Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

BTnode Application for automated Link Measurements

Martin Wirz

TERM THESIS

Winter Term 2006/07

Supervisor:	Andreas Meier
Professor:	Dr. Lothar Thiele
Start Date:	23th of October 2006
Issue Date:	4th of February 2007

Abstract

Link-quality measurements provide insight into the radio-channel behaviour in a wireless sensor network. We perform packet-error tests in order to measure the link quality. During such a test, packets are sent from a transmitting node to a receiving node. Counting the correct received packets allows characterizing the link quality between the nodes.

In this thesis an application for automated link measurements was implemented on the BTnode using the Chipcon CC1000 low-power radio. Our application performs packet-error tests and provides additional tracing functionality to determine packet-loss time dependency. These tests are controlled by the DSNAnalyzer based on the DSN infrastructure.

In addition, we provide a short case study comparing the packet-delivery ratio of the BTnode (CC1000), A80 (CC1020) and the Tmote Sky (CC2420) nodes.

Contents

1: Introduction *1*
1.1 Wireless Sensor Networks 1
1.2 Chapters Overview 1
1.3 Motivation 2
1.3.1 Motivation for Link-Quality Measurements 2
1.3.2 Preliminary Work 2
1.3.3 Comparability 3
1.3.4 Goal of this Thesis 3

2: Platform *5*
2.1 Testbed Overview 5
2.2 BTnode 6
2.2.1 Overview 6
2.2.2 BTnut System Software 7
2.2.3 Low Power Radio 7
2.3 Deployment-Support Network 13
2.4 DSNAnalyzer 14

3: Concept *17*
3.1 Overview 17
3.2 Link-Quality Measurements 17
3.2.1 Scenario 17
3.2.2 Approach 19
3.2.3 Comparison to the Siemens' Approach 21
3.3 PET Data Analysis 21
3.4 Data Collection and Node Control in WSN 22
3.4.1 Client - Target Communication 22
3.4.2 DSN Node - Target Communication 23

4: Verification and Measurements *25*
4.1 Application Verification 25
4.2 Measurements 26

4.2.1 BTnode 26
4.2.2 A80 29
4.2.3 Tmote Sky 29
4.3 Conclusion 29

5: Conclusion *33*
5.1 Contributions 33
5.2 Summary 33

A: Specification *35*

B: Description Task *39*

Tables

1-1 CC1000 and CC1020 current consumption comparison 3

2-1 Frequency configuration registers 12
2-2 Output power settings and typical current consumption at 868Mhz. . . 13
2-3 Functionality of the DSNAnalyzer [12] 15

3-1 PET configuration parameters 19
3-2 Packet error test enhancements 20

4-1 Statistical analysis of the the verification test with six BTnodes 26
4-2 BTnode PET configuration 28
4-3 A80 PET configuration 29
4-4 Tmote Sky PET configuration 29

Figures

2-1 Testbed overview 5
2-2 BTnode rev3 system overview [4] 6
2-3 CC1000 microcontroller interface 8
2-4 RSSI voltage output 9
2-5 Received signal strength vs. RSSI voltage 10
2-6 Frequency parameters 12
2-7 DSNAnalyzer concept 14

3-1 DSN virtual connection 20
3-2 DSN virtual connection 22
3-3 JSON-RPC client/target sequence diagram 23

4-1 Visualization of the application's verification test 27
4-2 PET visualization of the BTnode sensor node 28
4-3 PET visualization of the A80 sensor node 30
4-4 PET visualization of the Tmote Sky sensor node 31

1

Introduction

”Seamlessly integrating computing with the physical world via sensors and actuators, physical computing systems promise to give society an improved living standard, greater security, and unparalleled convenience and efficiency.”

(Stankovic et al. 2005 [16])

1.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) consists of small, autonomous sensor devices with wireless networking capability. Because of their small physical dimensions, these sensor devices, referred to as sensor nodes, can be used to cooperatively monitor physical conditions without disturbing the observed environment.
However, the sensor nodes have to face major resource constraints:

- Only small batteries can be attached to a node
- The radio range is short because of embedded low-power transceivers
- The processing speed is limited
- Only a small amount of memory is available on the nodes

These limitations complicate the design of protocols and applications for WSNs.

1.2 Chapters Overview

The report of this thesis is structured in the following way. In Chapter 2 we present system specific properties of the platform on which we implemented our application and introduce the testbed we used for our measurements. In Chapter we deal with the concept of our application to perform link measurements. In Chapter **??** we provide specifics on the implementation. Finally, in Chapter 4 we verify our application and get a first glimpse at measurements performed in a office scenario and compare the results to other sensor nodes.

1.3 Motivation

1.3.1 Motivation for Link-Quality Measurements

Wireless communication is seen as very unpredictable due to the influence of reflection, fading and interference. All this influences limit the ability to model a wireless channel accurately for prediction of the radio behavior. In addition, communication quality can vary dramatically over time and change with slight spatial displacements as stated in [17].
Sensor nodes use low power radio transceivers for data transmission. This amplifies the unpredictability of the wireless communication channel.

To cope with this problem, reliable and robust communication protocols specially designed for wireless ad-hoc networks are necessary. To develop and optimize such protocols, a good understanding of the radio channel behavior is required. Measuring the link quality between different nodes in WSN is one way to determine the channel characteristics.

1.3.2 Preliminary Work

In cooperation with the ETH Zurich, Siemens Building Technologies (SBT) has developed a platform to perform link measurements in order to investigate the channel behavior of their wireless sensor node A80 [12].
The A80 sensor node is a proprietary device, developed by SBT. The A80 is based on a MSP430 microprocessor and has a Chipcon CC1020 [7] UHF radio transceiver. It is designed for low power applications in the Industrial, Scientific and Medical (ISM) domain. The CC1020 has a frequency range from 402MHz - 470MHz and 804MHz - 940MHz and complies with the regulations to allocate the ISM alarm band [13]. This band is exclusive for transmitting alarm message and requires channel spacing modulation of 25kHz and transmit power of at most 10mW with a duty cycle less than 0.1%.
For safety-critical applications, operating in this narrow frequency band is to advantage because of regulations.
One drawback of narrow-band systems are that their radio requires more time to calibrate to a certain frequency. A short calibration time however is crucial in time critical protocols.
A second drawback of narrow-band systems is the energy requirement. The CC1020 has a current consumption of 19.9mA in RX mode and 20.5mA in TX mode at 0dBm. This is significantly higher than the consumption of the comparable Chipcon CC1000 [6] transceiver that also operates in the ISM band at 868Mhz but is not allowed to use the alarm channel due to its broader channel spacing of 500kHz. Table 1-1 gives a comparison of the energy consumption between the CC1000 and the CC1020 transceiver.
The CC1000 is mainly targeted for very low power applications which are needed in battery-operated systems requiring 3 years or more of battery lifetime.

Operating at 868 MHz	CC1000	CC1020
Minimum channel spacing	500kHz	25kHz
Current consumption in RX mode	9.6mA	19.9mA
Current consumption in TX mode at 0dBm	16.8mA	20.5mA

Table 1-1: CC1000 and CC1020 current consumption comparison

1.3.3 Comparability

A system to investigate the link behavior in wireless sensor networks was developed by SBT and is already in use. With this testbed, the Cipcon CC1020 transceiver implemented on the A80 node has been investigated.
In a next step, one is interested in comparing the link quality of the CC1020 to the CC1000 using the same measurement method and the same platform.

1.3.4 Goal of this Thesis

The goal of this thesis is to develop an application on the BTnode sensor node that allows automated link-quality measurements in order to investigate the Chipcon CC1000 radio. These link-quality measurements should be controllable with the DSNAnalyzer based on the DSN infrastructure.

2 Platform

2.1 Testbed Overview

This chapter introduces the testbed we set up to perform link-quality measurements. A complete overview is given in Figure 2-1. In the next sections, all elements will be presented. In a first step, we focus on the sensor node on which we are going to implement our application. We will highlight the main components and their specifications. Then, with the DSN, we introduce a wireless cable replacement system able to control and monitor sensor nodes in a WSN. Finally, to retrieve and analyse the measurement data, a software running on top of the DSN will be presented.

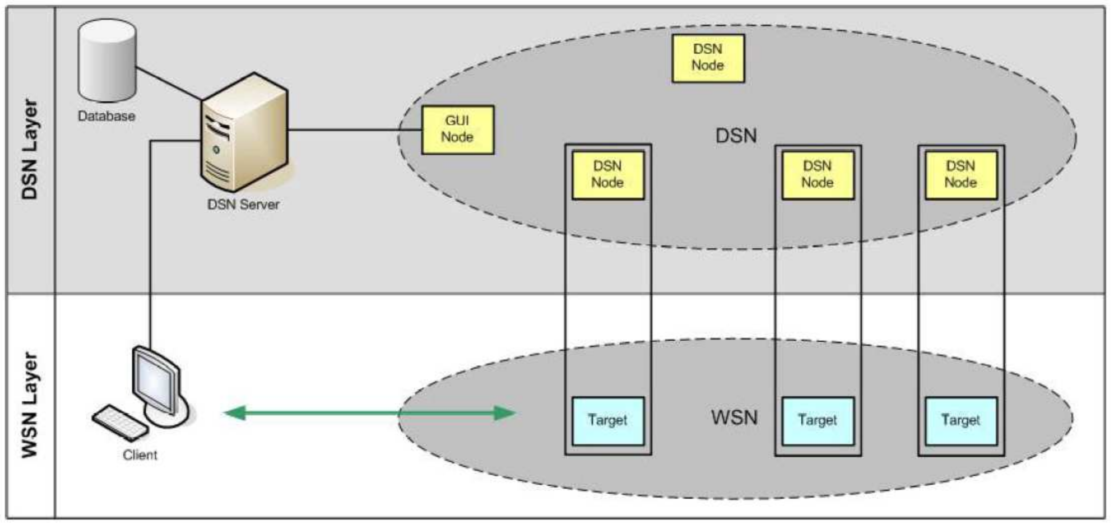


Figure 2-1
Testbed overview: The DSN is a background service providing a virtual connection from a host computer to each sensor node. [12]

2.2 BTnode

We intend to implement our application on the BTnode sensor node. The next sections introduce major hardware components with focus on communication features coming into operation in our application.

2.2.1 Overview

The BTnode is a small autonomous wireless communication and computing platform developed at the ETH Zurich by the Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems. It serves as a demonstration and prototyping platform for research in mobile, ad-hoc and distributed sensor networks.
An overview of the system is given in Figure 2-2.

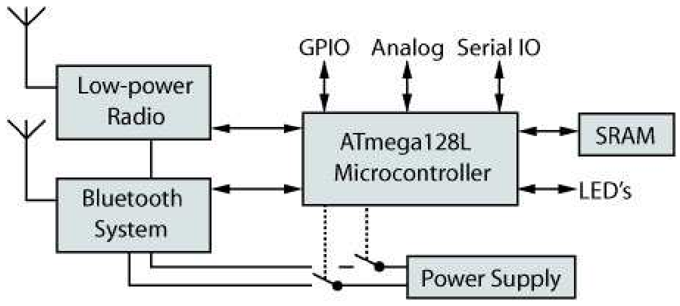


Figure 2-2
BTnode rev3 system overview [4]

The BTnode features two different radios: The Zeevo ZV4002 Bluetooth system and the Chipcon CC1000 low-power radio [6]. The two radios are controlled by an Atmel Atmega128 [5] microcontroller.

The Atmega128 is an 8-bit RISC microcontroller running at frequencies up to 8MHz. Most instructions are executed in a single clock cycle. On the BTnode, the system clock is generated by an external 7.3728MHz crystal oscillator. The Atmega128 provides 128kB in-system reprogrammable flash memory, 4kB EEPROM and 4kB internal SRAM. An addition of 240kB external SRAM is available on the BTnode.

The integrated peripherals include general purpose I/O pins, several 10-bit ADC inputs, JTAG, serial peripheral interface (SPI), two-wire serial interface (TWI1) and two universal asynchronous receiver transmitters (UART).

On the BTnode, the ATmega128 is connected over the second UART to the Bluetooth module. The first UART is available to applications. It is commonly used as debug and control interface using standard terminal emulator software.
The CC1000 transceiver is connected over SPI to the ATMega128. A detailed description is given in 2.2.3.

2.2.2 BTnut System Software

The BTnode runs with the BTnut system software [1], an expansion of Nut Operating System [9]. Nut/OS is an intentionally simple open source real-time operating system designed for embedded device development. Its key features include:

- Cooperative multithreading
- Event handling
- Periodic and one-shot timers
- Dynamic heap memory allocation
- Interrupt driven streaming I/O

The system is highly configurable and, with the BTnut system software, has been extended to provide BTnode specific drivers and libraries such as a Bluetooth stack and several communication protocols. BTnut provides an almost complete C standard library. This makes it possible to use native ANSI C for code development on the BTnode platform.

2.2.3 Low Power Radio

2.2.3.1 Overview

Besides the Bluetooth radio, the BTnode provides the low-power radio transceiver Chipcon CC1000. The CC1000 is a single-chip UHF transceiver designed for very low power and very low voltage wireless applications. The radio is intended for the ISM and SRD (Short Range Device) frequency bands at 315, 433, 868 and 915 MHz. Most operating parameters can be programmed via a serial bus interface. In combination with a microcontroller, the CC1000 becomes a highly flexible RF transceiver. Features of the radio include:

- Frequency selectable from 300-1000 MHz in steps of 250 Hz
- -110dBm sensitivity (at 2.4kBaud)
- Programmable output power from -20 to 10dBm (frequency dependent)
- Low supply voltage (2.1V to 3.6V)
- RSSI output
- FSK data rate up to 76.8kBaud
- FSK modulation spectrum shaping
- Integrated bit synchronizer

2.2.3.2 Communication with ATMega128

How does the microcontroller and the CC1000 communicate with each other?
In transmitting mode, the CC1000 receives data bytes from the ATmega128, modulates this signal onto an electromagnetic wave and transmits it into space.
In receiving mode, the procedure works the other way round. The radio receives an electromagnetic wave, demodulates it and decodes the signal. This digital signal is

then interpreted as a byte-stream and forwarded to the microcontroller.
The microcontroller itself is able to receive data bytes from the radio as well as to send bytes to radio.
In addition, the microcontroller can configure the CC1000 radio in order to set frequency, output power and other parameters.

CC1000 Microcontroller Interface

Now, we look a bit closer at the CC1000 interface to communicate with the microcontroller.
The CC1000 provides three pins for configuration, one pin for data transmission and one pin for the clocking. Figure 2-3 shows the interface schemata. PDATA (Programming Data), PCLK (Programming Clock) and PALE (Programming Address Latch Enabled) are used by the microcontroller to configure the radio. Together they form the transceiver's configuration interface.
The bidirectional pin DIO (Data Input/Output) is used for data reception and data transmission from and to the microcontroller. DCLK (Data Clock) provides the data timing.
The RSSI/IF pin is connected to the microcontroller's analog to digital converter. It is used to measure the received signal strength (see Section 2.2.3.3).

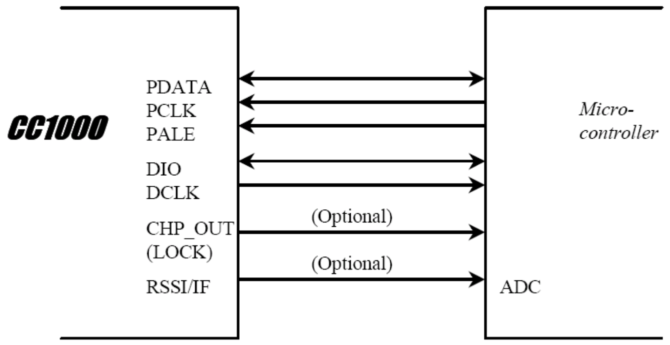


Figure 2-3
CC1000 microcontroller interface providing three configuration pins (PDATA, PCLK and PALE), one data pin DIO, one clocking pin DCLK and an analog RSSI output pin RSSI/IF. [6]

ATmega128 Serial Peripheral Interface

The communication between the CC1000 transceiver and the ATmega128 microcontroller runs over the Serial Peripheral Interface (SPI).
The SPI allows high-speed synchronous data transfer between the ATmega128 and peripheral devices. The SPI offers a full-duplex data transfer. A big benefit is the interrupt capability of this interface. After the transmission of every data byte over SPI, an interrupt occurs. This is crucial since the microcontroller just has to wait until an interrupt occurs to know that data arrived from an external component e.g. from the CC1000. After an interrupt occurred, the received data byte is stored in a register on the microcontroller and can be read.

To transmit data from the ATmega128 to the CC1000, one single data byte is stored in a designated register of the ATmega128. It will automatically be transfered via SPI to the CC1000 where the data byte will be sent into space. Here too, a SPI interrupt is triggered signaling the microprocessor to send the next data byte.

2.2.3.3 Received Signal Strength Indicator

Received Signal Strength Indicator (RSSI) is an indication of the received power level that is measured by the antenna. RSSI is an arbitrary value dependent on the specific device. Generally, the higher the RSSI level, the stronger the received signal. RSSI values are intended to be used at the physical and data link layers. For example some MAC protocols require this indicator to determine whether the channel is clear to send or if another device is using the channel.
It is possible to convert the RSSI into field energy (in dBm) to make the received signal strength comparable among different radios.

CC1000 has a built-in received signal strength indicator. When the RSSI function of the CC1000 is enabled, the chip gives an analog output signal at a the RSSI/IF pin. The Analog to Digital Converter (ADC) on the Atmega128 can measure the pin's output voltage over a resistor and convert it to a digital voltage level.

The measured RSSI output voltage is inversely proportional to the input RF signal strength. A higher voltage means a lower input RF signal strength. This fact is illustrated in Figure 2-4

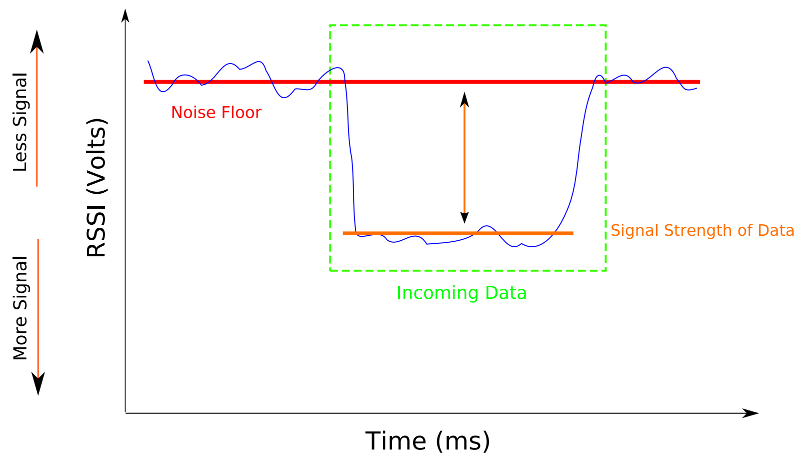


Figure 2-4
The RF signal strength is inversely proportional to the measured RSSI output voltage.[14]

A single analog to digital conversion is started by writing a logical 1 to the ADC Start Conversion 1-bit register on the microcontroller. This bit stays high as long as the conversion is in progress and will be cleared as soon as the conversion is completed. After conversion, the 10 bit digital value is stored in two 8-bit registers and can easily be accessed.

To retrieve the signal input power in dBm out of the measured RSSI voltage value, a simple conversion is needed. As mentioned before, RSSI values are not standardized. Therefore the conversion is different for each device type and highly hardware dependent. For the Chipcon CC1000 transceiver, a plot of RSSI voltage as a function of input power is shown in figure 2-5. A good approximation can be achieved by using the following conversion formulae:

$$\begin{aligned} P &= -51.3V_{\text{RSSI}} - 49.2 \quad [\text{dBm}] \qquad \text{at 433 MHz} \\ P &= -50.0V_{\text{RSSI}} - 45.5 \quad [\text{dBm}] \qquad \text{at 868 MHz} \end{aligned}$$

The CC1000 has the capability to detect and distinguish signal strength in a range from -105dBm to -45dBm.

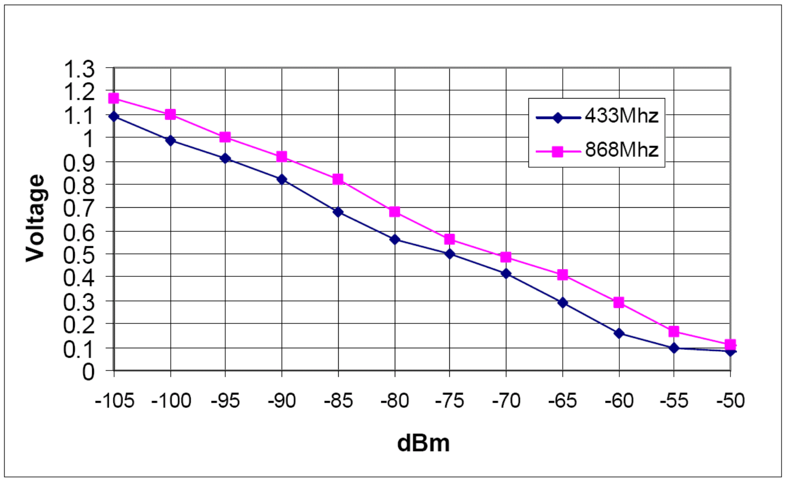


Figure 2-5
Received signal strength vs. RSSI voltage

2.2.3.4 Transmission Frequency

The transmission frequency of the CC1000 is set by programming the correspondent configuration registers. There are two frequency words, namely A and B, that can be programmed with two different carrier frequencies. One of the frequency words can be used for RX (local oscillator frequency) and the other for TX (transmitting frequency). This makes it possible to switch very fast between RX mode and TX mode.The CC1000 uses a Frequency Shift Keying (FSK) modulation scheme. Not only the carrier frequency f_{RF} can be programmed, but also a frequency separation f_{SEP}.

Figure 2-6 gives an illustration of all frequency parameters. In this section we always refer to this illustration.

Calculating the Frequency Parameters

In order to calibrate the radio to a certain frequency, a carrier frequency value FREQ and frequency separation value FSEP need to be programmed. The following steps show the procedure to calculate these settings. We start by defining some additional parameters:

f_{RF}:	The radio frequency f_{RF} is the carrier frequency. It can be chosen between 300Mhz and 1000Mhz.
f_{IF}:	The intermediate frequency f_{IF} is 150kHz. This value is fix and can not be changed.
f_{VCO}:	f_{VCO} is either the Local Oscillator (LO) frequency f_{LO} in receive mode, or the f_0 frequency in transmit mode.
f_{SEP}:	The frequency separation f_{SEP} is the distance in Hz, between lower and upper FSK frequency.
f_0:	f_0 is the lower FSK frequency.
f_1:	The upper FSK transmission frequency is given by $$f_1 = f_0 + f_{\mathrm{SEP}}$$
f_{LO}:	The LO frequency f_{LO} must be equal to $f_{\mathrm{RF}} - f_{\mathrm{IF}}$ or $f_{\mathrm{RF}} + f_{\mathrm{IF}}$ giving low-side or high side LO injection respectively.

These parameters can't be stored on the CC1000 directly. The CC1000 needs to know the values with reference to the internal crystal oscillator clock. Therefor a transformation need to be done.

1. The reference frequency f_{ref} is the crystal oscillator clock frequency f_{OSC} divided by REFDIV, a number between 2 and 15. REFDIV should be chosen that
$$1.00\mathrm{MHz} \leq f_{\mathrm{ref}} \leq 2.40\mathrm{MHz}$$
Thus, the reference frequency f_{ref} is given by:
$$f_{\mathrm{ref}} = \frac{f_{\mathrm{osc}}}{\mathrm{REFDIV}}$$
2. FSEP is related to the frequency separation f_{sep}. The equation
$$f_{\mathrm{sep}} = f_{\mathrm{ref}} \cdot \frac{\mathrm{FSEP}}{16384}$$
gives the relation. FSEP has to be chosen to fulfill this condition.
Therefore
$$\mathrm{FSEP} = \frac{16385 \cdot f_{\mathrm{sep}}}{f_{\mathrm{ref}}}$$
3. The frequency word FREQ can be calculated from:
$$f_{\mathrm{VCO}} = f_{\mathrm{ref}} \cdot \frac{\mathrm{FREQ} + \mathrm{FSEP} \cdot \mathrm{TXDATA} + 8192}{16384}$$
where TXDATA is 0 or 1 in transmit mode depending on the data bit to be sent. In receiving mode TXDATA is always 0.
4. The calculated values FREQ, REFDIV and FSEP can be stored in CC1000 registers according to Table 2-1.

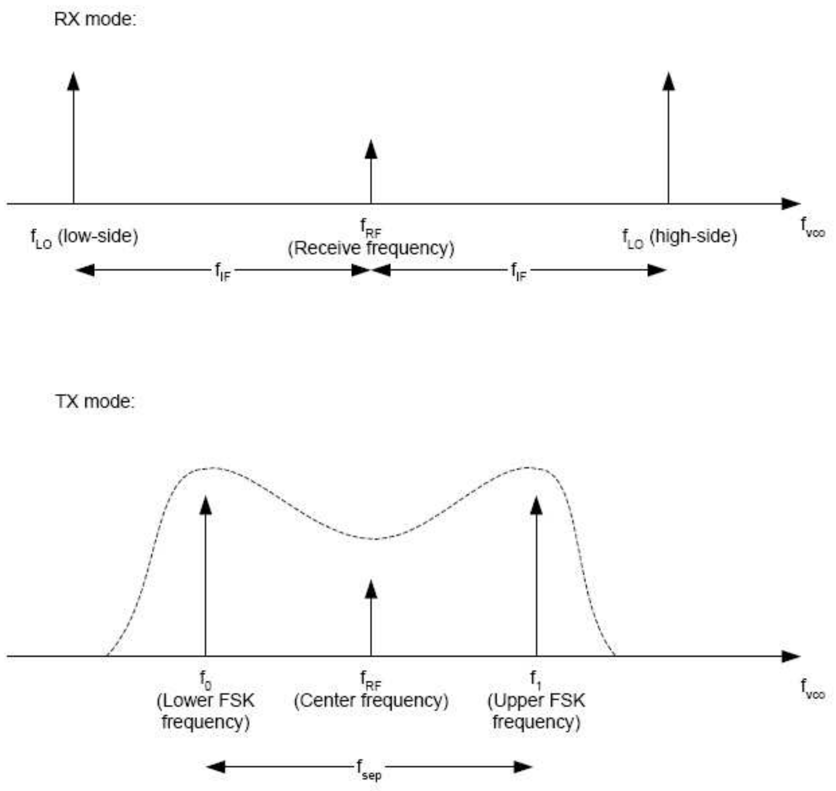

RX mode:
f_{LO} (low-side)
f_{RF} (Receive frequency)
f_{LO} (high-side)
f_{vco}
f_{IF}
f_{IF}
TX mode:
f_0 (Lower FSK frequency)
f_{RF} (Center frequency)
f_1 (Upper FSK frequency)
f_{vco}
f_{sep}

Figure 2-6
Illustration of the frequency parameters in RX and TX mode respectively.

Address	Register	Name	Description
01h	FREQ_2A[7:0]	FREQ_A[23:16]	8 MSB of the frequency control word A
02h	FREQ_1A[7:0]	FREQ_A[15:8]	Bit 15 to 8 of the frequency control word A
03h	FREQ_0A[7:0]	FREQ_A[7:0]	8 LSB of the frequency control word A
04h	FREQ_2B[7:0]	FREQ_B[23:16]	8 MSB of the frequency control word B
05h	FREQ_1B[7:0]	FREQ_B[15:8]	Bit 15 to 8 of the frequency control word B
06h	FREQ_0B[7:0]	FREQ_B[7:0]	LSB of the frequency control word B
07h	FSEP1[2:0]	FSEP_MSB[2:0]	3 MSB of frequency separation control
08h	FSEP0[7:0]	FREQ_LSB[7:0]	8 LSB of frequency separation control
0Ch	PLL	REFDIC[3:6]	Reference divider (4 bit)

Table 2-1: Frequency configuration registers

2.2.3.4.1 Frequency Setting in ISM Band

With the previously mentioned procedure we are able to set the CC1000 to a desired frequency. However, it is recommended to use predefined settings for the operating frequencies to ensure optimal configuration for best sensitivity in receiving mode. Chipcon offers a tool to generate an optimal setting for a certain radio frequency.
A table with sample settings is given in [6]. (See Section **??**)

2.2.3.5 RF Output Power

The RF output power is programmable and controlled by the CC1000 PA_POW register. Table 2-2 shows some values for output power and the typical current consumption at this level.
The complete table is given in [6].

Output power (dBm)	RF frequency 868 MHz	
	PA_POW (hex)	Current consumption (mA)
-20	02	8.6
-15	05	9.3
-10	09	10.1
-5	40	13.8
0	80	16.8
5	FF	25.4

Table 2-2: Output power settings and typical current consumption at 868Mhz.

2.3 Deployment-Support Network

We are now ready to use the BTnode hardware and develop applications. However, it is infeasible to run distributed applications in a WSN with the lack of a central control entity. Our application requires such a system. In this section we present the WSN control platform used in this thesis.

Classic approaches to develop and deploy WSNs use serial cables for reprogramming, testing, monitoring and controlling of the sensor nodes. Although successful in lab setups this method does not scale very well. When a WSN is getting larger than just a few nodes, prototyping and surveillance using serial cable connection to each sensor node is not feasible anymore.
In [11], the Deployment-Support Network (DSN) platform is proposed as a tool for the deployment of large WSNs. The DSN is a wireless cable replacement that allows controlling and validation of WSN applications and to monitor arbitrary WSN target devices in a real world scenario. The DSN provides a virtual connection to each node in a WSN. Figure 2-1 illustrates the concept of the DSN.

The DSN consists of nodes, referred to as DSN nodes, which are attached to the investigated sensor nodes, the so-called target nodes. Both, the DSN and the target nodes span up an independent wireless network. The DSN network can be considered as a wireless backbone network. A host, e.g. a personal computer, can be used to access the DSN by connecting to one of the DSN nodes. With this setup, a target node can be controlled, monitored and programmed as if it were connected to a host computer using a cable. In [10] an implementation, referred to as JAWS, of such a DSN was developed based on the BTnode platform. It uses Bluetooth to construct a tree-based multi-hop wireless network.

We use the JAWS DSN as a service to run and analyze link measurements in a WSN.

2.4 DSNAnalyzer

The DSNAnalyzer [12] is a graphical backend software application for the DSN that allows controlling and monitoring the target nodes over a GUI. Additionally, the DSNAnalyzer features a system to generate, run and analyze link-quality tests among target nodes. Table 2-3 shows the functionality of the DSNAnalyzer.

The DSNAnalyzer is in use to evaluate the A80 target node at SBT (See Section 1.3.2). Since we are interested in comparing the BTnode to the A80, it is recommendable to use the same analysis tools. The the DSNAnalyzer fully achieves all our requirements for link-quality measurements and has therefore been chosen as the tool to go with. Figure 2-7 illustrates the concept of the DSNAnalyzer. The BTnode targets are connected to the DSN that acts as a wireless cable replacement providing virtual connections to each node. With the host computer running the DSNAnalyzer software, we are able to directly communicate with each target node.

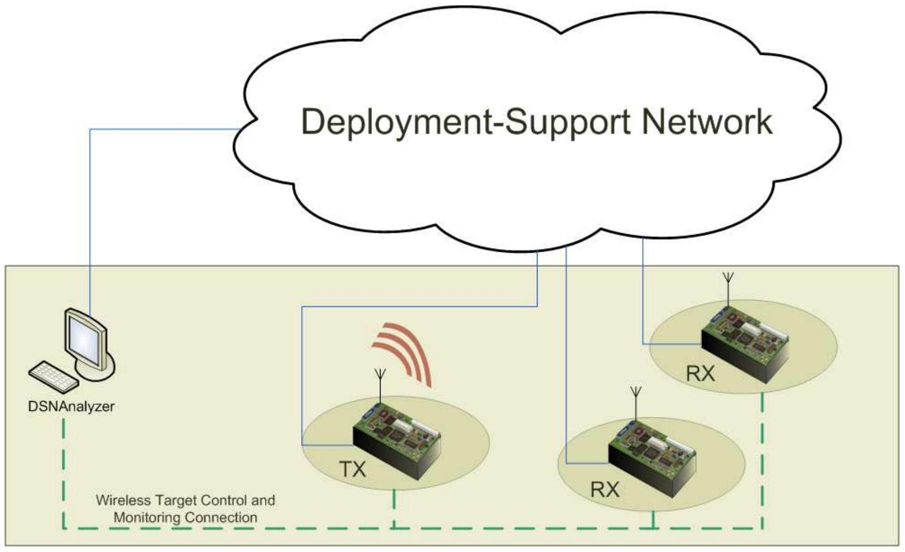

Deployment-Support Network
DSNAnalyzer
TX
RX
RX
Wireless Target Control and Monitoring Connection

Figure 2-7
DSNAnalyzer provides a virtual connection to every target node for link-quality measurements

Use Case	Description
Create Test	The operator creates a test for link-quality measurements which is stored in a text file. This test includes parameters like the number of packets, the period, the sending power and the participating Targets.
Execute Test Series	Execute several tests one after another by specifying the tests, their order and an optional delay before and after the test.
Execute Test	The operator executes a test that is specified in a text file.
Program Target	The operator loads a new binary file to a Target and flashes it.
Program Targets	Program several Targets with a new binary.
Control Target	Control a Target by setting its power state.
Execute Target Command	Execute an arbitrary command on one or a set of targets.
View Trace	The operator loads a trace of events from the DSN-Server of an external trace file. The events are displayed in different representation in several windows.
Analyze Test Result	Use charts to display the results of a test including the ranges of the signal strength of the correct received frames, the faulty received frames and the noise.
Analyze Link	Show the quality of a specified link.
Analyze Packet Errors	Show detailed information about the time and quantity of frame errors.
Analyze Bit Errors	Show detailed information about the time and quantity of bit errors.

Table 2-3: Functionality of the DSNAnalyzer [12]

3

Concept

3.1 Overview

In this chapter, we develop a concept for our Link-Quality Measurement (LQM) application on the BTnode. We will first state the requirements and then present our approach.
The first section devotes itself to specify link-quality measurements. In the following section, we deal with the concept of data analysis and finally, in the last section we investigate the node-controlling and data-gathering issue in a WSN.

3.2 Link-Quality Measurements

3.2.1 Scenario

Link measurements are a method to gain insight into the radio-channel behavior. The basic concept is to have one sending and at least one receiving node in a WSN. The radio of the transmitting node sends data on a certain radio channel whereas the receiving node is continuously listening to this channel. If data has been detected by the receiver, it is logged and preprocessed for analysis. Additional measurements, such as RSSI, can be recorded for more accurate interpretations. The measurement analysis should yield to a prediction of the link quality between nodes. In the following section we will develop a concept for a test procedure to perform such LQM. This test procedure should run on the testbed described in Chapter 2. We start by defining the requirements and will later present our own approach.

It is important to define the right measurement methods and to specify what exactly should be measured in a link-quality test in order to be able to do the right analysis afterwards.
The very basic concept behind our approach for such link-quality measurements is that a receiver analyzes and verifies data received from a sender. A fundamental decision that has to be made is whether to go for bit error statistics or for packet error statistics for link measurements by analyzing the received packets.

By performing a Bit Error Test (BET), the sender transmits a sequence of predefined bits. The receiver tries to detect a data stream and hopefully finds the expected bit sequence. Because of the receiver's knowledge about the expected bit sequence, single bit errors can be detected just by comparing the received stream with the expected sequence. It is even possible to locate a single bit error in the data stream.

With the Packet Error Test (PET) approach one is not interested in detecting each bit error. Most modern communication protocols are packet oriented. If one bit isn't received correctly and can't be corrected with redundancy, the whole packet is useless and considered as defective. The packet has to be discarded. The amount of bit errors is not important.
Packet error tests are therefore only interested in whether a packet was received correctly or not. No further byte or even bit investigation of a faulty received packet is done.
In order to assure the correctness of a received packet, the Cyclic Redundancy Checksum (CRC) of the the received data is calculated and compared to the CRC value transmitted with the packet. Cyclic redundancy check is a type of hash function to generate a checksum, that allows the detection of errors in a packet. If both, the received and the computed values are identical, then there is a very high probability that the received data bytes are equal to the sent data and have therefore not been corrupted on the way from the transmitter to the receiver.

The big advantage coming along with CRC comparison is the flexibility in the packet structure. Not the packet content itself is being verified on receiver side but a checksum. This enables the possibility to add variable data to a packet. Such data could include the transmitter's unique hardware address, a sequence number for each packet and other relevant information.
This additional information makes it possible to set up a tracer:

- With a sequence number it is possible to detect which packets got lost. Without, it is only possible to say at the end of a test how many packets out of the total contained errors or did not make it to the receiver. A sequence number included in every packet allows analyzing the packet losses time dependency to see whether packet loss occurred in bursts or are randomly distributed over time.
- The sender ID helps to find out what source the received packet is coming from. Even if only one transmitting device is specified in a link test, this information is useful in order to filter the measurements.

With this information, the complete packet flow is traceable. Each packet is uniquely identifiable.
Obviously, the ability to compute the Bit Error Rate (BER) is not possible with this method. Thus, one could think of combining BET and PET by investigating a predefined byte sequence within the packet to count bit errors. This would require a more complex application where the receiver needs to know what packet is expected next in order to verify that no error occurred in the variable data of the packet. We did not implement such a system, because we believe that BER isn't more beneficial than the Packet Error Rate (PER) in link-quality measurements.

We are interested in knowing how many packets did arrive at the receiver and which packed couldn't be detected. However, it is also interesting to see under what conditions errors occurred. For such an investigation, additional data need to be measured and included into the test. In [15] it is stated that RSSI is an important value to investigate the radio channel behavior.
The test procedure can be extended in a way that if a packet arrives, the receiver not only checks the correctness of the packet but also measures the channel's RSSI value. In fact, two different RSSI values per packet should be determined: One value measured during the reception and the other shortly after the packet arrived. The second value is considered as the noise RSSI and gives prediction about the current noise situation of the channel when no transmission is going on.

3.2.2 Approach

Link-quality tests in WSNs have been performed by SBT and we would like to have a similar setup using the same methods with identical parameters to compare the results.
On the other hand, we would like to extend the test method to obtain more useful results for better interpretation. In the next section, we will first present our own approach to perform link-quality measurements and then compare it with the way Siemens measures the link quality.

Test Setup

The link measurement setup consists of one sending and at least one receiving node in a WSN. Each node is able to change freely between these two functions. It is therefore possible to run tests in a round-robin scheme where each node is once transmitter and receiver otherwise.

Node Configuration

We would like to analyze the effect of different radio configurations. Therefore, several parameters can be configured. Table 3-1 shows the configuration parameters and their range.

Parameter	Description	Range
chan	Radio channel	0 - 79
power	Transmit power	0 - 26
iter	Number of test packets to be sent	1 - 65535
txper	Interval between the transmission of two packets [ms]	100 - 65535
preamb	Preamble in 8-bit units	4 - 255

Table 3-1: PET configuration parameters

PET Data Structure

Our attempt lies in setting up a tracer. Therefore, we follow a Packet Error Test approach. We are not interested in detecting single bit errors. It is sufficient enough

to identify whether a packet was received correctly or not by comparing CRC values. The ability to add variable data to a packet allows us to set up a tracer by merging a sender identification as well as a sequence number into each packet.
Table 3-2 shows the data that is included in each test packet. The packet structure

Data	Description	Size
Sequence Number	Number of the packet in a PET	2 Bytes
Sender ID	Unique identification number (2 Least significant bytes of the Bluetooth MAC address of the BTnode)	2 Bytes
Payload	A series of predefined bytes as additional payload	26 Bytes
CRC	Packet's CRC value to detect errors at the receiver	2 Bytes

Table 3-2: Packet error test enhancements to provide tracer ability

is given in Figure 3-1

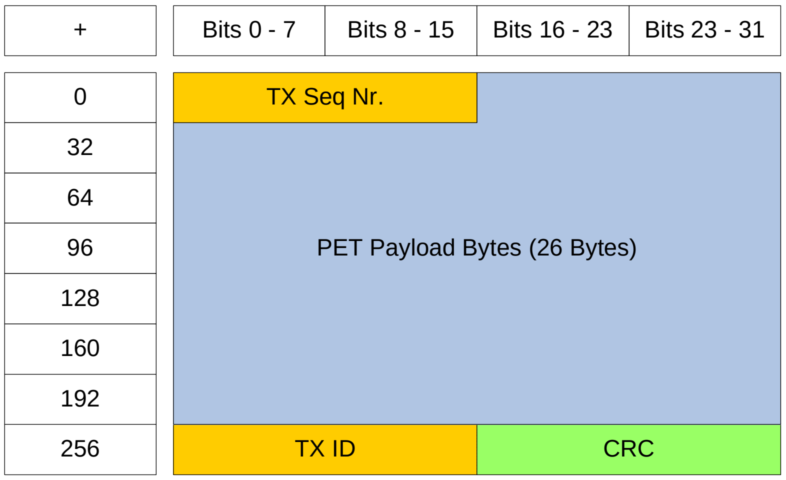


Figure 3-1
Test packet description

PET Transmitter Concept

The transmitter initializes itself with the previously configured parameters and starts a test. During a test, the transmitter is sending packets with a predefined packet rate.
In order to transmit such a packet, at first, a preamble sequence is sent directly followed by a Start Frame Delimiter (SFD) for synchronizing sender and receiver and also for signaling the beginning of the packet. Next, the packet is sent byte per byte according to **??**. After the transmission of the 32 bytes, the sender powers down the radio and waits a certain time before starting to send another test packet. Of course the preamble sequence as well as the SFD have to be transmitted again for receiver synchronization.

PET Receiver Concept

The receiver is permanently observing the radio channel. If a preamble sequence has been detected, the receiver waits until the SFD is registered. This signals the start of a packet and the receiver begins to record the next 32 bytes to catch the whole packet. The receiver verifies the obtained packet by CRC comparison.
Just after detecting the SFD, the receiver measures the RSSI value. This values is the signal strength during reception of a packet. After receiving the packet, the RSSI is measured again. This value is the signal strength of the noise.

3.2.3 Comparison to the Siemens' Approach

Siemens does not perform a packet error test but a bit error test for link measurements. Therefore, they can not add a sequence number and a sender ID to their packet. Siemens sends a frame of 26 predefined bytes and verifies them by comparison. There is no need to add CRC since the packet content is known by every participant in a test.

3.3 PET Data Analysis

In the previous sections we presented the concept of our BTnode application to perform PETs for link measurements. A further step is now to analyze this huge amount of date generated by the nodes during a test. In this section we will discuss what actually need to be analyzed in order to gain the most significant results out of a test.
To summarize, what we want to do is to trace packets in a WSN and analyze under what condition they are received correctly.

During a test, packets are sent from a transmitting node to receiving nodes. Each packet carries a sequence number, a sender ID, a predefined sequence of bytes and CRC as payload.The RSSI is measured twice for each received packet: during reception and shortly after to cover the actual noise floor.
We therefore log the following statistical values for later analysis:

Statistical Values of each received Packet

- The sequence number and the sender ID of each correctly received packet, its RSSI value and the noise value.
- The sequence number and the sender ID of each faulty received packet, its RSSI value and the noise value.

This information is logged automatically and sent back to the host after each reception of a packet but can also be deactivated if desired.

Statistical Values of a whole Packet Error Test for each Node

- The total number of correctly received frames
- The total number of faulty received frames

- The total number of missed frame (The total number sent frames subtracted by the number of received frames)

This information is logged and sent back to the host upon request after the transmitter has sent all its test packets.

In Section 2.4, the DSNAnalyzer was introduced as a tool for performing and visualizing link-quality measurement results. The DSNAnalyzer is capable of processing the above mentioned data set and provides plotting functionality to draw diagrams out of the data.

3.4 Data Collection and Node Control in WSN

We would like to have a direct connection to each node in the WSN. It should be possible to remote control and monitor a target.

3.4.1 Client - Target Communication

To communicate from a host computer with a WSN node, a communication mechanism has to be specified.
The idea is to establish a virtual connection to each node. We use the DSN for connection handling. Further, the DSN, working as a background service, is responsible for forwarding data streams from the client to the target and vice versa. For the host-node communication, a client-server protocol is used with the host as client and the target as server.
Figure 3-2 shows the connection schematics. For the communication between the

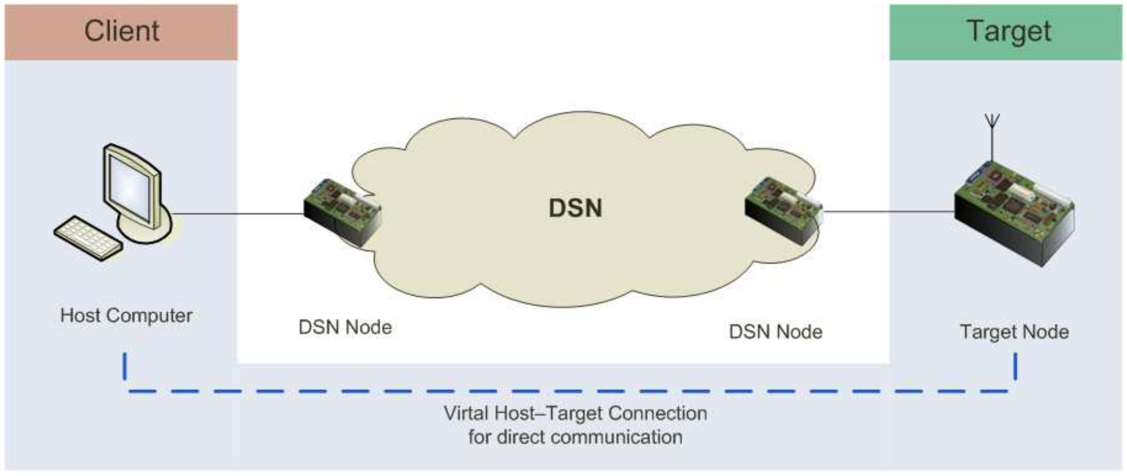
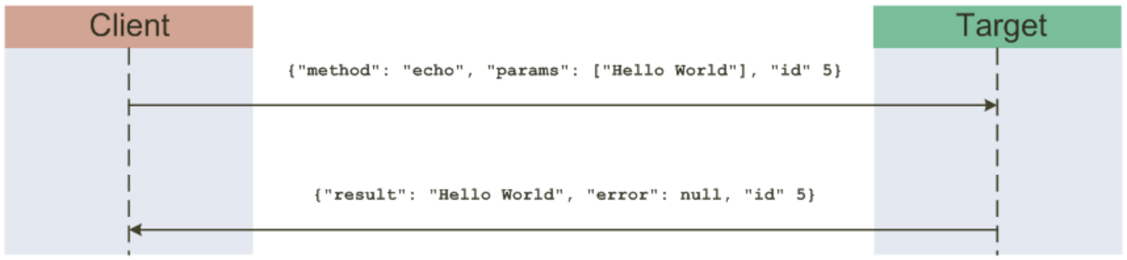


Figure 3-2
The DSN establishes a wireless virtual connection from a host computer acting as client to a WSN node acting as server

DSNAnalyzer and the target, a Remote Procedure Call (RPC) interaction has been chosen [12].
A simple lightweight RPC protocol is JSON-RPC [2]. JSON is an alternative to XML for transmitting structured information between client and server. A JSON-RPC parser does not require lot of resources and the JSON-RPC protocol is slim with less

overhead compared to XML.
JSON-RPC uses requests from the clients that are answered by responses from the server. A request includes the name of the method, the parameters and an ID. A response includes the result, an error message and the ID of the request. The exact data flow from the client to the target with a simple RPC interaction is shown in Figure 3-3.



```
{"method": "echo", "params": ["Hello World"], "id" 5}
{"result": "Hello World", "error": null, "id" 5}
```

Figure 3-3
JSON-RPC client/target sequence diagram

3.4.2 DSN Node - Target Communication

By using an RPC interaction between host and target as mentioned in Section 3.4.1, the target receives a plain string in JSON-RPC notation from its attached DSN node. This string can easily be parsed. To send logs back to the host, the target writes the data in JSON-RPC notation to the standard output stream. It will automatically find its way to the host.
Framing and stuffing of data is handled by the DSN itself and does not concern us.

4

Implementation

4.1 Overview

In this chapter we cover major implementation topics. We start by explaining some important mechanisms of the BTnut system software in order to understand the subtleties of our application. We then provide the basic idea behind the framework of our application before we focus on implemented key procedures.

4.2 Basic Mechanisms of BTnut

We introduced BTnut system software in Section 2.2.2 and listed its main features. In this section we give a very short introduction to the most relevant features we used for developing our application.

4.2.1 Cooperative Multithreading

BTnut supports the usage of multiple threads by switching CPU from the context of one thread to the context of another thread. This gives the appearance of simultaneously running threads. The thread with the highest priority always runs if it is not waiting for any event or explicitly yielding the CPU. Cooperative multithreading means, that a running thread is non-preemptive and only stops running if either an interrupt occurs or if it has to wait for an event.

4.2.2 Events

Threads are running as long as there is something to do. Very often, a thread does not have to work all the time but instead has to wait for something to happen. BTnut provides an event queue mechanism. Threads can line-up in such a queue when they are waiting for an event while other threads or interrupts can post events to these queues to wake up waiting threads.

4.2.3 Interrupts

Even if the running thread is not willing to pass the CPU to another thread, interrupt routines are executed immediately. If an interrupt occurs, an interrupt handler is called. This handler is similar to a function and has to be programmed by the developer. However, this handler has restricted access and is not able to execute all API functions. Interrupts can be disabled if desired.

4.3 PET Application Framework

In this section we present the framework of our application.
For better structure, we split our application into three modules:

- **PET module**
  Run packet error tests, manages packet flow and handles data transmission.
- **GATEWAY module**
  Communicates with the DSNAnalyzer by parsing JSON-RPC commands and generates log messages.
- **CONTROL module**
  Provides functionality for monitoring and controlling the target and packet error tests.

The PET module is implemented as a thread providing the flexibility to run multiple procedures in parallel. The CONTROL module and the GATEWAY module support the PET module and are together implemented as a second thread. The two threads are able to communicate with each other. Figure **??** illustrates the idea behind our basic framework.

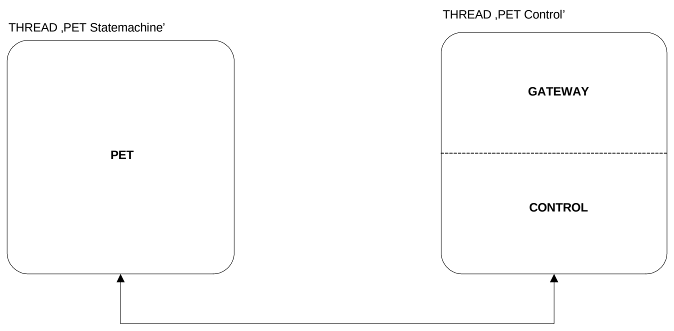


Figure 4-1
The principle behind the application framework: One thread to run PETs and another thread for its controlling

We will now step by step extend this framework towards the whole application. For a first glimpse see Figure **??**. It is showing an elaborate visualization of our application including all key elements.

4.4 Control Functionality of the PET Application

In this section we define the functionality of our application to control the target node using the DSN. We need functions to change the behavior of the target and we would like to be able to configure different parameters. This section provides a rough description of all implemented remote procedures that can be called by the DSNAnalyzer. For a complete list including description, syntax, parameters and corresponding log messages see Appendix A.

PET Control Procedures

By calling control procedures, we are able to change the mode of a node. We can set a node to receiving mode or start sending packet by putting a node into transmission mode. While being in either receiving or transmitting mode, it is possible to abort a test with a stop command. To reset a test (not the target node itself), a command to reinitialize the node is provided.

Command	Description
bet.tx_on	Starts the transmission of packets
bet.rx_on	Reset the results and start with reception of frames
bet.stop	Stop PET
bet.init	Sets all parameters to its values

Set Procedures

This procedures allow to change internal PET parameters.

Command	Description
set.chan	Sets the radio channel to be used
set.power	Sets the CC1000 transmit power
set.iter	Sets the number of test packets to be sent
set.txper	Sets the period test packets are transmitted
set.preamb	Sets the preamble in 8-bit units
set.i_output	Sets the output mask to be used

Get Procedures

Get procedures return data to the DSNAnalyzer. This functions are called to retrieve the configuration setting or to log PET results after a test.

Command	Description
get.betres	Logs PET results
get.betpar	Logs the PET parameters which are currently set
get.rssi	Returns the RSSI value

4.5 PET State Machine Thread

By referring to Figure **??**, we now deal with the most important module: The PET module. This module provides all the functionality that is directly related to sending and receiving data using the CC1000 transceiver. The module is also responsible for managing the packet flow of the application.

4.5.1 PET State Machine

We implemented the packet error test procedure as a state machine with 3 states An illustration is given in Figure **??**

- `PET_IDLE`: Initializing and configuration state
  While in `PET_IDLE` mode, the thread waits for changing to another state. The configuration of a PET can only be done in this state.
- `PET_TRANSMIT`: Sending packets
  Entering `PET_TRANSMIT` starts a PET by sending packets according to the configuration. If the PET finishes, the thread changes back to `PET_IDLE` state automatically.
- `PET_RECEIVE`: Receiving and analyzing packets
  If the thread is in `PET_RECEIVE` state, it is permanently trying to detect arriving packets. If a packet was received it gets analyzed and the thread tries to detect the next packet.

Controlling of the PET state machine is implemented in the PET control thread and explained in Section **??**. As soon as the PET control thread forces a state change, the state machine exits the current state and changes into the new one.
The state diagram graph in Figure **??** also shows which RPC commands are allowed to be executed in a state and which force to change state. The PET state machine thread is all the time in one of the three states. If there is nothing to work off, the thread yields CPU and lines-up in a queue waiting for reactivation. In fact, the thread is sleeping most of the time. The thread is only running while sending a packet in `PET_TRANSMIT` or while analyzing a received packet in `PET_RECEIVE`.
The simplified implementation of the state machine thread looks as follows

```
THREAD(PET_statemachine, arg) {
  for(;;) {
    switch(statemachine_state){
    case PET_IDLE:
      NutEventWait(&SIGNAL_BETSM_Do, NUT_WAIT_INFINITE);
    break;

    case PET_TRANSMIT:
      betmac_transmit();
    break;

    case PET_RECEIVE:
      betmac_receive();
    break;
    }
  }
}
```

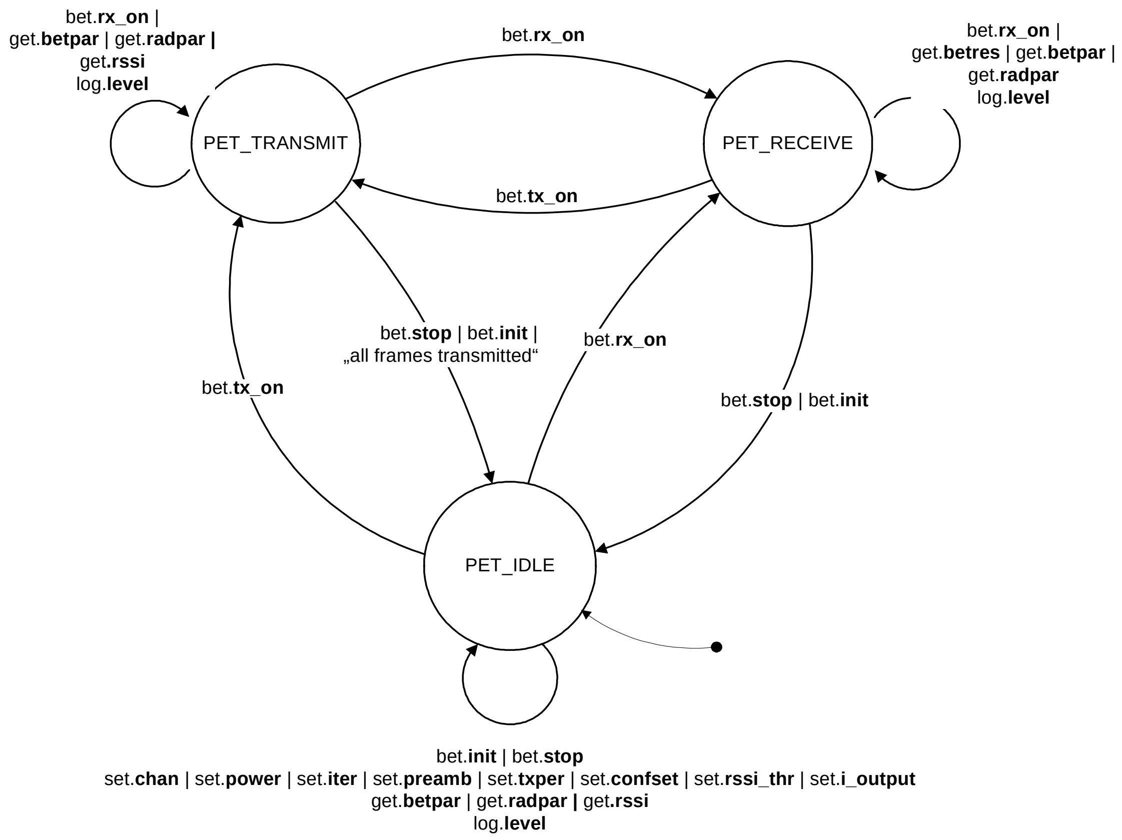

bet.rx_on | get.betpar | get.radpar | get.rssi log.level
bet.rx_on
bet.rx_on | get.betres | get.betpar | get.radpar log.level
PET_TRANSMIT
PET_RECEIVE
bet.tx_on
bet.stop | bet.init | „all frames transmitted“
bet.rx_on
bet.tx_on
bet.stop | bet.init
PET_IDLE
bet.init | bet.stop
set.chan | set.power | set.iter | set.preamb | set.txper | set.confset | set.rssi_thr | set.i_output
get.betpar | get.radpar | get.rssi
log.level

Figure 4-2
PET state graph including procedure calls for state transitions and parameter settings

4.5.2 PET Packet Handling

Packet handling describes the procedure from assembling a packet by the sender until statistical analysis on receiver side.
Packet handling is an integral part of the PET module and is therefore implemented in the PET state machine thread.
In this section we will first introduce the used Media Access Control (MAC) protocol and then present the implementation of the packet flow trough our application on sender and receiver side.

4.5.2.1 PET MAC Protocol

MAC protocols directly interact with the radio module and provide an abstraction layer to upper layer applications to hide the radio handling.
The PET MAC protocol provides only basic functionality:

General Functionality

- Set radio channel
- Set transmission power
- Change radio module between sending and receiving mode
- Reading RSSI values

Sending Functionality

- Assembling packets (Sequence number, Sender ID, additional payload bytes)
- Calculating CRC and add it to the packet
- Sending of the preamble sequence and SFD before transmitting packets

Receiving Functionality

- Receiving bytes from the radio
- Synchronizing receiver and transmitter by detecting preamble sequence and SFD
- Calculating CRC of the received packet
- Providing packets to the upper layer

The CC1000 is not able to detect a whole packet at once but receives it byte per byte and forwards each byte to the microcontroller. It is the duty of the MAC protocol to detect and record packets. It is therefore worth to take a deeper look into how the MAC protocol detects received bytes and how it announces the recording of a whole packet. We use hardware interrupts and their dedicated interrupt handler for this. We implemented an interrupt handler responsible for recording bytes and announcing the reception of a packet.
The CC1000 radio forwards each received byte over SPI to the microcontroller (See 3.4.1). As soon as a byte arrives at the ATmega128, a SPI hardware interrupt is triggered. The interrupt handler becomes active, records the byte and checks if the packet was received completely. If not, the interrupt handler finishes and becomes

active again as soon as the next byte was received.
After the complete reception of a packet, the interrupt handler signals an event to a waiting queue. An enqueued thread wakes up and can process the packet. In our application, only the PET state machine thread is waiting for this event. The SPI interrupt is permanently disabled while not being in receiving mode and will be enabled by changing state to `PET_RECEIVE`. How this procedure is embedded into the RX packet flow is described in Section **??**.

4.5.2.2 TX PET Packet Flow

Figure **??** illustrates the packet flow for transmitting packets. This is identical to the procedure described in Section 3.2.2. In a first step, the packet is generated by assembling the sequence number, the sender ID, the predefined bytes and the CRC. Then, the radio is set to transmission mode and is now ready to send bytes. First, a preamble sequence is sent followed by the start frame delimiter in order to synchronize the receiver to the sender. Now, the radio starts sending the packet payload. After sending the last byte, the radio powers down in order not to occupy the communication channel anymore. A log message about the successful transmission is sent back to the DSNAnalyzer if enabled for this test. This finishes the transmission of one packet. The thread pauses an arbitrary time before sending the next packet accordingly.
This procedure is iterated as often as there are packets to send and as long as the PET state machine is in `PET_TRANSMIT` state.

4.5.2.3 RX PET Packet Flow

Figure **??** illustrates the packet flow for receiving packets. By entering `PET_RECEIVE`, the PET state machine thread turns on the radio on and yields itself waiting for an event. Meanwhile, the SPI interrupt handler is trying to receive packets as described in **??**. When a packet was completely received, the handler signals an event to the waiting queue, to wake up the state machine thread. The thread verifies the received packet and updates the internal statistics. The thread might send an optional log message about the reception of a packet. Then, the thread yields and waits for the next event.

4.6 PET Control Thread

The main task of the PET control thread is to receive JSON-RPC commands, parse them, change PET parameters, control the PET state machine thread and send log messages.
In order to let this thread control the state of the PET state machine thread, we implemented an inter-thread communication mechanism:
In order to change state, the PET control thread sets the new state as active and signals an event to the state machine thread. The state machine thread does not recognize state changes without signaling because otherwise a resource intensive polling method would run permanently.

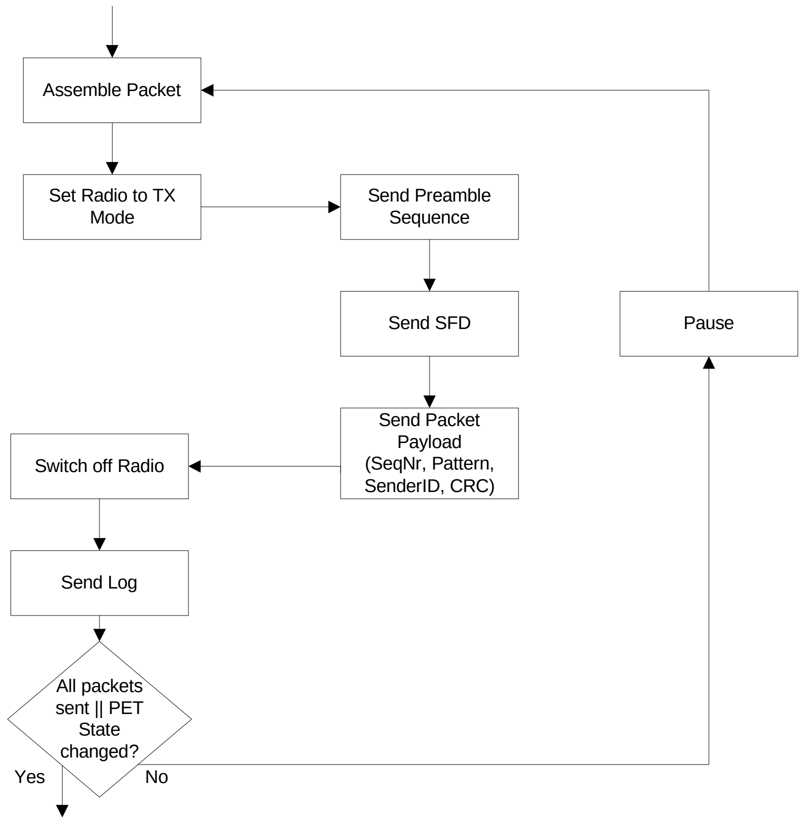


Figure 4-3
TX packet flow: This procedure is iterated for each packet

4.6.1 RPC, Parser and Logger

The GATEWAY module is implemented in the PET control thread and is responsible for the communication between the DSNAnalyzer and the CONTROL module.
As stated in Section 3.4.1, we intend to use an RPC interaction for the communication of our client-server system.
Therefore, the BTnode interprets the received command, executes the corresponding procedure and replies with a log message.
Figure **??** shows the procedure of a sample RPC interaction starting with the arrival of a command and ends with sending the corresponding log message.
In addition, there are some special occurrences where logs are sent back to the host without a request call. Those occurrences are triggered when a packet was sent or received during a test. These log messages are optional and sending them back to the DSNAnalyzer can be switched off by an RPC command (See Appendix A). This is useful in order not to overload the DSN.

SBT has implemented a JSON-RPC parser and a logger that works with on their A80 node. They kindly provided us with their implementation of the parser and logger allowing to use in our application.

We include the parser in such a way, that our application detects arriving RPC command coming from the host and triggers an interrupt to call the interrupt handler.

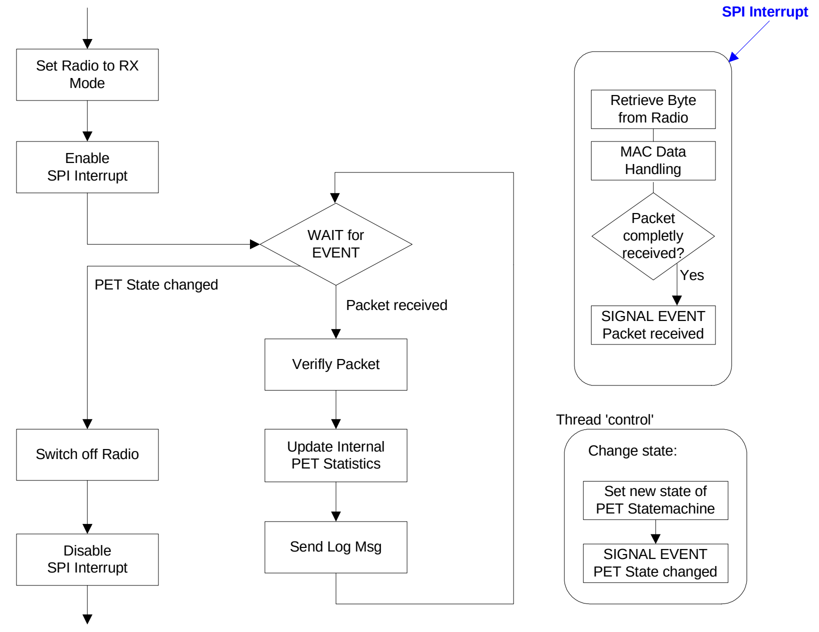
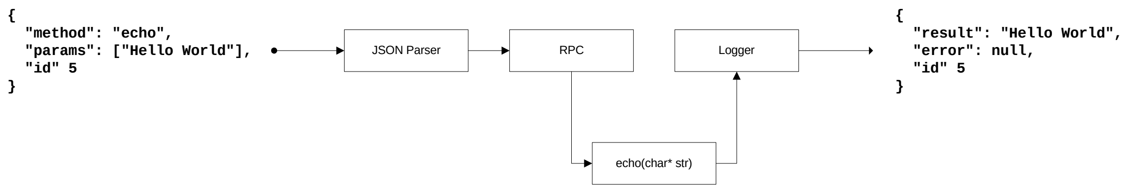


Figure 4-4
RX packet flow: The PET state machine thread is waiting until a packet was received or the control thread changed state



```
{
  "method": "echo",
  "params": ["Hello World"],
  "id" 5
}
```

```
{
  "result": "Hello World",
  "error": null,
  "id" 5
}
```

Figure 4-5
Procedure of handling an JSON-RPC command: The parser interprets the command and the RPC part calls the corresponding procedure. Its return value is sent to the logger, where a message in JSON notation is generated

RPCs are strings received over UART (See 3.4.1). Whenever the target receives a character from the attached DSN node, an interrupt occurs and the UART interrupt handler we implemented is called. Our handler buffers the received character and awakens the control thread to check whether a complete RPC command is stored in the buffer or if there are still characters to expect. If a command did arrive completely, the thread parses the RPC and executes the corresponding procedure. After sending the log messages back to the DSNAnalyzer, the thread yields CPU and enters sleeping mode again.

4.7 Schematic Overview of the PET Application

To sum up, Figure **??** illustrates the interaction between the PET state machine thread and the PET control thread interaction as well as the UART and SPI interrupt handling.

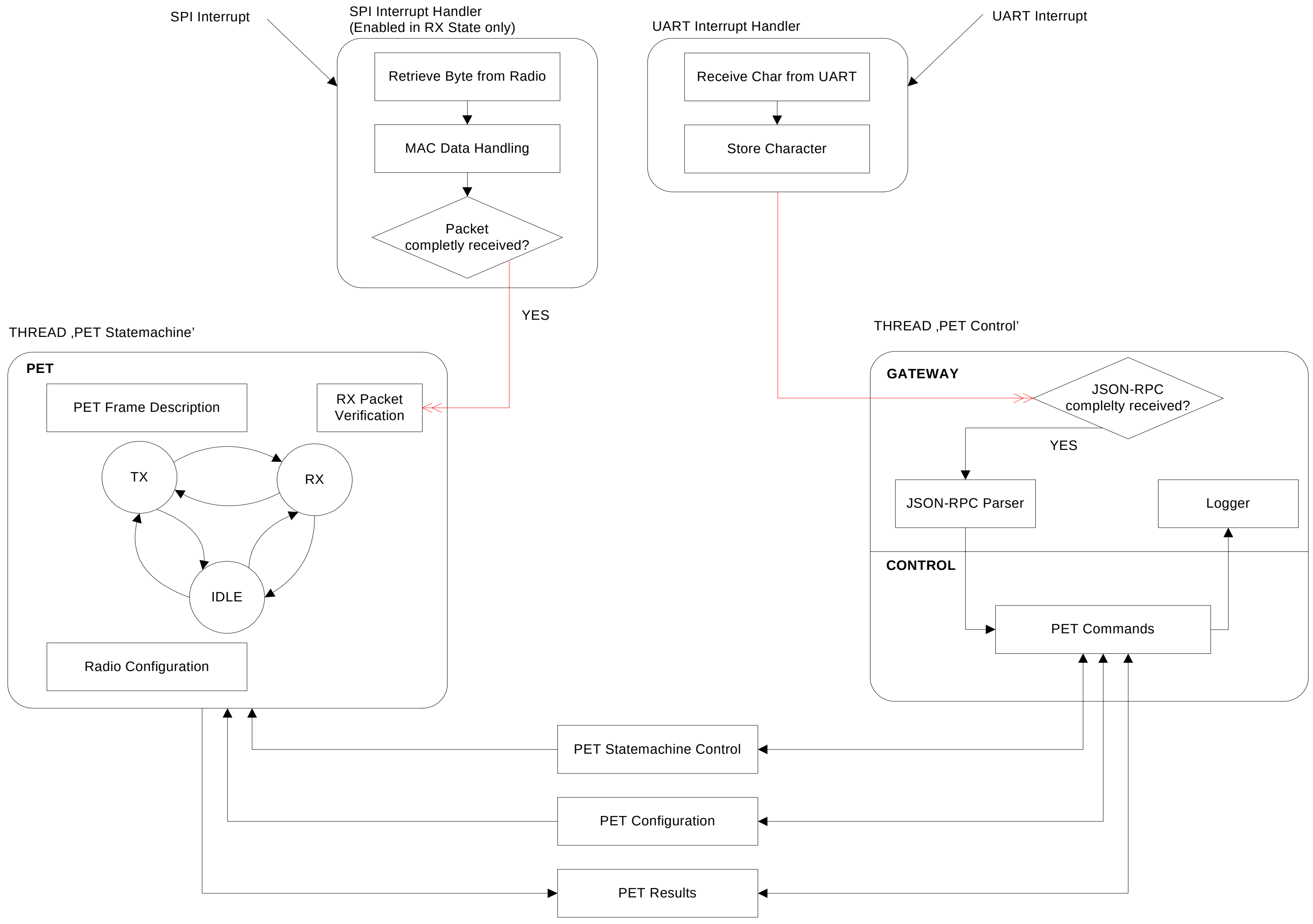


Figure 4-6
Packet error test event handling and inter-thread communication

4.8 Further Implementation Topics

In this section we present several implementation topics that are not directly related to the core application, but nevertheless important for our application.

4.8.1 CC1000 Configuration and Usage

BTnut provides basic functions to configure and use the CC1000 transceiver.
In 2.2.3.4 we have seen that it is rather tricky to program the Chipcon to a certain frequency. Additionally, among all frequency settings, just a few are optimized for best sensitivity. Such frequencies and the corresponding register settings are listed in the CC1000 data sheet or can be calculated with a tool[1] provided by Chipcon. BTnut does not provide calculating frequency register configuration out of MHz values. We therefore decided to implement a look-up table with optimized frequency sets. This table can easily be modified and extended.

4.8.2 Storage of the Bluetooth MAC address in EEPROM

The ATmega128 microcontroller does not have a unique identification number. However, this is required according to our PET specification.
The BTnode features a Bluetooth radio. The Bluetooth module provides a unique 48-bit MAC address for every device. We therefore have chosen the last 16 bits of this address as the number to identify the sender.
We do not want to power up the Bluetooth module at every start-up of the node because it takes approximately 5 seconds to read the address. Therefore, while booting, the BTnode checks whether the ID is stored in the EEPROM[2] or not. If not, the BTnode initializes the Bluetooth module, retrieves the address and stores it directly in
EEPROM.

[1]CC1000 Optimal Frequency Calculator:
http://www.chipcon.com/files/CC1000_Optimal_Frequency_Calculator_1_2.xls

[2]Electronically Erasable Programmable Read-Only Memory (EEPROM), is a non-volatile storage chip to store small amounts of data.

5 Verification and Measurements

This chapter does not provide an in-depth analysis of link-quality measurements but should give a first glimpse and should disclose tendencies in what direction further, more detailed measurements could go.

5.1 Application Verification

Before we can start using our application to measure the link quality in WSNs, we have to be sure that our implementation works reliable. This is an important requirement since we want to be sure that packet losses do not occur due to bugs in our application.
We verified our software and performed a round-robin PET with six BTnodes where each node sends 1500 packets with full transmission power. In a round-robin PET, each node is once transmitter and receiver otherwise. With this method it is possible to measure every link combination among all nodes in a WSN.
We arranged the nodes in a small circle on a table in order to have only a short distance between. This minimizes the influences of reflection, fading and interference and allows testing whether our application works reliable.
A round-robin test with six nodes measures 30 different links. A visualization of the test results is given in Figure 4-1(a). Each bar represents a link and is a plots of measurement results. Visualized are the amount of correctly received packets, the number of faulty received packets as well as the number of missed packets.
Table 4-1 shows a statistical analysis of the test whereas the associated histogram in Figure 4-1(b) illustrates the number of links that correctly received a certain percentage of packets.

The test results show that the reception rate is below 100%. Although a packet only has to overcome a short distance from sender to receiver, external influence cannot be excluded completely. A verification of our application in an EMC [1] cell would give

[1] Electromagnetic compatibility

PET with 30 Links	Absolute	Percentage [%]
Packets sent per link	1500	100
Minimal received packets	1493	99.5
Maximal received packets	1500	100
Mean value of received packets	1497.5	99.8
Standard deviation	1.9	0.1

Table 5-1: Statistical analysis of the the verification test with six BTnodes

a more precise conclusion. However, the results of our verification lies within the accepted tolerance.

5.2 Measurements

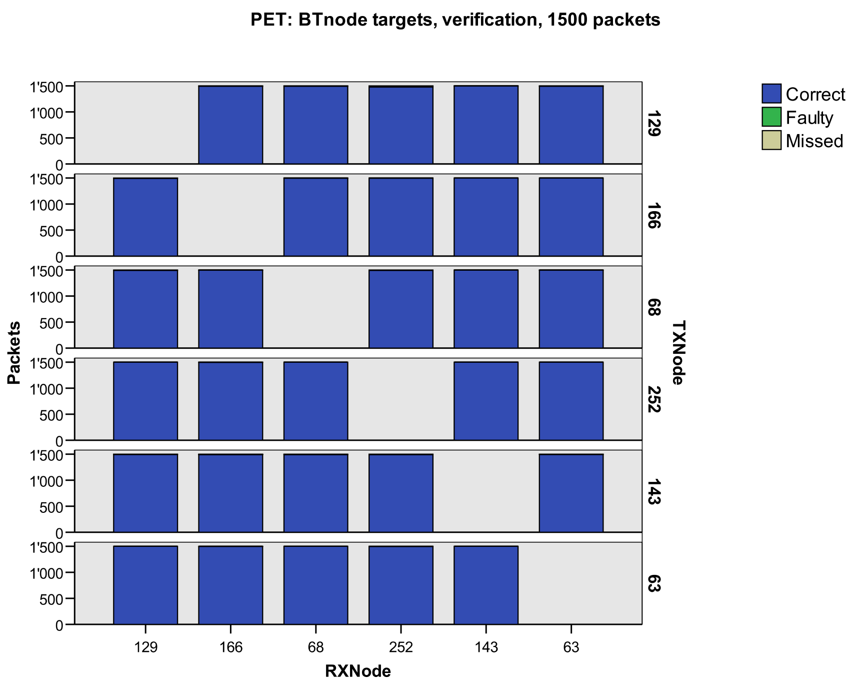
Our interest in measuring the link quality goes beyond analyzing packet error tests performed on a desk. That's why we investigated the channel behavior in a more realistic, office-like scenario.
In addition, we want to compare the link quality between the BTnode and other sensor nodes.

We distributed BTnodes, A80 and Tmote Sky [3] nodes in several offices over the on the ETH ETZ G floor. At each location, a BTnode, an A80 and a Tmote Sky node were placed just next to each other in order to have an identical setup for comparable measurement results. We performed link-quality tests with each sensor node type separately.

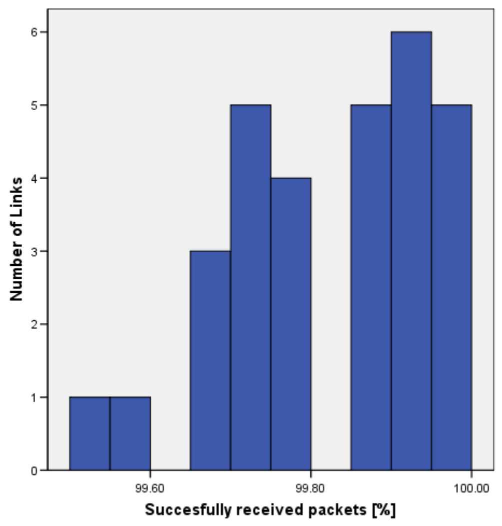
In the next section we present the results of the link measurements of different sensor nodes.

5.2.1 BTnode

We performed a PET with the settings given in Table 4-2. The location of the targets is given in Figure 4-2(b) whereas Figure 4-2(a) shows that the link quality among the nodes vary dramatically. The symmetry along the diagonal axis shows that if the link from an arbitrary node A to another node B is bad, the link in the other direction from node B to A is bad as well. This implies symmetric links. The figure also shows that more isolated nodes have a worse link quality. In particular, node 0063 has only bad link, in both directions.
Another important fact our test revealed is, that a node either receives a very high percentage of the packages or almost nothing. An abrupt transition divides the receiving-area from the non-receiving area. This resulted in having two overlapping clusters in which the included nodes receive almost all packages whereas nodes not including hardly receive anything. The first cluster consists of the Nodes 0129, 0168 and 0068. The second cluster includes the nodes 0166, 0068, 0252 and 0143.



(a) Statistics

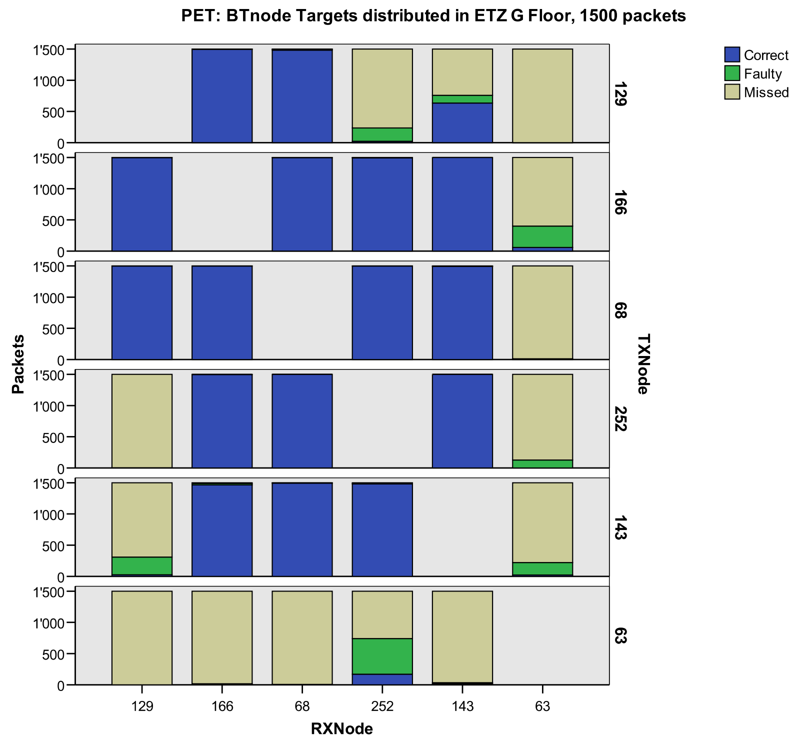


(b) Histogram

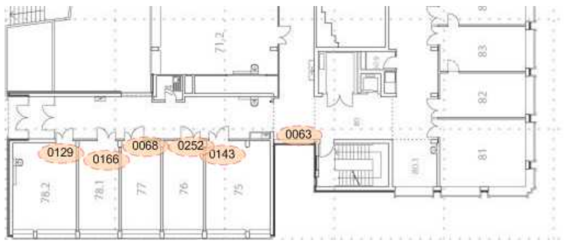
Figure 5-1
Visualization of the application's verification test

Parameter	Value
Packets per link	1500
Transmitting power (Max. power)	5dBm
Frequency	868MHz

Table 5-2: BTnode PET configuration



(a) Statistics



(b) Target placing

Figure 5-2
PET visualization of the BTnode sensor node

5.2.2 A80

We performed the same test we presented in 4.2.1 with the A80 node. The settings are given in Table 4-3. The location of the targets is given in Figure 4-3(b).

The link quality among all nodes in the A80 WSN is very good. The percentage of correctly received packets is almost 100% on all links. Plot (b) in 4-3 shows a magnification of the bar chart for better visualization.

Parameter	Value
Packets per link	1500
Transmitting power (Max. power)	5dBm
Frequency	434MHz

Table 5-3: A80 PET configuration

5.2.3 Tmote Sky

In addition to the BTnode and the A80, we also present link measurements of the Tmote Sky sensor node. The Tmote Sky communicates in the 2.4GHz frequency band using an Cipcon CC2024 radio.
Similar to the BTnode measurements, there is a symmetry along the diagonal axis identifiable. This implies symmetric links.

Parameter	Value
Packets per link	1000
Transmitting power (Max. power)	0dBm
Frequency	2.4GHz

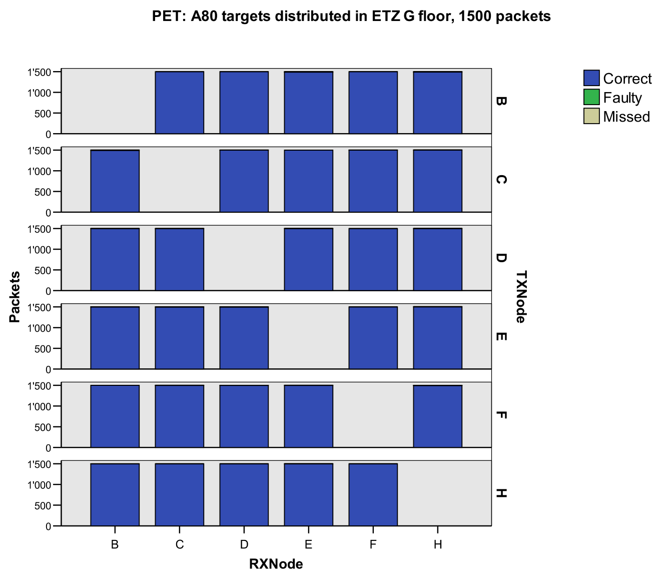
Table 5-4: Tmote Sky PET configuration

5.3 Conclusion

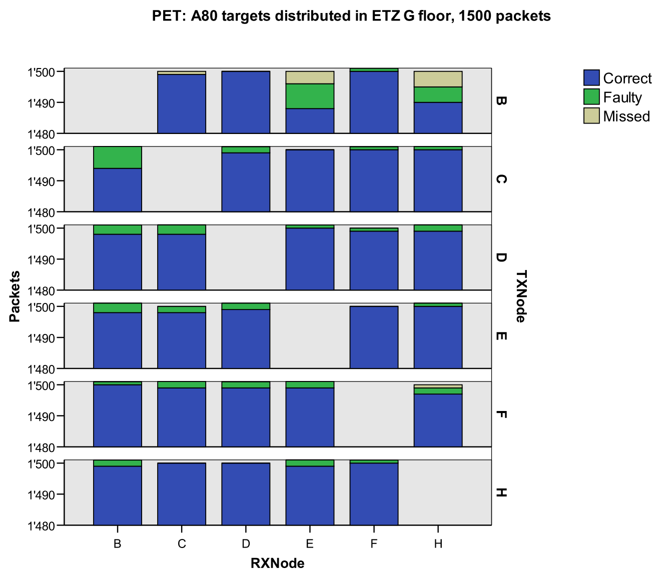
Recapitulating the conclusions from the previous sections, we can say, that

- the A80 sensor node does not have problems receiving packets in our test setup.
- the Tmote Sky shows some indications of degrading link quality with distance.
- the BTnode has range limitation and poor links to more isolated nodes. In addition, the BTnode has a rather sharp transition from being a good link to being a bad link. The test also shows a tendency towards symmetric links.

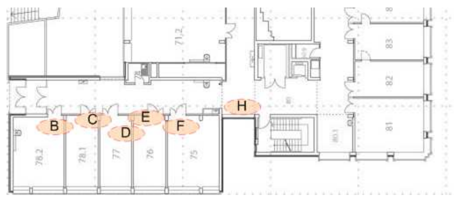
A first comparison of the link quality of the BTnode, the A80 and the Tmote Sky revealed substantial differences in the link quality. However, we only made a few



(a) Statistics

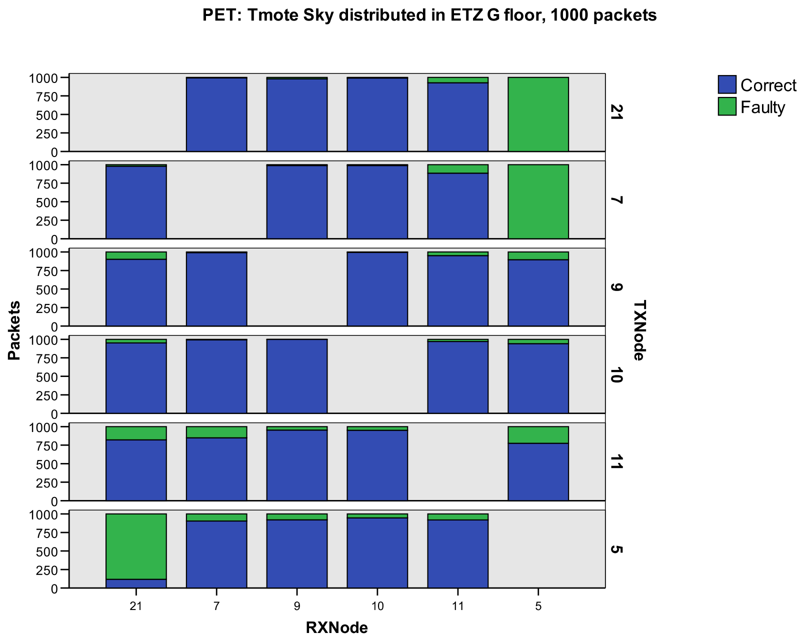


(b) Zoom

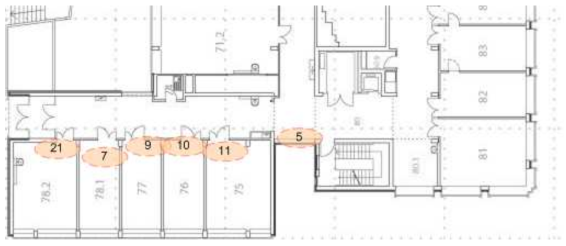


(c) Target placing

Figure 5-3
PET visualization of the A80 sensor node



(a) Statistics



(b) Target placing

Figure 5-4
PET visualization of the Tmote Sky sensor node

measurements. It is therefore not possible to provide a profound conclusion. The observations addressed in this chapter provide hints for further measurements. Such measurements should include evaluating the effect of small spacial displacement of a node and the effect of varying the external antenna position.
To retrieve meaningful results from the A80 node, a larger area for placing the nodes or altering of the transmission power should be considered. It would be interesting to see if the A80 also has an abrupt transition from the receiving area to the non-receiving area as the BTnode or if it is more smooth like Tmote Sky's behavior.
Other measurements could include a larger test setup with more nodes.

6 Conclusion

6.1 Contributions

The contribution of this thesis is an application able to perform link measurements in form of packet error tests on the BTnode. Its main features include

- Configuration and control from a host computer using the DSNAnalyzer
- Analysis of the measurements using the DSNAnalyzer
- MAC protocol to perform packet error tests using the Cipcon CC1000 radio transceiver

6.2 Summary

At the beginning of this thesis, a great effort was invested to become familiar with

- Embedded application development
- Embedded debugging
- The BTnode platform including BTnut system software and programming of the Chipcon CC1000 radio

This was very time time-consuming but helped to understand subtleties of the subject matter and led towards efficient concept design and implementation of our application.

After the implementation, a verification showed that the application is working reliable and able to measure the link quality. We then performed measurements in an office-like scenario in order to compare the results with the A80 and the Tmote Sky sensor nodes. The results revealed major differences between the BTnode and the A80 regarding packet losses on a link. Detailed measurements are required for an in-depth analysis of the link quality.

A

Specification

RPC Command

```
{"method": "<command>", "params": [<params>], "id": <id>}
```

PET control procedures

Command	Description	Parameters
bet.tx_on	Start with transmission of frames	None
bet.rx_on	Reset thest results and start with reception of frames	<tsn> Serial number of RX Btnode
bet.stop	Stop PET	None
bet.init	Sets all parameters to ist default values	None

Set procedures

Command	Description	Parameters
set.chan	Set the radio configuration set to be used	<set>: Radio configuration set which shall be used
set.power	Sets the CC1000 transmit power	<power>: Transmit power [0-26] See CC1000 data sheet table S.29
set.iter	Sets the number of test frames to be sent once the transmission is started	<itr>:Nr. of testframes [0 -65535]
set.txper	Sets the period test frames are transmitted	<period>:Transmit frame period [100 - 65535ms]
set.pream	Sets the preamble in 8-bit units	<preamb>: 01010101 seq used as preamble [0 - 255]
set.i_output	Sets the output mask to be used	<i_output>:Decimal value of the output mask

Get procedures

Command	Description	Parameters
get.betres	Logs PET results	None
get.betpar	Logs the PET parameters which are currently set	None
get.rssi	Returns the RSSI value.	None

i_output settings

`i_output` is a masking to only send designated log messages
The decimal value is generated by converting the bit sequence

Bit Nr	Deximal
Bit0=1:	Output information for every correct frame received
Bit1=1:	Output information for faulty frames received
Bit2=1:	Output information on receive frame error
Bit3=1:	Output information for every frame transmitted
Bit4=1:	Output information when all frames transmitted
Bit5=1:	Output information when transmit frame error
Bit6=1:	Not yet used
Bit7=1:	Not yet used

E.g. if `i_output` = 0 during a PET, then no log messages are automatically generated during the test.
This functionality is useful in order not to overloading the DSN if a large amount of nodes participate in a test.

B

Description Task

Institut für Technische Informatik (TIK)

Wintersemester 2006/07

SEMESTERARBEIT

für
Martin Wirz

Betreuer: Andreas Meier

Ausgabe: 23. Oktober 2006
Abgabe: 4. Februar 2007

BTnode Application for Automated Link Measurements

Einleitung

Ein drahtloses Sensor Netzwerk (WSN—Wireless Sensor Network) besteht aus einer Vielzahl von kleinen resourcenbeschränkten Knoten welche mit Funkmodul und Sensoren bestückt sind. Diese werden in der Umwelt (z.B. in einem Haus) verteilt und erstellen möglichst autonom ein Netzwerk. Ein solches Netz ermöglicht den Knoten Sensor-Messungen auszutauschen und diese Daten gemeinsam zu verarbeiten. Nach einer Vision von Stankovic et al. [1] soll dies die 'nahtlose Integration von Rechner mit der Umwelt mit Hilfe von Sensoren und Aktoren ermöglichen'.

In verschiedenen Projekten [2, 3] konnte in den vergangenen Jahren Erfahrungen mit solchen Multihop Sensor-Netzwerken gemacht werden. Dabei wurde festgestellt, dass eine Vielzahl der Packete nicht bei ihrer Destination (z.B. Senke) angekommen sind. Ein Grund dafür liegt im kabellosen und daher unsicheren und schwer einschätzbaren Übertragungskanal, dessen Linkqualität infolge Interferenz und Fading sehr stark variieren kann. In verschiedenen Forschungsgruppen wurde dieses Verhalten untersucht [4, 5]. Dabei wurde zum Beispiel festgestellt, dass in einem gewissen Abstand, der sogenannten Grey Area, die Linkqualität bei sehr kleiner Änderung der Position stark variieren kann.

Für viele Applikationen und Netwerkprotokolle ist eine gewisse Linkqualität notwendig um eine zuverlässige Funkionalität zu gewährleisten. Im Hintergrund soll dabei eine Applikation stehen die über eine lange Zeit mit niedriger Datenrate Nachrichten verschickt. Oft ist man mit der vorteilhaften Situation konfrontiert, dass man aus einer Vielzahl von Links die Zuverlässigsten Auswählen kann. Dabei is es wünschenswert Links zu benutzen die auch über längere Zeit stabil sind.

Es stellt sich also die Frage, wie man solche zuverlässigen Links auswählen kann. Das sollte möglichst wenig Zeit und insbesondere wenig Energieresourcen in Anspruch nehmen. Hierzu ist es notwendig weiterführende Messungen wie in [4, 5] vorzunehmen. So könnte es beispielsweise Sinn machen, das Verhalten auf verschiedenen Kanälen zu betrachen.

Beim Aufbau eines WSNs ist man mit der Problematik konfrontiert, dass man nicht genau weiss was in den einzelnen Knoten geschieht. Es wäre zwar prinzipiell möglich zusätzliche Information über das WSN verschicken, jedoch wird dies höchstwahrscheinlich das Verhalten des Netzwerkes verändern. Zudem kann es auch gut sein, dass die Kommunikation noch nicht zuverlässig funktioniert, und es deshalb gar nicht erst möglich ist, Zugang zum Netzwerk und somit den Knoten zu erhalten. Eine Möglichkeit für das Fehlersuchen, die Datenerfassung wie auch das Softwareupdaten ist, ein so genanntes 'Deployment Support Network' (DSN) [6] zu benutzen. Ein DSN ist ein kabelloses Sekundärnetzwerk und ermöglicht die Entwicklung, das Testen, und die Validierung von Sensor-Applikationen. Dazu werden die WSN/DSN Knotenpaare gebildet welche mit einem kurzen Kabel verbunden sind. Die DSN Knoten bauen eigenständig ein drahtloses Netzwerk auf und ermöglichen somit, wie in Abbildung 1 dargestellt, einen einfach Zugang zu den angehängten WSN Knoten.

In dieser Arbeit soll eine Plattform aufgebaut werden, die ein möglichst autonomes Messen und Analysieren der Linkqualität des CC1000 [7] auf dem BTnode [8, 9] erlaubt. Dazu soll die Messplatform DSNAnalyser [10] ausgebaut werden, welche mittels Deployment Support Netzwerk (DSN) [6] solche automatisierte Tests ermöglicht. Hierzu ist es notwendig auf dem BTnode eine Applikation zu entwickeln, welche die Befehle des DSNAnalysers versteht und entsprechend ausführt. Je nach der zur verfügung stehender Zeit, soll in einem zweiten Teil der Arbeit die Messplattform untersucht und validiert werden. So ist es zum Beispiel denkbar, Linktests in einer EMV-Zelle durchzuführen um jegliche Interferenzen von Aussen auszuschliessen.

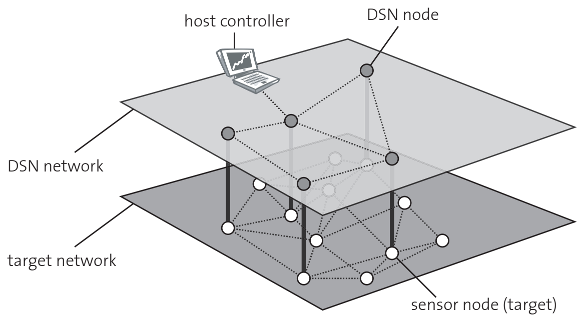


Abbildung 1: Die Knoten des Target Netzwerkes sind per Kabel mit den Knoten des 'Deployment Support Network' (DSN) verbunden. Dieses Sekundärnetz erlaubt einen einfachen zugriff auf die Targets, insbesondere erlaubt es Firmware updates, loggen von Nachrichten sowie das Versenden von Commands and die Targets.

Aufgabenstellung

1. Erstellen Sie einen Projektplan und legen Sie Meilensteine sowohl zeitlich wie auch thematisch fest [11]. Erarbeiten Sie in Absprache mit dem Betreuer ein Pflichtenheft.

2. Machen Sie sich mit den relevanten Arbeiten im Bereich Sensornetze, Systeme, Linkqualitätsmessungen sowie Linkqualitätsabschätzung vertraut. Führen Sie eine Literaturrecherche durch. Suchen Sie auch nach relevanten neueren Publikationen. Vergleichen Sie bestehende Konzepte anderer Universitäten. Prüfen Sie welche Ideen/Konzepte Sie aus diesen Lösungen verwenden können.

3. Die Applikation soll auf dem BTnode [8] entwickelt werden. Arbeiten Sie sich in die Softwareentwicklungsumgebung der Knoten ein. Machen Sie sich mit den erforderlichen Tools vertraut und benutzen Sie die entsprechenden Hilfsmittel (Versionskontrolle, Bugtracker, online Dokumentation, Mailinglisten, Application Notes, Beispielapplikationen). Schauen Sie dazu insbesondere das BTnode Tutorial [12] und die BTnode Website [9] an.

4. Nehmen Sie das JAWS Deployment-Support Network [6, 9] auf einigen Knoten in Betrieb und testen Sie dieses auf Zuverlässigkeit und Leistung.

5. Machen Sie sich mit dem DSNAnalyzer [10] vertraut. Nehmen sie dazu die Testumgebung mit Siemens A80 Knoten sowie Adapterboard und BTnode in Betrieb.

6. Erstellen Sie ein Konzept für die Applikation auf dem BTnode, welche mit Hilfe des DSNAnalysers automatisierte Linktests durchführt.

7. Setzen Sie dieses Konzept um, d.h. implementieren Sie die Applikation auf dem BTnode.

8. Validieren Sie die erstellte Applikation und Messplattform.

9. Dokumentieren Sie Ihre Arbeit sorgfältig mit einem Vortrag, einer kleinen Demonstration, sowie mit einem Schlussbericht.

Durchführung der Semesterarbeit

Allgemeines

- Der Verlauf des Projektes Semesterarbeit soll laufend anhand des Projektplanes und der Meilensteine evaluiert werden. Unvorhergesehene Probleme beim eingeschlagenen Lösungsweg können Änderungen am Projektplan erforderlich machen. Diese sollen dokumentiert werden.
- Sie verfügen über einen PC mit Linux/Windows für Softwareentwicklung und Test. Für die Einhaltung der geltenden Sicherheitsrichtlinien der ETH Zürich sind Sie selbst verantwortlich. Falls damit Probleme auftauchen wenden Sie sich an Ihren Betreuer.
- Stellen Sie Ihr Projekt zu Beginn der Semesterarbeit in einem Kurzvortrag vor und präsentieren Sie die erarbeiteten Resultate am Schluss im Rahmen des Institutskolloquiums.
- Besprechen Sie Ihr Vorgehen regelmässig mit Ihren Betreuern.
- Sie führen ein Researchtagebuch in welchem sie die Fortschritte täglich protokollieren.

Abgabe

- Geben Sie vier unterschriebene Exemplare des Berichtes, das Researchtagebuch sowie alle relevanten Source-, Object und Konfigurationsfiles bis spätestens am 4. Februar 2007 dem betreuenden Assistenten oder seinen Stellvertreter ab. Diese Aufgabenstellung soll im Bericht eingefügt werden.
- Räumen Sie Ihre Rechnerkonten soweit auf, dass nur noch die relevanten Source- und Objectfiles, Konfigurationsfiles, benötigten Directorystrukturen usw. bestehen bleiben. Der Programmcode sowie die Filestruktur soll ausreichen dokumentiert sein. Eine spätere Anschlussarbeit soll auf dem hinterlassenen Stand aufbauen können.

Literatur

[1] J. A. Stankovic, I. Lee, A. K. Mok, and R. Rajkumar, "Opportunities and obligations for physical computing systems.," IEEE Computer, vol. 38, no. 11, pp. 23–31, 2005.

[2] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, "Habitat monitoring with sensor networks," Commun. ACM, vol. 47, no. 6, pp. 34–40, 2004.

[3] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," in c-sensys05, (New York, NY, USA), pp. 51–63, ACM Press, 2005.

[4] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in First Int'l Workshop on Embedded Software (EMSOFT 2001), pp. 1–13, 2003.

[5] N. Reijers, G. Halkes, and K. Langendoen, "Link Layer Measurements in Sensor Networks," in Proc. 1st Int'l Conf. on Mobile Ad-hoc and Sensor Systems (MASS), Oct. 2004.

[6] J. Beutel, M. Dyer, L. Meier, and L. Thiele, "Scalable topology control for deployment-sensor networks," in Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05), pp. 359–363, IEEE, Piscataway, NJ, Apr. 2005.

[7] Chipcon, CC1000, Single Chip Very Low Power RF Transceiver, April 2002.

[8] J. Beutel, O. Kasten, and M. Ringwald, “BTnodes – a distributed platform for sensor nodes,” in Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003), pp. 292–293, ACM Press, New York, Nov. 2003.

[9] “BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks.” http://www.btnode.ethz.ch.

[10] P. Oehen, “DSNAnalyzer: Backend for the Deployment Support Network,” Master’s thesis, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, Sept. 2006.

[11] E. Zitzler, “Studien- und Diplomarbeiten, Merkblatt für Studenten und Betreuer.” Computer Engineering and Networks Lab, ETH Zürich, Switzerland, Mar. 1998.

[12] J. Beutel, P. Blum, M. Dyer, and C. Moser, “Btnode programming — an introduction to btnut applications.” Computer Engineering and Networks Lab, ETH Zürich, Switzerland, May 2006.

[13] “NCCR-MICS: Swiss National Competence Center on Mobile Information and Communication Systems.” http://www.mics.org.

[14] TIK, “Notengebung bei Studien- und Diplomarbeiten.” Computer Engineering and Networks Lab, ETH Zürich, Switzerland, May 1998.

Bibliography

[1] Btnode website. http://www.btnode.ethz.ch.

[2] Json rpc website. http://www.btnode.ethz.ch.

[3] Moteiv corporation. http://www.moteiv.com.

[4] Btnode rev3 - product brief, April 2005.

[5] Atmel Corporate. *ATmega128 Datasheets*, 2005.

[6] Chipcon AS. *CC1000 Single Chip Very Low Power RF Transceiver Data Sheet*, 2.3 edition, 2005.

[7] Chipcon AS. *Single Chip Low Power RF Transceiver for Narrowband Systems*, 1.8 edition, 2005.

[8] Lukas Winterhalter Daniel Hobi. Large-scale bluetooth sensor-network demonstrator. Master's thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2005.

[9] Egnite Software GmbH. Nut/os rtos for the atmega128. http://www.ethernut.de/.

[10] L. Meier J. Beutel, M. Dyer and L. Thiele. Scalable topology control for deployment-sensor networks. Technical report, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, November 2004. Technical Report 208.

[11] L. Meier M. Ringwald J. Beutel, M. Dyer and L. Thiele. Next-generation deployment support for sensor networks. Technical report, Computer Engineering and Networks Lab, ETH Zürich, Switzerland, November 2004. Technical Report 207.

[12] Patrice Oehen. Dsnanalyzer: Backend for the deployment support network. Master's thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2006.

[13] OFCOM (Swiss Federal Office of Communications). Radio interfaces regulation, alarms 868.600 - 868.700 mhz, January 2007.

[14] Joe Polastre. Tinyos radio stacks. Presentation at Network Embedded Systems Technology Winter 2004 Retreat, January 2007.

[15] K. Srinivasan and P. Levis. RSSI is Under Appreciated. *Proceedings of the Third Workshop on Embedded Networked Sensors*, 2006.

[16] John A. Stankovic, Insup Lee, Aloysius K. Mok, and Raj Rajkumar. Opportunities and obligations for physical computing systems. *IEEE Computer*, 38(11):23–31, 2005.

[17] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. *ACM SenSys 03*, November 2003.