section contains an overview of the simulation program and a thorough description of the experiments that its *Interference Builder* is based on. A user manual for the program can be found in appendix E.

## 5.1 Overview

The simulation program is written in Java and has a graphic interface. The documentation for the program could be found in [42]. Figure 28 shows the construction of the program. The main parts of the Simulation

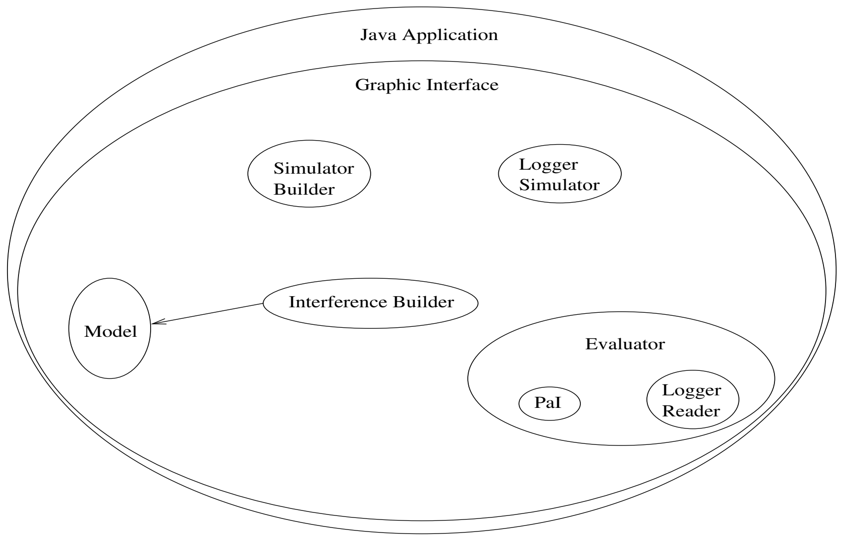


Figure 28: Simulation Program Overview

program are:

- **The Model:** This model described in section 4 is the NS2-UCBT simulation system with extensions. The *Interference Builder* is an flexible way to model loss probability of a specific channel. The *Interference Builder* is described thoroughly in section 5.2.
- **Evaluator:** The evaluator is used to perform simulations and compare them with experiments. The evaluator has two parts; the *PaI Simulator* (Packet at Interval) which sends a Bluetooth packet at a specific interval and the *Logger Reader* which reads the DL141E logger. The *PaI Simulator* is described more thorougly in section 5.2.1 and the Logger Reader in section 5.4.

- **Logger Simulator :** The *Logger Simulator* is described more thoroughly in section 5.3.
- **Simulator Builder :** The simulation program that creates OTcl script files for the NS2 simulation. This is handled automatically by the Simulation Program, thus the user does not need to know anything about NS2.

A graphic interface is wrapped around these features to allow the user to perform many experiments and quickly examine their results. Figure 29 shows an example of a single sin wave of 2Hz connected to a logger which captures 500 samples at 1000Hz each second. The topmost plot in figure 29 shows the sinus signal that one would like to measure and the plot in the bottom shows how well the logger reproduces it. It is possible to look in slow motion at the signal and how the logger reproduce it, it is then possible to change logger configuration to try to find the most appropriate configuration for the current measuring system.Note that the signal is purely fictional. The frequency, amplitude, phase, and offset is tuneable. The purpose of the signal is to give the user of the simulation program the opportunity to see how a signal could be reconstructed by the Bluetooth logger. Off course it is not always the case that a sinus signal should be measured, but tests with the simulation program has shown that a sinus signal makes it easy to overview the performance of a logger configuration. The Simulation program also calculates and displays the throughput and the maximum delay of a measurement value.

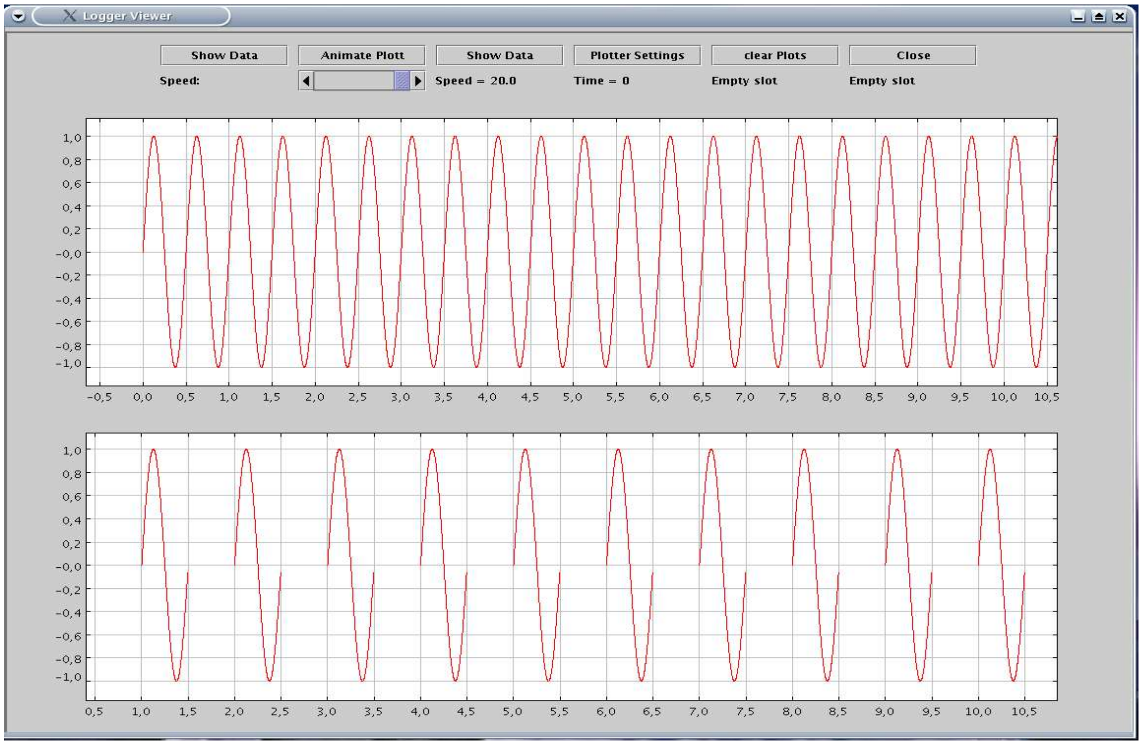


Figure 29: Simulation Program Window Example

## 5.2 Interference Builder

One very important part of the simulation program is the *Interference builder*. The interference builder's task is to construct a correct *lossfile* (described in section 4.2.1). With the Simulation program it is possible to either load the default *FERvsDist* table that is used by BlueHoc [40] or the user can add packet loss probabilities at different distances by clicking on a graph. However, the limitation of this last approach is that all packets will get the same loss treatment which will result in a mean loss probability for all packets. The simulation program also supports building the *FERvsChan* table which can construct in three ways:

1. **Manually :** the user fills in exactly which loss probability should be used for a packet sent in a given frequency channel.

2. **Traffic summary:** Using the *Frequency distribution* table in the *Merlin traffic summary* (see Appendix B) from an experiment the simulation program calculates the *divergence expressed in percentage from the mean value of number of transmitted packets at a given frequency* [11]. These values are then used to calculate the probability of a lost packet at each frequency. The simulation program uses this to construct the *lossfile*. If the experiment is performed correct this will provide the most accurate lossfile.

3. **Spectra:** By adding one or more interference sources at specific frequencies with specific levels or by loading spectral, results from experiments are used to construct the *lossfile*(see section 5.2.2). This is the most important and complex of the three interference builder approaches.

Even though the extension to NS2-UCBT supports different packet lost probabilities for different packets (see section 4.2.1) the simulation program only supports the same packet lost probability for each packet type at a certain frequency. The reason for this is that the interference builder is constructed by using results from the *Frequency distribution* table in the *Merlin traffic summary* (see Appendix B) and these values only give an average value for all types of packets at a given frequency. Figure 30 shows an example of the dialog window that the user of the simulation program uses to add his/her interference sources. In this example the use of a spectra for adding interference could make it possible to define a certain loss probability at a specific channel or distance or use a *Merlin Traffic Summary* to define the loss probability of different Bluetooth channels.

### 5.2.1 PaI Simulator

The *PaI Simulator* (Packet At Interval simulator) is included in the simulation program. It is mostly used to verify results from the experiments with results from the simulations. The *PaI Simulator* sends one DM1,2,5 or one DH1,2,5 Bluetooth packet at a given interval. The *PaI Simulator* makes it easy to iterate through several simulations and hence to simulate a large number of experimental scenarios. The code below shows an example of a DM5 packet being sent every 1/100 second with interference at frequency 2450MHz at a level of -65dBm to -36dBm.

[11]This is an important value and will be refereed to as *Divergence expressed as Percentage At Frequency* (DPAF)

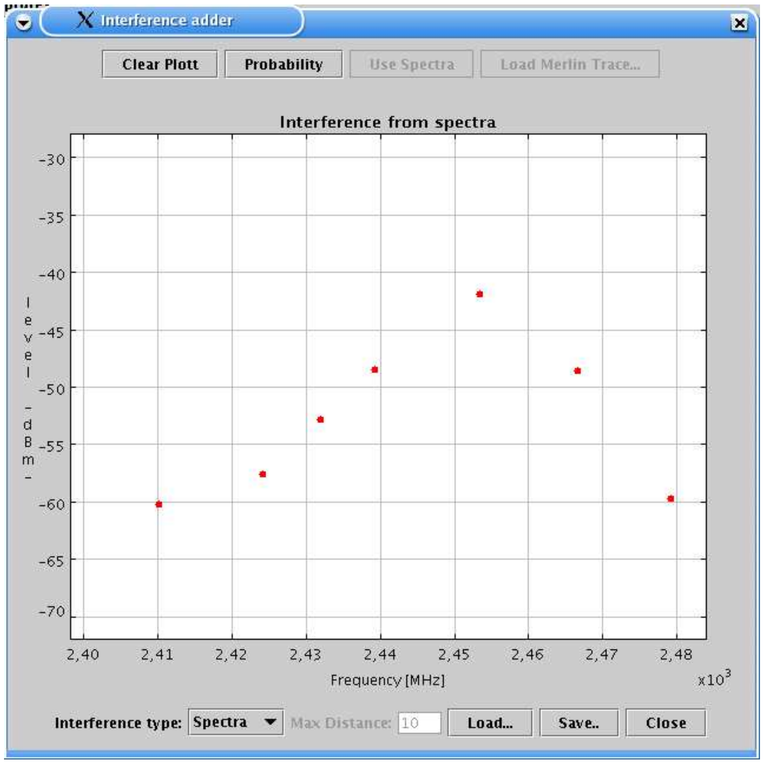


Figure 30: Loss Handler Dialog Example

```
SimParameters sp = new  SimParameters();
sp.sendPacket = "DM5";
sp.recPacket  = "DM5";
sp.trStart    = 0;
sp.simFinish  = 30;
sp.dist       = 0;

for(int level=-65; level<-35; level++) {
  sp.lf          = new LossFile(0,30);
  sp.lf.addInterferenceSource(2450,level);
  PaISimulator sim = new PaISimulator(sp, 0.01);
  sim.simulate();
  evaluateSimulation();
}
```

The simulation (`sim.simulate()`) outputs a traffic summary from which a frequency distribution table can be generated and compared with the frequency distribution table obtained from an experiment.

The Divergence expressed as Percentage At Frequency (DPAF) value is a crucial concept of this section. In the *Frequency distribution* table in the *Merlin traffic summary* (see Appendix B) it is possible to see how many *Total Good packets* have been sent at each frequency. By taking the mean of all *Total Good packets* at all frequencies the divergence in percent at each frequency can be calculated. The *DPAF* from a simulation could then be compared with the *DPAF* from an experiment, we call this the *diff-DPAF*. The *DPAF* is an important measure because in the PaI-simulations a channel is interfered with by an interference source at different levels. The *DPAF* measure tells how much the channel is affected related to other channels that are not exposed to this interference. The *diff-DPAF* will be of importance in section 5.2.2 where experiments and simulations are compared. The example code below shows how the *diff-DPAF* are calculated for two *Traffic summary* files.

```
TrafficSumReader tr1 = new TrafficSumReader(trsum1.html);
TrafficSumReader tr2 = new TrafficSumReader(trsum2.html);
int[] packet1 = tr1.getPacketAtChans();
int[] packet2 = tr2.getPacketAtChans();

double mean1=0;
double mean2=0;
for(int i=0; i<79; i++) {
  mean1+=(double) packet1[i];
  mean2+=(double) packet2[i];
}
mean1/=79; mean2/=79;

for(int i=0; i<79; i++) {
  int res=(int) (10000*Math.abs((1-(double) packet1[i]/mean1)-
     (1-(double) packet2[i]/mean2)));
  System.out.print((double) res/100 + " ");
}
System.out.print("\n");
```

The *TrafficSumReader* class reads the *Traffic summary* file and extracts the information provided from the *Traffic summary*.

### 5.2.2 Interference Experiments

The simulation program offers the user the ability to add interference at one or more frequencies at different levels. Which packet loss probability an interference of this kind produces are derived by the simulation program based on a number of experiments. The *DPAF* and *diff-DPAF* (described in section 5.2.1) from these experiments are used to determine packet loss probability and to compare simulated with actual experimental results.

During the experiments two connectBlue Bluetooth class 3 Serial Port Adapters [43] were used. During these experiments these two units sent a DM5 packet roughly every 0.012s from the slave to the master. A Rhode&Schwarz signal generator was used as an interference generator. Figure 31 shows an example of the interference spectra. As shown in the figure both interference signals the upper, spectra example 1 and the lower, spectra example 2, has a narrow peak but not so narrow that it only has noise in channel 38 to 40. Thus noise in channel 39 will effect adjacent channels. Figure 32 shows an example of *DPAF* for interference at three different levels near Bluetooth channel 39. As one can see in the zoomed plots to the right an interference as high as -38dBm effects the *DPAF* in channels 38,39,40,41, and 42. My simulation program treats the interference at frequency 2441MHz (channel 38) as a single interference, and lets the effects on the channels nearby be a result of this interference. This is **not** an optimum way of handling an interference source in the model, but it is the compromise between:

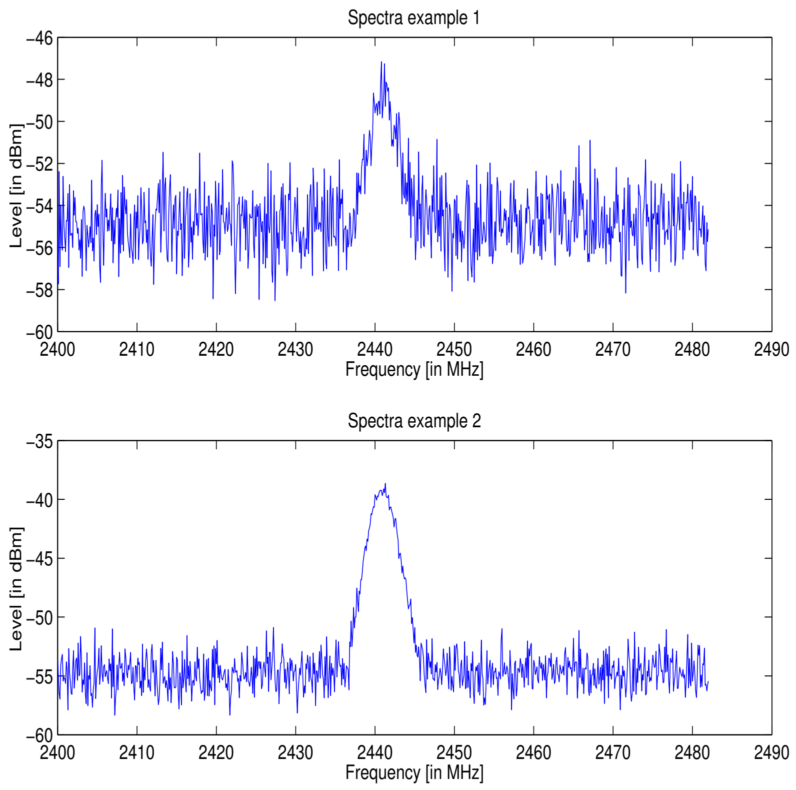


Figure 31: Two Interference Spectra Examples

- **Flexibility:** It is possible to generate an arbitrary spectra to use for the simulation.
- **Programming:** The simulation program uses five tables for looking up packet loss probabilities. This was simple to implement. If the user would like, the tables could easily be replaced with results from another set of experiments.
- **Correctness:** It would be better if one used individual spectra's for each case to determine packetloss probability. For instance loading the entire example spectra 2 shown in figure 31 will produce different results than just a -38dBm source at frequency 2441MHz. The problem when there are several sources of interference (as when a entire spectra is loaded) is explained more thorough below.
- **Experiment equipment :** it is not possible to have all the interference spectra needed. This thesis was limited to the spectra which the two signal generators offered and a more or less random spectrum representing a microwave oven.

The next figure (figure 33) shows the *DPAF* at the five effected channels with interference over the entire Bluetooth band. For example: In the top plot at x-value frequency 2450 MHz an interference source at frequency 2450MHz with level -51dBm has been used resulting in a *DPAF* of about 100% for channel 48 ($ch$ in the plot), 30% for channel 47 ($ch-1$ in the plot) 10% for channel 49 ($ch+1$ in the plot), etc. As figure 33 shows the *DPAF* is about the same through the whole Bluetooth band, or at least it does not indicate any specific peak. This shows that the table for these effected channels could be applied to any channel in the Bluetooth spectra, i.e. it would be enough to look at one channel and the interferences around it, the result could then be applicable at any of the 79 Bluetooth channels. Next a number of experiments

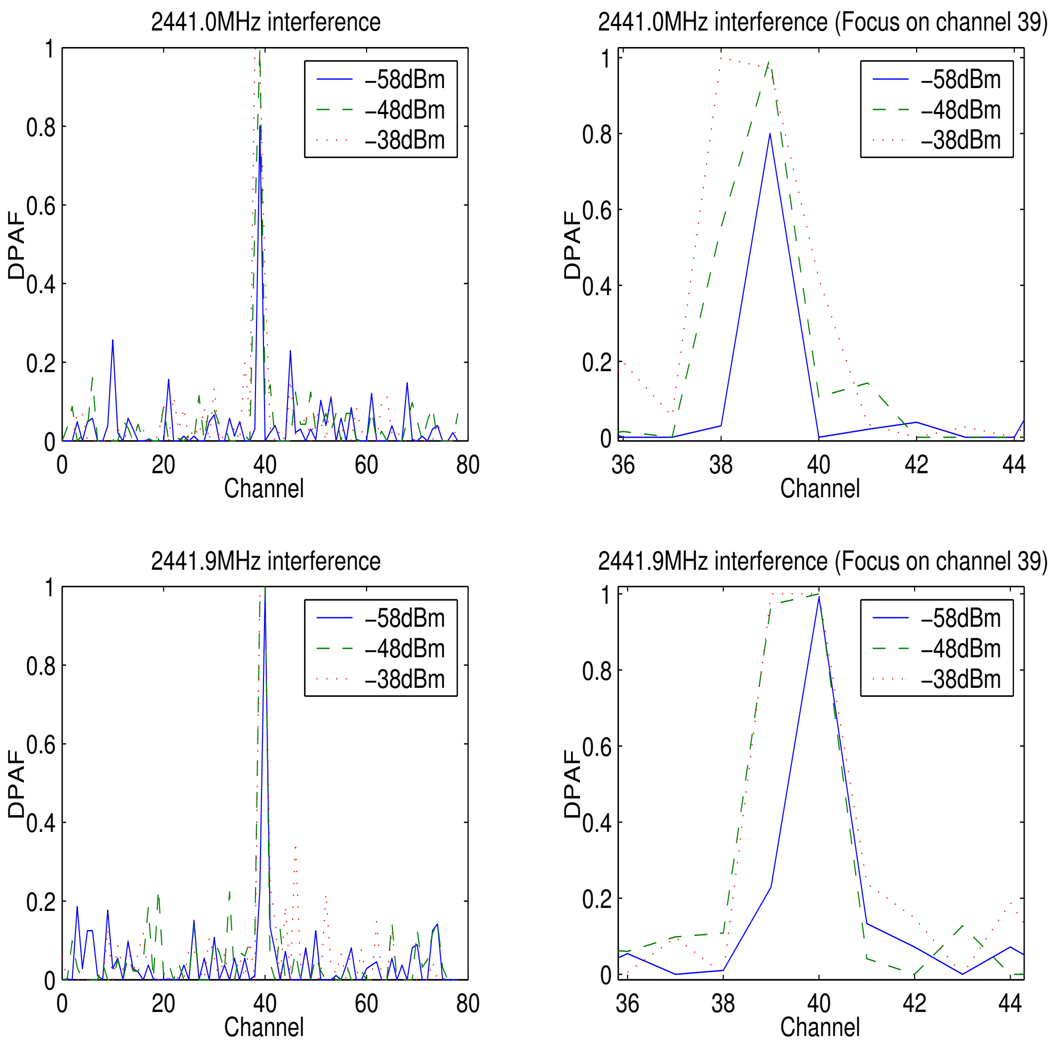


Figure 32: Interference DPAF Example

were performed to obtain the tables for the five effected channels. The experiment was performed with interference frequency $2441.0MHz, 2441.1MHz, 2441.2MHz \ldots 2441.9MHz$. At each interference frequency the level was increased by one step from -65dBm up to -36dBm, i.e. a total number of 360 experiments were performed. The *DPAF* was determined for each experiment and translated into packet loss probability which was stored in five 10x36 tables. The following steps were performed to obtain the packet loss probability from an interference frequency $f$ at a specific level $l$:

- Round $f$ to the nearest tenth of a MHz. $\lfloor f \rfloor - f$ is now a number 0-9 which is the column number that should be read in the tables.
- The row is calculated by taking $l + 65$.
- The channel, $ch$ is calculated by taking $\lfloor f \rfloor - 2402$.
- Finally read the five tables for the five effected channels and update $ch - 1$ to $ch + 3$.

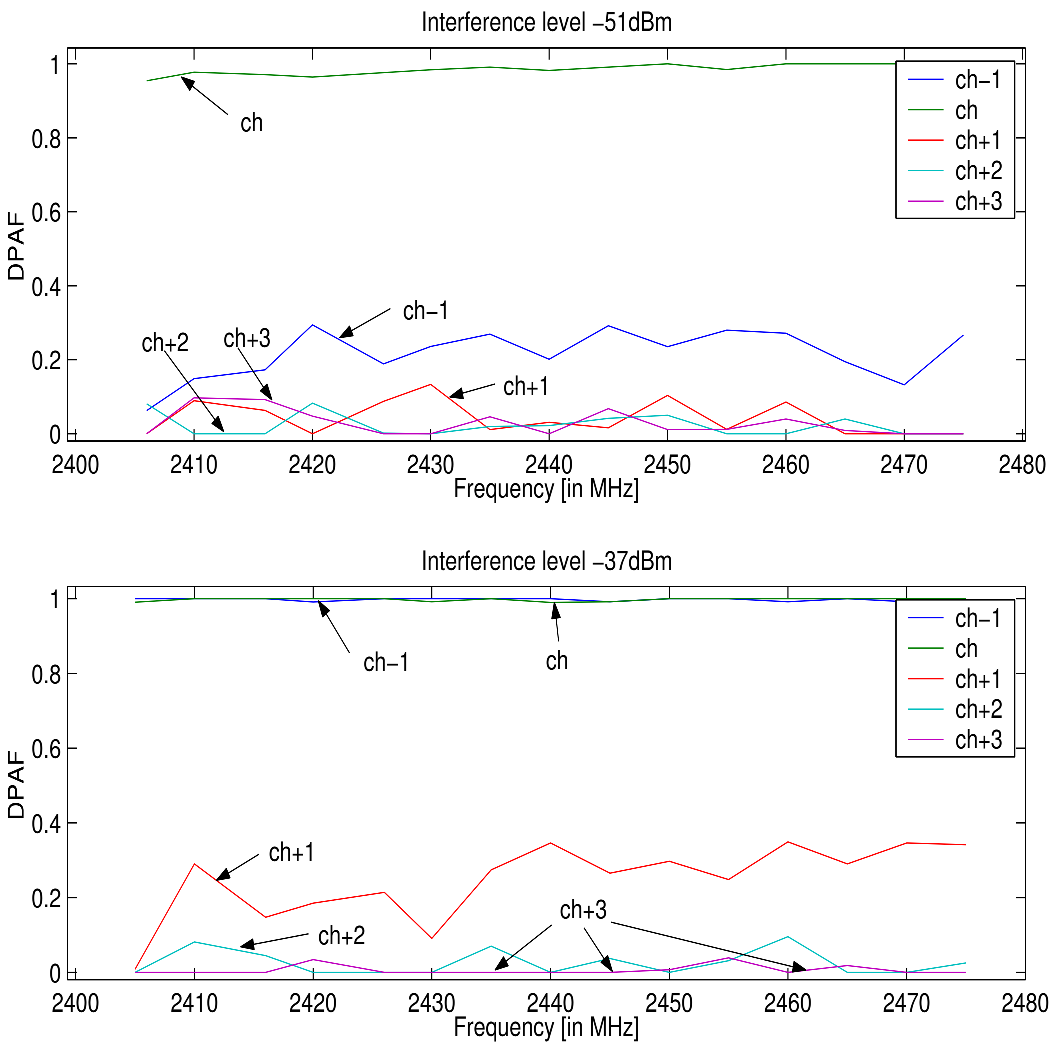


Figure 33: All channels Example

Figure 34 shows a packet loss probability table for the effected channel $ch$ and figure 35 shows the tables for the nearby channels. The x-axis shows the level, the y-axis the interference frequency offset from the affected channel and the z-axis the packet loss probability. These tables are the same for all 79 channels in the Bluetooth band. For example: if an interference source at 2458.64MHz at level -50dBm is used the effected channel ($ch$) will be 56. At the x-value $-50$ and the y-value $0.6$ at the table in figure 34 the packet loss probability for channel 56 could be found in the z-axis ($\approx 30\%$). In a similar way the packet loss probability for channels 55, 57, 58 and 59 could be found in figure 35.

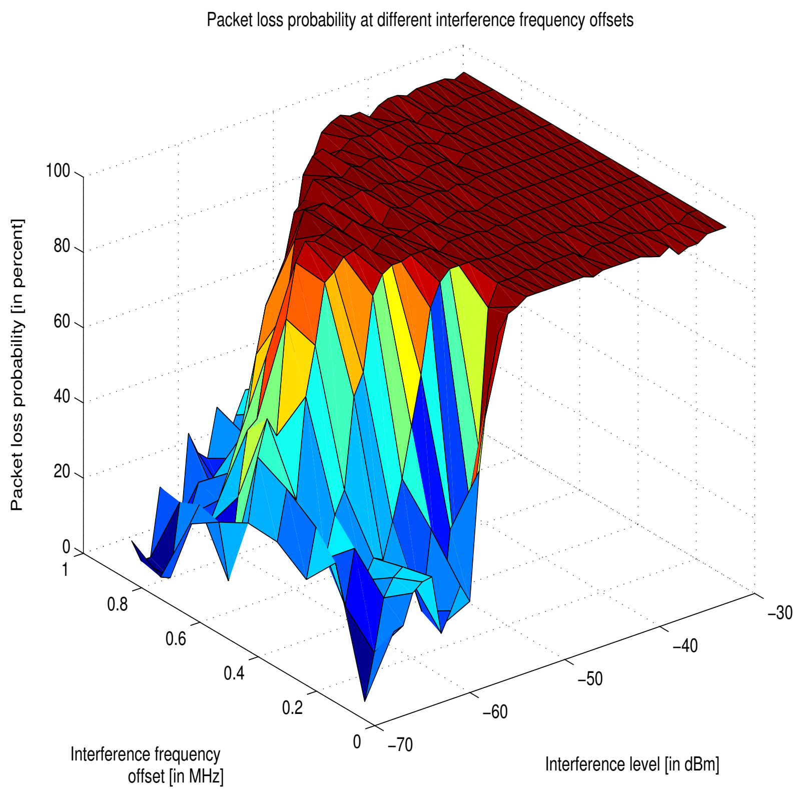


Figure 34: Packet Loss Probability for a channel $ch$ with an interfernce source at a range of frequence offsets (derived from channel 48).

To evaluate the interference model a number of experiments were performed and then compared to a *PAI-simulation* (see section 5.2.1) with the same properties. The difference between the experiments and the simulations are expressed in *diff-DPAF* as described in section 5.2.1. Figure 36 shows the *DPAF* of an experiment, the *DPAF* of the simulation, and the *diff-PDAF* of both. Both experiment and simulation were performed **without** interference, still there is quite a large difference between them. Note that there is also a quite large *DPAF* between the channels in both the experiment and the simulation. This has to do with how the Bluetooth module handles frequency hoping. Which channel to transfer on is decided randomly and there is no guarantee that transmissions are spread equally among the the channels. This explains why the *DPAF* diverges among the channels, but also the relative high *diff-PDAF* between the experiments and the simulation. Comparisons between experiments with two different Bluetooth modules would probably have the same result. In figures 37, 38, and 39 an number of experiment with one interference source at different frequencies over the entire Bluetooth band have been compared to experiments. The *diff-PDAF* of the five most effected channels for each interference frequency are shown in the figures.

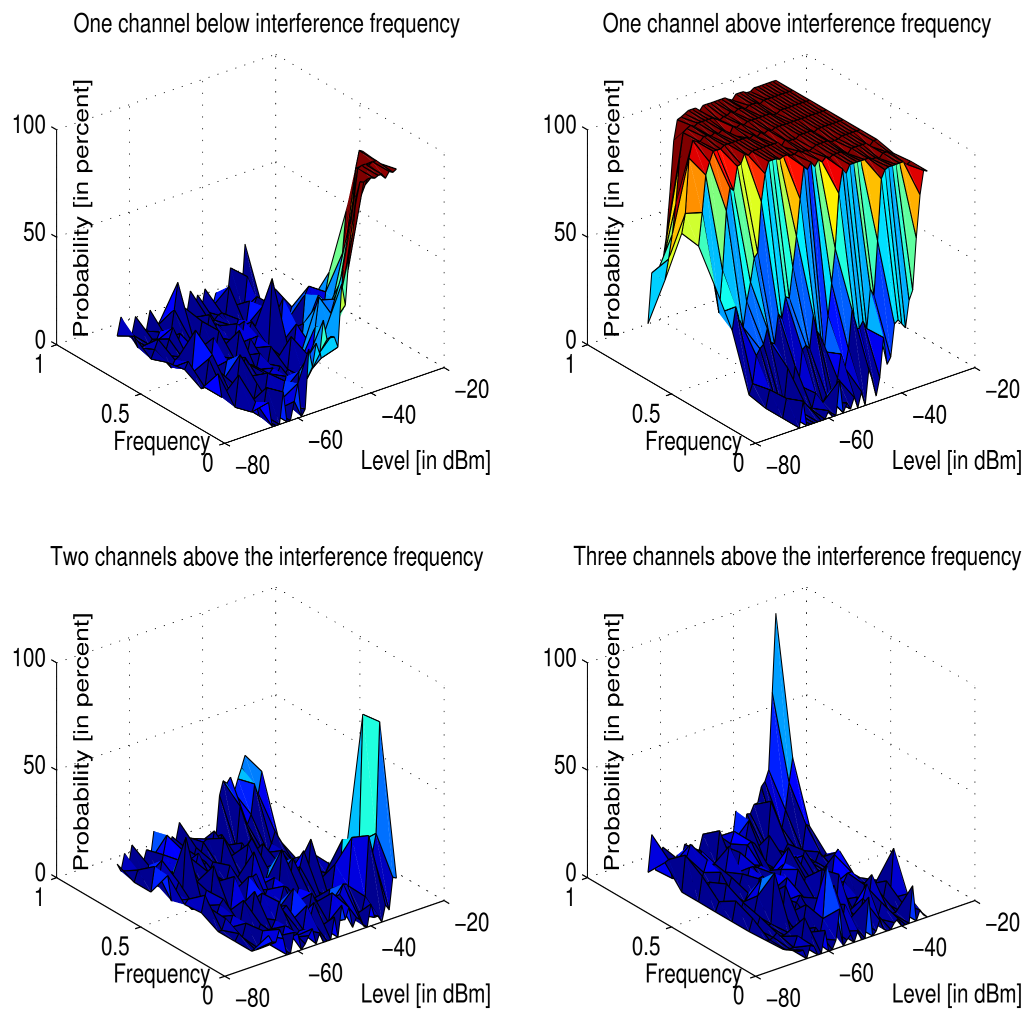


Figure 35: Packet Loss Probability for adjacent channels $ch-1, ch+1, ch+2$ and $ch+3$

The *diff-DPAF* can for some experiments in the figures seem quite large; note that the *DPAF* can differ by over ten percent between the channels in the same experiment/simulation. Also remember the compromises discussed in the beginning of this section. The goal was not to construct a hundred percent correct interference builder, but rather the goal is that using another level or interference frequency than the actual one should not perform better than the actual interference frequency and level. Finally, the most important issues are that a comparison between two logger configurations should not result in a answer that suggest a different setting than would perform better compared to an actual experiment. This is examined in more detailed later in section 5.3.

Figure 40 shows an example of a simulation with one interference source at channel 63 with levels -44dBm, -43dBm ,-42dBm, -41dBm and -40dBm have been compared to an experiment with interference at channel 63 with level -42dBm. Figure 40 shows clearly that simulation with the real interference level performs best.

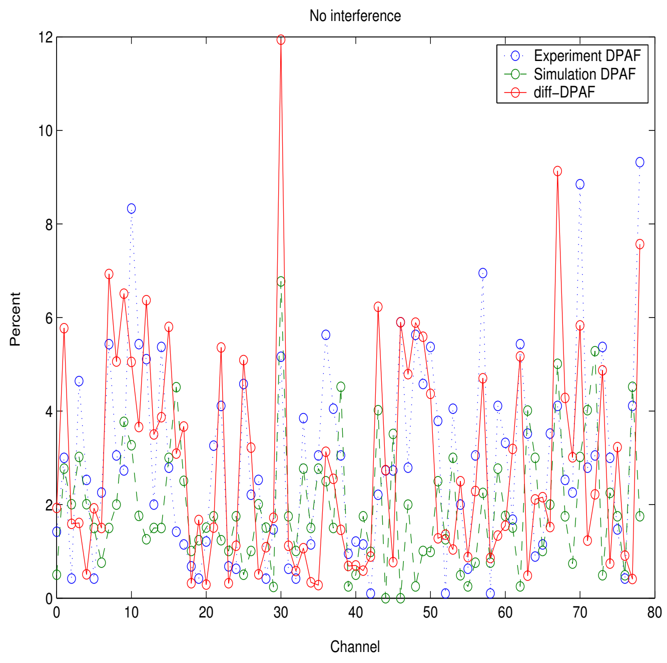


Figure 36: Comparison between simulation and experiment without interference

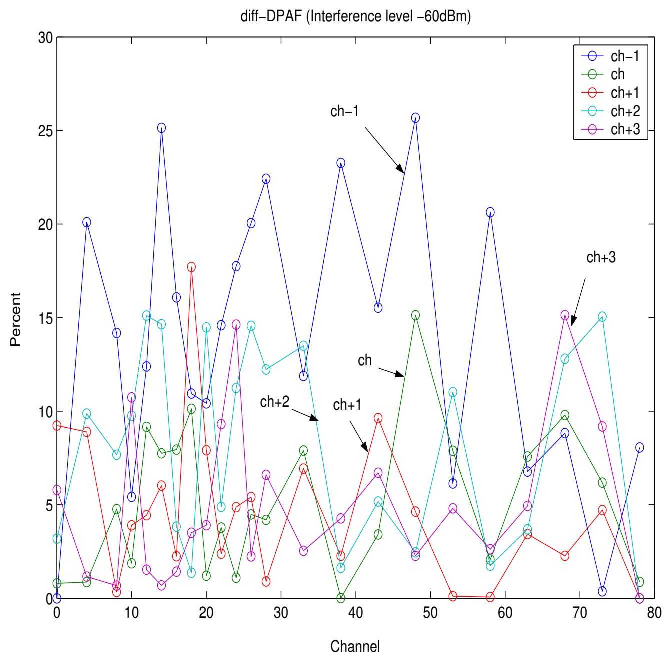


Figure 37: Comparison between simulation and experiment (-60dBm)

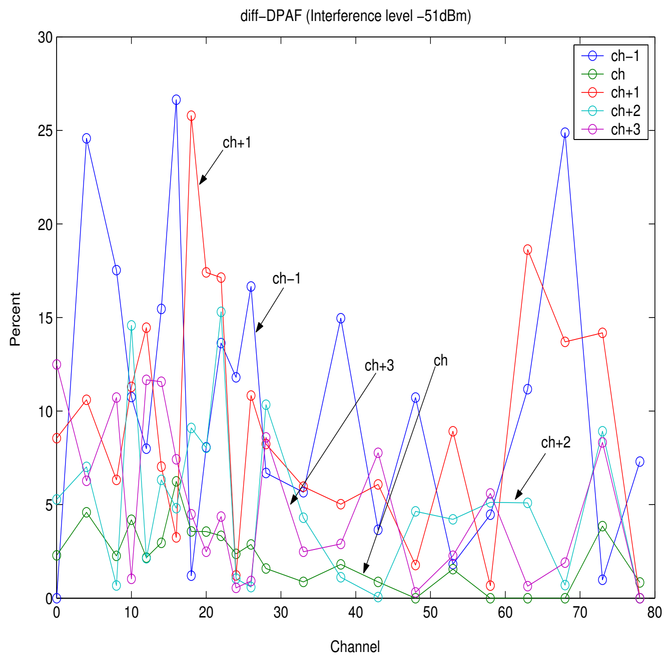


Figure 38: Comparison between simulation and experiment (-51dBm)

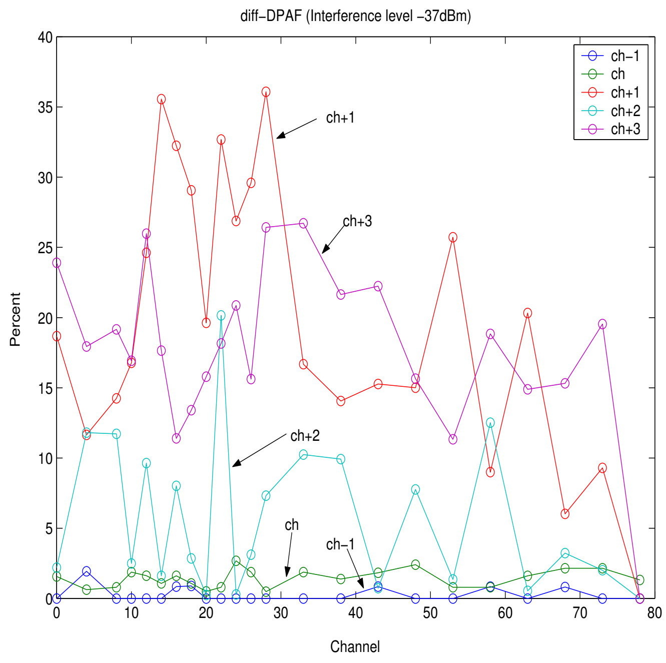


Figure 39: Comparison between simulation and experiment (-37dBm)

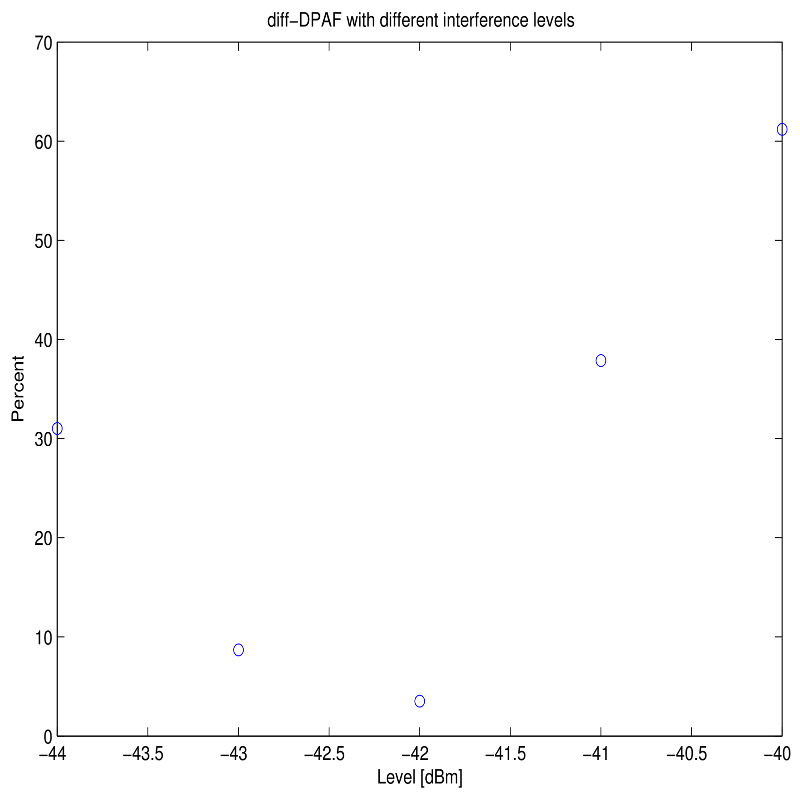


Figure 40: Comparison between simulation and experiment (different levels)

The last of the single-interference source comparisons (figure 41) shows interferences at three different levels at frequencies that increase in small steps from frequency 2465.0MHz to 2465.82MHz. The result is overall quite good. One quite large peak could be found in the topmost plot at frequency 2465.4MHz. The reason for this peak could be for multiple reasons. It could be caused by something in the experiment or by frequency hopping as described earlier. However, the most likely explanation is that frequency 0.4MHz from base frequency 2465.4MHz at level -61dBm ends up in a large slope in figure 34 and it could happen that a simulated packet loss probability will diverge from the actual one.

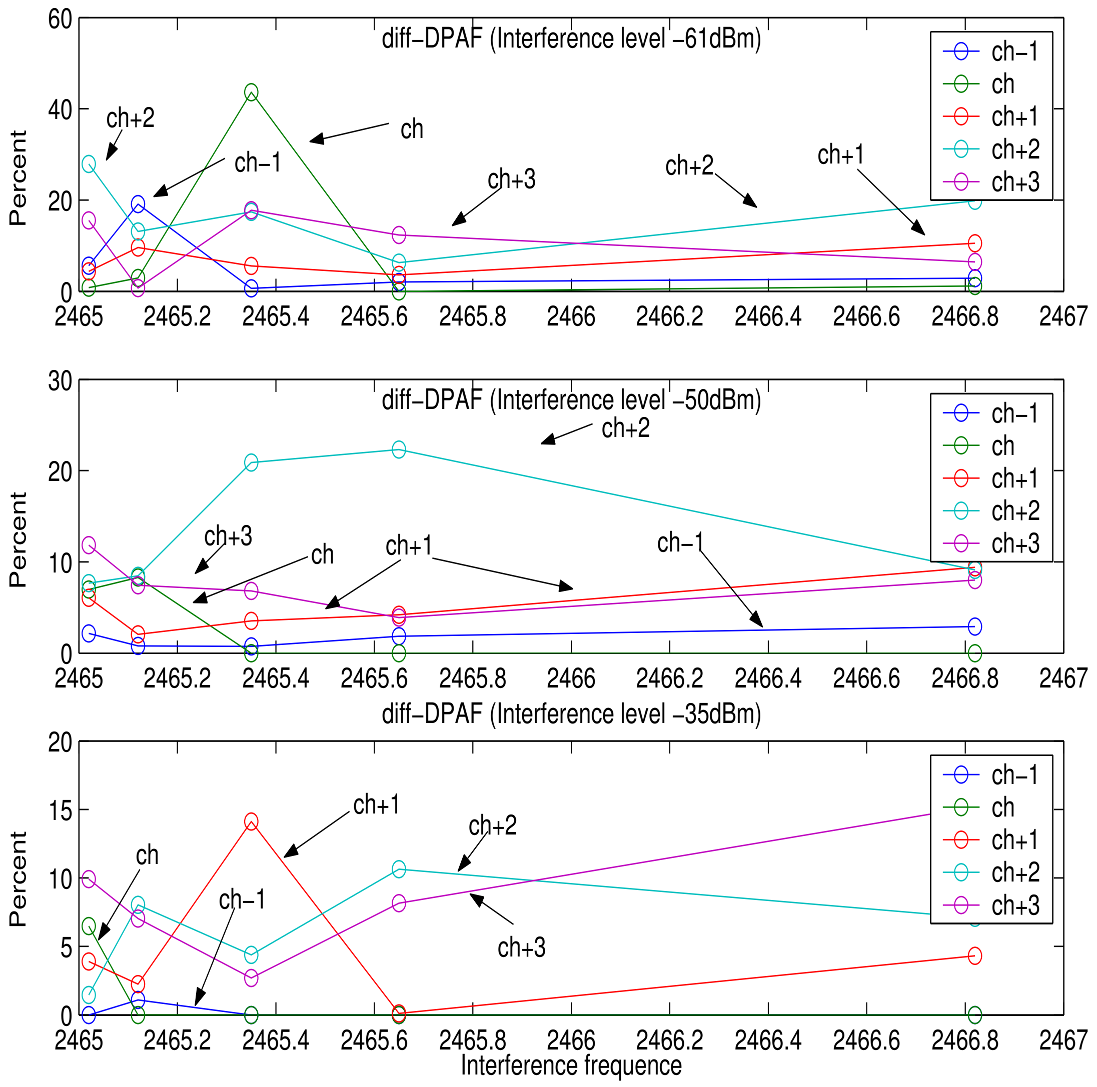


Figure 41: Comparison between simulation and experiment (small frequency steps)

So far only single interference sources have been treated. Imagine that one adds several interference sources at almost the same frequency (divering by less than 0.1MHz). It matters how one measures the spectra. If one has a high sampling frequency with many samples, it will appear to the *Loss Handler* as if many sources of interference. This needs to be handled properly by the *Loss Handler*. The case of several interference sources are handled in the following way:

- Only one source of interference is allowed in each channel.
- If there are several interference sources in the same channel a mean value is calculated.
- A channel is affected by at most one source of interference. If several interference sources interfere with the same channel (via adjacent channels as described earlier) the interference producing the highest packet loss probability is used.

Using this approach, 16 experiments were performed and compared with similar simulations. The properties of the 16 experiments are shown in table 8. Note that in experiment 16 two sources of interferences have been used (as in figure 20), but the *Interference Builder* will treat them as if they where a single interference source. Figures 42 and 43 shows the results from the comparisons between the experiments in table 8 and similar simulations. The x-axis shows the number of the experiment from table 8 and on the y-axis the mean and the highest *diff-DPAF* from the *affected channels* stated for each experiment in table 8.

| **Experiment** | **Interference Frequency [in MHz]** | **Affected Channels** |
|---|---|---|
| 1 | 2433.00 and 2471.00 | 30-35 and 68-72 |
| 2 | 2436.00 and 2468.00 | 33-38 and 65-69 |
| 3 | 2440.00 and 2464.00 | 37-41 and 61-65 |
| 4 | 2444.00 and 2460.00 | 41-45 and 57-61 |
| 5 | 2446.00 and 2458.00 | 43-47 and 55-59 |
| 6 | 2449.50 and 2454.50 | 46-55 |
| 7 | 2450.00 and 2454.00 | 47-55 |
| 8 | 2450.50 and 2453.50 | 47-54 |
| 9 | 2451.00 and 2453.00 | 48-54 |
| 10 | 2451.25 and 2452.75 | 48-53 |
| 11 | 2451.50 and 2452.50 | 48-53 |
| 12 | 2451.60 and 2452.40 | 48-53 |
| 13 | 2451.70 and 2452.30 | 48-53 |
| 14 | 2451.75 and 2452.25 | 48-53 |
| 15 | 2451.85 and 2452.15 | 48-53 |
| 16 | 2452.00 and 2452.00 | 49-53 |

Table 8: Experiment Interference Frequency and affected channels

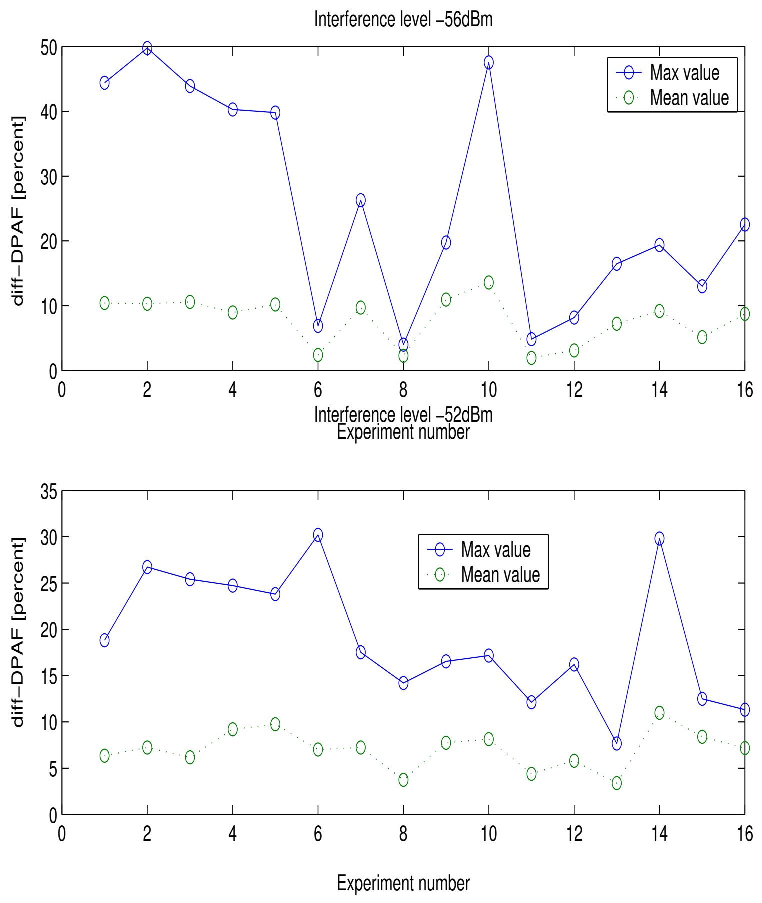


Figure 42: Highest and mean diff-DPAF of the affected channels, level -56 and -52dBm

The highest *diff-DPAF* are at some points quite high while the mean *diff-DPAF* are overall satisfying. The high *diff-DPAF* depends most likely on the same factors described earlier. However, note that none of the large peaks are located at the last experiments which are the most important because in this the two interference sources affect the same channels which was what these experiments and simulations should examine.

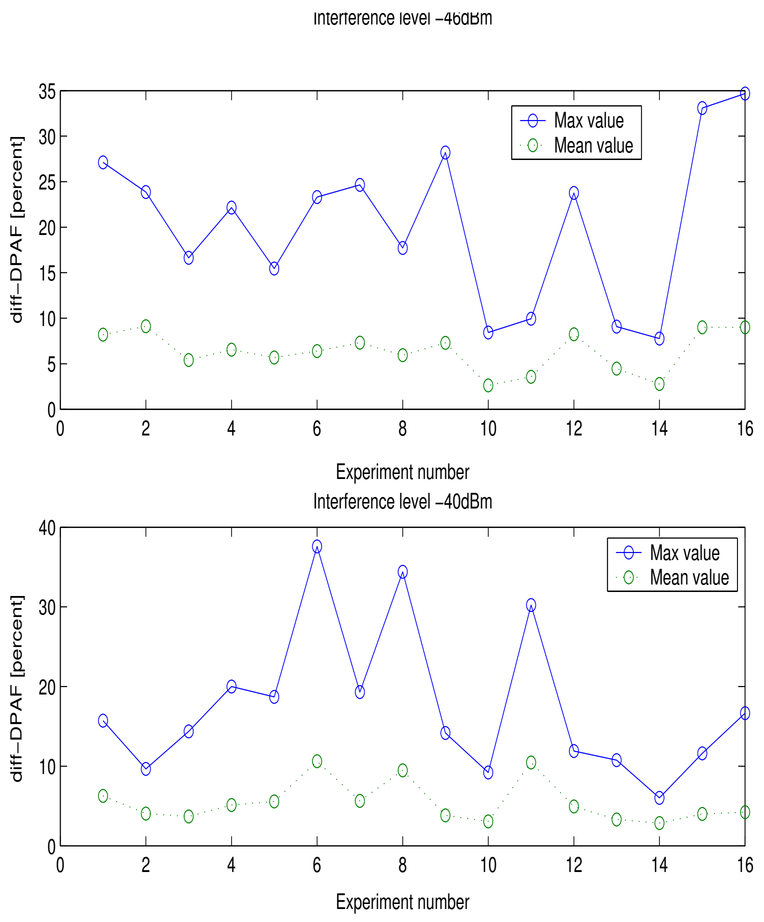


Figure 43: Highest and mean diff-DPAF of the affected channels, level -46 and -40dBm

As one can see in the figures 42 and 43 additional sources of interference produce more accurate simulations (the *diff-DPAF* is lower). This is probably related to the large slope viewable in figures 34 and 35. This slope indicates that the difference in interference level from when an interference source results in a low to a high packet loss probability is quite small.

The last experiment-simulation comparisions in this section are devoted to logger experiments compared with logger simulations. How the model handles logger simulations is described more thoroughly in section 4.2.3 where 15 logger simulations are compared to 15 experiments. The settings of these experiments and simulations can be found in table 9. Both simulations and experiments uses two sources of interference at frequency 2450MHz and 2460MHz, both with level -45dBm.

Figures 44 and 45 shows the result from both experiments and simulations with a reading intervals of one and two seconds (respectively). The value of the x-axis specifies the simulation number in table 9. The results are overall satisfying, with divergences of only a few samples/s.

| Experiment | Sampling rate [in Hz] | Number of Samples |
|---|---|---|
| 1 | 5000 | 50 |
| 2 | 5000 | 100 |
| 3 | 5000 | 250 |
| 4 | 5000 | 500 |
| 5 | 5000 | 1000 |
| 6 | 10000 | 50 |
| 7 | 10000 | 100 |
| 8 | 10000 | 250 |
| 9 | 10000 | 500 |
| 10 | 10000 | 1000 |
| 11 | 25000 | 50 |
| 12 | 25000 | 100 |
| 13 | 25000 | 250 |
| 14 | 25000 | 500 |
| 15 | 25000 | 1000 |

Table 9: Logger Simulations and Experiments with interference

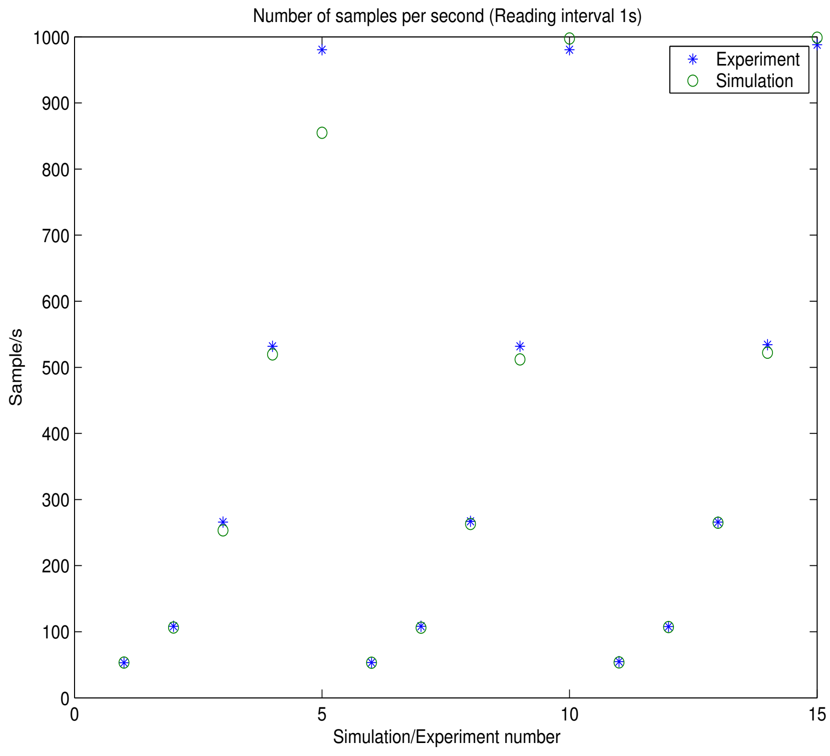


Figure 44: Logger Simulation vs. Experiment with interferences with a reading interval of 1s

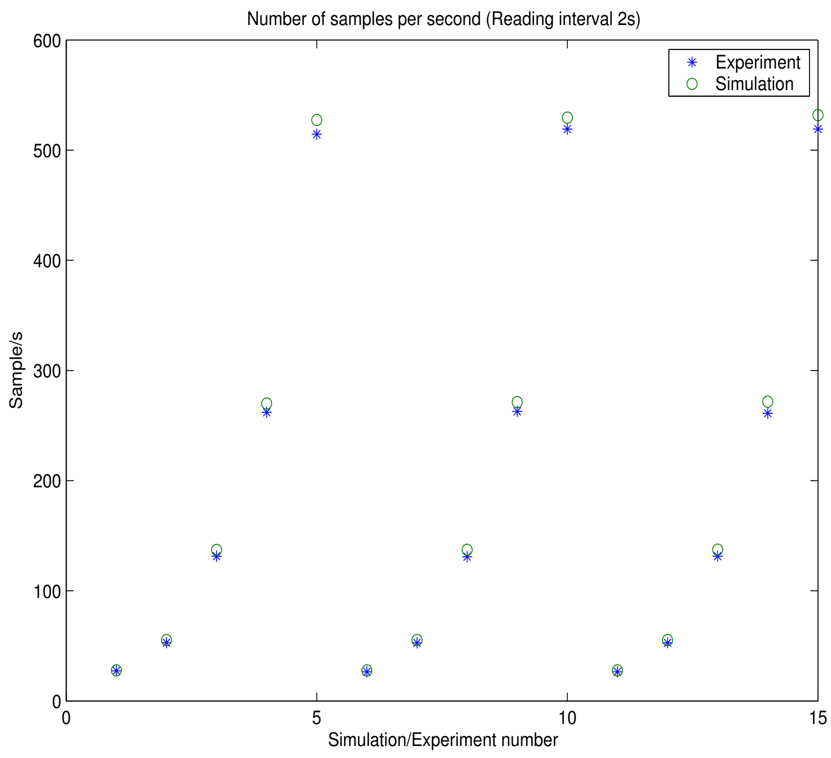
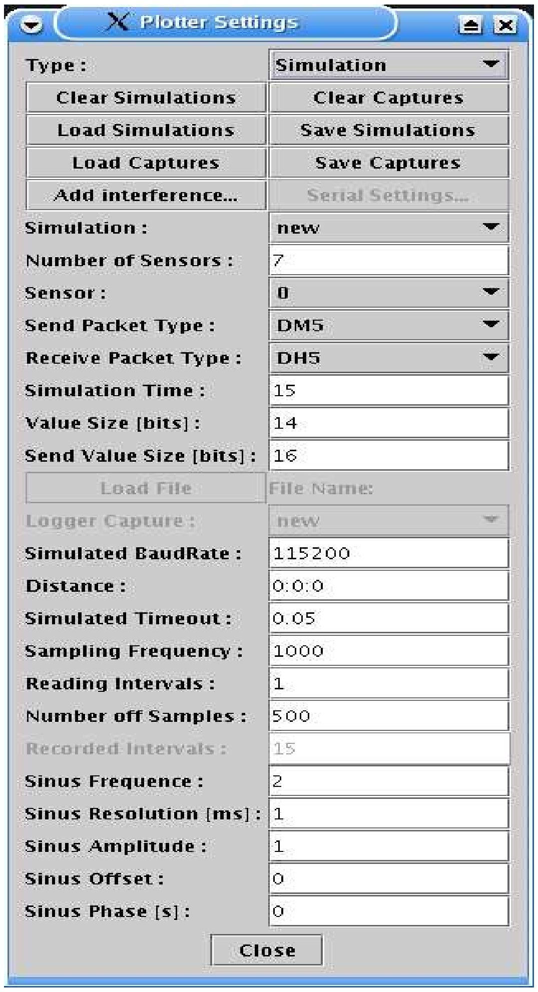


Figure 45: Logger Simulation vs. Experiment with interferences with a reading interval off 2s

## 5.3 Logger Simulator

The Logger Simulator is simply a Dialog Window where the user can fill in the desired properties and then the Simulation Program will construct an OTcl script file for NS2 for the desired simulation. The simulation program will, during the NS2 simulation, read the output from the *Logger Protocol* (see section 4.2.3, evaluate it and then show the results as a plot as shown at the bottom of figure 29. Once a simulation has been performed it is possible to save it and load it for later studies. Figure 46 shows an example of the Dialog Window that the user of the Simulation Program uses to define his/her simulation.

A more thorough explanation of the different fields and how to use the Simulation settings in the dialog can be found in the *Simulation Program Manual* in appendix E.

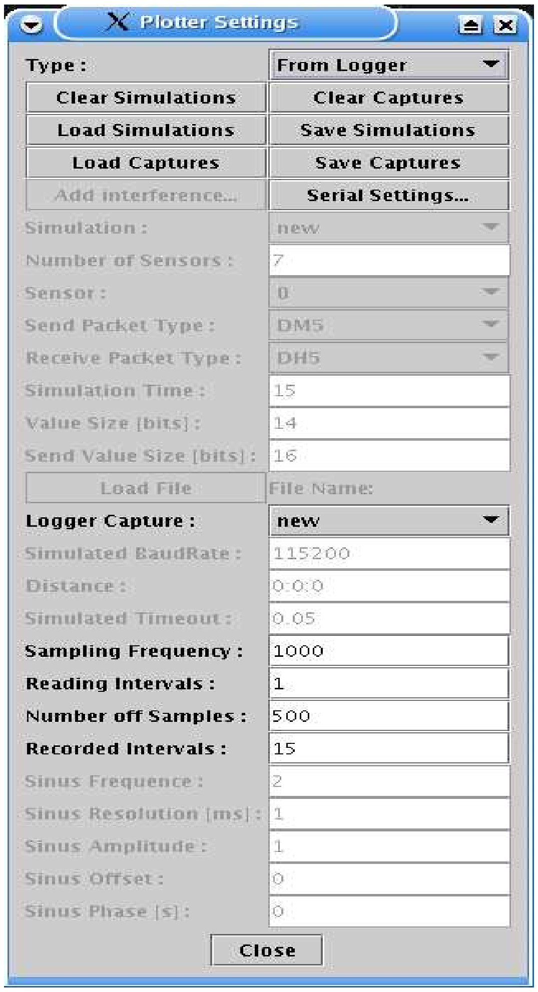


| | |
|---|---|
| Type : | Simulation |
| Simulation : | new |
| Number of Sensors : | 7 |
| Sensor : | 0 |
| Send Packet Type : | DM5 |
| Receive Packet Type : | DH5 |
| Simulation Time : | 15 |
| Value Size [bits] : | 14 |
| Send Value Size [bits] : | 16 |
| Logger Capture : | new |
| Simulated BaudRate : | 115200 |
| Distance : | 0:0:0 |
| Simulated Timeout : | 0.05 |
| Sampling Frequency : | 1000 |
| Reading Intervals : | 1 |
| Number off Samples : | 500 |
| Recorded Intervals : | 15 |
| Sinus Frequence : | 2 |
| Sinus Resolution [ms] : | 1 |
| Sinus Amplitude : | 1 |
| Sinus Offset : | 0 |
| Sinus Phase [s] : | 0 |

Figure 46: Logger Simulation Dialog Example

## 5.4 Logger Reader

For evaluation purposes (and for other purposes) it is possible that the simulation program use results from an actual DL141E logger. The simulation program programs the DL141E and reads its measured values. The *Logger Reader* uses the *javax comm API* [44] for the serial communication with the DL141E[12]. The *Logger Reader* keeps track of the correct time for a sample value, but it **also** logs the time when a value is received. Thus it is possible to view how a signal sampled with the DL141E is received by the simulation program in slow motion or fast forward. Figure 47 shows the *Logger Reader Dialog* for setting up *Logger Reader* sessions.



| | |
|---|---|
| Type : | From Logger |
| Simulation : | new |
| Number of Sensors : | 7 |
| Sensor : | 0 |
| Send Packet Type : | DM5 |
| Receive Packet Type : | DH5 |
| Simulation Time : | 15 |
| Value Size [bits] : | 14 |
| Send Value Size [bits] : | 16 |
| Load File | File Name: |
| Logger Capture : | new |
| Simulated BaudRate : | 115200 |
| Distance : | 0:0:0 |
| Simulated Timeout : | 0.05 |
| Sampling Frequency : | 1000 |
| Reading Intervals : | 1 |
| Number off Samples : | 500 |
| Recorded Intervals : | 15 |
| Sinus Frequence : | 2 |
| Sinus Resolution [ms] : | 1 |
| Sinus Amplitude : | 1 |
| Sinus Offset : | 0 |
| Sinus Phase [s] : | 0 |

Figure 47: Logger Reader Dialog Example

[12]The Java serial communication works well and reads the loggers' for commercial purposes beyond this thesis.

The *Logger Reader* has two limitations:

1. It supports only one active channel. This is sufficient because the *Logger Reader* is mainly an evaluation tool and viewing one channel should be enough.

2. The reading interval can not be set to less than one second. This means that there could be a dead time when the DL141E is neither sampling nor being read. However, it is possible to program the logger so that it starts sampling after a reading period is over. Unfortunately, the DL141E does not support sending samples at beginning of a reading period and so it will then be impossible to know the exact time when the sampling was started. Knowing this time is crucial for determining a sample's time which is crucial for the *Logger Reader*, in order to evaluate the DL141E.

# 6 Results

The results from this thesis should help answer the question: **Which Logger configuration performs best in a specific scenario?** It is impossible to determine a specific configuration that always out performes all other configurations at all time. Trivially using high sampling frequency and a large number of samples results in the highest bit rate, however it also results in the largest delay of a measured value and the largest gap in the measured signal. This is due to the lack of multitasking in the DL141E.

If one would like to set up a table containing the throughput for all configurations it would have to include:

- **Number of samples :** directly related to the buffer size. The number of samples could be every integer from 1 to 256000, i.e. up to 256000 entries.
- **Sampling frequency :** Could be every integer from 1 to 30000, i.e up to 30000Hz.
- **Sources of interference :** This is a the most difficult part to treat. Should one use same packet loss probability for all Bluetooth packets? For all channels? And how does the system developer know which packet loss probability that is the correct for his/her setting.

Describing the throughput for all configurations would then in worst case require a $256000 \cdot 30000 = 7.68 \cdot 10^9$ entries table for each desired interference case. However, it is not neccesary to investigate all sampling frequencies and all buffer sizes. Note that the DL141E is new on the market and no one knows exactly which configurations that will be its main market.

Instead of publish a huge table in this report, this thesis provides a simulation program that could be used and understood by **anyone** who has basic computer knowledges. The simulation program (described in section 5) provides the user with results from specific cases specified by the user while in the end of this section gives *general evaluation results* and *optimization results* of the logger.

Trivially a large number of samples and high sampling frequency results in the highest throughput, thus a large number of samples reduces the overhead in the Logger and high sampling frequency reduces the time when the logger samples are enqueued. However, because of the long delay collecting the measured values of a large number of samples it might not be possible to use the *full buffer* approach in all cases. Thus it would be interesting to find a number of samples were the delay is low and the overhead in multiple reading intervals is negligible. Figure 48 shows results from simulations with different buffer approaches at 0%, 25%, and 50% packet loss probability. In these simulations the logger has used *text mode* (with all samples of the size 48 bits) and sampling frequency of 30kHz has been used. Figure 48 shows that up to about 1000 samples per sampling interval throughput increases as the number of samples increases. However, using buffer sizes of over 1000 samples has small effects on the throughput. Note also that at high packet loss probability, the point were larger number of samples per transfer stops increaseing the throughput is lower.

Figure 49 shows the time it takes to fill and send a buffer for the logger at the different buffer sizes from figure 48. Figures 48 and 49 indicate that using a buffer size with approximately 1000 samples produces the best compromise between maximum delay and throughput. Here the delay will be 33ms to fill the buffer and 683ms to send it.

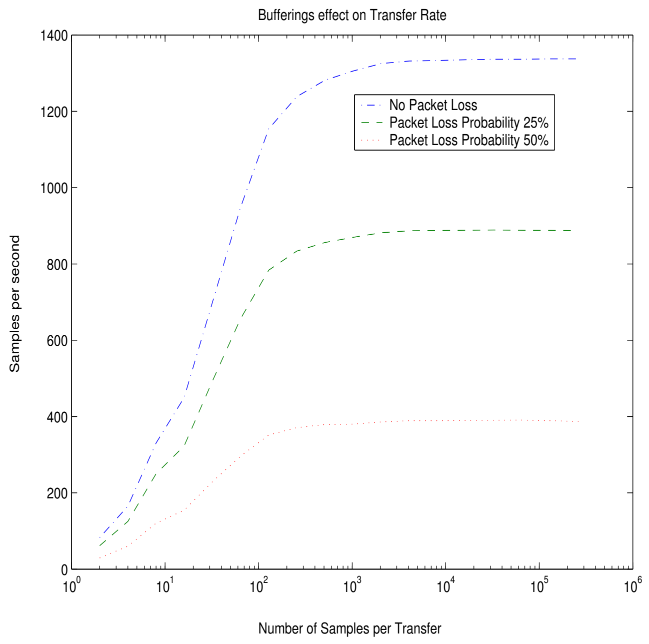


Figure 48: Samples per Second with Different Buffer sizes

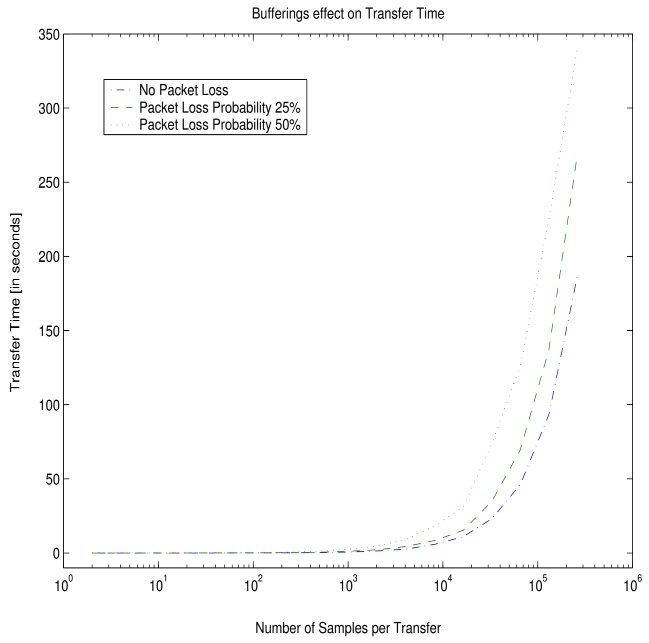


Figure 49: Transfer Time with Different Buffer sizes

The rest of this section will focus on 30 kHz sampling with 1000 samples per sampling interval. Two logger configurations will be derived and compared:

1. **Current logger configuration :** The logger configuration that is used today by logger reader software.
2. **Optimized available logger configuration :** The best logger configuration that could be achieved with the settings that DL141E allows.

We know that the highest possible net bit rate for Bluetooth data traffic is 723.2kbit/s (see table 3), achieved using *DH5* packets. Because each sample are 14 bits the **highest** throughput in sample per second should be $\frac{723.2 \cdot 1000}{14} = 51657.1 samples/second$. Trivially we see that the DL141E uses *DM5* which according to table 3 only has a net rate of 477.9kbit/s, the max sampling rate is 30kHz, thus one can only achieve 30 000samples/s, and the DL141E uses more than 14 bits to send each value. In the end of this section we will examine more thorough were in the logger configuration that throughput are lost and how large the losses are.

## 6.1 Optimizing the DL141E

The easiest optimization that could be done in the frame of the available settings of the DL141E is to change from *text mode* to *rawdata mode*. The current logger reader programs uses *text mode*, but some tests have been performed with the *java logger reader* mentioned in section 5.4. In *text mode* each sample uses up to 56 bits (but on average 48, which is the value used in the experiments in this section) even though each sample is sampled with only a 14 bit ADC. However, the *rawdata mode* uses only 16 bits which will be a factor of three improvement. Figure 50 shows number of samples per second for 1000 samples, 30kHz sampling interval at packet loss probability from 0-75% for *rawdata* and *text mode*. The point were the throughput has reached its half value compared to zero packet loss probability are highligted for both curves. *Text mode* has reach its half value at approximately 36% packet loss propability while *rawdata mode* has reached its at 38% packet loss propability, so non of the both performs obviously better in interference.

Because the *text mode* uses three times more data than the *rawdata mode* one would expect that the *rawdata mode* would perform three times better than the textmode. However, figure 51 shows that a three times improvement never is reached, but at higher packet loss probability the improvement is better than with smaller loss probabilities. The reason is when using less bits per transfer there are fewer frames to transfer hence it is less sensitive to interference. The reasons for the variances in figure 51 is that each interference level is simulated 25 seconds and then the throughput is calculated. It can then varier how many reading intervals that are read during a simulation and were in the sampling or sending of a interval that the simulation is aborted. This has effects on the throughput. The fitted line in figure 51 is a least squares fit of the improvement values.

Another large advantage of using *rawdata mode* instead of *text mode* is that receiving and typecasting values from the logger in a logger reader program is handled more effective when *rawdata mode* is used. This is however out of the scope for this thesis but interesting enough to be described briefly in appendix F. This could also indirect effects the throughput thus many reader programs receives data from the logger, process the data and then starts the logger sampling again, resulting in a delay when the logger does nothing.

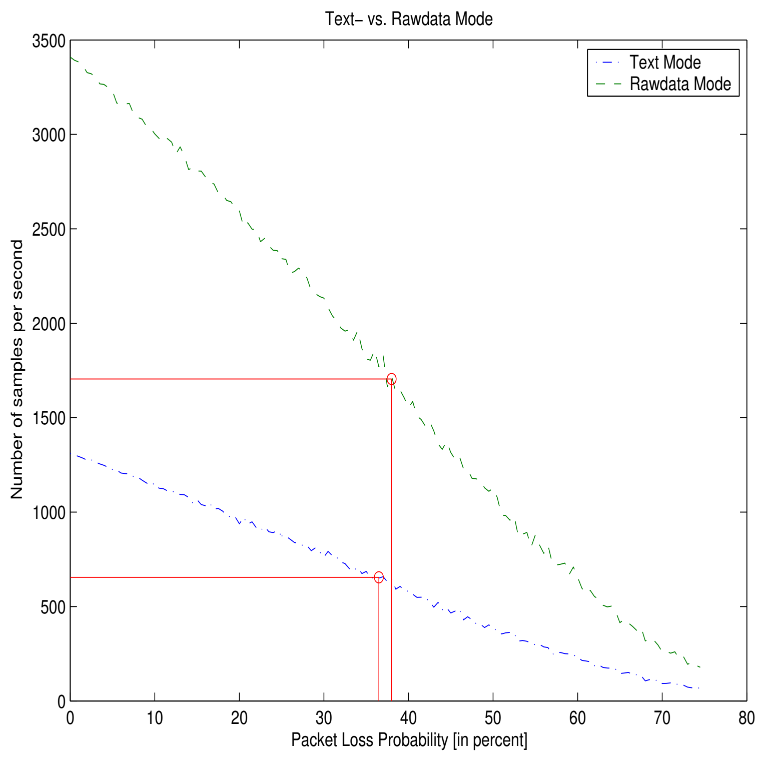


Figure 50: Samples Per Second with Text- vs. Rawdata Mode

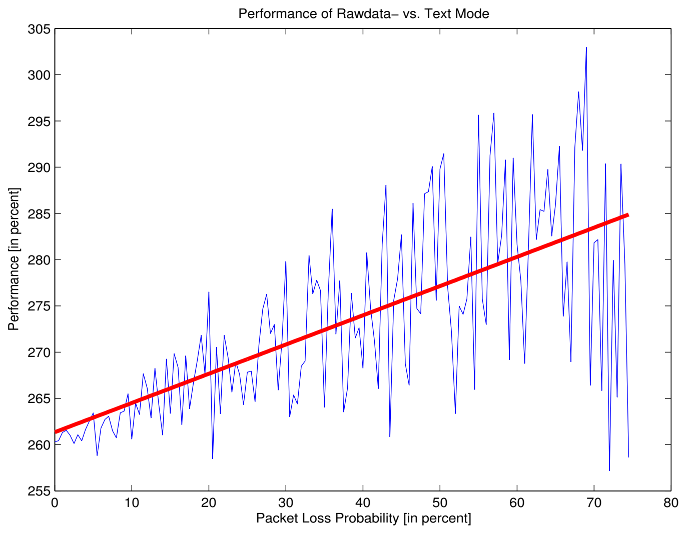


Figure 51: Rawdata Mode's Effectiveness Compared to Text Mode

For a *DM5* packet the maximum net bit rate is according to table 3 477900bit/s. However, the current logger configuration however uses only 115200bit/s which results in low use of the Bluetooth bandwidth. In figure 52 the baudrate of the interface between the logger's buffer and the Bluetooth module has been increased in three steps: First from the standard RS-232 baudrate 115200bits/s to the standard baudrate 230400kbit/s. Second to the standard baudrate 460800kbits/s, and Finlay to the baudrate 480000kbits/s which is slightly above the highest reachable net bit rate. The packet loss probability at which the half Baudrate is reached compared to no packet loss is marked for each baudrate in figure 52, however it does not point at any specific baudrate which out performs the others.

Using *rawdata mode* and the higher baudrate is the easiest ways of increase the throughput for the DL141E without changing it's fixed configuration. The DL141E which uses *rawdata mode* and 480000kbits/s baudrate is called *the optimized logger*. Figure 53 shows the buffering's effects on the throughput for *the optimized logger*. The highest throughput is about 8000samples/seconds which is far from the highest achievable throughput at 51657samples/seconds. What needs to be changed in the DL141E to achieved maximum throughput will be described in next section (section 6.2).

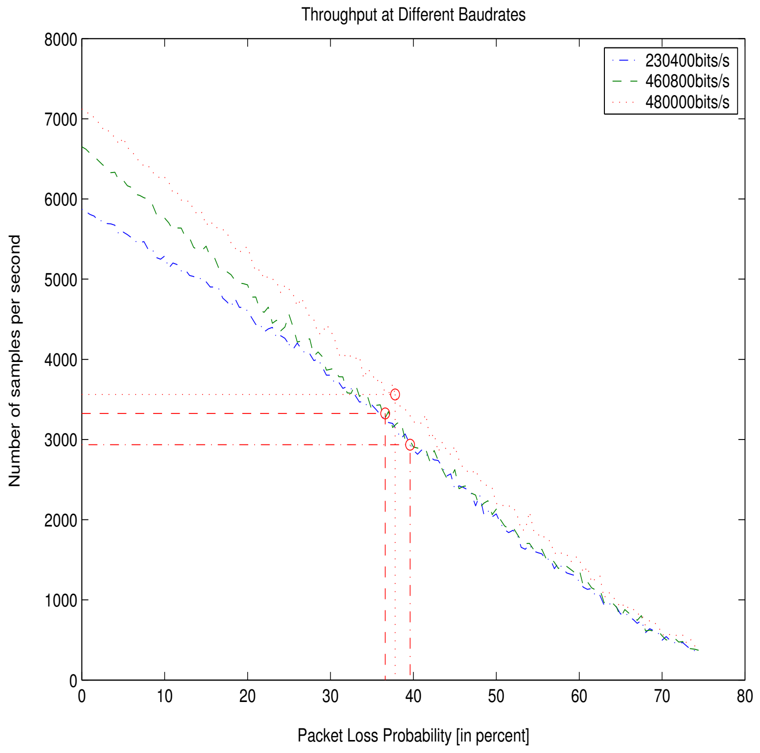


Figure 52: Samples Per Second with Higher Baudrates

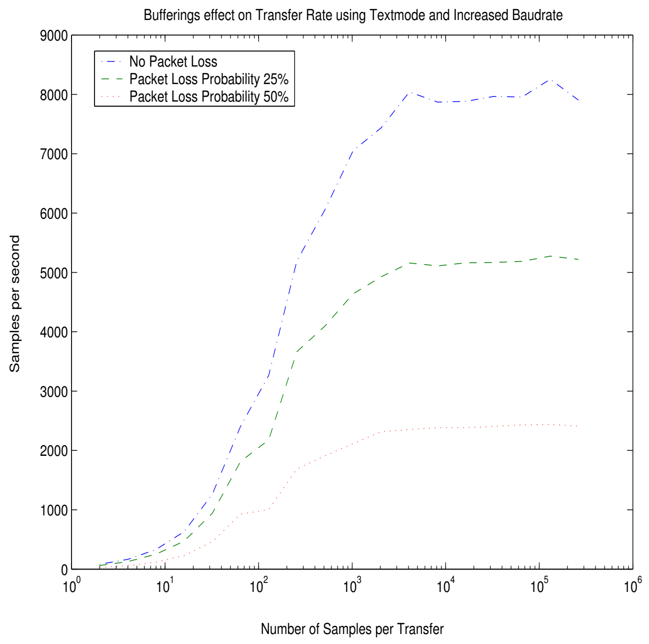


Figure 53: Buffering's Effects on the Optimized Logger

## 6.2 Achieving Maximum Throughput

The DL141E is constructed in a way that prevents it from using the full capacity of the Bluetooth link. In this section, different configurations will be investigated by repeatedly changing the configuration – until the highest achievable throughput at about 51600samples/second (derived from maximum net bit rate of 723.2kbit/s as shown in table 3) will be reached. The optimizations are performed in an order starting with the change which is believed to be the easiest change to the current logger's construction and ending with the change which is believed to be the most difficult construction & configuration change. Note that all configurations in this section are assumptions concerning a future logger configuration. Actual configurations are expected to only slightly diverge in throughput from these estimates. Figure 54 shows three changes of *the optimized logger* with the purpose to increase the throughput.

- **DM5 :** Make sure that each *DM5* Bluetooth packet is filled with data, i.e. according to table 1 each packet should have 224 bytes of data. This increases the throughput at 1000 samples per reading interval by about 300 samples/second.
- **DH5 :** Change to the more bandwidth effective Bluetooth packet *DH5*, and make sure that the payload in each packet are filled, i.e. contains 339 bytes of data according to table 1. This increases the throughput at 1000 samples per reading interval by about 2000samples/seconds.
- **14 bits :** Only send 14 bits per sample, reducing the transmission overhead by two extra bits. This results in a 1000 samples/second gain.

The packet loss probability where each configuration has reached its half non interference value is marked for each configuration in figure 54. The point increases slightly for each improvement of the configuration.

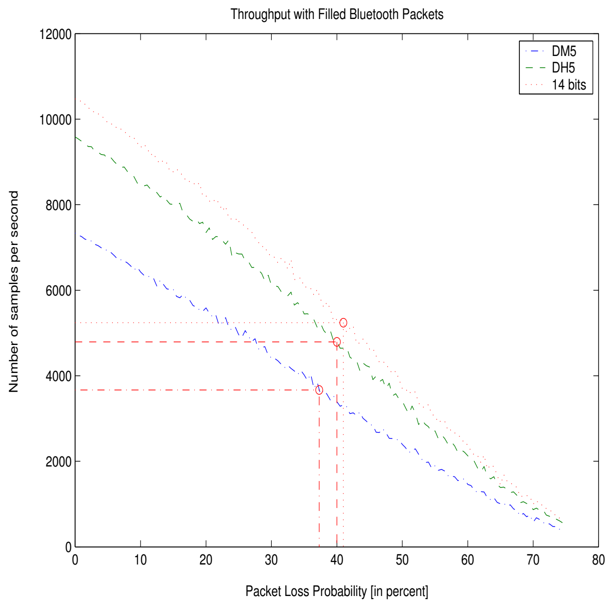


Figure 54: Throughput with Filled Bluetooth Packets

Still we have only reached less then a $\frac{1}{4}$ of the maximum achievable throughput. The results of the last and most effective three configuration changes are shown in figure 55. The three different configurations are:

1. **Multitasking :** Multitasking: Allows the logger to sample and send at the same time. This reduces the idle time, i.e. when the Bluetooth Module is waiting for values to be ready to send, resulting in a significant increase in throughput . This configuration change has also many other benefits, e.g. it is now possible to measure continuous signals, the maximum delay of measured values will decrease, the logger will be easier to program, etc. The number of samples in each reading interval is no longer important – once we exceeded the number of samples will produce large Bluetooth frames (about 1000 samples/second). However, there is still room for improvement; specifically by about 7000 samples/second to achieve 30000 samples/second which is the maximum sampling frequency of the ADC. Because a DL141E with multitasking sampling and transmission does not exist it is not possible to tell exactly how this logger would work. As to buffering, the approach in the simulation of this configurtion drops as many samples as necessary, when the buffer is full and new values should be added, which is the simplest approach.

2. **Logger, Bluetooth Logger Interface :** Logger, Bluetooth Logger Interface: The rest of the samples are delayed in the interface between the logger's buffer and the logger's Bluetooth interface. By allowing the Bluetooth module to fill a Bluetooth packet and send it whenever it is ready, results in the second curve in figure 55 are achieved. Note that when the Bluetooth module is in control of the buffer packet losses of upto 20% do not results in lower throughput. This is because the Bluetooth module now has the ability to use the entire Bluetooth bandwidth.

3. **Increase the sampling frequency :** Given the above two improvements it is now possible to increase the sampling frequency upto (almost) 52kHz, resulting in roughly the maximum number of samples per second (51657.1samples/s) which the link can support.

The packet loss probability where each configuration has reached its half non interference value varies in figure 55. The first configuration which uses multitasking sample and send, but the serial interface between the logger's buffer and Bluetooth module has the same half throughput point as we have seen in earlier configurations. However, in the second configuration, where the Bluetooth module are in complete control of the buffer the half throughput point is higher (about 44%). The reason is that the Bluetooth module now can send more values than the logger can produce, which allows the device to compensate for lost packets which need to be retransmitted. The last configuration has the lowest half throughput point as we have seen so far. The reaon is that here the maximum net bit rate of the Bluetooth protocol is used and every packet lost will directly effect the throughput.

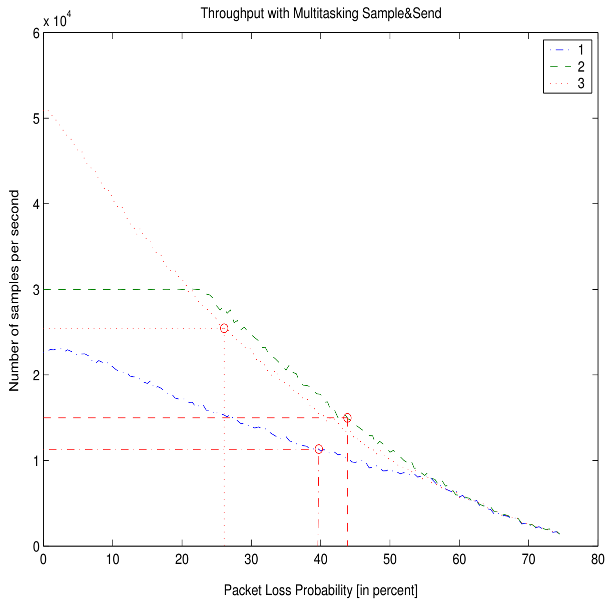


Figure 55: Throughput with a multitasking Logger

# 7 Conclusion

This section will summarize the work that has been done in this master's thesis. In section 7.1 the *conclusions of the thesis* are discussed, in section 7.2 proposals of *future work* are discussed, and finally in section 7.3 the *shortcomings* of this thesis are discussed.

## 7.1 Thesis Conclusions

Generally speaking the DL141E is easy to use, produces accurate measurements, and responds to its commands. However there are still a lot of optimizations that needs to be done to really use the bandwidth available in the Bluetooth protocol.

The conclusions of this thesis are divided in three groups:

1. **General :** includes conclusions from the work in general. These are described in section 7.1.1.
2. **Interferences :** Interference is an important component of this masters thesis, sources of interferences have been used in both simulations and experiments. The conclusions from this are described in section 7.1.2.
3. **Logger Configurations :** as shown in section 6 these configurations have a large effect on the throughput. In section 7.1.3 conclusions about these configurations are discussed.

### 7.1.1 Overview

The difficulties with this masters thesis could be summarized in the following items:

- **Experiments :** performing experiments is very time consuming. However evaluating the experiments is even more time consuming. For example, to evaluate the first sets of experiments that were performed in this thesis took about two weaks, the result was the determination that a different set of experiments were needed. The problem of evaluating experiments properly includes also figuring out a way of making the results available for others. An example of this is the *Interference Builder* (see section 5.2) that uses experiments to derive packet loss probabilities at Bluetooth channels for the simulations.
- **The Logger :** The DL141E was only recent developed and has not been used extensively. This has sometimes made it difficult to know exactly how logger works and to use all of its functions. For example, all the existing programs that read data from the DL141E use *text mode*, making it difficult to perform experiments in *rawdata mode*. Tests were made with a program designed specifically for this thesis that would read the logger in *rawdata mode*, but problems reprogramming (i.e. sending the sample and send command shown in table 5) the logger after each reading interval made it impossible to get accurate throughput measurements in *rawdata mode*.

### 7.1.2 Interference

Interferences are very difficult to categorize. It is not possible to say that a specific interference has a specific effect on the Bluetooth device. Much research has been performed regarding how different Bluetooth networks responds to different packet loss probabilities, but how does one know what packet loss probability exist? Additionally this can vary between different scenarios.

One of the first results of this thesis was that to easily describe interference source one needs a flexible way of adding interference sources to a model. This lead to the construction of the *Interference builder* (described in section 5.2) which holds the results from a large number of experiments and produces a packet loss probability based on these experiments. The *Interference builder* allows the user to come to conclusions of his/her system based on experiments performed in this thesis. However, the cost of the flexibility is the loss of correctness. As shown in section 5.2.2 the packet sometimes diverge from an experiment compared to a simulation.

### 7.1.3 Logger Configurations

The original question for this thesis was: “How does the size of the DL141E’s buffer affect the data throughput, especially if there are interferences?” However in figure 48 in section 6 we have seen that the design of the DL141E so far does not use more then about $\frac{1}{40}$ of the available Bluetooth bandwidth. Thus rather than being concerned with the effects of interference it is more important to optimize the logger as much as possible. There are several optimization that could be performed without changing the design of the DL141E and there are also a number of optimization that quite easily could be changed in the design of the DL141E. These are discussed further in section 7.2.1.

Section 6 also showed that one cannot simply focus on throughput when one considers the properties of a logger system. Due to the lack of multitasking, i.e. the logger can not send and sample at the same time, there will be delays in the measured values. These delay will increase as the number of samples each sampling period increases, but the throughput will be lower using small number of samples per reading interval, then using a large number of samples. About 1000 samples per interval was shown in section 6 to be a good compromise between bandwidth and throughput (see figure 48 and 49).

The requested throughput and maximum allowed delay does however vary depending on which system one would like to build. This lead to the construction of the simulation program (described in section 5) where the user could perform his/her own simulations with different configurations and compare them to other configurations. The simulation program was found to be the best way to:

- Fast and easy get an overview from a simulation.
- Allow users not familiar with NS2-UCBT (described in 4.1) with its extensions (described in section 4.2) to perform simulations.
- Utilize information from experiments to add interference sources (in form of packet loss probabilities) to a simulation.

In section 6.2 we saw that increasing the number of samples per reading interval has less significance when the logger is capable of multitasking. One only needs to make sure that enough samples are produced for the Bluetooth module. While the simulation program and the figures in section 6 could help setting up the most efficient logger configuration for the existing DL141E, further optimizations of the DL141E design would have a far larger improvement on the logger’s throughput.

## 7.2 Future Work

This section describes future work in the subject. It is divided in two sections:

1. **Logger Improvements :** proposals to optimize the DL141E. This section is mainly for the developers of the DL141E.
2. **Thesis Continuations :** proposals of future work directly associated with this thesis. This section is for future thesis and research projects that continue from where this work ends.

### 7.2.1 Logger Improvements

This thesis has shown that there are a number of optimizations that should be made in the DL141E configuration and design. Table 10 summarizes the optimizations that were performed in section 6. With

| Configuration | Throughput | Percent of maximum throughput | Improvement |
|---|---|---|---|
| Current logger configuration | 1 300samples/s | 2.5% | |
| Rawdata mode | 3 400samples/s | 6.6% | 261% |
| 480kbit/s baudrate | 7 000samples/s | 13.6% | 206% |
| Filled DM5 packets | 7 300samples/s | 14.1% | 104% |
| Filled DH5 packets | 9 500samples/s | 18.4% | 130% |
| 14 bits per sample | 10 500samples/s | 20.3% | 111% |
| Multitasking sample/send | 23 500sample/s | 45.5% | 224% |
| Optimized logger to Bluetooth module transfer | 30 000sample/s | 58.1% | 128% |
| Increased sampling rate | 51 600sample/s | 100% | 220% |

Table 10: Logger Improvements

table 10 and section 6 as background this thesis propose the following future work in the logger design:

1. Ensure that all programs that read from the logger can use *rawdata mode*. This is the easiest change that could be performed, but it has the largest improvement (calculated in percent) of the changes actually performed in this thesis.
2. Investigate properly how the interface between the Logger's buffer and the Bluetooth module is working. This thesis has shown that many throughput improvements could be found here. Start with trying to achieve higher baudrates, then try to make sure each Bluetooth packet is filled, and finally make sure that the Bluetooth Module is fed with all values it can consume at all time. Achieving some of these goals or only half way of them will increase the performance of the logger substantially.
3. Try to optimize the code for the microprocessor in the DL141E. To change so that only 14 bits are send should not be very hard. It should not be impossible to increase the multitasking in the current microprocessor to allow sampling and sending at the same time at least not if one could settle with lower sampling rate. It should also be investigated properly if the sampling rate could be increased, it is possible that changing the code correctly would increase the sampling rate. In section 6 this thesis have shown that a sampling rate up to 52kHz is necessary to obtain maximum throughput. When all investigations of the current DL141E source code has been performed and still full multitasking at 52kHz has not been achieved an investigation about how much the cost will be for updating the hardware in the logger to allow full performance should be done. The results from this investigation should tell how future DL141E design should be performed.

The DL141E in its current configuration is robust, easy to use, offers high sampling rate, and a large memory. For the proposed systems today with the DL141E the design is probably more than enough. However, the main result of this thesis is that one should not perform any Bluetooth optimization until at least these three aspects have been properly investigated.

### 7.2.2 Thesis Continuations

The characterization of sources of interference and their affect upon Bluetooth could be investigated further. This thesis suggests that future investigations focus (as this thesis has done) more on how a given interference source changes the packet loss probability. Rather than has been done earlier where the focus has been on how the loss probability affects Bluetooth's throughput. Additionally, further characterizations as performed in this thesis could be done. The following four subjects are examples of what could be investigated further:

1. How different Bluetooth packets responds to signal interferences (generated from a signal generator, as in this thesis) at different frequencies with different levels.
2. Investigate the difference between the different Bluetooth transmit power classes (see table 2).
3. Try to find out what the most common sources of interference for a Bluetooth device are, focused on future systems with the DL141E
4. In the future when there are several DL141E systems in use it would be interesting to measure some of them to see how they differ with respect to throughput and packet loss.

The list could be made longer, thus it is necessary to limit the experiments to the scenarios that one are interested in and limit the sources of interference that one would like to investigate to sources of interference that are present in the environment that one would like to examine.

Future systems using the DL141E may use many loggers connected to the same computer. Although the current version of the logger supports constructions of piconets, it is not fully implemented yet - as the developers wish to examine more thoroughly which scheduling algorithm to use. A reader program that reads multiple loggers also needs to have a suitable scheduling algorithm to help deciding which order to read the loggers. The scheduling algorithm should be selected after performing simulations with extensions to the model and simulation program provided by this thesis. This thesis suggest that another thesis project should be undertaken, for example a 10 credit C-level thesis, suitable for two students. A more formal description of this proposed thesis can be found in appendix G.

Another possible thesis that continues the work from this thesis is to examine the possibilities of constructing simple scatternets with the DL141E. Such a scatternet thesis would focus on the scheduling algorithm thesis (described above), while build its foundations on this thesis and using the simulation program and model to perform simulations that should provide the information necessary to construct simple scatternets with the DL141E. The scatternet thesis is probably a C-level 10 credits thesis, appropriate for one or two students. A more formal description of the thesis can be found in appendix H.

## 7.3 Shortcomings

The shortcomings of this thesis are mainly associated with the experiments. It would have been interesting performing more experiments with more sources of interferences and several Bluetooth devices. The reason why many of the desired experiments were omitted are the following:

1. It takes long time performing experiments and evaluating them in an appropriate way. It is important, not only that the right experiment is performed, but that the experiment can be followed by an accurate evaluation allowing others to take advantage of the result.

2. Whenever experiments are made one are limited to the resources at the lab. In this thesis there were good access of equipment, but still one can not perform all experiments that one would desire. E.g. IUC did not have the resources to measure large networks, i.e. networks containing many loggers, nor was it possible to add all types of interferences that were desired.

# References

[1] Bluetooth SIG. Bluetooth specification. Available: www.bluetooth.com, accessed Aug. 2004.

[2] The Network Simulator - ns-2. www.isi.edu/nsnam/ns/index.html, Accessed Febuary, 2005.

[3] UCBT - Bluetooth extension for NS2 at the University of Cincinnati. www.ececs.uc.edu/ cdmc/ucbt/ucbt.html, Accessed 2005.

[4] Jennifer Bray and Charles F. Sturrman. *Bluetooth 1.1, Connect Without Cables*. Prentice Hall PTR, 2 edition, 2002.

[5] Gerald Q. Maguire Jr. 2G1330 Mobile and Wireless Network Architectures: Bluetooth. Lecture notes, Royal Institute of Technology (KTH), March 2004.

[6] S Zürbes. Considerations on link and system throughput of bluetooth networks. *Proc. IEEE Personal, Indoor and Mobile Radio Communications Conf., London*, 2, Sept. 2000.

[7] M.C. Valenti, M. Robert, and J.H. Reed. On the throughput of Bluetooth data transmissions. *Proc. IEEE Wireless Commun. & Networking Conf. (WCNC), (Orlando, FL)*, pages 119–123, Mar. 2002.

[8] M. C. Valenti and M. Robert. Custom Coding, Adaptive Rate Control, and Distributed Detection for Bluetooth. *Proc. IEEE Vehicular Tech. Conf. (VTC), (Vancouver, BC)*, pages 918–922, Sept. 2002.

[9] M.C. Valenti and M. Robert. Improving the QoS of Bluetooth through Turbo coding. *Proc. IEEE Military Communication Confference (MILCOM), (Los Angeles, CA)*, pages 1057–1061, Oct. 2002.

[10] A. Das, A. Ghose, V. Gupta, A. Razdan, H. Saran, and R. Shorey. Adaptive Link-level Error Recovery Mechanisms in Bluetooth. *Proc.of the IEEE International Conference on Personal Wireless Communications*, pages 85–89, Dec. 2000.

[11] J. Kim, Y. Lim, Y. Kim, and J. S. Ma. An adaptive segmentation scheme for the Bluetooth-based wireless channel. *Proc. IEEE International Conference on Computer Communications and Networks (ICCCN 2001), Scottsdale, AZ*, 2001.

[12] Mark Francis D'Souza. Residential Microwave Oven Interference on Bluetooth Data Performance. Master's thesis, Center for Wireless Telecommunications, 2002.

[13] IEEE Coexistent Task Group. www.iee802.org, Accessed Febuary, 2005.

[14] N. Golmie, R. E. Van Dyck, A. Soltanian, A. Tonnerre, and O. Rébala. Interference evaluation of Bluetooth and IEEE 802.11b systems. *Wireless Networks*, 9(3):201–211, May 2003.

[15] C. F. Chiasserini and R. R. Rao. Coexistence mechanisms for interference mitigation between IEEE 802.11 WLANs and Bluetooth. *Proc. of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, 2:590–598, June 2002.

[16] N. Golmie, N. Chevrollier, and O. Rebala. Bluetooth and WLAN coexistence: challenges and solutions. *Wireless Communications, IEEE*, 10(6):22–29, Dec. 2003.

[17] Amre El-Hoiydi. Interference Between Bluetooth Networks-Upper Bound on the Packet Error Rate. *IEEE Communications letters*, 5(6), June 2001.

[18] B. S. Peterson, R.O. Baldwin, J.P. Kharoufeh, and R. A. Raines. Refinements to the packet error rate upper bound for Bluetooth networks. *IEEE Communications Letters*, 7(8):382–384, Aug. 2003.

[19] Ting-Yu Lin and Yu-Chee Tseng. Collision analysis for a multi-Bluetooth picocells environment. *IEEE Communications Letters*, 7(10), Oct. 2003.

[20] Kihong Kim and Gordon L. Stüber. Interference Mitigation in Asynchronous Slow Frequency Hopping Bluetooth Networks. *Wireless Personal Communication*, 28(2):143–159, Jan. 2004.

[21] G. Pasolini. Analytical investigation on the coexistence of Bluetooth piconets. *IEEE Communications Letters*, 8(3):144–146, March 2004.

[22] Björn Jonsson. Bluetooth QoS Scheduler. Master's thesis, Royal Institute of Technology, Stockholm, 2003.

[23] Niklas Johansson, Ulf Körner, and Per Johansson. Performance Evaluation of Scheduling Algorithms for Bluetooth. *IFIP TC6 Fifth International Conference on Broadband Communications, Hong Kong*, 5:139–150, Nov. 1999.

[24] A. Capone, M. Gerla, and R. Kapoor. Scheduling schemes for Bluetooth Picocells. *International Workshop on 3G Infrastructure and Services", Athens, Greece*, July 2001.

[25] Rachid Ait Yaiz and Geert Heijenk. Polling Best Effort Traffic in Bluetooth. *Proceedings 4th International Symposium on Wireless Personal Multimedia Communications (WPMC'01), Aalborg, Denmark*, Sept. 2001.

[26] Ka Lok Chan, Vojislav B. Misic, and Jelena Misic. Efficient Polling Schemes for Bluetooth Picocells Revisited. *Proc. of the 37th Annual Hawaii International Conference on System Sciences, Big Island, Hawaii*, Jan. 2004.

[27] Jelena Misic and Vojislav B. Misic. Modeling Bluetooth Piconet Performance. *IEEE Communications letters*, 7(1), 2003.

[28] N. Golmie. Bluetooth dynamic scheduling and interference mitigation. *Mobile Network Applications*, 9(1):21–31, Feb. 2004.

[29] Daniele Miorandi, Andrea Zanella, and Gianfranco Pierobon. Performance evaluation of Bluetooth polling schemes: an analytical approach. *Mobile Networks and Applications*, 9:63–72, Feb. 2004.

[30] M. Perillo and W. Heinzelman. ASP: An Adaptive Energy Efficient Polling Algorithm for Bluetooth Piconets. *Proc. of the 36th Hawaii International Conference on System Sciences, Big Island, Hawaii*, 2003.

[31] J. Kim, J. Kim, Y. Lim, J. Yoon, S. L. Min, and J. S. Ma. A Bluetooth-based high-performance LAN access point incorporating multiple radio units. *Proc. of the Second International Conference on Communications in Computing (CIC' 2001), Las Vegas, NV*, June 2001.

[32] J. Kim, Y. Lim, S. L. Min, and J. S. Ma. Performance evaluation of the Bluetooth-based public Internet access point. *Proc. of the 15th International Conference on Networking (ICOIN-15), Beppu, Japan,*, Jan.. 2001.

[33] Joy Ghosh, Vivek Kumar, Xin Wang, and Chunming Qiao. BTSpin - Single Phase Distributed Bluetooth Scatternet Formation. Technical report, Department of Computer Science and Engineering, University at Buffalo, Dec. 2003.

[34] Computer Access Technology Corporation. *Merlin Protocol Analyzer User's Manual*, 1.8 edition, Dec. 2001. Available: www.catc.com/support.html, accesed Aug. 2004.

[35] Rohde&Schwarz. *Universal Radio Communication Tester R&S CMU200*, 06.00 edition, May 2004. Available www.rohde-schwarz.com/, accesed Aug 2004.

[36] NS2 Tutorial. www.isi.edu/nsnam/ns/tutorial/, Accessed Febuary, 2005.

[37] NS by Example. nile.wpi.edu/NS/, Accessed Febuary, 2005.

[38] Otcl and tclcl home. otcl-tclcl.sourceforge.net/, Accessed Febuary, 2005.

[39] Martin Leopold. Evaluation of bluetooth communication: Simulation and experiments. Technical Report 02-03, Dept. of Computer Science, University of Copenhagen, 2002.

[40] BlueHoc Web site. http://oss.software.ibm.com/bluehoc/, Accessed Febuary, 2005.

[41] Matlab Web site. www.mathworks.com, Accessed Febuary, 2005.

[42] Simulation Program Documentation. www.bluecenter.se, Accessed Febuary, 2005.

[43] connectBlue's Web site. www.connectblue.se, Accessed Febuary, 2005.

[44] Java Sun Web site. http://java.sun.com, Accessed Febuary, 2005.

# A Word List

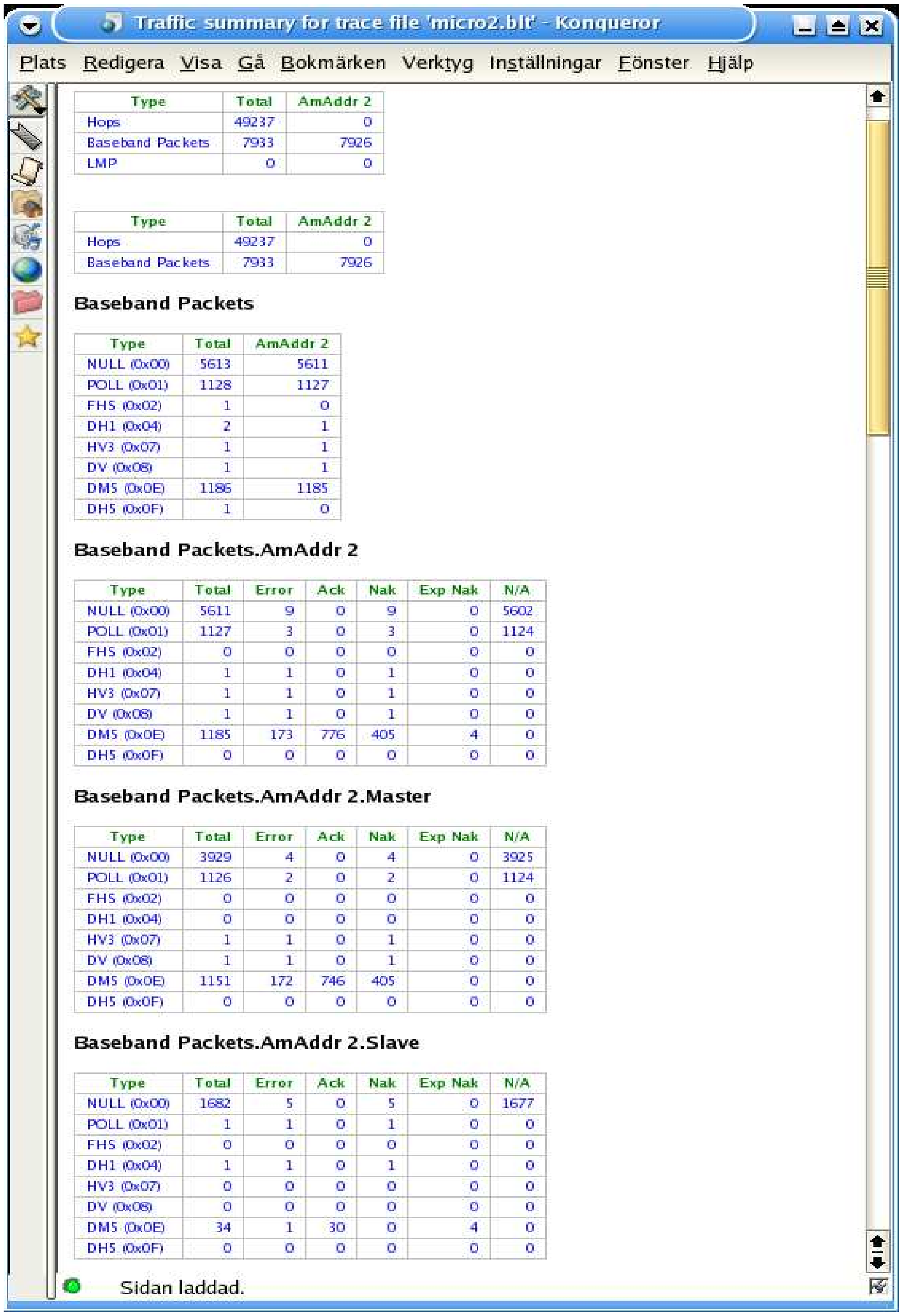
| | |
|---|---|
| *affected channels* | The Bluetooth channels that are affected by a source of interference. |
| *Asynchronous Connectionless (ACL)* | Bluetooth's service for data transmission. |
| *BlueCenter/IUC* | IUC is short for *Industrial Development Centre* which is the company in *Olofström* for which this thesis has been performed. *BlueCenter* is an apartment of *IUC* that performs researches and developments with Bluetooth. |
| *Bluetooth* | Protocol for short distance wireless communication. The data communication between the logger and the logger reader uses Bluetooth. |
| *Bluetooth data packets* | Bluetooth has six different data packets, see *DMx* and *DHx* |
| *buffer* | In this thesis, the memory of the *logger* i.e. the *DL141E*. The *DL141E* uses three buffer approaches: *immediate* were collected values are sent when they arrive without buffering, *k-buffering* where *k* values are stored in the buffer, and *full-buffering* where the buffer is filled before the values are transfered. |
| *CMU200* | A Bluetooth protocol analyser. |
| *connectBlue* | The company that delivered the Bluetooth module in the *DL141E*. |
| *DHx* | High rate Bluetooth packet. Could be DH1, DH3 or DH5 |
| *DL141E* | The name of the logger (developed by *BlueCenter/IUC*) that this thesis has focused on. |
| *diff-DPAF* | difference in *DPAF* between (often) a simulation and an experiment. See *DPAF*. |
| *DMx* | Forward Error Corrected (FEC) Bluetooth packet. Could be DM1, DM3 or DM5 |
| *DPAF* | Divergence expressed as Percentage At Frequency. Used in this thesis to describe difference in number of packets send in the different Bluetooth channels. |
| *Faraday's cage* | A RF-screened off cage for performing experiments with no unknown sources off interferences. |

| | |
|---|---|
| *FERRvsDist table* | The table (located in the *lossfile*) that specifyes the packet loss probability for a specific Bluetooth packet at a specific distance. |
| *FERRvsChan table* | The table (located in the *lossfile*) that specifyes the packet loss probability for a specific Bluetooth packet at a specific Bluetooth channel. |
| *Frequency hoping* | Bluetooth's strategy of avoiding interferences by hoping between 79 transfer channels. |
| *Industrial Development Centre* | The company in *Olofström* for which this thesis has been performed. See also *BlueCenter/IUC* |
| *interference Builder* | A Java dialog window based on experiments for constructing a *lossfile* for the *loss handler*. |
| *ISM* | Industrial, Scientific and Medical. The free band in which Bluetooth operates. |
| *Logger protocol* | An extension of *NS2-UCBT* for enable logger simulations. |
| *Logger Simulator* | A part of the simulation program performing logger simulations. |
| *Logger's reading interval* | The interval in seconds by which the logger starts sample and send. |
| *Loss Handler* | An extension of *NS2-UCBT* for handle losses of Bluetooth packets. |
| *maximum throughput* | Maximum achievable number of samples per seconds, i.e. 51657.1 samples/s for the *DL141E*. |
| *Merlin* | A Bluetooth traffic sniffer. |
| *Model* | In this thesis, *NS2-UCBT* with extensions and *Interference adder* for performing *Logger* simulations. |
| *Multitasking* | In this thesis the ability of the *logger* tosample and send at the same time. |
| *Network Simulator 2* | The simulation program used in this thesis for all simulations. |
| *NS2* | See *Network Simulator 2*. |
| *NS2-Agent* | The device connected to the *NS2-node* in this thesis often a logger agent or a logger sink agent. |
| *NS2-Node* | A node in the *NS2* simulation. In this thesis Bluetooth node are used. |

| | |
|---|---|
| *NS2-UCBT* | See *Network Simulator 2* and *UCBT*. |
| *Logger* | A device that collects samples stores them in its memory and sends them (in this thesis) wireless to e.g. a computer. See also *DL141E*. |
| *Logger Reader* | A program that reads the values from a logger. |
| *lossfile* | The file that specifies the packet loss probability for a specific distance, Bluetooth packet and channel. The file is used by the *loss Handler*. |
| *optimized logger* | The best available throughput with the current *DL141E* design. |
| *OTcl/Tcl* | The script language that one uses to construct simulations with *NS2*. |
| *packet loss probability* | The probability (in percent) that a Bluetooth packet is lost. |
| *packet trace* | The tracefile showing all transferred packets in a Bluetooth network. The *packet trace* are produced by *Merlin* and *Virtual Merlin* |
| *PaI Simulator* | A part of the simulation program that sends a Bluetooth packet at a specific interval. The *PaI Simulator* are used to evaluate the *Interference Builder*. |
| *piconet* | A Bluetooth master can connect from one up to seven slaves, forming a *piconet*. |
| *point-to-point* | A *piconet* with only two bluetooth devices. |
| *point-to-multipoint* | A *piconet* with multiple bluetooth devices. |
| *power classes* | Bluetooth supports three different power classes for different distances between the Bluetooth devices. |
| *rawdata mode* | A mode were the *DL141E* uses 16 bits per sampled value when transferring. |
| *scatternet* | A Bluetooth device acting as slave in one net and master in another can bridge packets from one piconet to another. These Bluetooth devices form a *scatternet* |
| *Simulation Builder* | Part of the simulation program that creates the *OTcl/Tcl* script to perform *NS2* simulations. |
| *Sources of Interference* | In this thesis, interferences that results in retransmissions of Bluetooth packets. |

| | |
|---|---|
| *Syncronous Connection Oriented (SCO)* | Bluetooth's service for real-time applications, especially voice. |
| *text mode* | A mode the DL141E sends its sampled values as text which leads to that up to 56 bits are used for transfer a sampled value. |
| *traffic summary* | The HTML file showing a summary of a Bluetooth network. The *traffic summary* are produced by *Merlin* and *Virtual Merlin* |
| *UCBT* | The Bluetooth extension of NS2 from the University of Cincinnati. |
| *Virtual Merlin* | An extension of *NS2-UCBT* that produces *Merlin* similar statistics in HTML format. |

## B Merlin Traffic Summary


Traffic summary for trace file 'micro2.blt' - Konqueror
Plats Redigera Visa Gå Bokmärken Verktyg Inställningar Fönster Hjälp

| Type | Total | AmAddr 2 |
|---|---|---|
| Hops | 49237 | 0 |
| Baseband Packets | 7933 | 7926 |
| LMP | 0 | 0 |

| Type | Total | AmAddr 2 |
|---|---|---|
| Hops | 49237 | 0 |
| Baseband Packets | 7933 | 7926 |

**Baseband Packets**

| Type | Total | AmAddr 2 |
|---|---|---|
| NULL (0x00) | 5613 | 5611 |
| POLL (0x01) | 1128 | 1127 |
| FHS (0x02) | 1 | 0 |
| DH1 (0x04) | 2 | 1 |
| HV3 (0x07) | 1 | 1 |
| DV (0x08) | 1 | 1 |
| DM5 (0x0E) | 1186 | 1185 |
| DH5 (0x0F) | 1 | 0 |

**Baseband Packets.AmAddr 2**

| Type | Total | Error | Ack | Nak | Exp Nak | N/A |
|---|---|---|---|---|---|---|
| NULL (0x00) | 5611 | 9 | 0 | 9 | 0 | 5602 |
| POLL (0x01) | 1127 | 3 | 0 | 3 | 0 | 1124 |
| FHS (0x02) | 0 | 0 | 0 | 0 | 0 | 0 |
| DH1 (0x04) | 1 | 1 | 0 | 1 | 0 | 0 |
| HV3 (0x07) | 1 | 1 | 0 | 1 | 0 | 0 |
| DV (0x08) | 1 | 1 | 0 | 1 | 0 | 0 |
| DM5 (0x0E) | 1185 | 173 | 776 | 405 | 4 | 0 |
| DH5 (0x0F) | 0 | 0 | 0 | 0 | 0 | 0 |

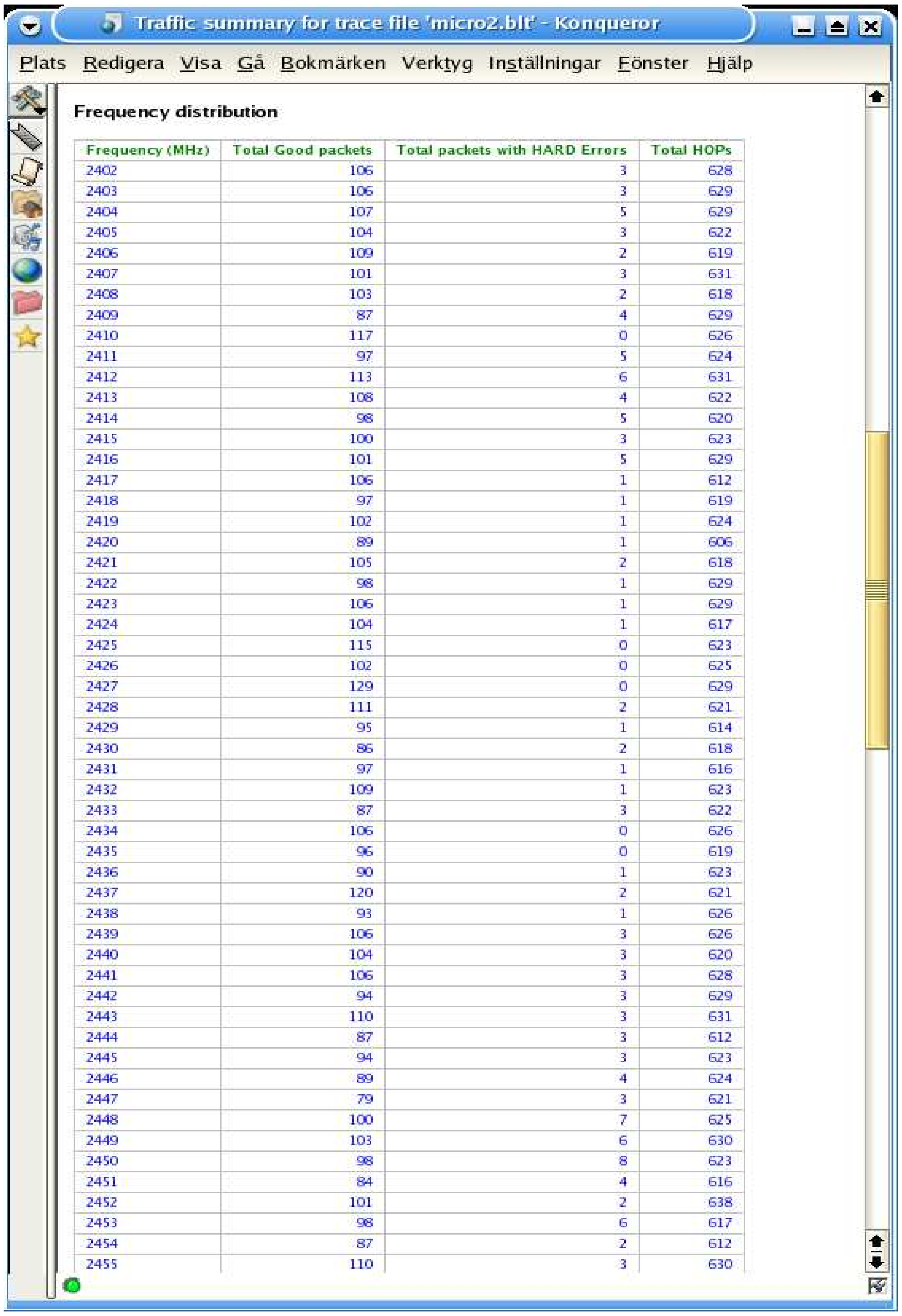
**Baseband Packets.AmAddr 2.Master**

| Type | Total | Error | Ack | Nak | Exp Nak | N/A |
|---|---|---|---|---|---|---|
| NULL (0x00) | 3929 | 4 | 0 | 4 | 0 | 3925 |
| POLL (0x01) | 1126 | 2 | 0 | 2 | 0 | 1124 |
| FHS (0x02) | 0 | 0 | 0 | 0 | 0 | 0 |
| DH1 (0x04) | 0 | 0 | 0 | 0 | 0 | 0 |
| HV3 (0x07) | 1 | 1 | 0 | 1 | 0 | 0 |
| DV (0x08) | 1 | 1 | 0 | 1 | 0 | 0 |
| DM5 (0x0E) | 1151 | 172 | 746 | 405 | 0 | 0 |
| DH5 (0x0F) | 0 | 0 | 0 | 0 | 0 | 0 |

**Baseband Packets.AmAddr 2.Slave**

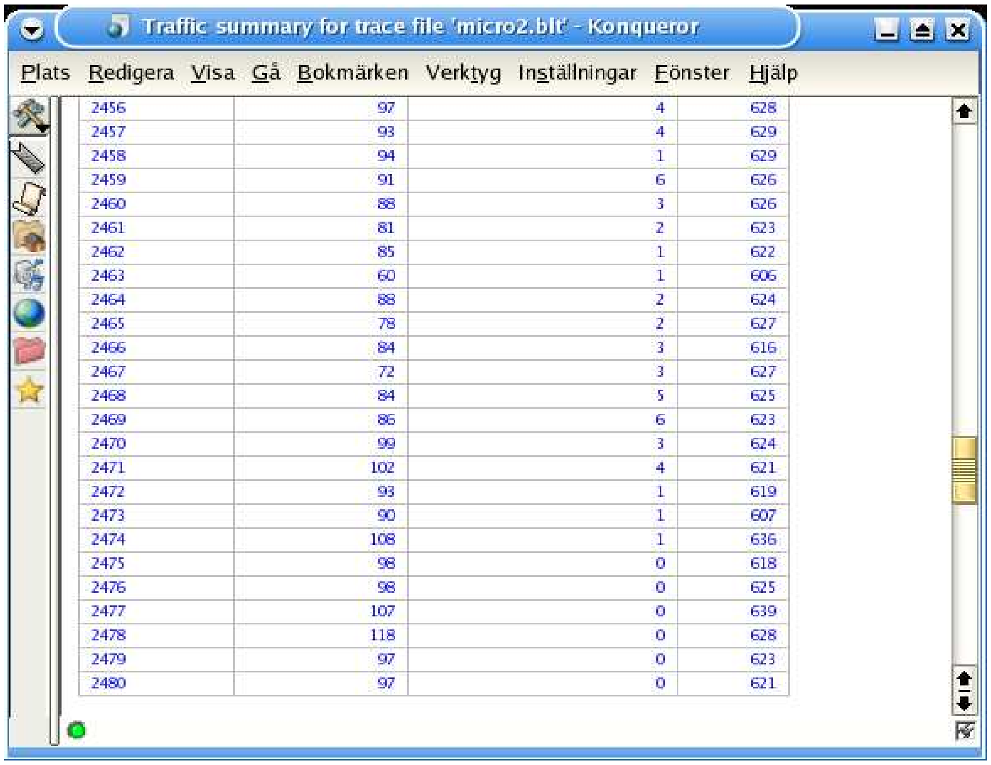
| Type | Total | Error | Ack | Nak | Exp Nak | N/A |
|---|---|---|---|---|---|---|
| NULL (0x00) | 1682 | 5 | 0 | 5 | 0 | 1677 |
| POLL (0x01) | 1 | 1 | 0 | 1 | 0 | 0 |
| FHS (0x02) | 0 | 0 | 0 | 0 | 0 | 0 |
| DH1 (0x04) | 1 | 1 | 0 | 1 | 0 | 0 |
| HV3 (0x07) | 0 | 0 | 0 | 0 | 0 | 0 |
| DV (0x08) | 0 | 0 | 0 | 0 | 0 | 0 |
| DM5 (0x0E) | 34 | 1 | 30 | 0 | 4 | 0 |
| DH5 (0x0F) | 0 | 0 | 0 | 0 | 0 | 0 |

Sidan laddad.

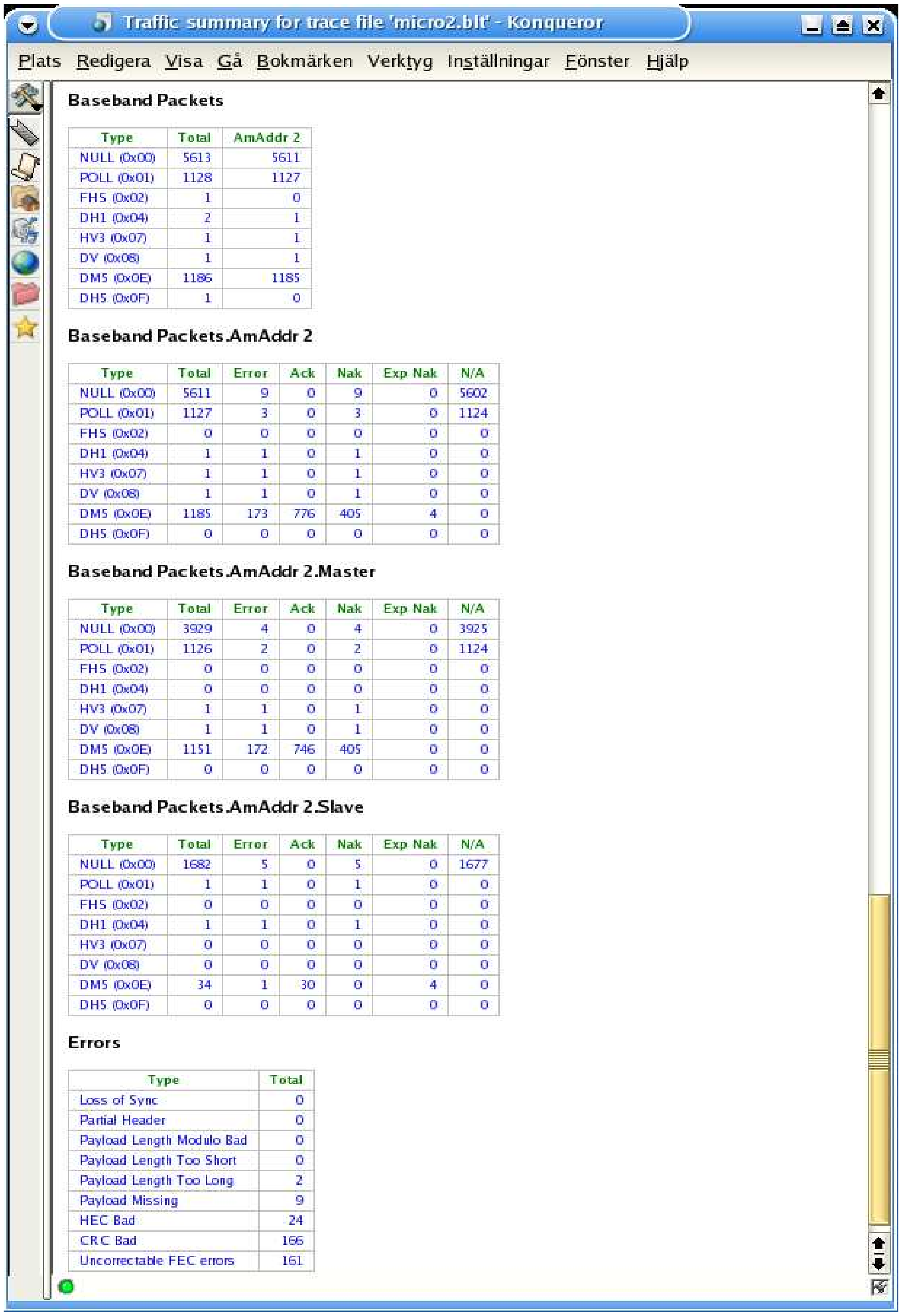

Traffic summary for trace file 'micro2.blt' - Konqueror
Plats Redigera Visa Gå Bokmärken Verktyg Inställningar Fönster Hjälp

Frequency distribution

| Frequency (MHz) | Total Good packets | Total packets with HARD Errors | Total HOPs |
|---|---|---|---|
| 2402 | 106 | 3 | 628 |
| 2403 | 106 | 3 | 629 |
| 2404 | 107 | 5 | 629 |
| 2405 | 104 | 3 | 622 |
| 2406 | 109 | 2 | 619 |
| 2407 | 101 | 3 | 631 |
| 2408 | 103 | 2 | 618 |
| 2409 | 87 | 4 | 629 |
| 2410 | 117 | 0 | 626 |
| 2411 | 97 | 5 | 624 |
| 2412 | 113 | 6 | 631 |
| 2413 | 108 | 4 | 622 |
| 2414 | 98 | 5 | 620 |
| 2415 | 100 | 3 | 623 |
| 2416 | 101 | 5 | 629 |
| 2417 | 106 | 1 | 612 |
| 2418 | 97 | 1 | 619 |
| 2419 | 102 | 1 | 624 |
| 2420 | 89 | 1 | 606 |
| 2421 | 105 | 2 | 618 |
| 2422 | 98 | 1 | 629 |
| 2423 | 106 | 1 | 629 |
| 2424 | 104 | 1 | 617 |
| 2425 | 115 | 0 | 623 |
| 2426 | 102 | 0 | 625 |
| 2427 | 129 | 0 | 629 |
| 2428 | 111 | 2 | 621 |
| 2429 | 95 | 1 | 614 |
| 2430 | 86 | 2 | 618 |
| 2431 | 97 | 1 | 616 |
| 2432 | 109 | 1 | 623 |
| 2433 | 87 | 3 | 622 |
| 2434 | 106 | 0 | 626 |
| 2435 | 96 | 0 | 619 |
| 2436 | 90 | 1 | 623 |
| 2437 | 120 | 2 | 621 |
| 2438 | 93 | 1 | 626 |
| 2439 | 106 | 3 | 626 |
| 2440 | 104 | 3 | 620 |
| 2441 | 106 | 3 | 628 |
| 2442 | 94 | 3 | 629 |
| 2443 | 110 | 3 | 631 |
| 2444 | 87 | 3 | 612 |
| 2445 | 94 | 3 | 623 |
| 2446 | 89 | 4 | 624 |
| 2447 | 79 | 3 | 621 |
| 2448 | 100 | 7 | 625 |
| 2449 | 103 | 6 | 630 |
| 2450 | 98 | 8 | 623 |
| 2451 | 84 | 4 | 616 |
| 2452 | 101 | 2 | 638 |
| 2453 | 98 | 6 | 617 |
| 2454 | 87 | 2 | 612 |
| 2455 | 110 | 3 | 630 |


Traffic summary for trace file 'micro2.blt' - Konqueror
Plats Redigera Visa Gå Bokmärken Verktyg Inställningar Fönster Hjälp

| | | | |
|---|---|---|---|
| 2456 | 97 | 4 | 628 |
| 2457 | 93 | 4 | 629 |
| 2458 | 94 | 1 | 629 |
| 2459 | 91 | 6 | 626 |
| 2460 | 88 | 3 | 626 |
| 2461 | 81 | 2 | 623 |
| 2462 | 85 | 1 | 622 |
| 2463 | 60 | 1 | 606 |
| 2464 | 88 | 2 | 624 |
| 2465 | 78 | 2 | 627 |
| 2466 | 84 | 3 | 616 |
| 2467 | 72 | 3 | 627 |
| 2468 | 84 | 5 | 625 |
| 2469 | 86 | 6 | 623 |
| 2470 | 99 | 3 | 624 |
| 2471 | 102 | 4 | 621 |
| 2472 | 93 | 1 | 619 |
| 2473 | 90 | 1 | 607 |
| 2474 | 108 | 1 | 636 |
| 2475 | 98 | 0 | 618 |
| 2476 | 98 | 0 | 625 |
| 2477 | 107 | 0 | 639 |
| 2478 | 118 | 0 | 628 |
| 2479 | 97 | 0 | 623 |
| 2480 | 97 | 0 | 621 |


Traffic summary for trace file 'micro2.blt' - Konqueror
Plats Redigera Visa Gå Bokmärken Verktyg Inställningar Fönster Hjälp

**Baseband Packets**

| Type | Total | AmAddr 2 |
|---|---|---|
| NULL (0x00) | 5613 | 5611 |
| POLL (0x01) | 1128 | 1127 |
| FHS (0x02) | 1 | 0 |
| DH1 (0x04) | 2 | 1 |
| HV3 (0x07) | 1 | 1 |
| DV (0x08) | 1 | 1 |
| DM5 (0x0E) | 1186 | 1185 |
| DH5 (0x0F) | 1 | 0 |

**Baseband Packets.AmAddr 2**

| Type | Total | Error | Ack | Nak | Exp Nak | N/A |
|---|---|---|---|---|---|---|
| NULL (0x00) | 5611 | 9 | 0 | 9 | 0 | 5602 |
| POLL (0x01) | 1127 | 3 | 0 | 3 | 0 | 1124 |
| FHS (0x02) | 0 | 0 | 0 | 0 | 0 | 0 |
| DH1 (0x04) | 1 | 1 | 0 | 1 | 0 | 0 |
| HV3 (0x07) | 1 | 1 | 0 | 1 | 0 | 0 |
| DV (0x08) | 1 | 1 | 0 | 1 | 0 | 0 |
| DM5 (0x0E) | 1185 | 173 | 776 | 405 | 4 | 0 |
| DH5 (0x0F) | 0 | 0 | 0 | 0 | 0 | 0 |

**Baseband Packets.AmAddr 2.Master**

| Type | Total | Error | Ack | Nak | Exp Nak | N/A |
|---|---|---|---|---|---|---|
| NULL (0x00) | 3929 | 4 | 0 | 4 | 0 | 3925 |
| POLL (0x01) | 1126 | 2 | 0 | 2 | 0 | 1124 |
| FHS (0x02) | 0 | 0 | 0 | 0 | 0 | 0 |
| DH1 (0x04) | 0 | 0 | 0 | 0 | 0 | 0 |
| HV3 (0x07) | 1 | 1 | 0 | 1 | 0 | 0 |
| DV (0x08) | 1 | 1 | 0 | 1 | 0 | 0 |
| DM5 (0x0E) | 1151 | 172 | 746 | 405 | 0 | 0 |
| DH5 (0x0F) | 0 | 0 | 0 | 0 | 0 | 0 |

**Baseband Packets.AmAddr 2.Slave**

| Type | Total | Error | Ack | Nak | Exp Nak | N/A |
|---|---|---|---|---|---|---|
| NULL (0x00) | 1682 | 5 | 0 | 5 | 0 | 1677 |
| POLL (0x01) | 1 | 1 | 0 | 1 | 0 | 0 |
| FHS (0x02) | 0 | 0 | 0 | 0 | 0 | 0 |
| DH1 (0x04) | 1 | 1 | 0 | 1 | 0 | 0 |
| HV3 (0x07) | 0 | 0 | 0 | 0 | 0 | 0 |
| DV (0x08) | 0 | 0 | 0 | 0 | 0 | 0 |
| DM5 (0x0E) | 34 | 1 | 30 | 0 | 4 | 0 |
| DH5 (0x0F) | 0 | 0 | 0 | 0 | 0 | 0 |

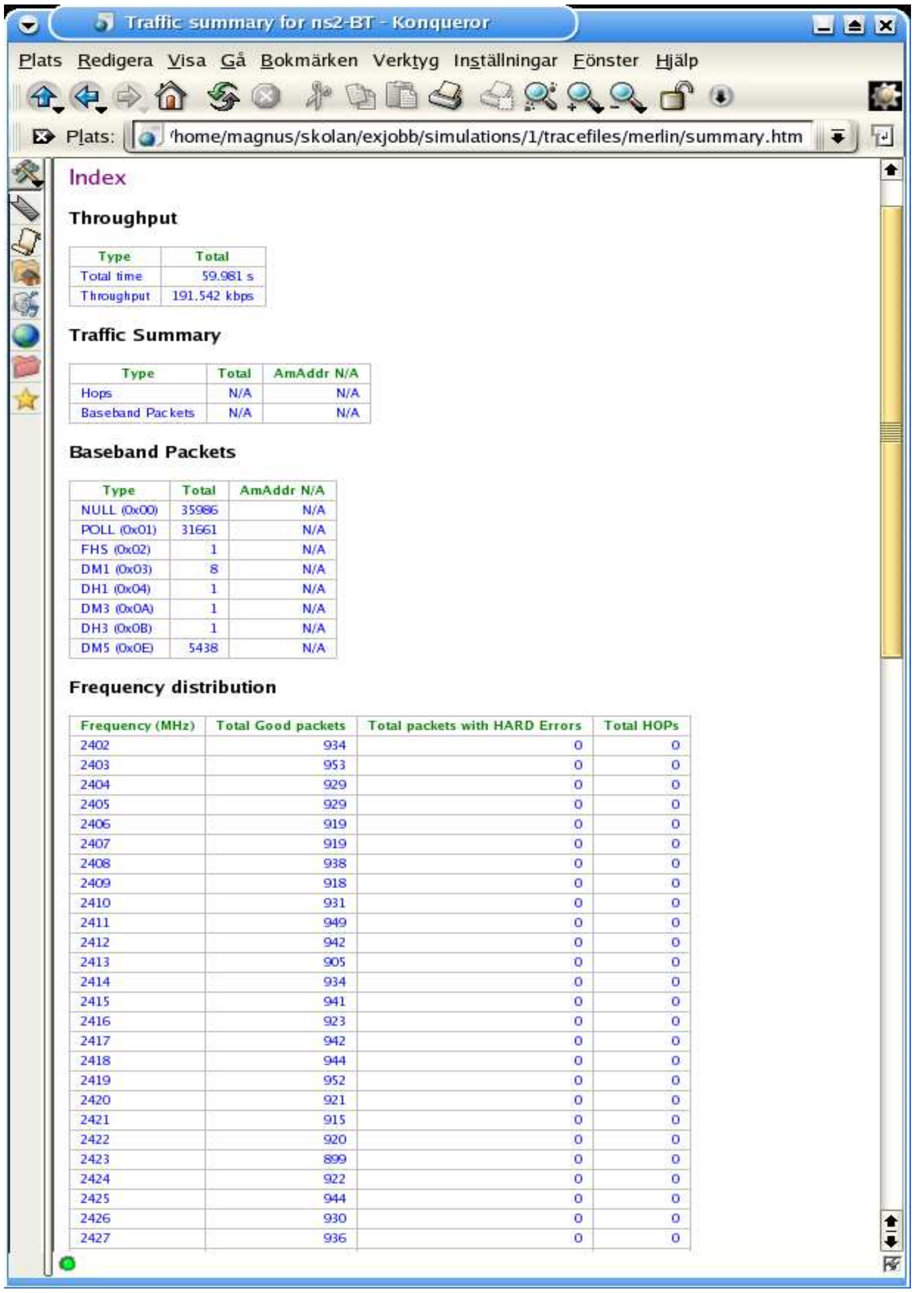
**Errors**

| Type | Total |
|---|---|
| Loss of Sync | 0 |
| Partial Header | 0 |
| Payload Length Modulo Bad | 0 |
| Payload Length Too Short | 0 |
| Payload Length Too Long | 2 |
| Payload Missing | 9 |
| HEC Bad | 24 |
| CRC Bad | 166 |
| Uncorrectable FEC errors | 161 |

# C Virtual Merlin Example

Examples of outputs generated by the Virtual Merlin sniffer. Note that neither the traffic summary nor the packet trace is as detailed as the actual Merlin traces.

## C.1 Traffic Summary


Traffic summary for ns2-BT - Konqueror
Plats Redigera Visa Gå Bokmärken Verktyg Inställningar Fönster Hjälp
Plats: /home/magnus/skolan/exjobb/simulations/1/tracefiles/merlin/summary.htm

Index

Throughput

| Type | Total |
|---|---|
| Total time | 59.981 s |
| Throughput | 191.542 kbps |

Traffic Summary

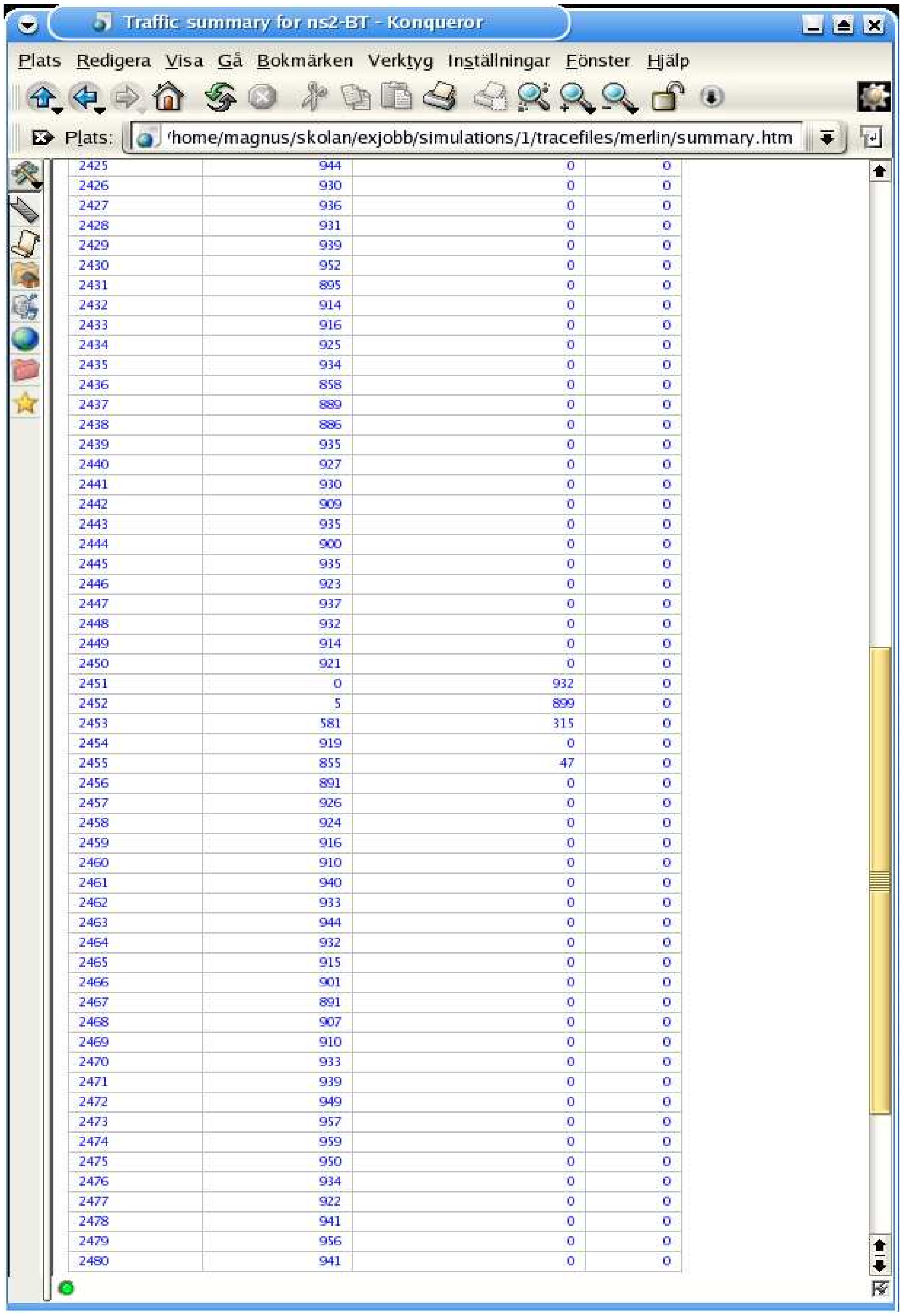
| Type | Total | AmAddr N/A |
|---|---|---|
| Hops | N/A | N/A |
| Baseband Packets | N/A | N/A |

Baseband Packets

| Type | Total | AmAddr N/A |
|---|---|---|
| NULL (0x00) | 35986 | N/A |
| POLL (0x01) | 31661 | N/A |
| FHS (0x02) | 1 | N/A |
| DM1 (0x03) | 8 | N/A |
| DH1 (0x04) | 1 | N/A |
| DM3 (0x0A) | 1 | N/A |
| DH3 (0x0B) | 1 | N/A |
| DM5 (0x0E) | 5438 | N/A |

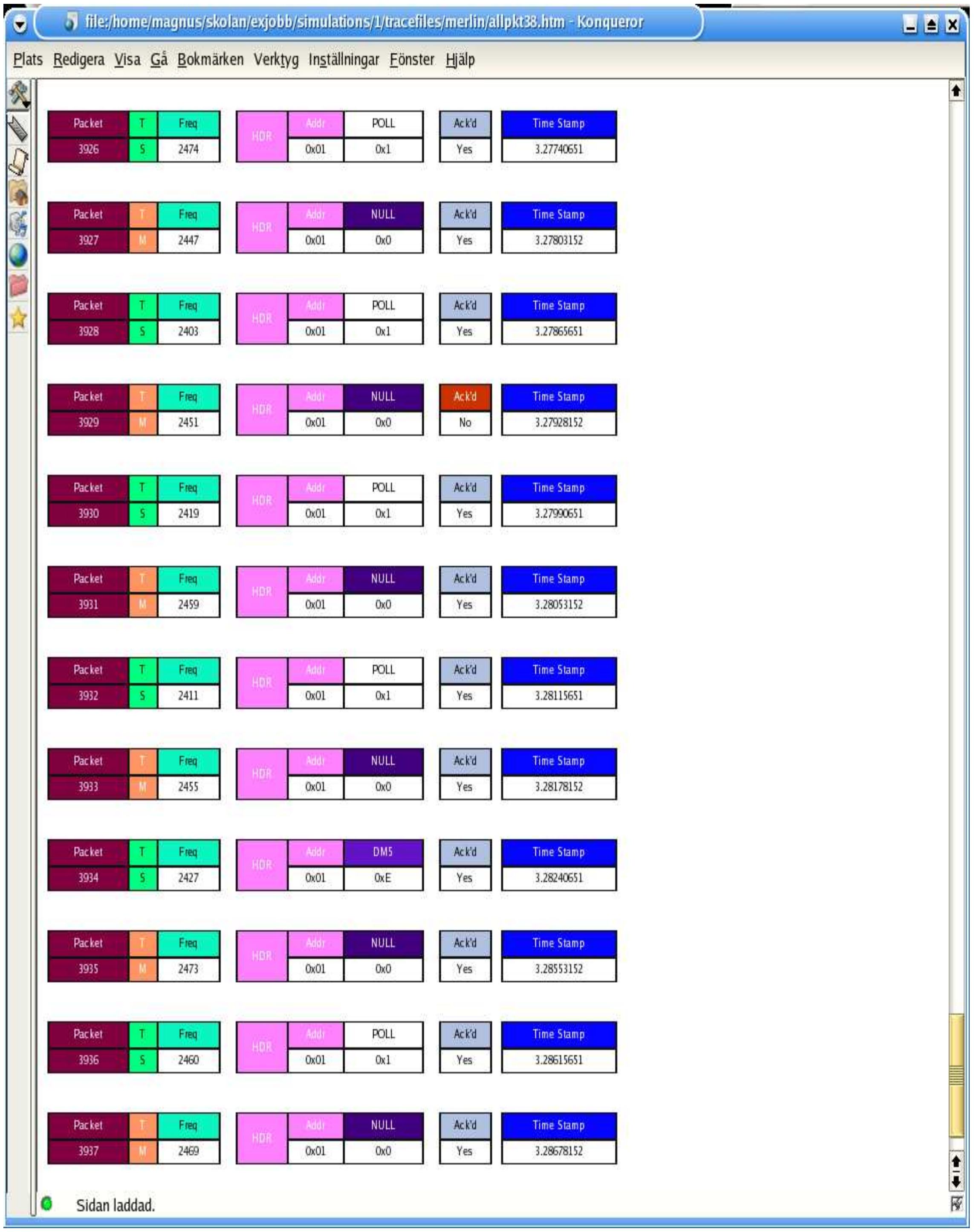
Frequency distribution

| Frequency (MHz) | Total Good packets | Total packets with HARD Errors | Total HOPs |
|---|---|---|---|
| 2402 | 934 | 0 | 0 |
| 2403 | 953 | 0 | 0 |
| 2404 | 929 | 0 | 0 |
| 2405 | 929 | 0 | 0 |
| 2406 | 919 | 0 | 0 |
| 2407 | 919 | 0 | 0 |
| 2408 | 938 | 0 | 0 |
| 2409 | 918 | 0 | 0 |
| 2410 | 931 | 0 | 0 |
| 2411 | 949 | 0 | 0 |
| 2412 | 942 | 0 | 0 |
| 2413 | 905 | 0 | 0 |
| 2414 | 934 | 0 | 0 |
| 2415 | 941 | 0 | 0 |
| 2416 | 923 | 0 | 0 |
| 2417 | 942 | 0 | 0 |
| 2418 | 944 | 0 | 0 |
| 2419 | 952 | 0 | 0 |
| 2420 | 921 | 0 | 0 |
| 2421 | 915 | 0 | 0 |
| 2422 | 920 | 0 | 0 |
| 2423 | 899 | 0 | 0 |
| 2424 | 922 | 0 | 0 |
| 2425 | 944 | 0 | 0 |
| 2426 | 930 | 0 | 0 |
| 2427 | 936 | 0 | 0 |


Traffic summary for ns2-BT - Konqueror
Plats Redigera Visa Gå Bokmärken Verktyg Inställningar Fönster Hjälp
Plats: 'home/magnus/skolan/exjobb/simulations/1/tracefiles/merlin/summary.htm

| | | | |
|---|---|---|---|
| 2425 | 944 | 0 | 0 |
| 2426 | 930 | 0 | 0 |
| 2427 | 936 | 0 | 0 |
| 2428 | 931 | 0 | 0 |
| 2429 | 939 | 0 | 0 |
| 2430 | 952 | 0 | 0 |
| 2431 | 895 | 0 | 0 |
| 2432 | 914 | 0 | 0 |
| 2433 | 916 | 0 | 0 |
| 2434 | 925 | 0 | 0 |
| 2435 | 934 | 0 | 0 |
| 2436 | 858 | 0 | 0 |
| 2437 | 889 | 0 | 0 |
| 2438 | 886 | 0 | 0 |
| 2439 | 935 | 0 | 0 |
| 2440 | 927 | 0 | 0 |
| 2441 | 930 | 0 | 0 |
| 2442 | 909 | 0 | 0 |
| 2443 | 935 | 0 | 0 |
| 2444 | 900 | 0 | 0 |
| 2445 | 935 | 0 | 0 |
| 2446 | 923 | 0 | 0 |
| 2447 | 937 | 0 | 0 |
| 2448 | 932 | 0 | 0 |
| 2449 | 914 | 0 | 0 |
| 2450 | 921 | 0 | 0 |
| 2451 | 0 | 932 | 0 |
| 2452 | 5 | 899 | 0 |
| 2453 | 581 | 315 | 0 |
| 2454 | 919 | 0 | 0 |
| 2455 | 855 | 47 | 0 |
| 2456 | 891 | 0 | 0 |
| 2457 | 926 | 0 | 0 |
| 2458 | 924 | 0 | 0 |
| 2459 | 916 | 0 | 0 |
| 2460 | 910 | 0 | 0 |
| 2461 | 940 | 0 | 0 |
| 2462 | 933 | 0 | 0 |
| 2463 | 944 | 0 | 0 |
| 2464 | 932 | 0 | 0 |
| 2465 | 915 | 0 | 0 |
| 2466 | 901 | 0 | 0 |
| 2467 | 891 | 0 | 0 |
| 2468 | 907 | 0 | 0 |
| 2469 | 910 | 0 | 0 |
| 2470 | 933 | 0 | 0 |
| 2471 | 939 | 0 | 0 |
| 2472 | 949 | 0 | 0 |
| 2473 | 957 | 0 | 0 |
| 2474 | 959 | 0 | 0 |
| 2475 | 950 | 0 | 0 |
| 2476 | 934 | 0 | 0 |
| 2477 | 922 | 0 | 0 |
| 2478 | 941 | 0 | 0 |
| 2479 | 956 | 0 | 0 |
| 2480 | 941 | 0 | 0 |

## C.2 Packet Trace


file:/home/magnus/skolan/exjobb/simulations/1/tracefiles/merlin/allpkt38.htm - Konqueror
Plats Redigera Visa Gå Bokmärken Verktyg Inställningar Fönster Hjälp

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Packet | T | Freq | HDR | Addr | POLL | Ack'd | Time Stamp |
| 3926 | S | 2474 | | 0x01 | 0x1 | Yes | 3.27740651 |
| Packet | T | Freq | HDR | Addr | NULL | Ack'd | Time Stamp |
| 3927 | M | 2447 | | 0x01 | 0x0 | Yes | 3.27803152 |
| Packet | T | Freq | HDR | Addr | POLL | Ack'd | Time Stamp |
| 3928 | S | 2403 | | 0x01 | 0x1 | Yes | 3.27865651 |
| Packet | T | Freq | HDR | Addr | NULL | Ack'd | Time Stamp |
| 3929 | M | 2451 | | 0x01 | 0x0 | No | 3.27928152 |
| Packet | T | Freq | HDR | Addr | POLL | Ack'd | Time Stamp |
| 3930 | S | 2419 | | 0x01 | 0x1 | Yes | 3.27990651 |
| Packet | T | Freq | HDR | Addr | NULL | Ack'd | Time Stamp |
| 3931 | M | 2459 | | 0x01 | 0x0 | Yes | 3.28053152 |
| Packet | T | Freq | HDR | Addr | POLL | Ack'd | Time Stamp |
| 3932 | S | 2411 | | 0x01 | 0x1 | Yes | 3.28115651 |
| Packet | T | Freq | HDR | Addr | NULL | Ack'd | Time Stamp |
| 3933 | M | 2455 | | 0x01 | 0x0 | Yes | 3.28178152 |
| Packet | T | Freq | HDR | Addr | DM5 | Ack'd | Time Stamp |
| 3934 | S | 2427 | | 0x01 | 0xE | Yes | 3.28240651 |
| Packet | T | Freq | HDR | Addr | NULL | Ack'd | Time Stamp |
| 3935 | M | 2473 | | 0x01 | 0x0 | Yes | 3.28553152 |
| Packet | T | Freq | HDR | Addr | POLL | Ack'd | Time Stamp |
| 3936 | S | 2460 | | 0x01 | 0x1 | Yes | 3.28615651 |
| Packet | T | Freq | HDR | Addr | NULL | Ack'd | Time Stamp |
| 3937 | M | 2469 | | 0x01 | 0x0 | Yes | 3.28678152 |

Sidan laddad.

# D Logger Protocol

## D.1 Logger Agent code

### D.1.1 logger.h

```
/*
 * File: Header File for a new 'Logger' Agent Class for the ns
 *       network simulator (with UCBT extensions)
 * Author: Magnus Karlsson (mk11bq@telia.com), January 2005
 *
 */


#ifndef NS_LOGGER_H
#define NS_LOGGER_H

#include "timer-handler.h"
#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"

// Header for the packets send by the Logger
struct hdr_logger {
  int size;        // Packet size
  bool ok;         // Is this a packet containing OK?
  bool change;     // Should som settings be changed
  char command;    // Which settings?
  int value;       // New value

  static int offset_;    // offset for this header
  inline static int& offset() { return offset_; }
  inline static hdr_logger* access(Packet* p) {
    return (hdr_logger*) p->access(offset_);
  }
};

class LoggerAgent;


// Sender uses this timer to
// schedule next logger send time
class SendTimer : public TimerHandler {
 public:
        SendTimer(LoggerAgent* t) : TimerHandler(), t_(t) {}
        inline virtual void expire(Event*);
 protected:
        LoggerAgent* t_;
};

// Sender uses this timer to
// schedule next logger read time
class ReadTimer : public TimerHandler {
 public:
        ReadTimer(LoggerAgent* t) : TimerHandler(), t_(t) {}
        inline virtual void expire(Event*);
 protected:
        LoggerAgent* t_;
};

// The logger agent class. Build to simulate the DL141E logger
// from Bluecenter IUC, öOlofstrm.
class LoggerAgent : public Agent {
 public:
  LoggerAgent();
  void send_pkt();  // called by SendTimer:expire (Sender)
  void read_logger(); // called by readTimer:expire
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*); // Receivs packets from the logger-sink
 protected:
  // Agent/Logger sampleNumber_  (tcl variable)
  int nsamp_;       // Number of samples
  // Agent/Logger sampleSize_  (tcl variable)
  int sampSize_;    // Size of the samples 16 or 48(expected) for DL141E
  // Agent/Logger samplingRate_  (tcl variable)
  int sampFrq_;     // Sampling frequency
  // Agent/Logger samplingPeriods_  (tcl variable)
  int rint_;        // Reading interval of the logger
  // Agent/Logger baudRate_  (tcl variable)
  int baud_;        // Baudrate for the RS232 interface
  // Agent/Logger timeout_  (tcl variable)
  double timeout_; // Timeout for the RS232 interface
 private:
  SendTimer snd_timer_; // Send timer
  ReadTimer rd_timer_;  // Read&send timer
  bool sending;     // Is logger sending
  bool stop;        // Is logger running

  double oldtime;
  int bufferSize;
};


#endif
```

### D.1.2 logger.cc

```
/*
 * File: Code for a new 'Logger' Agent Class for the ns
 *       network simulator (with UCBT extensions)
 * Author: Magnus Karlsson (mk11bq@telia.com), January 2005
 *
 */


#include "logger.h"
#include "logger-sink.h"
#include <math.h>

int hdr_logger::offset_;

static class LoggerHeaderClass : public PacketHeaderClass {
public:
  LoggerHeaderClass() : PacketHeaderClass("PacketHeader/Logger",
                                          sizeof(hdr_logger)) {
                bind_offset(&hdr_logger::offset_);
  }
} class_loggerhdr;

static class LoggerClass : public TclClass {
public:
  LoggerClass() : TclClass("Agent/Logger") {}
  TclObject* create(int, const char*const*) {
    return (new LoggerAgent());
  }
} class_logger;

// When snd_timer_ expires call LoggerAgent:send_pkt()
void SendTimer::expire(Event*)
{
  t_->send_pkt();
}

// When rd_timer_ expires call LoggerAgent:read_logger()
void ReadTimer::expire(Event*)
{
  t_->read_logger();
}

// LoggerAgent constructor
LoggerAgent::LoggerAgent() : Agent(PT_LOGGER), snd_timer_(this),
                             rd_timer_(this)
{
  sending = false;
  stop = true;
  bind("sampleNumber_",&nsamp_);
  bind("sampleSize_",&sampSize_);
  bind("samplingRate_",&sampFrq_);
  bind("baudRate_",&baud_);
  bind("samplingPeriods_",&rint_);
  bind("timeout_",&timeout_);
}

// Execute tcl commands
int LoggerAgent::command(int argc, const char*const* argv)
{
  if(argc == 2) {
    if(strcmp(argv[1], "start") == 0) {
      if(stop==true) {
        stop=false;
        read_logger();
      }
      return (TCL_OK);
    }
    else if(strcmp(argv[1], "read") == 0) {
      if(stop==true) {
        stop=false;

        // Check if the logger is sending
        if(sending == true)
          return (TCL_OK);


        sending = true;

        double ts = (double)1/sampFrq_;
        snd_timer_.resched(ts*(double)nsamp_+timeout_);
        bufferSize=(int)ceil((double)(nsamp_*sampSize_)/8);
        bufferSize+=5; // "DATA:"
        oldtime = Scheduler::instance().clock()+ts*(double)nsamp_;
      }
      return (TCL_OK);
    }
    else if(strcmp(argv[1], "stop") == 0) {
      stop=true;
      return (TCL_OK);
    }
  }
  else if(argc == 4) {
     if(strcmp(argv[2], "sample_size") == 0) {
        if(stop==true)
          sampSize_=(int)round(atof(argv[3]));
        return (TCL_OK);
      }
      else if(strcmp(argv[2], "baud_rate") == 0) {
        if(stop==true)
baud_=(int)round(atof(argv[3]));
        return (TCL_OK);
      }
      else if(strcmp(argv[2], "timeout") == 0) {
        if(stop==true)
          timeout_=(int)round(atof(argv[3]));
        return (TCL_OK);
      }
  }
  // If the command hasn't been processed by LoggerAgent()::command,
  // call the command() function for the base class
  return (Agent::command(argc, argv));
}

// Receives responds from the LoggerSink
void LoggerAgent::recv(Packet* pkt, Handler*)
{
  hdr_loggersink* hdr = hdr_loggersink::access(pkt);
  if(hdr->change==true) {
    stop=true;
    if(hdr->command=='a')
      sampFrq_ = hdr->value;
    else if(hdr->command=='b')
      nsamp_ = hdr->value;
    else if(hdr->command=='h')
      rint_ = hdr->value;
    Packet* pktret = allocpkt();
    hdr_logger* hdr_l =  hdr_logger::access(pktret);
    hdr_l->ok       = true;
    hdr_l->change   = true;
    hdr_l->command  = hdr->command;
    hdr_l->value    = hdr->value;
    size_ = 224; //Force to use DM5
    send(pktret,0);
  }
  else if(hdr->okReceived==true) {
    sending=false;
    if(rint_==0)
      rd_timer_.resched(0.00375);
  }
  else {
    // Reschedules
    double time = Scheduler::instance().clock();
    if((time-oldtime)>timeout_)
      snd_timer_.resched(time-oldtime);
    else
      snd_timer_.resched(timeout_);
  }

  Packet::free(pkt);
}

// Sends the Buffer
void LoggerAgent::send_pkt()
{
  double time;
  int sendSize, numDM5Pkt;

  if(stop==true) {
    sending = false;
    return;
  }

  if(bufferSize>0) {
    time = Scheduler::instance().clock();

    sendSize = (int)ceil(timeout_*baud_/8);
    if(bufferSize<sendSize)
      sendSize = bufferSize;

    bufferSize-=sendSize;

    oldtime = time;

    // Forces UCBT to use the same packes as the DL141E
    numDM5Pkt = (int)ceil((double)(sendSize+9)/224);
    size_ = 224*numDM5Pkt+(numDM5Pkt-1)*40+13;

    // Create a new packet
    Packet* pkt = allocpkt();
    // Access the Ping header for the new packet:
    hdr_logger* hdr =  hdr_logger::access(pkt);
    // Set header values
    hdr->size    = sendSize;
    hdr->ok      = false;
    hdr->change = false;


    // Send the packet
    send(pkt, 0);


  }
  else { // Sends the OK packet.
    // Create a new packet
    Packet* pkt = allocpkt();
    // Access the Ping header for the new packet:
    hdr_logger* hdr =  hdr_logger::access(pkt);
    size_ = 224;
hdr->size = 11;
    hdr->ok      = true;
    hdr->change = false;
    send(pkt, 0);
  }
}


// Schedule to sample&send each intervall if the logger
// is done sending and if it is not stopped.
void LoggerAgent::read_logger() {
  // Reschedules next interval
  if(stop==false && rint_>0)
    rd_timer_.resched(rint_);

  // Check if the logger is sending
  if(sending == true)
    return;

  sending = true;

  double ts = (double)1/sampFrq_;
  snd_timer_.resched(ts*(double)nsamp_+timeout_);
  bufferSize=(int)ceil((double)(nsamp_*sampSize_)/8);
  oldtime = Scheduler::instance().clock()+ts*(double)nsamp_;
}
```

## D.2 Logger SinkAgent code

### D.2.1 logger-sink.h

```
/*
 * File: Header File for a new 'Logger' Sink Agent Class for the ns
 *       network simulator (with UCBT extensions)
 * Author: Magnus Karlsson (mk11bq@telia.com), January 2005
 *
 */


#ifndef NS_LOGGER_SINK_H
#define NS_LOGGER_SINK_H

#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"

// The header for the LoggerSinkAgent packet.
struct hdr_loggersink {
  bool okReceived;  // Is OK packet received

  bool change;       // Is change packet received
  char command;      // Command for change packet
  int value;         // New value

  static int offset_;    // offset for this header
  inline static int& offset() { return offset_; }
  inline static hdr_loggersink* access(Packet* p) {
    return (hdr_loggersink*) p->access(offset_);
  }
};

class LoggerSinkAgent : public Agent {
 public:
  LoggerSinkAgent();
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
 private:
  int nbRecPkt;    // Numbre of received packets
  bool first;
};


#endif
```

### D.2.2 logger-sink.cc

```
/*
 * File: Code for a new 'Logger' Sink Agent Class for the ns
 *       network simulator (with UCBT extensions)
 * Author: Magnus Karlsson (mk11bq@telia.com), January 2005
 *
 */


#include "logger-sink.h"
#include "logger.h"
#include <math.h>

int hdr_loggersink::offset_;

static class LoggerSinkHeaderClass : public PacketHeaderClass {
public:
  LoggerSinkHeaderClass() : PacketHeaderClass("PacketHeader/LoggerSink",
                                              sizeof(hdr_loggersink)) {
              bind_offset(&hdr_loggersink::offset_);
  }
} class_loggersinkhdr;


static class LoggerSinkClass : public TclClass {
public:
  LoggerSinkClass() : TclClass("Agent/LoggerSink") {}
  TclObject* create(int, const char*const*) {
    return (new LoggerSinkAgent());
  }
} class_loggersink;

// LoggerSinkAgent constructor
LoggerSinkAgent::LoggerSinkAgent() : Agent(PT_LOGGERSINK)
{
  nbRecPkt=0;
  first = true;
}

// Execute commands
int LoggerSinkAgent::command(int argc, const char*const* argv)
{
  if(argc == 4) {
    if(strcmp(argv[1], "change") == 0) {
        Packet* pkt = allocpkt();
        hdr_loggersink* hdr =  hdr_loggersink::access(pkt);
        hdr->okReceived=false;
        hdr->change=true;

        if(strcmp(argv[2], "sampling_rate") == 0) {
          hdr->command = 'a';
          hdr->value=(int)round(atof(argv[3]));
          size_ = 224; //force to use DM5;
          send(pkt,0);
          return (TCL_OK);
        }
        else if(strcmp(argv[2], "number_of_samples") == 0) {
          hdr->command = 'b';
          hdr->value=(int)round(atof(argv[3]));
          size_ = 224; //force to use DM5
          send(pkt,0);
          return (TCL_OK);
        }
        else if(strcmp(argv[2], "sampling_periods") == 0) {
          hdr->command = 'h';
          hdr->value=(int)round(atof(argv[3]));
          size_ = 224; //force to use DM5;
          send(pkt,0);
          return (TCL_OK);
        }
    }
  }
  // If the command hasn't been processed by LoggerAgent()::command,
  // Call the command() function for the base class
  return (Agent::command(argc, argv));
}


void LoggerSinkAgent::recv(Packet* pkt, Handler*)
{
  double time = Scheduler::instance().clock();

  hdr_logger* hdr_l = hdr_logger::access(pkt);

  nbRecPkt++;

  if(hdr_l->change==true) {
    if(hdr_l->command=='a') {
      printf("FREQ=%d\n", hdr_l->value);
    }
    else if(hdr_l->command=='b') {
      printf("SAMPLES=%d\n", hdr_l->value);
   }
    else if(hdr_l->command=='h') {
      printf("Seconds=%d\n", hdr_l->value);
    }
  }
  else if(hdr_l->ok==false) {
    if(first==true)
```

```
      printf("DATA:\n");
first = false;
    printf("rl %f %d\n", time, hdr_l->size);
  }

  if(nbRecPkt>=11) {
    // Create a new packet
    Packet* pktret = allocpkt();
      // Access the Ping header for the new packet:
    hdr_loggersink* hdr_ls =  hdr_loggersink::access(pktret);
    hdr_ls->okReceived = hdr_l->ok;
    if(hdr_l->ok==true) {
      printf("OK\n");
      first = true;
    }
    size_ = 224; //Force to use DM5
    send(pktret,0);
    nbRecPkt=0;
  }
  else if(hdr_l->ok==true) {
    printf("OK\n");
    first = true;
    // Create a new packet
    Packet* pktret = allocpkt();
    // Access the Ping header for the new packet:
    hdr_loggersink* hdr_ls =  hdr_loggersink::access(pktret);
    hdr_ls->okReceived = true;
    size_ = 0; // Force to use DM1
    send(pktret,0);
  }
  else {
    // Create a new packet
    Packet* pktret = allocpkt();
    // Access the Ping header for the new packet:
    hdr_loggersink* hdr_ls =  hdr_loggersink::access(pktret);
    hdr_ls->okReceived = false;
    size_ = 0; // Force to use DM1
    send(pktret,0);
  }
  Packet::free(pkt);
}
```

# E Simulation Program Manual

With the simulation program it is possible to:

1. Simulate different settings of the DL141E.
2. Display contents of a text file capture of the DL141E.
3. Capture values from a DL141E via a RS232 interface.
4. Add sources of interference to the simulation.
5. Compare simulations with experiments.

The simulation program measures *throughput* and *maximum delay*. It also shows the results from the simulations in form of how an sinus signal should be reproduced by the logger. The frequency, resolution, amplitude, offset and phase of the sinus signal could be set. It is also possible to see in slowmotion how the values arrive from the logger and how the signal is reproduced. While many graphic simulation program focus on showing how packets arrive to from the sender to the receiver this simulation program shows a sinus signal instead. The reason is that this is more likely to show the actual case even though it is not only sinus signal one would like to measure with the DL141E. In figure 56 the initial window of the simulation program is shown. The contents of the window in figure 56 are:

1. **Plotter Settings... :** Displays the dialog window for setting up simulations, reading of logger captured text files, and logger captures. Examples of plotter setting dialogs are shown in figure 57, 58 and 59
2. **PaI-Simulator :** Displays the window for performing PaI-Simulations. These simulations are intended to be used for comparing packet losses in simulations and experiments. The *PaI-Simulator* are described more thorough in section E.1.
3. **Clear Plots :** Clears the current curves in the plots.
4. **About :** Shows a window with information about the simulation program.
5. **Exit :** Exits the simulation program.

To start a new *Simulation*, *logger file reading* or *logger capture* press the button *Plotter Settings...*. The dialog shown in figure 57 are displayed.

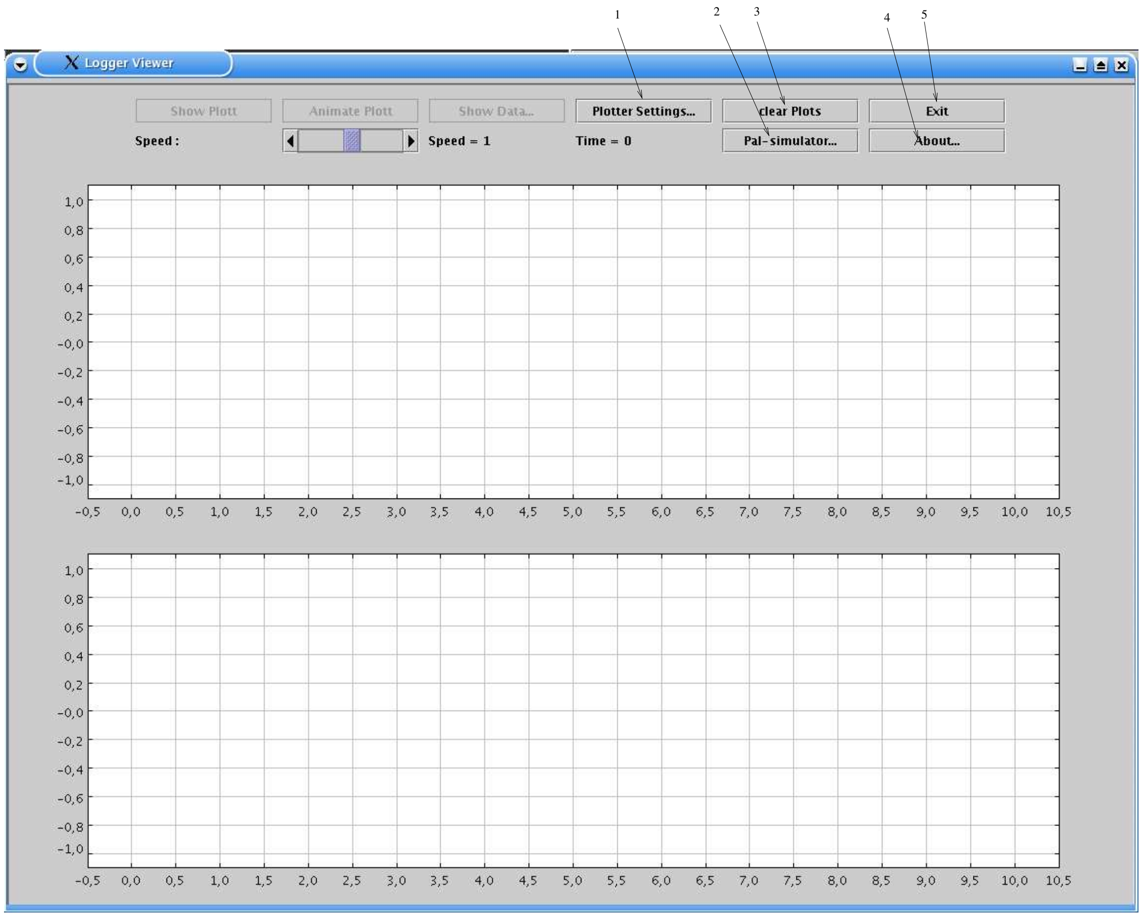
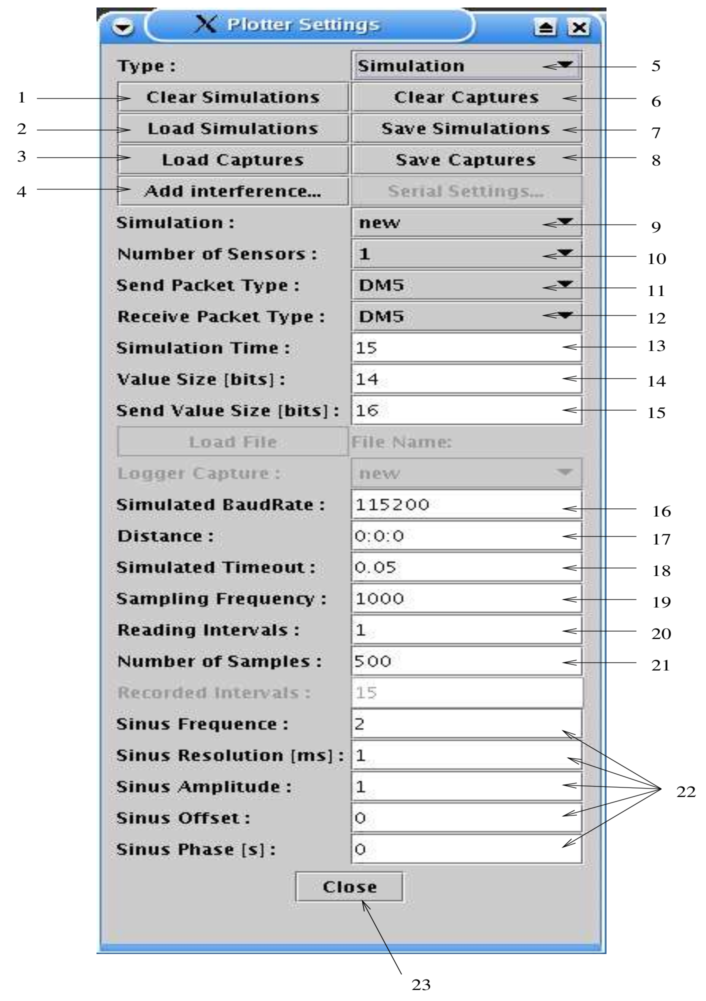


Figure 56: Simulation Program: Initial Window

The contents of the dialog in figure 57 are:

1. **Clear Simulations :** Removes all simulations performed.
2. **Load Simulations :** Load simulations from a previous performed simulations saved on a file.
3. **Load Captures :** Load logger captures from a previous performed capture saved on a file.
4. **Add Interference... :** Add sources of interferences to a simulation. The interference adder are described in section E.2
5. **Type :** The type chooser. The types could be *Simulation* for performing simulations, *From File* for reading captured logger text files, or *Logger Captures* for reading values directly from a DL141E.
6. **Clear Captures :** Removes all captures performed.
7. **Save Simulations :** Save all the simulations performed.
8. **Save Captures :** Save all the logger captures performed.



Type : Simulation
Simulation : new
Number of Sensors : 1
Send Packet Type : DM5
Receive Packet Type : DM5
Simulation Time : 15
Value Size [bits] : 14
Send Value Size [bits] : 16
Logger Capture : new
Simulated BaudRate : 115200
Distance : 0:0:0
Simulated Timeout : 0.05
Sampling Frequency : 1000
Reading Intervals : 1
Number of Samples : 500
Recorded Intervals : 15
Sinus Frequence : 2
Sinus Resolution [ms] : 1
Sinus Amplitude : 1
Sinus Offset : 0
Sinus Phase [s] : 0

Figure 57: Simulation Program: Logger Plotter Dialog - Simulation

9. **Simulation :** Choose which simulation that should be investigated. By choosing *new* a new simulation is performed. When a simulation has been run it is possible to return to it and compare it with other simulations.

10. **Number of Sensors :** Number of sensors connected to the simulated logger.

11. **Send Packet Type :** The Bluetooth packet type that should be used by the sender i.e. the logger in the simulation.

12. **Receive Packet Type :** The Bluetooth packet type that should be used by the receiver, i.e. the computer.

13. **Simulation Time :** Sets how long the simulation should continue.

14. **Value Size [bits] :** The size of each sampled value in bits. The DL141E' ADC uses 14 bits, thus the DL141E's samples is of size 14 bits.

15. **Send Value Size [bits] :** Sets the size necessary to send one sample. The DL141E uses from 16 up to 56 bits for sending its 14 bit sample. To simulate the logger in *text mode* it is recommended to use a 48 bits send value and for *rawdata mode* one should use 16 bits.

16. **Simulated Baudrate :** Sets the baudrate between the logger's buffer and the Bluetooth module.

17. **Distance :** Sets the distance between the logger and the device that reads the logger. The distance should be on the form: *start*:*step*:*stop* where *start* indicates the start distance, *stop* the final distance and *step* the step that should be taken from start to stop distance. The different distanced are divided equally in the simulation time. The distances are related to the packet loss probability at the specific distances edited by the interference builder (described in section E.2)

18. **Simulated Timeout :** The timeout when the samples that the Bluetooth module has received is send, even though the Bluetooth packet is not filled.

19. **Sampling Frequency :** The logger's sampling frequency

20. **Reading Intervals :** Number of seconds between reading intervals.

21. **Number of Samples:** Number of samples each reading interval.

22. **Sinus :** Properties of the captured sinus signal.

23. **Close :** Closes the *Plotter Settings* dialog and make it possible to run or view a simulation.

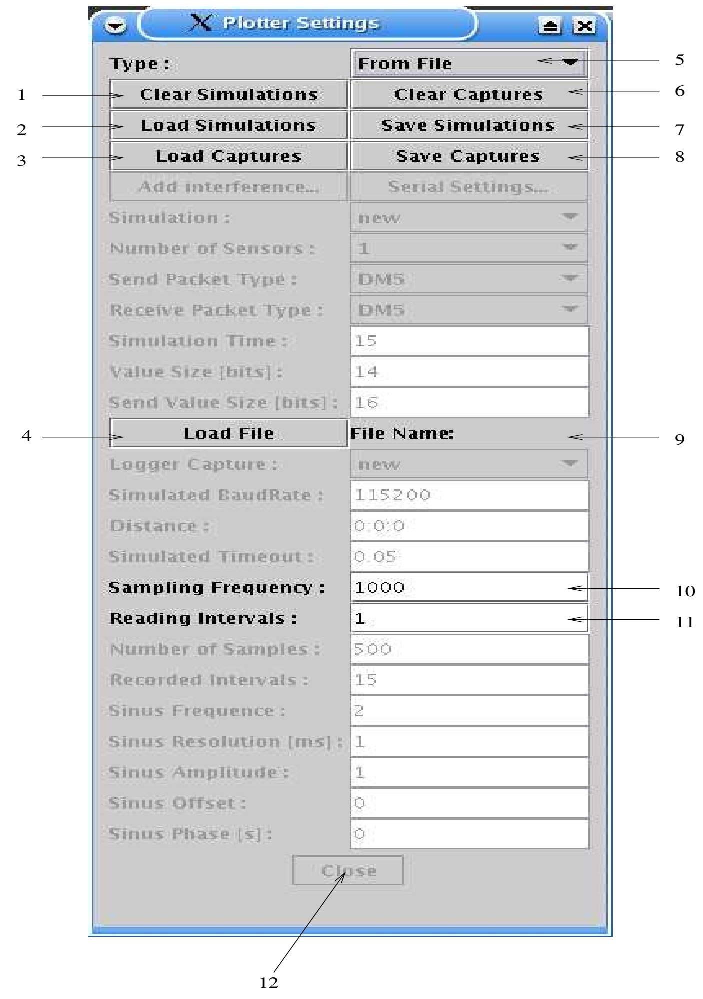


Figure 58: Simulation Program: Logger Plotter Dialog - From File

Figure 58 shows the *Plotter Settings Dialog* for reading a captured text file from a logger. A textfile like this could be captured with e.g. Microsoft HyperTerminal and requires that the logger uses *text mode*. The contents of the dialog in figure 58 are:

1. **Clear Simulations :** Removes all simulations performed.
2. **Load Simulations :** Load simulations from a previous performed simulations saved on a file.
3. **Load Captures :** Load logger captures from a previous performed capture saved on a file.
4. **Load File :** Loads a captured text file from a logger.
5. **Type :** The type chooser. The types could be *Simulation* for performing simulations, *From File* for reading captured logger text files, or *Logger Captures* for reading values directly from a DL141E.
6. **Clear Captures :** Removes all captures performed.
7. **Save Simulations :** Save all the simulations performed.

8. **Save Captures :** Save all the logger captures performed.

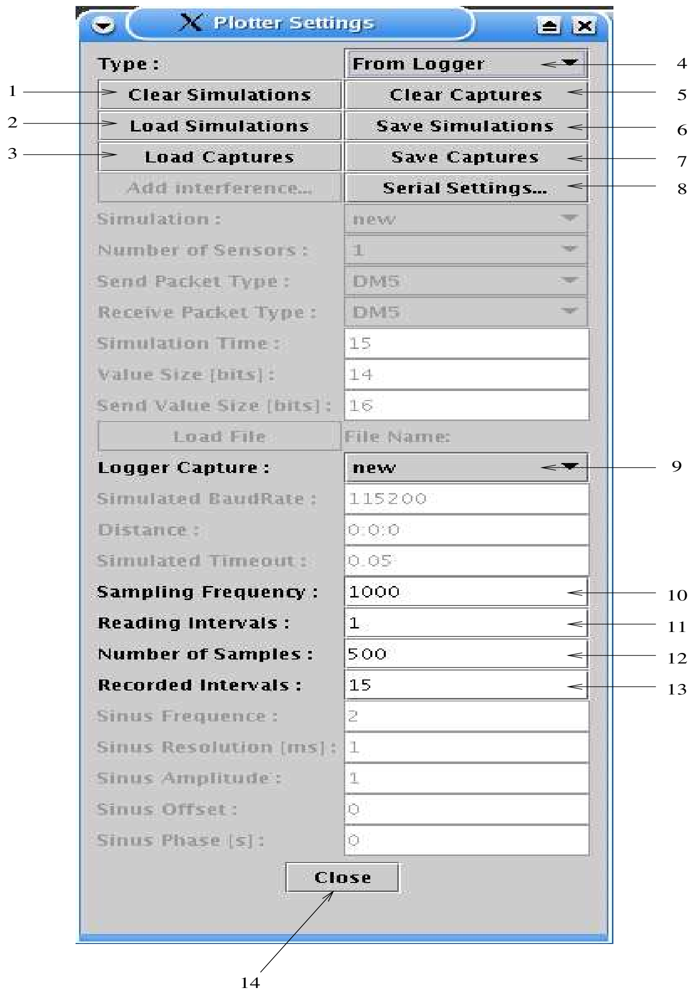
9. **File Name :** Displays the filename of the captured text file from a logger. The contents of this file could then be shown in the logger *plotter window* shown in figure 56

10. **Sampling Frequency :** The sampling frequency used when capturing the textfile from the logger.

11. **Reading Intervals :** The sampling reading intervals used when capturing the text file from the logger.

12. **Close :** Closes the *Plotter Settings* dialog and make it possible to run or view the captured textfile as graphs. This button is not enabled until a file has been chosen.

Figure 59 shows the *Plotter Settings Dialog* for reading a logger directly via a bluetooth device connected to a serial port in the computer. The contents of the dialog in figure 59 are:



Type : From Logger
Simulation : new
Number of Sensors : 1
Send Packet Type : DM5
Receive Packet Type : DM5
Simulation Time : 15
Value Size [bits] : 14
Send Value Size [bits] : 16
File Name:
Logger Capture : new
Simulated BaudRate : 115200
Distance : 0:0:0
Simulated Timeout : 0.05
Sampling Frequency : 1000
Reading Intervals : 1
Number of Samples : 500
Recorded Intervals : 15
Sinus Frequence : 2
Sinus Resolution [ms] : 1
Sinus Amplitude : 1
Sinus Offset : 0
Sinus Phase [s] : 0

Figure 59: Simulation Program: Logger Plotter Dialog - From Logger

1. **Clear Simulations :** Removes all simulations performed.

2. **Load Simulations :** Load simulations from a previous performed simulations saved on a file.
3. **Load Captures :** Load logger captures from a previous performed capture saved on a file.
4. **Type :** The type chooser. The types could be *Simulation* for performing simulations, *From File* for reading captured logger text files, or *Logger Captures* for reading values directly from a DL141E.
5. **Clear Captures :** Removes all captures performed.
6. **Save Simulations :** Save all the simulations performed.
7. **Save Captures :** Save all the logger captures performed.
8. **Serial Settings... :** Opens the dialog to choose *serial settings*.
9. **Logger Capture :** Choose which logger capture that should be investigated. By choosing *new* a new logger capture is performed. When the logger capture has been run it is possible to return to it and compare it with other captures.
10. **Sampling Frequency :** The logger's sampling frequency
11. **Reading Intervals :** Number of seconds between reading intervals.
12. **Number of Samples:** Number of samples each reading interval.
13. **Recorded Intervals :** An approximated value of how many reading intervals that should be recorded.
14. **Close :** Closes the *Plotter Settings* dialog and make it possible to start the logger capture.

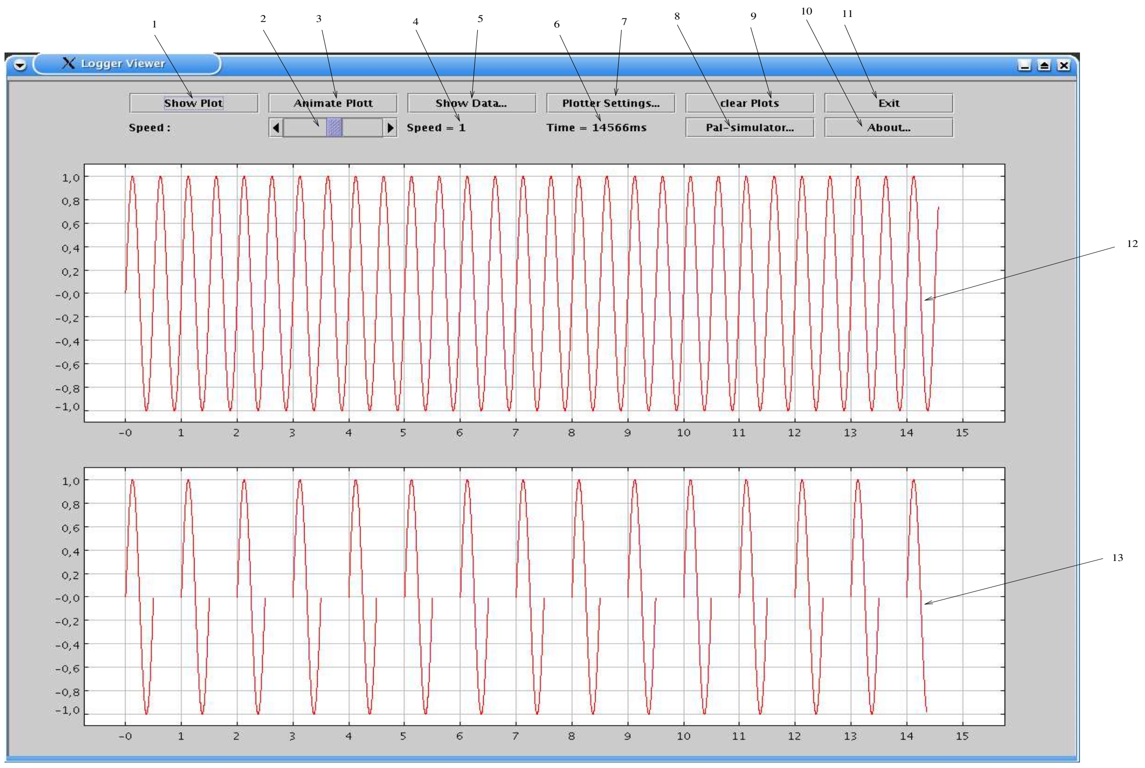


Figure 60: Simulation Program: Simulation Window after performed simulation

Figure 60 shows the *Simulation Window* after a performed simulation. The contents of the *Simulation Window* are:

1. **Show Plot :** Shows the plot for the artificial sinus signal and the plot showing how the logger reproduces it based on the simulation.
2. **Speed changer :** Change the speed of the signal animation.
3. **Animate Plot :** Animate the plot for the artificial sinus signal and the plot showing how the logger reproduces it based on the simulation.
4. **Current speed :** Shows the current speed of the animator.
5. **Show Data... :** Shows throughput and maximum delay derived from the simulation.
6. **Current Time :** Shows the current time when performing an animation.
7. **Plotter Settings :** Displays the *Logger Plotter Dialog* shown in figure 57, 58 and 59.
8. **PaI-Simulator... :** Displays the window for performing PaI-Simulations. These simulations are intended to be used for comparing packet losses in simulations and experiments. The *PaI-Simulator* are described more thourough in section E.1.
9. **Clear Plots :** Clear all curves in the plots.
10. **About... :** Shows a window with information about the simulation program.

11. **Exit :** Exits the simulation program.
12. **Plot 1 :** Shows the artificial original sinus signal if a *simulation* is performed, the values of a text file if *From File* is chosen, and the values read from the logger if *From Logger* is chosen.
13. **Plot 2 :** Shows the simulated logger's reproduction of the artificial sinus signal.

## E.1 PaI Simulator

The purpose with the Packet at Interval (PaI) simulator is to enable the user to perform simulations with added sources of interferences and compare them with respect to *diff-DPAF* with Merlin traffic summaries captured by experiments. Figure 61 shows the *Initial window* of *PaI Simulator*. Before any simulation could be performed the simulations parameters must be set. The *New Simulation* button opens the dialog for setting up simulation parameters (see figure 62) The contents of the *PaI Simulator Dialog* is:

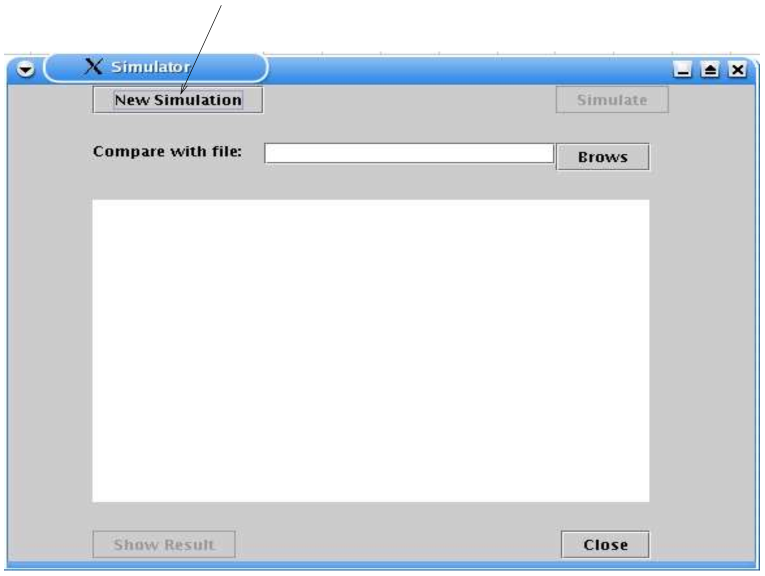
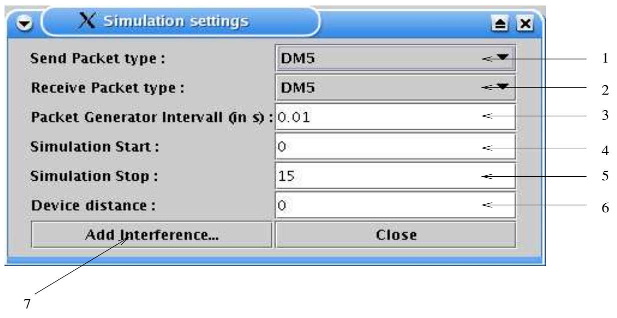


Figure 61: Simulation Program: Initial PaI Simulator Window



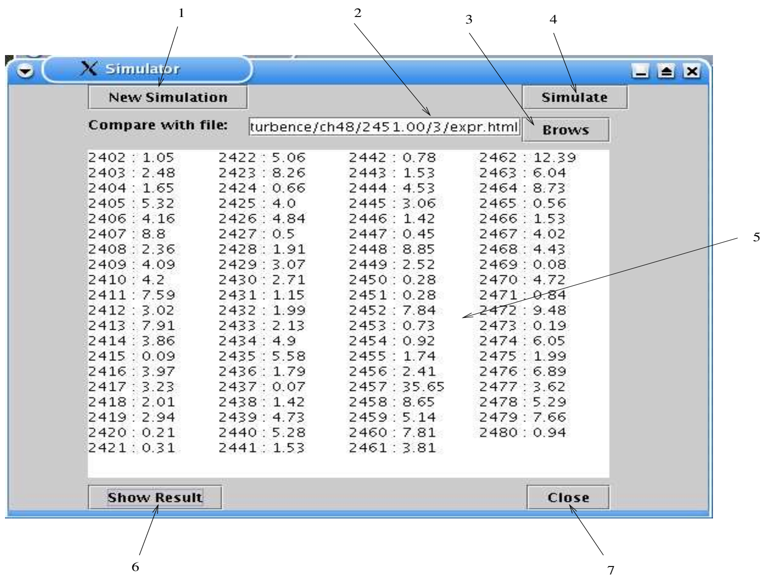
Send Packet type : DM5
Receive Packet type : DM5
Packet Generator Intervall (in s) : 0.01
Simulation Start : 0
Simulation Stop : 15
Device distance : 0

Figure 62: Simulation Program: PaI Simulator Dialog

1. **Send Packet Type :** The Bluetooth packet type that should be used by the sender.
2. **Receive Packet Type :** The Bluetooth packet type that should be used by the receiver.
3. **Packet Generator Interval (in s) :** The interval that the Bluetooth packets from the sender should be generate with.
4. **Simulation Start :** Start time of the simulation.

5. **Simulation Stop :** Stop time of the simulation.
6. **Device distance :** Distance between the Bluetooth devices.
7. **Add Interference... :** Add sources of interferences to a simulation. The interference adder are described in section E.2

Figure 63 shows the *PaI Simulator Window* after a performed simulation and comparision with an experiment.



```
turbence/ch48/2451.00/3/expr.html

2402 : 1.05    2422 : 5.06    2442 : 0.78    2462 : 12.39
2403 : 2.48    2423 : 8.26    2443 : 1.53    2463 : 6.04
2404 : 1.65    2424 : 0.66    2444 : 4.53    2464 : 8.73
2405 : 5.32    2425 : 4.0     2445 : 3.06    2465 : 0.56
2406 : 4.16    2426 : 4.84    2446 : 1.42    2466 : 1.53
2407 : 8.8     2427 : 0.5     2447 : 0.45    2467 : 4.02
2408 : 2.36    2428 : 1.91    2448 : 8.85    2468 : 4.43
2409 : 4.09    2429 : 3.07    2449 : 2.52    2469 : 0.08
2410 : 4.2     2430 : 2.71    2450 : 0.28    2470 : 4.72
2411 : 7.59    2431 : 1.15    2451 : 0.28    2471 : 0.84
2412 : 3.02    2432 : 1.99    2452 : 7.84    2472 : 9.48
2413 : 7.91    2433 : 2.13    2453 : 0.73    2473 : 0.19
2414 : 3.86    2434 : 4.9     2454 : 0.92    2474 : 6.05
2415 : 0.09    2435 : 5.58    2455 : 1.74    2475 : 1.99
2416 : 3.97    2436 : 1.79    2456 : 2.41    2476 : 6.89
2417 : 3.23    2437 : 0.07    2457 : 35.65   2477 : 3.62
2418 : 2.01    2438 : 1.42    2458 : 8.65    2478 : 5.29
2419 : 2.94    2439 : 4.73    2459 : 5.14    2479 : 7.66
2420 : 0.21    2440 : 5.28    2460 : 7.81    2480 : 0.94
2421 : 0.31    2441 : 1.53    2461 : 3.81
```

Figure 63: Simulation Program: PaI Simulator Window after Performed Simulation

1. **New Simulation :** Shows the *PaI Simulation Dialog* shown in figure 62.
2. **File name :** The filename of the *Merlin trace file* from an experiment that should be compared to the trace file from the simulation.
3. **Brows :** Choose the *Merlin trace file* from a experiment that should be compared with the tracefile from the simulation.
4. **Simulate :** Run the simulation.
5. **diff-DPAF table :** The *diff-DPAF* between the *Merlin trace file* from the experiment and the trace file from the simulation.
6. **Show Result :** Shows the *diff-DPAF* table.
7. **Close :** Closes the *PaI Simulator*.

## E.2 Interference Adder

With the *Interference Adder* it is possible to add different sources of interference to the simulation. Interferences can be added in three ways:

1. By adding a interference spectra.
2. By defining the packet loss probability at a specific Bluetooth channel.
3. By defining a packet loss probability at a specific distance.

All interferences are added by clicking on the plot or loading graphs from a file. Figure 64 shows the

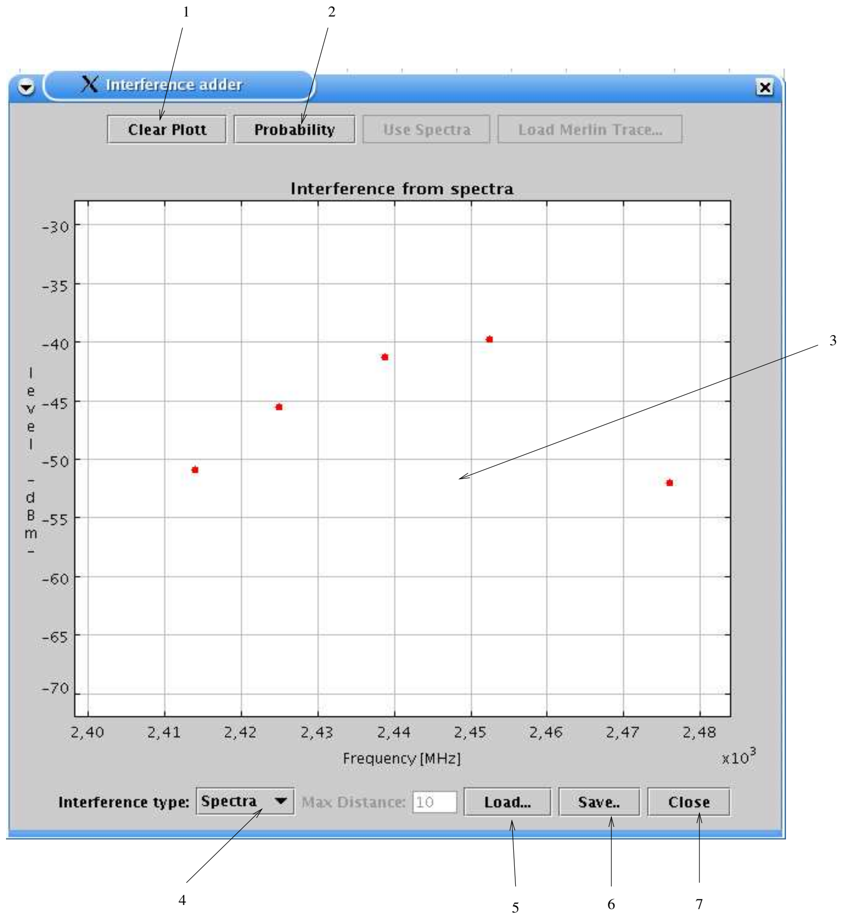


Figure 64: Simulation Program: Interference Adder - Interference from Spectra

*Interference Adder Window* for adding interference based on a interference spectra. The contents of the *Window* is:

1. **Clear Plott :** Clears all added Interferences.
2. **Probability :** Change from *spectra* interference adder to *probability* interference adder.
3. **Plott :** The interferences, frequency and level.

4. **Interference type :** Type of interference. Could be either *spectra* which add interference to the Bluetooth channel, or *Distance* which add interference (equal for all channels) depending on the distance between the Bluetooth devices.

5. **Load... :** Loads a curve.

6. **Save... :** Saves a curve.

7. **Close :** Closes the Interference adder.

Figure 65 shows the *Interference Adder Window* for adding interferences in form of packet loss probabilities at specific Bluetooth channels. The contents of the *Window* in figure 65 is:

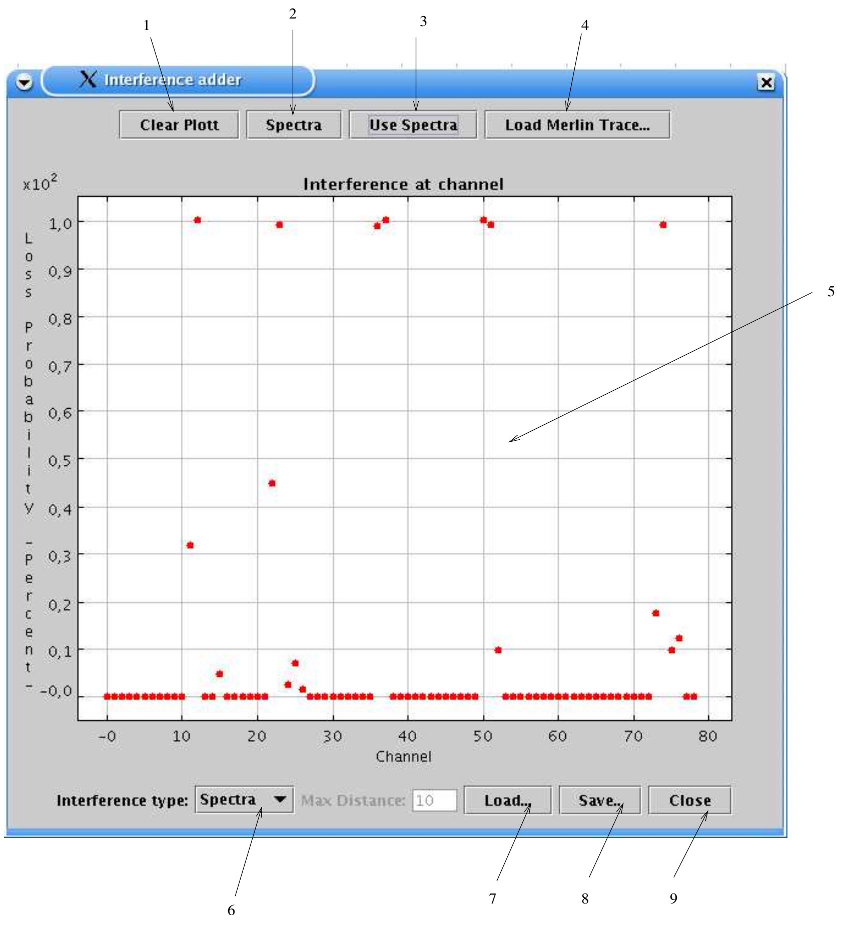


Figure 65: Simulation Program: Interference Adder - Interference at Channel

1. **Clear Plott :** Clears all added Interferences.

2. **Spectra :** Change from *probability* interference adder to *spectra* interference adder.

3. **Use Spectra :** Uses the packet loss probability calculated from the interferences added in the spectra adder (see figure 64).

4. **Load Merlin Trace... :** Load a *Merlin Trace* from an experiment and base the packet loss probability from the *DPAF* in the *Merlin Trace* file.
5. **Plott :** The interferences, channel and packet loss probability.
6. **Interference type :** Type of interference. Could be either *spectra* which add interference to the Bluetooth channel, or *Distance* which add interference (equal for all channels) depending on the distance between the Bluetooth devices.
7. **Load... :** Loads a curve.
8. **Save... :** Saves a curve.
9. **Close :** Closes the Interference adder.

Figure 66 shows the *Interference Adder Window* for adding interferences in form of packet loss probabilities when the Bluetooth devices are located at a specific distance from each other. The *Interference*

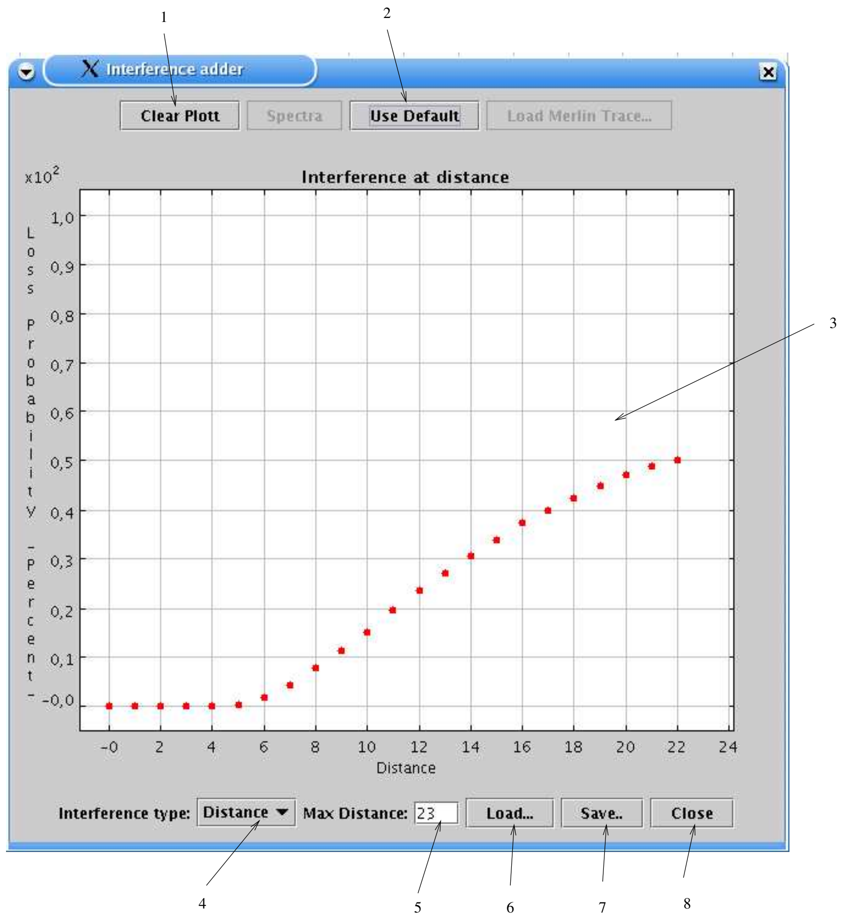


Figure 66: Simulation Program: Interference Adder - Interference at Distance

*at Distance* adder could be used to change packet loss probability during a simulation. The contents of the *Window* in figure 66 is:

1. **Clear Plott :** Clears all added Interferences.
2. **Use Default :** Use the default values from *BlueHoc's ferloss tables*. Note, this table has different packet loss probabilities for different Bluetooth packets which the other interference types do not.
3. **Plott :** The interferences, distance and packet loss probability.
4. **Interference type :** Type of interference. Could be either *spectra* which add interference to the Bluetooth channel, or *Distance* which add interference (equal for all channels) depending on the distance between the Bluetooth devices.
5. **Max Distance :** The maximum distance between the Bluetooth modules.
6. **Load... :** Loads a curve.
7. **Save... :** Saves a curve.
8. **Close :** Closes the Interference adder.

## F Text- vs. Rawdata Mode in Logger Reader Software

When a Logger reader program receives values from a logger each byte needs to be transformed to a float value that could be treated in different ways depending on the purpose with the logger reader. In *text mode* the received bytes are an ASCII code for the number and in *rawdata mode* each value is two bytes which should be concatenated to a 16 byte value where the two first bits are set to zero. When using *text mode* each sample could be one up to six bytes, separated by space. This means that one cannot know how much memory needs to be allocated and if a received number is the first, second, third, forth, fifth or last in the sample value. In the Java logger reader program associated with this thesis the proceeding of typecasting for *text mode* is performed in the following way.

1. The received bytes are typecast to a java string.
2. The java string are split at the space, forming one string for each sample.
3. Each string are type casted to a double.

This procedure is easy programmed, but needs a lot of resources. In the first step a very large string object is created, in the second as many string objects as there are samples are created, and finally in the third step as many double objects that there are samples are created. To creates as many objects as this consumes a lot of resources and memory. It could probably be done in a better way than this, but the advantage of using *rawdata mode* makes it more interesting to focus on *rawdata mode* instead. To transform two received bytes ($b_1$ and $b_2$) simply do:

$$((b_1 \wedge 0x3F) << 8) \vee (b_2 \wedge 0x00FF)$$

These simple bit operations are performed very fast by a computer. In figure 67 the execution time of both strategies are compared with focus on milliseconds to typecast a certain number of samples for *rawdata mode* and *text mode*. Figure 67 speaks for it self, the *rawdata mode* type casting is more than 200 times faster then the *text mode* type casting. The tests were performed on a Linux computer with a 1700MHz Intel Pentium 4 processor.

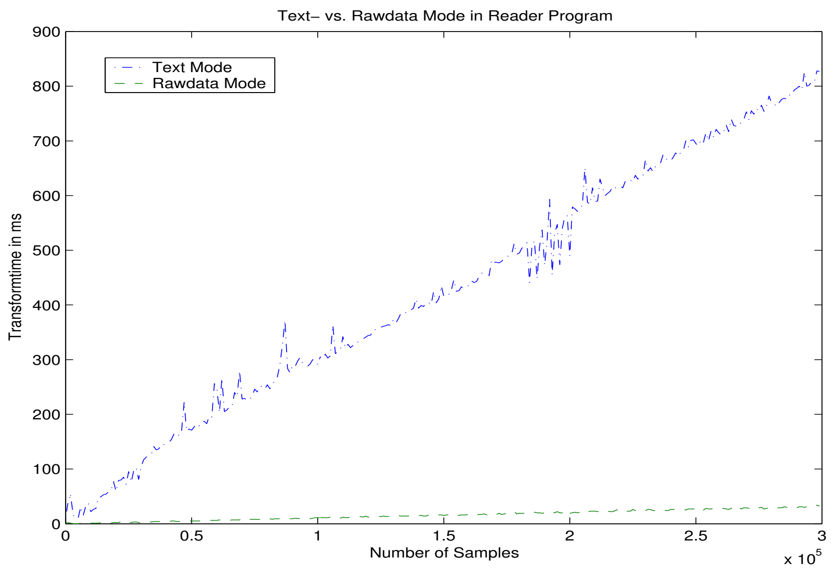


Figure 67: Text- and Rawdata Mode in Logger Reader Software

# G Scheduling Algorithms for the DL141E Bluetooth Logger

*BlueCenter/Industrial Development Centre, Olofström AB has developed a Bluetooth measurement value logger - the DL141E. The DL141E are capable of collecting samples from up to eight channels, sampled at up to 30kHz, and can store up to 256 000 samples in its buffer. The logger sends the values from its buffer to e.g. a computer wireless via a Bluetooth channel. A today only point-to-point system configuration exists, but in the future large systems with several DL141Es will be set up. Thus it is important to investigate different approaches for constructing large systems with the DL141E.*

The main task with this thesis is to investigate scheduling algorithms suitable for possible future system with the DL141E Bluetooth measurement logger. The thesis continues the earlier performed master's thesis: *Modelling and Evaluation of a Bluetooth Data Logger in the Presence of Interference Sources*. The investigation should focus on throughput and maximum delay of a sample based on simulations. The simulations should be performed in *Network Simulator 2 (NS2)* with a Bluetooth extension *(UCBT)* and a model constructed in the earlier thesis. The scheduling algorithms should be investigated for two different cases:

1. A logger reader program collects values from multiple loggers. The investigation should focus on which scheduling algorithm this reader program should use to read the loggers in the most effective way. The work could use parts of a Java based simulation program constructed in the earlier thesis.

2. In a Bluetooth network on Bluetooth device acts as master and could be connected from one to seven slaves. The order in which the slaves are polled needs to be decided by a scheduling algorithm. The investigation should focus on which scheduling algorithm that is most suitable for the DL141E and its systems.

The thesis should investigate classic scheduling algorithms, e.g. PRR (Pure Round Robin), ERR (Exhaustive Round Robin), LRR (Limited Round Robin), and WRR (Weighted Round Robin), but also a few other algorithms that are decided after a thorough literature study. The algorithms needs only to be simulated, but it is of great importance that the chosen approach is implement able in the DL141E logger with the available resources. The goal is to find the most appropriate scheduling algorithm that produces the best result with respect to throughput, maximum delay, sensitiveness for sources of interference, and that are implement able with the available resources.

**Keywords :** Bluetooth, Piconet, Scheduling Algorithms, Simulation, NS2, C++ and Java.

The investigations should be performed as a *C-level 10 credits thesis* suitable for *two students* who needs to do a thesis in *Computer science and/or telecommunications*. The students should have good knowledge's in C++ and Java programming, knowledge's in algorithms, implementation and evaluation of algorithms, and basic knowledge's about Bluetooth and telecommunications. Because the thesis is about simulating different algorithm implementations it is important that the students can describe the simulation and result thorough and clear in a report written in Swedish or English.

## H Simple Scatternet with the DL141E Bluetooth Logger

*BlueCenter/Industrial Development Centre, Olofström AB have developed a Bluetooth measurement value logger - the DL141E. The DL141E are capable of collecting samples from up to eight channels, sampled at up to 30kHz, and can store up to 256 000 samples in its buffer. The logger sends the values from its buffer to e.g. a computer wireless via a Bluetooth channel. Today only point-to-point system configuration exists, but in the future large systems with several DL141Es will be set up. Thus it is important to investigate different approaches for constructing large systems with the DL141E.*

The main task of this thesis is to investigate, via simulations how one constructs scatternets suitable for the DL141E logger. The thesis should continue the work performed in the master's thesis *Modelling and Evaluation of a Bluetooth Data Logger in the Presence of Interference Sources* performed earlier at BlueCenter/IUC. The former thesis provides a simulation model for the DL141E logger by adopting the simulation program *Network Simulator 2 (NS2)* with a Bluetooth extension (UCBT). Strategies for scatternet with the DL141E logger should be derived based on the following:

- The types of scatternets that possible will be needed by BlueCenter/IUC's customers.
- Investigation about what is possible with today's DL141E design.
- Suitable routing protocol for the DL141E logger.
- Actions that needs to be taken when a logger in a scatternet are disconnected, i.e. fails or run out of power.
- Additional tasks or solutions to problems that occur during the work.

The different scatternet strategies that are derived should be simulated and evaluated. The strategies need only to be implemented in simulations, but it is of great importance that the chosen approach are implement able in the DL141E logger with the available resources. The goal is to find a scatternet approach that produces the best result with respect to throughput, maximum delay, sensitiveness for sources of interference, and that are implement able with the available resources.

**Keywords :** Bluetooth, Scatternet, Simulation, NS2, C++ and Java.

The investigations should be performed as a *C-level 10 credits thesis* suitable for *one or two students* who needs to do a thesis in *Computer science and/or telecommunications*. The students should have good knowledge's in C++ and Java programming, knowledge's in algorithms, implementation and evaluation of algorithms, and basic knowledge's about Bluetooth and telecommunications. Because the thesis is about simulating different scaternet configurations it is important that the students can describe the simulation and result thorough and clear in a report.

IMIT/LCN 2005-08

www.kth.se