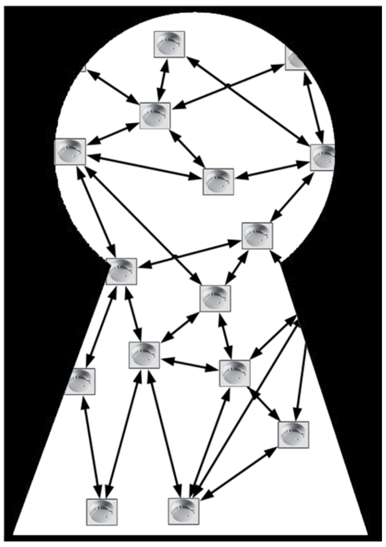
Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Computer Engineering and Networks Laboratory (TIK)
Prof. Dr. Lothar Thiele

Distributed Spy-Software Tool WSNSpy



MASTER'S THESIS
MA-2007-32

JULY 2007

Author:

Severin Hafner

Supervisors:

Kevin Martin
Dr. Simon Künzli
Andreas Meier

Contents

Tables

Figures

Abstract

In this thesis we developed a system that allows to determine the health state of a wireless sensor network in a fast and easy way without any instrumentation of nodes.

Due to limited energy resources and memory constraints, usually no additional functionality for debugging purposes can be inserted into wireless sensor networks, which implies hardly any observability in case of failure.

To improve observability, we designed a system that is able to passively overhear sensor network traffic using a distributed spy network. We neither depend on the instrumentation of sensor nodes nor does the system generate traffic in the frequency band of the sensor network. Additionally, we developed the evaluation tool WSNSpy, which reconstructs the sensor network from the overheard message transmissions by only depend on low-level information, i.e. sender and receiver of a message transmission. The obtained network we use as the basis for the network state evaluation which is done by checking certain conditional properties on its fulfillment. The properties to check can modularly be added allowing the user to specify the definition of network health individually for his sensor network.

We evaluated the system within this thesis achieving high capturing characteristics and a well matching relation between the reconstructed network and reality, providing a good basis for checking various properties.

Preface

The work presented in this document was developed within a master's thesis of the Swiss Federal Institute of Technology (ETH) in Zurich. The location for the development was the Communication and Wireless Group of the Research and Development department of Siemens Building Technologies (SBT) in Zug.

I would like to thank Kevin Martin, R&D Engineer at SBT Zug, Dr. Simon Künzli, Senior Engineer at SBT Zug, and Andreas Meier, Ph.D. student at ETH Zurich for supervising my work and for their great support during this thesis. I also wish to thank Siemens SBT for the opportunity to do my thesis in the exciting area of fire detection systems.

Zug, July 2007

Severin Hafner

1

Introduction

Humans often rely on sensors in order to get information of various kinds from their environment. Sensors could be used actively by reading e.g. a thermometer to adapt the clothing to the weather, or passively by deploying a fire detector in order to get alerted in case of a fire. As one single sensor only provides information of a small area, several sensors are needed for larger environments. In this case, multiple sensors can be combined to a sensor network which is usually done, by applying radio devices to the sensor in order to achieve a wireless sensor network.

1.1 Wireless Sensor Networks

Wireless Sensor Networks (WSN)[1] are an emerging technology in scientific and in industrial environments. A network consisting of several sensor nodes is able to monitor a larger area than a single sensor. The wireless communication guarantees an easy deployment of the sensor nodes even in harsh environments. These attributes make WSNs an interesting and promising technology for various applications in different surroundings. Examples for deployed WSNs are Bird Observation on Great Duck Island, Maine, USA [6], Ocean Water Monitoring [15] or glacier monitoring [7] as a selection out of numerous applications.

1.2 Problem Statement

However, the development of a WSN is not as easy as its operation should be in the end. Network deployments vary heavily in environmental condition and utilization which leads to demanding requirements for the hardware and software. Usually, WSNs consist of tiny sensor nodes which should be able to run autonomously

[1]Wireless Sensor Network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at different locations. [11]

for several months or even years. Therefore, WSNs are equipped with very limited hardware, i.e. they are built with little memory and small processors due to energy limitations. As these devices are built especially for one single application they usually become highly optimized for their purpose. The hardware is selected considering the requirements of the application and limitations are taken into account. If the system is working at the end, everything is fine. However in case of failure, it is very difficult to identify the buggy part and to provide a solution. The main reason for this difficulty is that systems with limited resources like senor nodes provide hardly any *observability*.

Observability mostly implies either additional hardware, e.g. LEDs for system state feedback, or additional code for debugging outputs which always results in a demand for more memory and energy consumption. But these are the critical parameters for tiny sensor nodes as the costs raise massively if enhancements are included only for debugging purposes. Therefore, other techniques must be found that allow to get detailed knowledge about the network without interfering with the sensor nodes.

One possibility to obtain this functionality is to make use of a so-called *Deployment Support Network (DSN)* [3]. The DSN is a second network laid out on top of the original sensor network using a different frequency range. Through the DSN the sensor nodes can be configured and logging events can be collected for further analysis. More powerful nodes with a higher bandwidth than the sensor nodes itself can be used for the DSN, because battery lifetime is not that much an issue, here. The DSN itself could have the same drawbacks as stated above for the WSN concerning the observability. But for the DSN, a fully working WSN could be used, for which no debugging should be necessary anymore.

However, the concept of the DSN requires additional DSN nodes attached to the sensor nodes. Hence, the DSN can only be used during development. For the debugging of deployed WSNs a concept is needed which provides information about the network state without interfering and instrumentation of nodes.

Ringwald et al. showed in [10] that many of the experienced problems concerning the deployment of a sensor network can be detected by overhearing and analyzing sensor network traffic without the need for sensor node instrumentation. The possibility to passively listen to the traffic generated by the sensor network, can be achieved by deploying special spy nodes in the area of the sensor network. This has the main advantage that the spy nodes do not interfere with the sensor network. As a single spy node cannot record the communication of the entire network, several spy nodes need to be deployed. However, for controlling the spy network and for collecting overheard data the concept of the DSN is well suited.

1.3 Scope of this Thesis

The scope of this thesis was to develop a system that is on the one hand able to overhear traffic from a sensor network and on the other hand to process the captured data in order to gain further knowledge of the state of the sensor network. For practical reasons, the spy network should be easy deployable and configurable

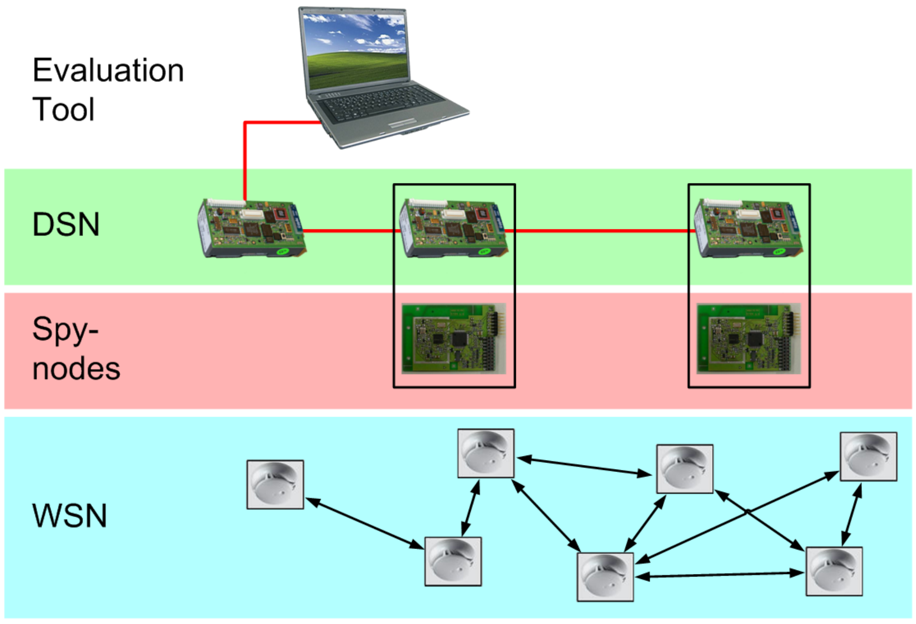


Figure 1-1: **Three-Layer Concept**

which can be achieved by using the DSN for the spy nodes instead of for the sensor nodes. This leads to a three-layer concept as shown in Figure 1-1.

The lowest layer contains the WSN which runs a certain application. In the area of the WSN spy nodes are deployed for overhearing sensor network traffic. As the spy nodes forward the captured traffic to the directly connected DSN nodes, the spy nodes can be seen as the connection between WSN and DSN. The DSN is used to configure the spy network and to collect the data overheard by the spy nodes.

In a second step, the overheard data is used to reconstruct the network based on low-level information, i.e. only using sender and receiver of a message transmission. The reconstructed network is then used to check certain user defined properties which are evaluated for every node on fulfillment. By this, statements concerning the "health" state of the network can be made based on the captured traffic.

1.4 Related Work

This section surveys previous work in relation with monitoring and debugging of WSNs. There exist various propositions and tools for this purpose. However most of them were designed from an active approach in which the sensor nodes themselves need to be instrumented and/or traffic is generated for monitoring and debugging capabilities in the frequency range used by the WSN.

In the following sections several approaches that are in one way or the other similar to the one presented in this thesis are discussed. Besides the similarities, we highlight especially the differences and the thesis is put into context of the related approaches.

1.4.1 SNIF - Sensor Network Inspection Framework

The *Sensor Network Inspection Framework (SNIF)* [10] provide functionality for inspecting a deployed sensor network, consisting of a distributed network sniffer and a data-stream-based framework for online traffic analysis. The tool uses the DSN for configuring the spy nodes and for collecting the overheard data.

Two major differences can be stated while comparing SNIF with the system developed within this thesis.

- First, the sniffing part, designed for the BTnode platform [16] has a radio and a Bluetooth module combined as one single device. This has the drawback that the hardware for overhearing the sensor network traffic is not modularly replaceable. In case of an improvement of the radio module not only the spy nodes but the DSN nodes as well need to be replaced. It was a crucial requirement for this thesis's outcome that new hardware can be introduced fast and easily. Therefore, we preferred a solution where the spy node and the DSN node are deployed on two separate platforms connected via an adaptor board.
- Second, in the SNIF, the part for the evaluation of captured traffic in the backend tool is mainly based on higher-layer information. E.g. in order to trace a packet on its way to the sink, either the path to the sink which is not contained in the header of the *Medium Access Control (MAC)* header must be known or else the payload of the MAC packet must be considered. The system developed in the scope of this thesis, is designed to work with as few information as possible. So, only source and destination of a message is used for network analysis, which is contained in the MAC header. No higher-level protocols are considered.

In contrast to the SNIF in which possibly an single packet is used for the detection of a certain case (e.g. a node reset due to a sequence number reset), the evaluation of our tool is based on all captured messages within a certain time frame. The goal was not to detect incidents based on individual messages, but to provide a network evaluation tool.

1.4.2 Emstar

Emstar [4] is a software environment designed for heterogeneous networks containing a mixture of extremely constrained 8-bit "motes" up to less resource-constrained 32-bit "microservers". It runs on a linux platform and consists of several libraries and tools which can modularly be combined and which support simulation, emulation, and visualization for WSNs.

The Emstar framework provides a rich variety of tools and services for the development of a WSN. However, it relies on a heterogeneous network including some powerful nodes, called "microservers". The WSN actually being developed only consists of the very resource-constrained nodes, on which no additional software for monitoring and debugging purposes can be placed. Besides, Emstar requires the instrumentation of nodes and uses the sensor network for transmitting secondary data containing monitoring information. Hence, for our purposes Emstar is not suited.

1.4.3 Sympathy and Memento

Sympathy and *Memento* are similar tools which both provide functionality to detect failures and for the debugging of WSNs.

Sympathy [9] enables failure detection by making use of selected metrics and by including an algorithm which localizes the source of a failure. This reduces the overall failure notification and narrows the search for the cause.

Memento [12] is a health monitoring system for WSNs by providing failure detection and symptom alerts. It consists of two parts, an energy-efficient protocol in order to deliver state summaries and a distributed module for the detection of failures.

However, both tools need instrumentation of sensor nodes as detecting failure is based on metrics obtained directly from the sensor nodes. This violates the stated concept to design a passively listening spy tool. Moreover, Sympathy and Memento use the same frequency band for transmitting debug information, which influences the behavior of the actual WSN application.

1.4.4 Deployment Support Network

The *Deployment Support Network (DSN)* is based on the concept in which a second device is attached to every sensor node building up a network on top of the sensor network operating in a different frequency range. The DSN is used for sending commands to and to collect logging information from the sensor nodes.

Unfortunately, the concept of the DSN requires additional DSN nodes attached to the sensor nodes. As this is not possible for deployed WSNs, the DSN is not suited. However, the DSN is well suited for controlling the nodes of the spy network.

1.5 Chapter Overview

This thesis is organized as follows. Chapter 2 explains the conceptual design of the spy system developed, by providing a system overview and by describing its purpose. In Chapter 3 we explain in detail how the system is implemented. The evaluation and testing of the system is documented in Chapter 4 and Chapter 5 gives a summary on the thesis and the conclusions obtained, as well as an outlook on future work.

2 Conceptual Design

The thoughts and work done during this thesis concerning the conceptual design are described in this chapter. First, we give an overview over the system, then the concept for the spy nodes, and afterwards the analysis and evaluation part resulting in the WSNSpy tool are presented.

2.1 System Overview

In order to spy on WSNs and to evaluate the captured traffic with the system proposed in this thesis, several individual components need to be put together. In this section the complete system is illustrated, focusing on the big picture.

The system is shown in Figure 2-1. Dark gray colored parts represent already existing components, described in Appendix A, which are integrated into the system "as is". Light shaded parts, however, denote the tools that are developed within the scope of this thesis, i.e. spy node and WSNSpy.

The system consists of the four components that are described below, starting on the right.

- **Spy Node**
  Before any traffic can be analyzed and evaluated, a device is needed that constantly listens to traffic and captures ongoing transmissions. Such a device is called a *spy* or *sniffer* and should ideally be permanently in receive state. However, the overheard traffic somehow has to be processed and delivered to

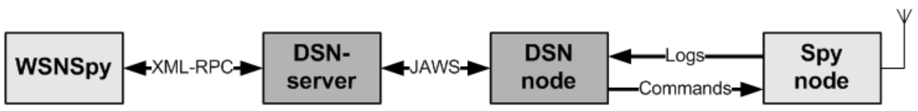


Figure 2-1: **System Overview**

the next element of the chain, which contradicts the principle of permanently receiving. Nevertheless, it is important to reduce the processing part done by the spy to a minimum.

- **DSN Node**
  The second element of the system is the DSN node which is directly connected to the spy node. Several DSN nodes build a DSN as described in Appendix A.5. Every output coming from the spy node is forwarded via the DSN to the *GUI node,* i.e. the root node. In the opposite direction, commands targeting the spy node are forwarded by the DSN node.
- **DSN-Server**
  The DSN-Server communicates with the DSN node via the GUI node. It uses the commands provided by the JAWS [1] software running on the DSN nodes. The DSN-Server provides a bunch of methods which could either be used to control the DSN or to get information out of the servers database [8]. For a more thorough description of the DSN-Server, see Appendix A.6.
- **WSNSpy (Evaluation Tool)**
  Finally, the WSNSpy can access the data stored in the database and process it in whatever way it is demanded. For the communication between the WSNSpy and the DSN-Server, the XML-RPC [22] interface is used.

After obtaining a general understanding of how these components work together the first and last element of the system are explained in very detail as they constitute the core part of this thesis.

2.2 Radio Spy

To be able to analyze packets and interpret the state of a network, traffic needs to be overheard and recorded. In a network in which all nodes communicate on the same frequency channel, the functionality of a spy node does not differ that much from a sensor node. The hardware usually is the same, as the spy must operate in the same frequency band to be able to receive any traffic. However, there also exists multi-frequency networks which change their communication channel from time to time. This alteration of the radio frequency may happen slowly, e.g. when changing to another channel due to interference, or even pretty fast, as it is done in *Bluetooth* [21], where the frequency is changed up to 1600 times per second.

Frequency-hopping is often used to make a network more resistant against interference and jamming devices. Unfortunately, it also complicates the task of overhearing traffic as a spy normally is not aware of the hopping sequence and therefore does not know the actual communication channel.

Due to the advantages described concerning security and reliability, a frequency-hopping application is planned for the WSN. This leads on the one hand to the need of a spy that is able to rapidly switch from frequency to frequency and to scan the medium for network traffic, and on the other hand to actually overhearing the traffic. This is a kind of a contradiction, because mostly when an ongoing communication is detected by scanning the medium, it is too late to start receiving as a part of the message has already been missed. Therefore, we propose to split the tasks scanning and overhearing into two phases and to do them sequentially.

In the following section the process of finding a feasible solution for a multi-frequency spy based on the A80 radio module is described.

2.2.1 Feasibility Study: Multi-Frequency Scan

Detailed specifications on how the multi-frequency scheme will look like for the final sensor network are not available, yet. Therefore, only vaguely defined constraints can be stated. Hard constraints for the feasibility study are only given from the hardware of the radio module, itself.

2.2.1.1 Guidelines for the Multi-Frequency Functionality

In connection with frequency selection the following guidelines are stated.

- The channel bandwidth is set to $25\ kHz$, which leads to 80 channels in the $2\ MHz$ wide ISM-Band in the range of $868 - 870\ MHz$.
- Every node chooses a certain number of channels out of the 80 possible channels. The selection process is not yet specified, but it will depend on the set of channels selected by neighboring nodes, the rating based on the link quality done earlier, and other factors.
- The format of transmitted packets is defined, e.g. header format, preamble length, etc. The minimal length of a message on the physical layer 3.1.1.1 is $12.5\ Bytes$ and the length of an acknowledgment is $9.5\ Bytes$ which results in an actual transmitting time of about $20\ ms$ at $5\ kBaud$ and $10\ ms$ at $10\ kBaud$.
- The hardware for the radio spy is given, which is the same device as the radio module of the sensor nodes. The module is specified in detail in Appendix A.1

As there must be assumed that any possible channel could be used for transmissions, every channel should be scanned for traffic all the time. This implicates that a single radio module for every channel running in parallel is needed, which is not practicable. Hence, several restrictions concerning the radio module must be taken into account.

2.2.1.2 Timing Constraints of the Radio Chip

The time needed to decide whether a transmission is in process at a certain moment on a single channel consists of the following subtasks.

1. **Frequency Programming**
   The time needed to program a new frequency is about $0.7\ ms$. It is possible to

further reduce the time needed for this step. The radio chip provides two frequency registers which allow to write the new frequency to the second register while the first one is used and vice versa. The setup of the new frequency could be done during step two of the previous iteration. The only action to be done in this step is to switch from one frequency register to the other.

2. **Measure RSSI Value**
   A transmission can be detected by measuring the *Received Signal Strength Indication (RSSI)* value. The time needed for the measurement depends on the required accuracy and as a consequence on the number of measured samples. When switching to a new frequency it takes some time until the RSSI value is stable, which is called RSSI attach time. Therefore, about $1.1\ ms$ are needed to decide whether a transmission is in process.

As a conclusion, we can state that at least $2\ ms$ are needed to perform a channel sensing action. If this is done by one single spy node sequentially for all channels, one round would take $160\ ms$. Compared with the message transmission time of $20\ ms$ for a low baud rate the probability of detecting the transmission is very low.

2.2.1.3 Broad-band Property of the Radio Chip

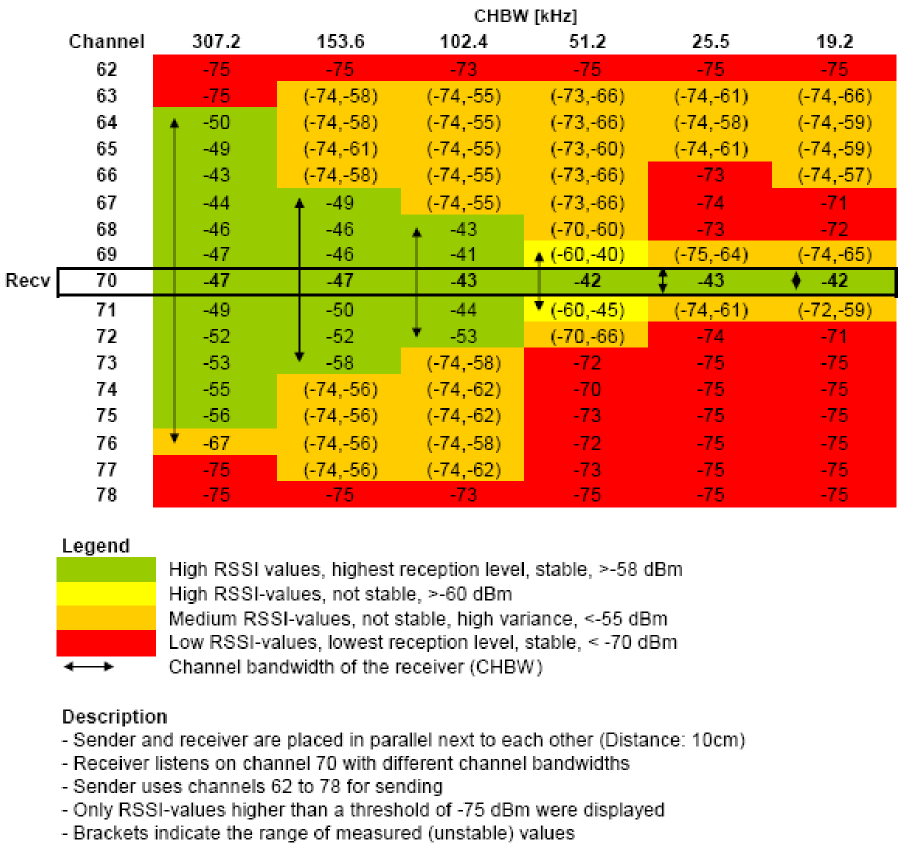
In order to improve the radio spy, the broad-band property of the radio chip can be exploited. The idea is to detect a transmission with a broad-band receiver scanning several channels at a time. In this case traffic is detected faster, however the correct channel needs to be specified later on with further measurements.

The CC1020 radio chip provides five different settings, for which the channel bandwidth varies between $19.2\ kHz$ and $307.2\ kHz$. There is a difference between channel spacing which denotes the width of a channel and the channel bandwidth which specifies the actually used frequency range for receiving. Usually, the latter one is smaller than the first one.

Figure 2-2 shows the RSSI values measured by a receiver while operating with different bandwidth settings. The center frequency of the receiver was always set to channel 70. The RSSI values were measured while a sender node operating with a narrow band setting of $19.2\ kHz$ for the channel bandwidth was transmitting on different channels. The test used to determine the values presented in the figure was done by placing the sender and receiver node directly next to each other.

From the figure we can conclude, there exists a hard level difference between in-band channels 64 to 76 and channels which are out-of-band for a channel bandwidth of $307.2\ kHz$. This is a promising fact in order to detect whether in the range of the receiver a transmission is going on. For smaller channel bandwidths the edge between in-band and out-of-band channels weakens. This makes it harder to decide on an ongoing transmission. Channel bandwidth settings of 25.5 and $51.2\ kHz$ are not useful for our purpose as no further knowledge could be gained than scanning with $19.2\ kHz$ bandwidth.

The observation stated above can be verified by Figure 2-3 which shows the measured RSSI values on all channels while transmitting on channel 50. The transmis-



		CHBW [kHz]					
	Channel	307.2	153.6	102.4	51.2	25.5	19.2
	62	🟥 -75	🟥 -75	🟥 -73	🟥 -75	🟥 -75	🟥 -75
	63	🟥 -75	🟧 (-74,-58)	🟧 (-74,-55)	🟧 (-73,-66)	🟧 (-74,-61)	🟧 (-74,-66)
	64	🟩 -50	🟧 (-74,-58)	🟧 (-74,-55)	🟧 (-73,-66)	🟧 (-74,-58)	🟧 (-74,-59)
	65	🟩 -49	🟧 (-74,-61)	🟧 (-74,-55)	🟧 (-73,-60)	🟧 (-74,-61)	🟧 (-74,-59)
	66	🟩 -43	🟧 (-74,-58)	🟧 (-74,-55)	🟧 (-73,-66)	🟥 -73	🟧 (-74,-57)
	67	🟩 -44	🟩 -49	🟧 (-74,-55)	🟧 (-73,-66)	🟥 -74	🟥 -71
	68	🟩 -46	🟩 -46	🟩 -43	🟧 (-70,-60)	🟥 -73	🟥 -72
	69	🟩 -47	🟩 -46	🟩 -41	🟨 (-60,-40)	🟧 (-75,-64)	🟧 (-74,-65)
Recv	70	🟩 -47	🟩 -47	🟩 -43	🟩 -42	🟩 -43	🟩 -42
	71	🟩 -49	🟩 -50	🟩 -44	🟨 (-60,-45)	🟧 (-74,-61)	🟧 (-72,-59)
	72	🟩 -52	🟩 -52	🟩 -53	🟧 (-70,-66)	🟥 -74	🟥 -71
	73	🟩 -53	🟩 -58	🟧 (-74,-58)	🟥 -72	🟥 -75	🟥 -75
	74	🟩 -55	🟧 (-74,-56)	🟧 (-74,-62)	🟥 -70	🟥 -75	🟥 -75
	75	🟩 -56	🟧 (-74,-56)	🟧 (-74,-62)	🟥 -73	🟥 -75	🟥 -75
	76	🟧 -67	🟧 (-74,-56)	🟧 (-74,-58)	🟥 -72	🟥 -75	🟥 -75
	77	🟥 -75	🟧 (-74,-56)	🟧 (-74,-62)	🟥 -73	🟥 -75	🟥 -75
	78	🟥 -75	🟥 -75	🟥 -73	🟥 -75	🟥 -75	🟥 -75

Legend

🟩	High RSSI values, highest reception level, stable, >-58 dBm
🟨	High RSSI-values, not stable, >-60 dBm
🟧	Medium RSSI-values, not stable, high variance, <-55 dBm
🟥	Low RSSI-values, lowest reception level, stable, < -70 dBm
↔	Channel bandwidth of the receiver (CHBW)

Description
- Sender and receiver are placed in parallel next to each other (Distance: 10cm)
- Receiver listens on channel 70 with different channel bandwidths
- Sender uses channels 62 to 78 for sending
- Only RSSI-values higher than a threshold of -75 dBm were displayed
- Brackets indicate the range of measured (unstable) values

Figure 2-2: **Channel Coverage for different Bandwidth Settings**

sion has visible impacts on about 25 channels in the range of channel 40 to 65. It has to be mentioned that the high signal strength measured on channel 20 originates from a second transmitter and must be omitted.

One major difficulty for deciding whether in a certain channel a transmission is going on, is that we do not know the distance between the two nodes. By only considering the signal strength measured on this single channel, it is impossible to decide whether a node in short distance is sending on an neighboring channel or whether a node far away is actually transmitting on the scanned channel.

2.2.1.4 Conclusions

To finalize, it is difficult to detect the channel on which a transmission is going on. Actually, only the channel bandwidth of $307.2\ kHz$ can be used for pre-sensing the medium, because smaller bandwidths do not have hard edges between in-band and out-of-band frequencies. To identify a single channel the last 13 channels must be scanned sequentially.

Due to this result we decided not to dig too much into a sophisticated multi-frequency spy, but to concentrate on the realization of a single channel spy, which

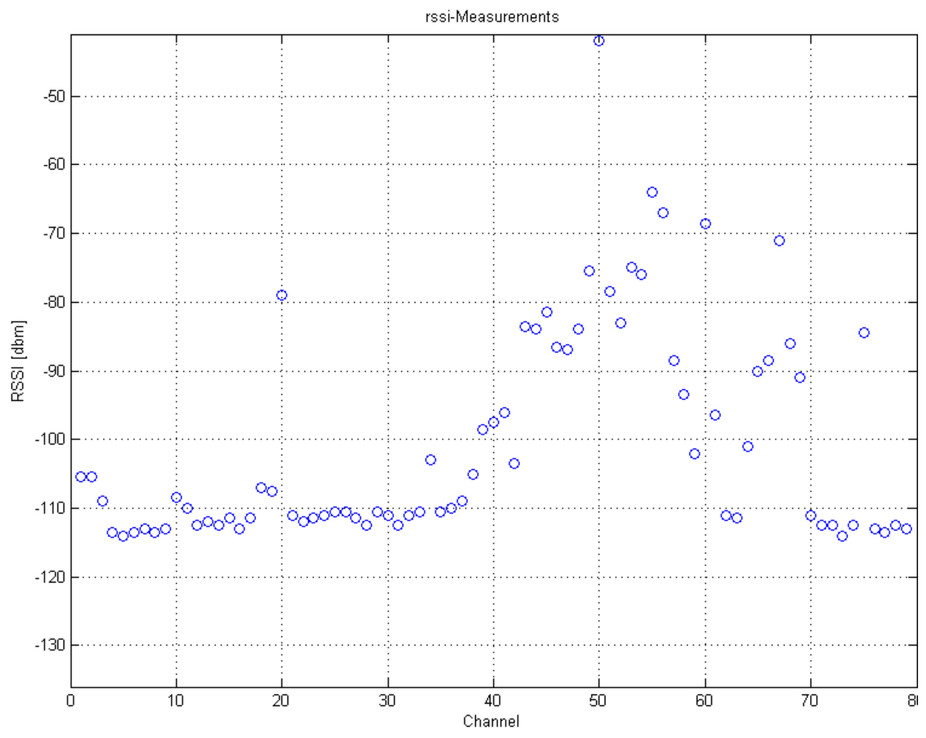


Figure 2-3: **RSSI Distribution over all Channels**
The figure shows the RSSI values measured on different channels while sending on channel 50. On channel 20 another system was active during the test.

covers the main focus of this thesis better. The development of a feasible multi-frequency spy is left open for future work.

2.2.2 *Spy Functionality*

In this section, we describe the functionality of the spy, which was designed for the A80 radio module. However, the focus is laid on the conceptual part by explaining the functionality provided to overhear traffic rather than discussing implementation specific topics. This is done in particular in Section 3.1.

2.2.2.1 *Message Assembly*

As the spy node only passively listens to the communication occurring in a WSN, we need to thoroughly discuss how a successful message transmission can be detected. Packets might be missed either by the sensor node or the spy node, or by both of them, which needs to be considered when interpreting the output of the spy. We list the cases that could occur in Table 2-1.

From the spy's point of view, it is necessary to receive a message and the corresponding acknowledgment to be sure that the message transmission was successful.

	Spy intercepts packet	**Spy misses packet**
MSG OK	The first part of a transmission was successful and was captured by the spy. ⇒ SPY LOG	The first part of a transmission was successful but was not intercepted by the spy. ⇒ NO SPY LOG
MSG LOST	No message was received and therefore no acknowledgment was generated. However, the spy intercepted the message and generated a log output. ⇒ SPY LOG	No message was received neither by the sensor node nor by the spy and therefore neither an acknowledgment nor a log output was generated. ⇒ NO SPY LOG
ACK OK	This case shows a completed transmission of a packet and its successful interception by the spy. ⇒ SPY LOG	A packet transmission was successful but the acknowledgment could not be overheard by the spy node. Therefore no log output is generated. ⇒ NO SPY LOG
ACK LOST	The acknowledgment was not received by the sensor node and will be retransmitted shortly. However, the spy intercepted the packet and generated a log output. ⇒ SPY LOG	The acknowledgment was not received neither by the sensor node nor by the spy but will be retransmitted shortly. No log output was generated. ⇒ NO SPY LOG

Table 2-1: **Packet Transmission and Spy Interception Scenarios**
The leftmost column states the result of a packet transmission. The terms "MSG LOST" and "ACK LOST" mean that the transmission was not successful but does not provide the reason for the failure. The columns two and three define whether the sent packet was overheard by the spy.

If only the message is considered, it is not possible to know if the message was received by the sensor node. Therefore, the acknowledgment must be considered. On the other hand, it is impossible for the spy to detect that the acknowledgment was missed by the receiver. But this does not matter, because the missed acknowledgment will be retransmitted in a while and the transmission will be successful for the sensor nodes as well as from the spy's point of view.

We decided to keep the spy as simple as possible and to stick to the minimal functionality needed to capture traffic. This means that the messages and acknowledgments are not assembled to a completed message transmission but to output every received packet one after another. The message assembly is done later on, when importing the data into the WSNSpy. The following scenario shows the advantage of

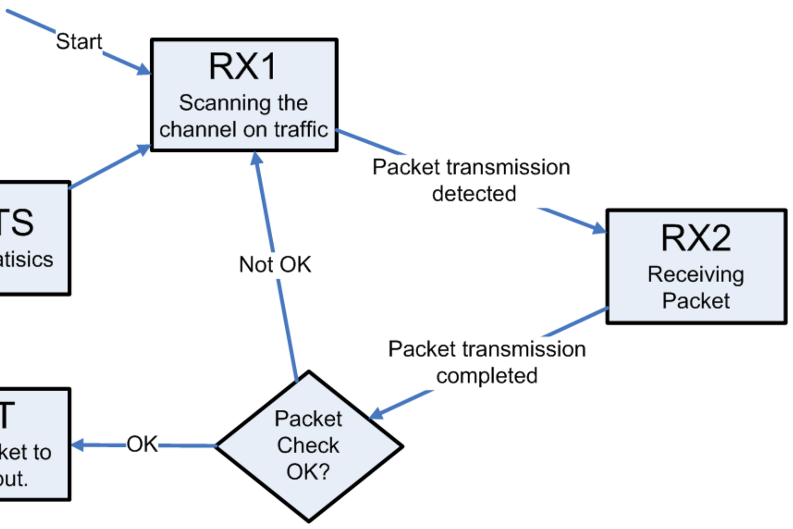

Start
RX1
Scanning the channel on traffic
Packet transmission detected
RX2
Receiving Packet
STATS
Update statisics
Not OK
Packet transmission completed
OUT
Write Packet to the output.
OK
Packet Check OK?

Figure 2-4: **A80-Spy Flow Chart**

this decision.

Assume that a spy node only receives one part of a communication, because only one node involved in an ongoing communication is in its reception range. The spy either receives the message or else the acknowledgment. In both cases, the spy does not have enough information to guarantee a successful message transmission between the two nodes. If a second spy node is placed that intercepts at least the other part of the communication, both spies together provide enough information for assembling a message. Therefore, the assembly must be done after the logs of the spy nodes have been merged.

2.2.2.2 Spy Concept

From the insights stated above, the algorithm running on the A80-Spy was developed. Figure 2-4 shows the layout of the algorithm.

On startup, the spy immediately goes into receive state **RX1** and remains in this state until the transmission of a packet is detected. In state **RX2** the packet is received and after completion the packet is checked on bit errors. If the packet is received correctly a log output is generated in **OUT** and the statistics are updated in **STATS**. Otherwise, if the packet contains errors no output is generated and the spy returns into **RX1** state waiting on overhearing the next packet.

How this algorithm is implemented in detail and what components of the software stack of the A80 are used is described in Section 3.1

2.3 WSNSpy Tool

While intercepting ongoing transmissions is an engineering task, the evaluation of the overheard traffic is much more complex. The purpose of the WSNSpy is to provide an environment to analyze and evaluate a WSN in order to make statements concerning the "health state". The term "healthy network" could be interpreted in many ways and its definition differs for every sensor network depending on the stated requirements.

In this section the design of the WSNSpy tool is presented. First the requirements from the user's point of view resulting in use case definitions are stated, followed by the definition of "health" in the context of fire detection sensor networks. Then, the actual design of the tool itself is presented, without going to much into detail concerning the implementation part.

2.3.1 Use Cases

Use cases are defined from the user's point of view. The user is interested in getting a response after he has done an action, rather than in its technical realization. In the following, we describe every use case for the WSNSpy.

1. **Network Reconstruction**
   One of the main tasks the tool must be able to perform, is to reconstruct a network based on the information delivered. It must not matter where the information comes from, however, a common data format must be defined in order to remain independent of the data source. The information delivered by the spy nodes consists of log outputs for every captured packet, distinguishing between messages and acknowledgments. The tool should assemble successful message transmissions out of these logs and extract sender node, receiver node, and the time at which the message has occurred. From this information the network should be reconstructed.
2. **Network Evaluation**
   The key functionality of the spy tool consists of checking certain properties applied to the network or a part of it. Properties should modularly be extendable and replaceable. For every property the evaluation result must be compared to a certain threshold. According to this comparison the property is fulfilled or not.
3. **Load Server Content**
   Besides importing file content, it should also be possible to make the evaluation based on information obtained directly from the database the DSN-Server uses to store the collected data.
4. **Continuously Importing Server Content**
   In order to have a certain flexibility and to be able to evaluate the network as fast as possible the analysis should be possible while the network still is running. Therefore, the tool must provide the functionality of ongoing data import. As real-time analysis is impossible due to the DSN, it should at least be possible to poll repetitively on database changes and to provide an ongoing data import.
5. **Load File Content**
   The tool should provide the functionality to load a file which contains the in-

formation the evaluation will rely on. This case is necessary for offline use, without connection to the DSN-Server.

6. **Time Window Adjustment**
   A certain flexibility is necessary for specifying the time frame which is considered for generating the network. If logs are imported continuously it should be possible to limit the time window to a certain size, e.g. considering only the messages intercepted within the last minute.
7. **Various Input Sources**
   The tool should be able to process various data input formats which leads to a multi-functional evaluation tool. Besides the already mentioned format of the spy logs, it should be possible to process logs generated by the sensor nodes, itself. As a third input source, log files originating from Glomosim [23] simulations must be possible to process. It is crucial to provide the possibility to compare obtained evaluation results based on spy inputs with results of other input sources. This helps to evaluate the tool itself, as well as to give inputs for simulation scenarios improvements.
8. **Examination of Messages**
   Besides the analysis of the network in general, the functionality of examining every single message is needed as well. The main focus of the tool should be the network evaluation which does not rely on the message content. However, it could be useful to provide functionality to examine message contents for further analysis.
9. **Statistical Figures**
   For in-depth network analysis certain key figures should be provided, for the whole network as well as only for parts of it or even for every single node and message. Key figures for a node are for example the number of sent and received messages, the time window in which this node was active and the number of neighbors. For a link, the number of successful transmissions as well as the time window the transmissions occurred are of interest.

So far, the requirements from a user's point of view are explained. The next section describes what tools are necessary to rate the state of a network.

2.3.2 Network "Health"

As already mentioned, there does not exist an absolute definition of a healthy network. The definition must rather be done by considering the requirements specified for the network, which highly depend on the purpose of the sensor network. The following properties can be used to determine the health of a network.

- **Connectivity**
  The degree of connectivity shows how many neighbors a node has to communicate with. The more neighbors the more redundant is the network. Node or link failures have less impact on a well connected network. However, many neighbors imply big routing tables and other neighbor-specific parameters a node

must remember. So, there is a trade-off between connectivity and resources needed.

- **Node Activity**
In an efficient WSN, communication is reduced to a minimum in order to save battery power. Therefore, high node activity often indicates an unintended behavior of the network. So, the number of messages sent by a node in a certain time is an indicator for the health of a network.

- **Network Partition**
The minimal requirement for a WSN is that at least it is connected in a way that every node could communicate with any other. If a network partition occurs this is not the case anymore. This is why it is important to check a network on partitioning.

- **Two Node-distinct Paths to the Sink**
This property states that for every node of a network at least two independent paths to the sink exist in a way that no intermittent node is visited twice. This property originates from the background of fire detection networks. These networks must guarantee that in case of a node failure the remaining network still is connected and that no partition occurred. This is proven by checking whether two node-distinct paths for each node of a network exists. So, this property is one level more restrictive than the network partition property and is therefore an indicator for a possibly even healthier network.

After considering the requirements defined from the user's point of view and the basic properties that need to be checked in order to be able to state the health of a network, we are ready to work out the tool design in detail.

2.3.3 WSNSpy Requirements

For the design of the WSNSpy the following features need to be treated as requirements in order to make an integration into the development environment feasible.

- **Using Deployed DSN**
A DSN which uses the DSN-Server for acquiring data is already set up as a development environment. Therefore, other implementations of a DSN were not taken into account for the realization of the distributed spy tool.

- **Integration into DSNAnalyzer**
As the DSNAnalyzer is a heavily used tool for configuring and debugging the WSN actually developed, the integration of the WSNSpy into the DSNAnalyzer should be possible. Apart from this, the DSNAnalyzer contains some components like the communication with the DSN-Server which could be reused in the WSNSpy. Therefore a combination of the two tools should be investigated.

- **Various Input Sources**
As stated earlier, to use the WSNSpy efficiently, it is important that the tool is able to process data from various input sources. Besides the already mentioned

format of the spy logs, it should be possible to process logs generated by the sensor nodes itself and as a third input source the log files originating from Glomosim simulations.

- **Evaluation based on Low-level Information**
  To be as independent as possible of later implemented protocols, the evaluation should be base on low-level information only. The implemented protocols so far for the A80 radio module are the physical (PHY) and the media access layer (MAC). To guarantee a successful message transmission the MAC layer is needed, as acknowledgments must be considered. Therefore, the MAC header format is necessary for the evaluation. However, higher-layer protocols can be omitted.

In the next section the functionality of the WSNSpy tool will thoroughly be discussed.

2.3.4 WSNSpy Design

The intention of developing the WSNSpy tool was to provide a tool that has the ability to analyze and evaluate data containing certain information about sensor node communication. The information needed by the tool must contain the sender, receiver, and the time stamp of a message transmission. The tool then reconstructs the network topology based on this information and allows to analyze the network by the user providing certain statistical numbers. In a second step the network can be evaluated by checking if one or multiple properties are fulfilled by the network.

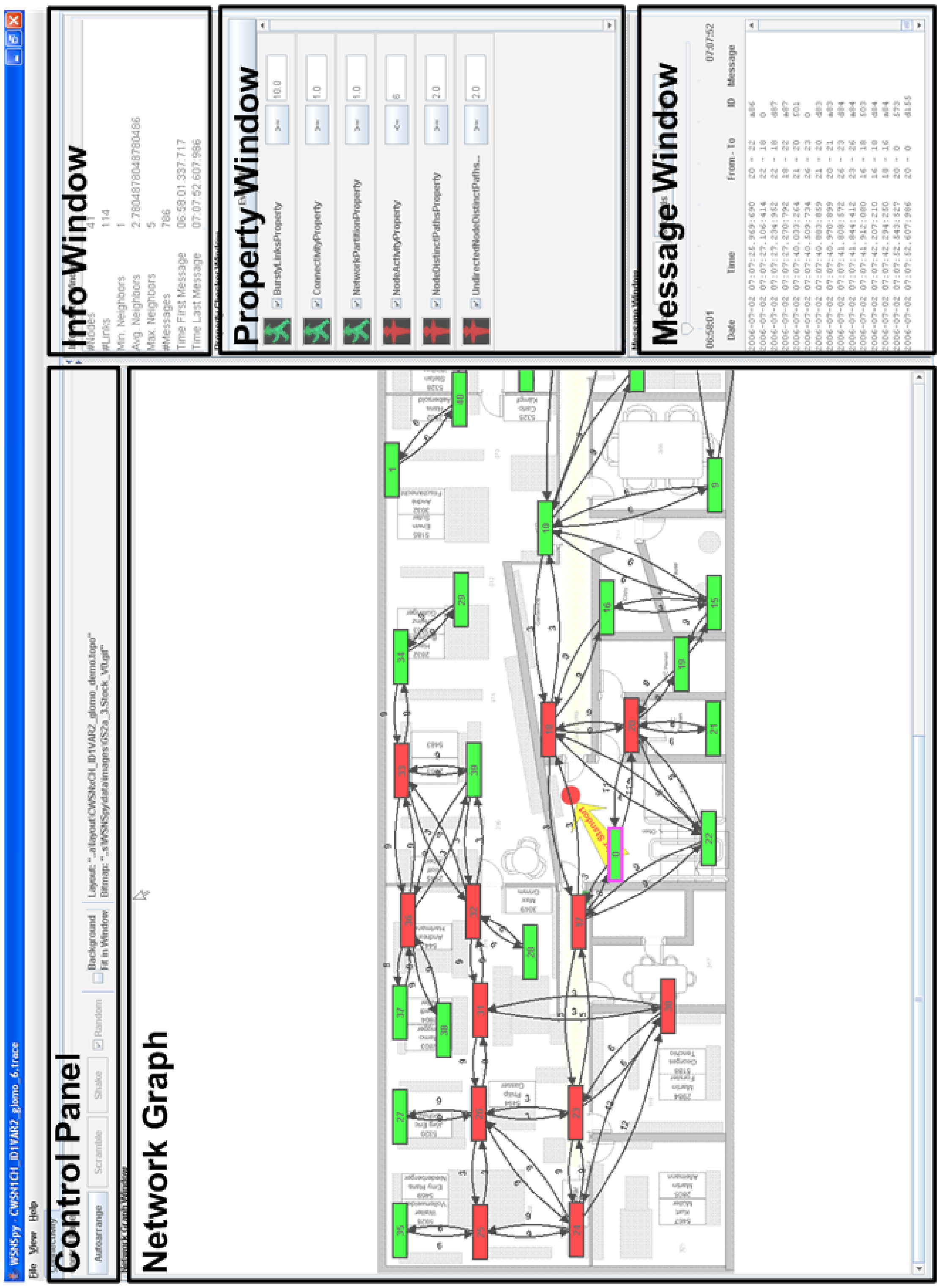
From the use cases defined in Section 2.3.1 the layout of the spy tool was designed which is shown in Figure 2-5. Every sub-window provides a certain functionality and is connected to one or several sub-windows. However, the layout of the tool does not explain how the components are connected and what the sub-windows purpose are. So, we are going to describe how the tool is designed.

The logical design of the WSNSpy is shown in Figure 2-6 in which the different modules of the tool and their dependencies are displayed. The size of the rectangles is an indicator of the complexity, size, and importance of the module. For the explanation the picture is traversed from top to bottom and from left to right. As already mentioned, the input data, originating from various sources like spy nodes, sensor nodes, and simulation, denotes the starting point of the WSNSpy.

2.3.4.1 Input Data Processor

The `InputDataProcessor` module operates in three consecutive phases which are repeated on new data input. The three phases are illustrated in Figure 2-7 and described below.

1. **Assemble Messages from Logged Events**
   In a first step the input data needs to be processed in order to get successful



Layout: "..\layout\CWSN\CH_ID1VAR2_glomo_demo.topo"
Bitmap: "..s\WSNSpy\data\images\GS2a_3.Stock_V0.gif"

#Nodes	41
#Links	114
Min. Neighbors	1
Avg. Neighbors	2.7804878048780486
Max. Neighbors	5
#Messages	786
Time First Message	06:58:01.337.717
Time Last Message	07:07:52.607.986

🟩	BurstyLinksProperty	>=	10.0
🟩	ConnectivityProperty	>=	1.0
🟩	NetworkPartitionProperty	>=	1.0
🟥	NodeActivityProperty	<=	6
🟥	NodeDistinctPathsProperty	>=	2.0
🟥	UndirectedNodeDistinctPaths...	>=	2.0

06:58:01 07:07:52

```
Date        Time               From - To   ID    Message
2006-07-02  07:07:25.969:690   20 - 22     484
2006-07-02  07:07:27.106:414   22 - 18     0
2006-07-02  07:07:27.234:952   22 - 18     487
2006-07-02  07:07:27.270:792   18 - 22     487
2006-07-02  07:07:40.033:264   21 - 20     501
2006-07-02  07:07:40.509:734   26 - 23     0
2006-07-02  07:07:40.883:859   21 - 20     483
2006-07-02  07:07:40.970:899   20 - 21     483
2006-07-02  07:07:41.808:872   26 - 23     484
2006-07-02  07:07:41.844:412   23 - 26     484
2006-07-02  07:07:41.912:080   16 - 18     503
2006-07-02  07:07:42.207:210   16 - 18     484
2006-07-02  07:07:42.294:250   18 - 16     484
2006-07-02  07:07:52.543:527   20 - 0      573
2006-07-02  07:07:52.607:986   20 - 0      4155
```

Figure 2-5: **WSNSpy Screenshot**

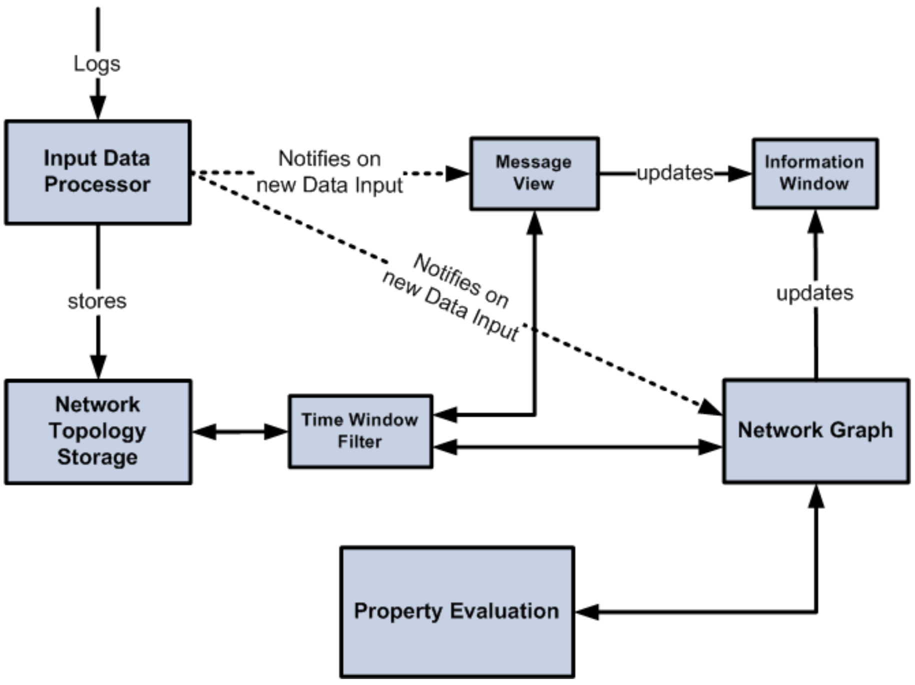


Figure 2-6: **WSNSpy Design**

message transmissions. This task depends on the kind of the input source and the following three cases are considered in the WSNSpy.

- **Sensor Node**
  Every node generates a log event if a message is received, and another log event if a sent message is acknowledged. Out of a send event and a receive event, successful message transmissions can be deduced as the logs contain origin, destination, and message ID.
- **Spy Node**
  Spy nodes cannot be certain if a packet is received by its intended receiver. Therefore, a message transmission cannot be assembled from a send and a receive event. Nevertheless, events generated from the overheard message and acknowledgment provide enough information to determine a successful message transmission.
- **Simulation**
  The logs originating from simulation cases are similar to the ones coming directly from the sensor node. As they only differ in the format of the logs, the concept described in the first case can be applied.

So, successful message transmissions are assembled by the `InputDataProcessor` module out of the provided events.

2. **Reconstruct Network out of Messages**
   A message contains certain information including its sender and receiver. The

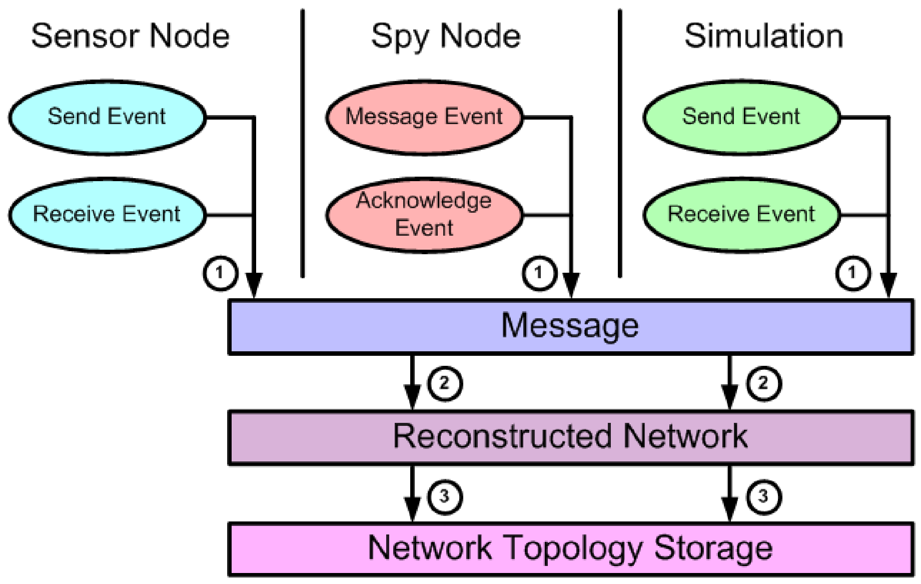


Figure 2-7: **WSNSpy Data Flow**

capture time is added to the log event by the DSN node. From this knowledge, we can reconstruct a part of the network as it is obvious that sender and receiver node must be neighbors. Otherwise they would not have been able to communicate with each other. By doing this for every message transmission, the network grows more and more until we have a complete representation of the physical network.

In case of full network coverage by the spies, we even can be sure that the complete network can be reconstructed, as all communicating nodes will be found sometime. On the other hand it can be stated that nodes not involved in any communication can hardly be declared as a part of the original network.

3. **Store Network Topology**
   Whereas the input of the WSNSpy is based on messages, the evaluation itself depends on the reconstructed network. Therefore, the messages are transformed again into the different network components, i.e. nodes and edges.

2.3.4.2 Network Topology Storage

The reconstructed network must be stored in a useful manner. In order to comfortably operate on the network it should be possible to navigate easily through the network by reaching from every node its attached edges and neighbors. To achieve this goal, the `NetworkTopologyStorage` is built up as shown in Figure 2-8.

For every node a reference to its incoming and outgoing edges are stored as well as a reference to its neighboring nodes. The same is done for the edges, by storing a reference to its source and destination node. As the edges were developed from messages, the time when the transmission has happened is stored in edges as well.

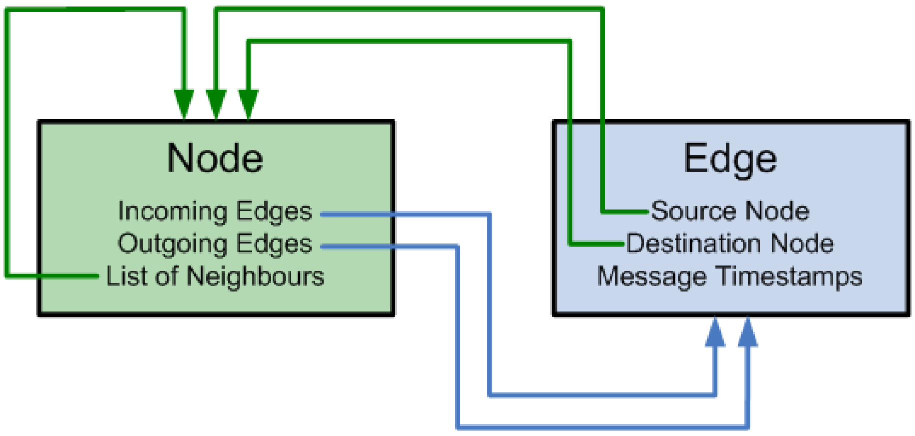


Figure 2-8: **Network Topology Storage**

2.3.4.3 *Time Window Filter*

For the analysis and evaluation of the network not all messages need necessarily to be considered. Therefore, the possibility must be provided to select a certain time frame. Only messages transmitted in this time frame are considered for the evaluation. Every `NetworkTopologyStorage` access occurs through the `TimeWindowFilter` module.

2.3.4.4 *Message View*

This module lists all messages of the network that complies with the timing requirements of the time window. The `MessageView` module belongs to the analyzing part of the tool and provides the functionality to investigate every single message in detail.

2.3.4.5 *Network Graph*

The `NetworkGraph` displays a graphical representation of the network and allows the user to arrange the nodes according to their physical positions. The network is updated every time the `NetworkTopologyStorage` has changed.

For analysis purposes, the module provides functionality to select certain network elements for further investigation. Moreover, the `NetworkGraph` is the starting point of the evaluation which is done in the `PropertyChecker` module.

2.3.4.6 *Information Window*

In this passive window information about the network is displayed. The numbers provided depend on the selected network components in the `NetworkGraph`. If no element is selected summary information about the whole network is provided.

The displayed information consists of several key numbers like the number of neighboring nodes, incoming and outgoing links, number of overheard messages and so on.

2.3.4.7 Property Evaluation

The core part of the WSNSpy tool, i.e. the evaluation, lies in the `PropertyEvaluation` module. The actual network is evaluated against certain properties that embodies an indicator for the health state of the network as described in Section 2.3.2. For each node every property is checked on its fulfillment. If all properties evaluate correctly the node is marked green, in case of an incorrect termination of at least one property the node is marked red. This way, nodes that do not comply to the defined restrictions are highlighted and consequently call the user's attention in order to do further investigations.

3

Implementation

This chapter describes the actual implementation of the first and last component of the system presented in Section 2.1. First, we describe the implementation of the A80-Spy and afterwards of the evaluation tool WSNSpy.

3.1 A80-Spy Implementation

For the spy nodes we use the A80 radio module, which is described in Appendix A.1. The main reasons for this decision are that

- the A80 radio module is available and actually used,
- in-deep knowledge about the hardware is available,
- a software stack or at least the layers needed by the spy are available, and
- the spy must be able to receive packets sent from nodes belonging to the sensor network. As these nodes contain the A80 radio module, it is advantageous to use the same module for the spy nodes.

While we could use the hardware of the A80 radio module "as is", we developed the spy software within this thesis. However, parts of the already existing software stack could be integrated into the spy software. Hence, we first give a short description of the software stack, before we present the implementation of the spy in detail.

3.1.1 A80 Software Stack

As discussed in Section 2.3.3, a spy must be able to receive packets and to extract the information about the sender and receiver. This information is hold by the *Medium Access Control (MAC)* header of a packet. The functionality for receiving packets is provided by the *Physical Layer (PHY)* and in order to extract sender and

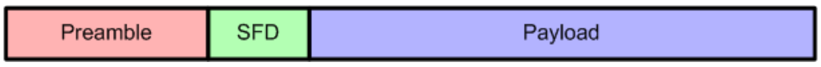


Figure 3-1: **Packet Format PHY**

receiver data the MAC layer functionality is needed. However, higher-layer data like neighborhood tables or routing information is not of interest. Therefore, the PHY and MAC layer of the A80 software stack is explained below.

3.1.1.1 Physical Layer (PHY)

The A80 PHY uses a preamble and a *Start Frame Delimiter (SFD)* to communicate as shown in Figure 3-1. The preamble is needed to signalize to the receiver that a packet will be sent shortly and to synchronize the receiver with the sender. The length of the preamble depends on the wake-up time synchronization between sender and receiver and is set by the MAC layer. The SFD marks the start of the actual packet transmission.

3.1.1.2 Medium Access Control Layer (MAC)

In WSNs, the MAC protocol is probably the most important protocol. This is due to the fact that energy is a very critical resource in WSNs. A sending or receiving node uses much more power than a node in sleeping state. Therefore, the following two points must be considered.

- Active states, i.e. sending and receiving should be as short as possible.
- Collisions due to simultaneous packet transmissions should occur as few as possible, because sending power is wasted.

Hence, it is crucial that the medium access is highly optimized. However, there does not exist the optimal MAC protocol for every WSN, because networks differ in various ways, like size, purpose, activity, etc.

For the actually developed WSN, WiseMAC [14] is used as MAC protocol. WiseMAC was designed for ultra low power multi-hop networks with very low data rate. It is based on Carrier Sense Multiple Access (CSMA) with collision avoidance. This is achieved by sensing the medium before a packet is sent. Moreover, WiseMAC uses a preamble sampling technique, which minimizes the power consumed when listening to an idle medium. The protocol stores the sampling schedule of a node's direct neighbors which allows to adapt the preamble length due to the knowledge of the neighboring wake-up times.

In Figure 3-2 we illustrate the format of a WiseMAC packet. It consists of a header, payload, and footer. The header contains a base header which is used to identify the packet by providing the overall length, a sequence number, and the distinction between message and acknowledgment type, among other information. The MAC header also includes the destination address and in case of type message (MSG) the

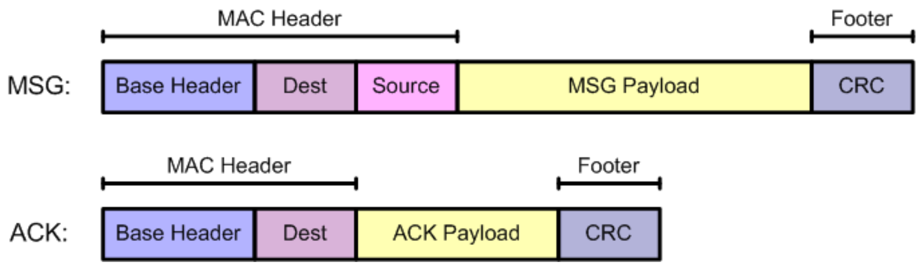


Figure 3-2: **Packet Format MAC**

source address as well. The PHY layer adds in front of the complete MAC packet the preamble and the SFD, as it was shown in Figure 3-1.

For the implementation of the A80-Spy, the distinction between messages and acknowledgments, the packet length and the information about sender and receiver address is used.

3.1.2 *Software Architecture*

In this section the implementation of the spy on the A80 radio module is presented. Below, we explain the algorithm used for the spy, followed by the output format for overheard messages and acknowledgments. Finally, we describe various aspects considered for the spy implementation.

3.1.2.1 *A80-Spy Algorithm*

We implemented the A80-Spy among the concept illustrated in Section 2.2.2.2. For intercepting packets the method to receive messages provided by the PHY layer is used. This method requires a callback method to be set which is called every time a packet is received. The spy remains in receiving state until a complete packet was overheard. Afterwards, the callback method processes the packet according to Algorithm 1.

In case of a received message or an acknowledgment reception the packet is stored in a buffer in order to free the receiving buffer for subsequent packets. For each type of the packet a separate buffer exists. Then, an event flag is set and the internal spy statistics are updated. Finally, the receive method of the PHY is called again.

During the processing of a packet, the spy is not in reception state. Therefore, it is crucial to keep the processing part as short as possible. The time between the end of a message transmission and the beginning of an acknowledgment is about $4\ ms$. This is the time needed by the receiving node for processing the message, to generate the acknowledgment, and to prepare its transmission. As the time for an output takes about $0.5\ ms$ per character at a rate of $19200\ Baud$, captured messages cannot be printed out immediately. In fact, intercepted packets are temporarily stored and an event is used to print the message out later on. Events are managed by an event-handler which periodically checks whether any flags for certain events were set. If

Algorithm 1: Processing Overheard Packet

```
Input: Packet Received
if ! ACK then
    // Packet is of type MSG;
    store MSG in msg_buffer;
    set msg_Event flag;
    update statistics;
else if ACK && ! NACK then
    // Packet is of type ACK;
    store ACK in ack_buffer;
    set ack_Event flag;
    update statistics;
else
    // Packet is of type NACK;
    drop packet;
end
// Re-init receiving;
phy_receive_frame();
```

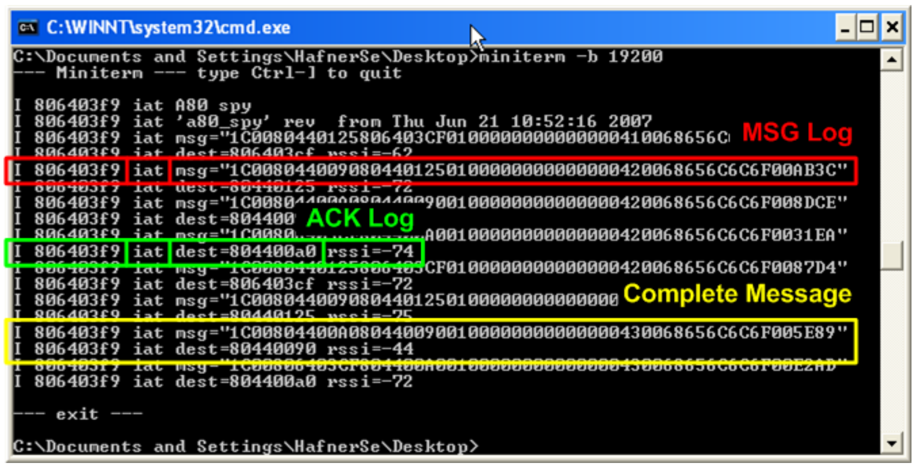
this is the case, the corresponding event is triggered. By this concept, receiving is not blocked anymore during the message output.

If the spy listens to message transmissions that occur shortly after each other, the spy might not be fast enough to output every message in time. To prevent the buffer being reused before the output event has finished, secondary buffers are introduced for each packet type. Every buffer is only rewritten after the output of its content has finished. If primary and secondary buffer are occupied due to high interception activity of the spy, the message will be discarded.

3.1.2.2 Spy Output Format

The output of the spy is a simple line of text which is different for messages and acknowledgments. The Figure 3-3 shows the output of several captured packets by one single spy and is used to illustrates the format of the spy logs.

The output marked with a red frame shows a spy log of a packet that was identified to be of type message. The first part is automatically generated by the A80 logging service and contains an identifier and the address of the spy node. The following three character are used to differentiate between the different logging schemes. Logs generated by the spy part of the A80 software are always marked as "iat". The message is printed in hexadecimal notation, in which every byte is encoded as a 2-digit hex number. This implies that the log output gets twice as long as the message has been, without considering the first part of the log previously explained. In order to keep the log output as short as necessary, we have to make sure that the spy is able to write more than twice as fast as the A80 is communicating via the wireless link.



```
C:\Documents and Settings\HafnerSe\Desktop>miniterm -b 19200
--- Miniterm --- type Ctrl-] to quit

I 806403f9 iat A80 spy
I 806403f9 iat 'a80_spy' rev  from Thu Jun 21 10:52:16 2007
I 806403f9 iat msg="1C0080440125806403CF01000000000000000041006865 6C
I 806403f9 iat dest=806403cf rssi=-62
I 806403f9 iat msg="1C008044009080440125010000000000000000420068656C6C6F00AB3C"
I 806403f9 iat dest=80440125 rssi=-72
I 806403f9 iat msg="1C00804400A080440090010000000000000000420068656C6C6F008DCE"
I 806403f9 iat dest=804400
I 806403f9 iat msg="1C0080                A0010000000000000000420068656C6C6F0031EA"
I 806403f9 iat dest=804400a0 rssi=-74
I 806403f9 iat msg="1C0080440125806403CF010000000000000000420068656C6C6F0087D4"
I 806403f9 iat dest=806403cf rssi=-72
I 806403f9 iat msg="1C00804400908044012501000000000000000
I 806403f9 iat dest=80440125 rssi=-75
I 806403f9 iat msg="1C00804400A080440090010000000000000000430068656C6C6F005E89"
I 806403f9 iat dest=80440090 rssi=-44
I 806403f9 iat msg="1C00806403CF804400A0010000000000000000430068656C6C6F00E2AD"
I 806403f9 iat dest=804400a0 rssi=-72

--- exit ---

C:\Documents and Settings\HafnerSe\Desktop>
```

Figure 3-3: **A80-Spy Log Format**

This is done by setting the baud rate of the serial interface of the radio module to 19200 *Baud*, which is four times as fast as the communication via the wireless link.

The log output for an intercepted acknowledgment, marked with a green frame, is shorter as it does not contain any payload data. The first two parts of the log, i.e. the spy's hardware address and the logger are the same as for message logs. Instead of the packet's content only the destination address and the RSSI value measured by the receiver of the preceding message are printed out.

Finally, the yellow frame shows the combination of the two different log formats of a message transmission and its corresponding acknowledgment. This can be verified as the destination address (80440090) of the acknowledgment is identical to the source address of the message. The source address (80440090) is contained in the message header after the base header (1C00) and the destination address (804400A0).

In the following we explain several facets of the implementation which do not belong to the core part of the spy, but still are worth to be mentioned.

3.1.2.3 Watch Dog

For redundancy reasons we implemented a watch dog which terminates and re-inits the receiving process if no packet is received for 60 seconds. This action was taken, because we observed a reduction of the spy's performance when running the spy for a long time including phases without any transmission activity. The watch dog has the disadvantage that a message is missed if it is transmitted at the same time as the watch dog fires. However, the risk of a message loss is reasonable compared to the benefits of the watch dog.

RPC Command	**Description**
spy.init	Initialization of the spy. All statistical values are set to zero.
spy.stop	Stops the spy from intercepting messages.
set.chan <param>	Sets the receiving channel of the spy. The spy.stop command must be executed in advance. The parameter contains the channel to be set [0...79].
set.ant <param>	Specifies the receiving antenna. The A80 radio module contains two orthogonally arranged antennas. The parameter contains the antenna to be used.
get.spystats	Prints the statistical numbers from the spy, i.e. the number of overheard messages and number of acknowledgments.
get.spypar	Returns the parameters used by the spy for the current configuration, i.e. the chosen channel and antenna are printed out.
get.rssi	This command triggers the measurement of the RSSI value, i.e. the actual signal strength is measured.

Table 3-1: **RPC Commands Table**

3.1.2.4 RPC Commands

The spy provides several commands executable through a *Remote Procedure Call (RPC)* interface in order to apply configurations to and to get information from the spy. In Table 3-1 the supported commands are listed.

3.1.2.5 Statistics

The spy keeps track of the number of received messages and acknowledgments by maintaining statistics. The statistics can be read out by the corresponding RPC command.

In the next section the implementation of the WSNSpy tool is presented.

3.2 WSNSpy Tool Implementation

For the implementation of the evaluation tool a development environment, including several tools and libraries are needed, which we describe first. Afterwards, the implementation itself is presented in detail.

3.2.1 Development Environment

In this section the environment we used for the spy tool development is presented. First, the chosen programming language is justified, then some libraries and modules used for the WSNSpy are explained.

3.2.1.1 Programming Language

In contrast to the spy software which is written in C, we decided to use Sun's Java [17] as programming language for the WSNSpy tool. We considered several factors in order to get to this decision, which are elucidated in the following.

- Apart from the implemented RPC commands, the spy node software does not need to provide functionality for user interaction as its only purpose is to print out overheard traffic. The spy tool, however, provides functionality to analyze and evaluate the overheard traffic, which leads to the insight that user interaction must be provided. In order to allow the user interaction to happen intuitively and comfortable, a *Graphical User Interface (GUI)* is advisable. Considering the Swing toolkit, Java provides a good environment for GUI development. Swing provides a rich variety of graphical objects (e.g. Frames, Buttons, Textareas, etc.) that can be assembled to a customized GUI.
- The evaluation tool is designed to run on an ordinary computer, for which nowadays performance is hardly a critical criterion anymore. Considering this, the focus for the WSNSpy development was laid on usability rather than resource consumption optimization. For our purposes, Java meets the stated constraints best.
- Considering the given development environment, Java is used for compatibility reasons as well. The WSN configuration and analysis tool DSNAnalyzer, which is comparable to the WSNSpy in the sense of utilization, is already implemented in Java. As a few parts of the DSNAnalyzer cover the needs of the WSNSpy, especially the communication between the tool and the DSN-Server, it was advisable to reuse certain modules of the DSNAnalyzer in the WSNSpy. However, this requires the use of Java as programming language.
- As the tool should not be restricted to a certain platform, Java as a platform independent programming language meets this requirement as well.
- Java provides many libraries which drastically facilitates programming. E.g. the JGraph library provides an interface for drawing network views including user interactions, enabling simple integration in a Java application.
- Finally, we already have a certain experience working with Java, which allowed us to reduce the phase of familiarization with the Java development environment to a minimum.

Considering the arguments explained, we decided to develop the WSNSpy as a Java application. The *Java Development Kit (JDK)* used for the evaluation tool is based on version 1.5.0_11. In order to comfortably work during the spy tool development, we used the *Integrated Development Environment (IDE)* of the Eclipse [18] Project.

3.2.1.2 JGraph Library

An important part of the evaluation tool is the reconstruction and display of the network. This is done by showing the nodes and links of the network as a network graph. For this purpose, we make use of the open source library *JGraph* [19], which provides a full-featured graph model interface for Java. Due to the open source principles, it is possible to manually extend and alter certain JGraph functionalities, which make JGraph even more useful. JGraph is based on cells of type node and edge which are specified by a rich variety of attributes concerning the appearance and operability of the cells. The network topology is defined by adding to every edge its source and target node cell.

3.2.1.3 Modules Used from the DSNAnalyzer

Into the WSNSpy some modules originating from the DSNAnalyzer application can be integrated. This is done on the one hand, because there is no sense in developing already existing work, and on the other hand to meet the requirement of compatibility of the WSNSpy to the DSNAnalyzer. The three modules mentioned below are integrated into the spy tool.

- **DSNCommunication**
  This module handles the communication between the tool and the DSN-Server. It provides the functionality to fetch events from the server's database that were collected via the DSN, and to configure the sensor nodes by executing RPC commands. In the WSNSpy only the first part of the DSNCommunication module, i.e. for database interaction is used.
- **EventList and Event**
  The events stored in the database of the DSN-Server need to be parsed on import for further use. As all events are of the same format, we were able to use this module for the spy events as well. The imported events are passed to the next module as a list of events.
- **Time Object**
  This module provides the functionality of working with date and time objects. It is similar to the classes provided by the JDK except that the Java *Application Programming Interface (API)* does not support microseconds.

3.2.1.4 Logger

For the development of the WSNSpy the *Apache Logging Service log4j* [20] was used. The logger enables logging during runtime and can be controlled by editing a configuration file.

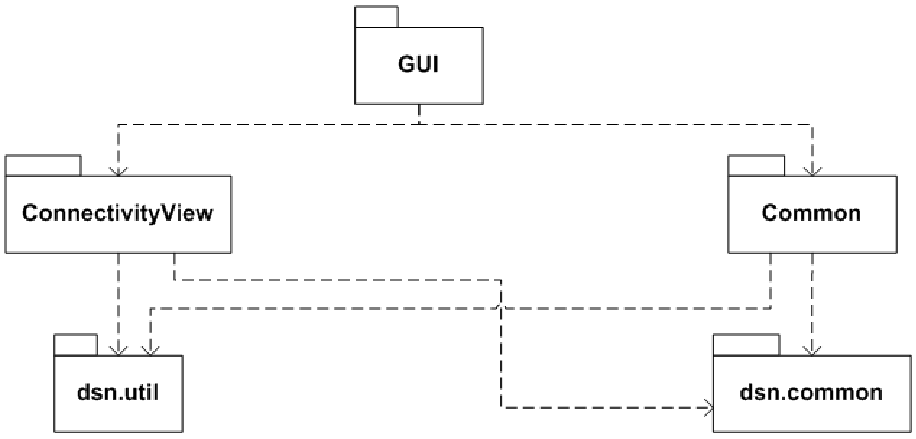


Figure 3-4: **WSNSpy Package Diagram**

After presenting the development environment used for the WSNSpy, the implementation of the evaluation tool is thoroughly discussed.

3.2.2 *Software Architecture*

Before going into detail, the implementation of the WSNSpy is presented by explaining how the tool is organized. The different packages and the relationship between them are shown in Figure 3-4.

The implementation is organized into five packages. The `GUI` package is the main package, from which the application is started. The rest of the tool is divided into a `Common` and a `ConnectivityView` package. The first package contains classes on which various parts of the tool are based on. In the second package all classes concerning the `ConnectivityView` tab are contained. The packages `dsn.util` and `dsn.common` are packages included from the DSNAnalyzer as described in Section 3.2.1.3. In the latter package classes closely related to the `ConnectivityView` tab are contained. In order to add a new tab as a tool extension later on, simply a new package must be included.

In the following, the three packages belonging to the WSNSpy implementation are explained in detail.

3.2.2.1 *GUI Package*

The `GUI` package contains the overall class `WSNSpyMain`. It initiates the `ConnectivityView` tab and potentially subsequent tabs. The `WSNSpyMain` class could be merged with the `DSNAnalyzerMain` class in order to have the two tools combined into a single one, divided into several tabs. Additionally, the property file of the WSNSpy needs to be merged with the one of the DSNAnalyzer.

3.2.2.2 Common Package

The Common package contains several classes that are used by various classes outside of this package. Following classes are a member of the Common package.

Input Data Processor

As described in Section 3.2.1.3, collecting the events is done in the DSNCommunication module of the DSNAnalyzer. So, the events collected in the EventList are provided to the InputDataProcessor via an Observable - Observer connection, over which the InputDataProcessor is notified on changes of the EventList.

The InputDataProcessor afterwards processes the input data sequentially according to the input source. The input source is specified by setting the mode property in the property file to

- **"dsn"** if logs originate directly from the sensor nodes,
- **"sim"** if the simulation is the input source, or
- **"spy"** if the logs are generated by spy nodes.

In "dsn" and "sim" mode two log events exist, a send and a receive event. A receive event is recognized if it contains "id=" and "source=" as string identifiers. If only "id=" is contained the event is treated as a send event.

Events originating from spy nodes are either message or acknowledgment events and are distinguished by the string identifiers "msg=" for a message event and "dest=" for an acknowledgment event.

The InputDataProcessor not only acts as an Observer of the EventList, but as an Observable class which notifies its Observer on network topology changes, as well. So, the ConnectivityView, which is registered as an Observer of the InputDataProcessor, gets notified every time a bunch of events are processed and messages are generated.

Network Topology Storage

The InputDataProcessor builds up the *Network Topology Storage* which is the base for all further investigation and processing. As illustrated in Figure 3-5, the storage consists of nodes and edges which are instances of the Node and Edge classes. In order to obtain the network topology the message interface IMessage is used which defines how the different types of messages are accessed.

In every node references to incoming and outgoing links are stored, as well as a list of its neighbors. On the other hand, every edge stores a reference of its origin and destination node. In addition, the time stamps are stored for every usage of the link. The references between nodes and edges connect the network elements together and allow an easy navigation through the network.

Every node gets a status assigned that is shown by the color if a node is highlighted. How the status is defined depends on the fulfillment of certain properties which are explained in Section 3.2.2.4.

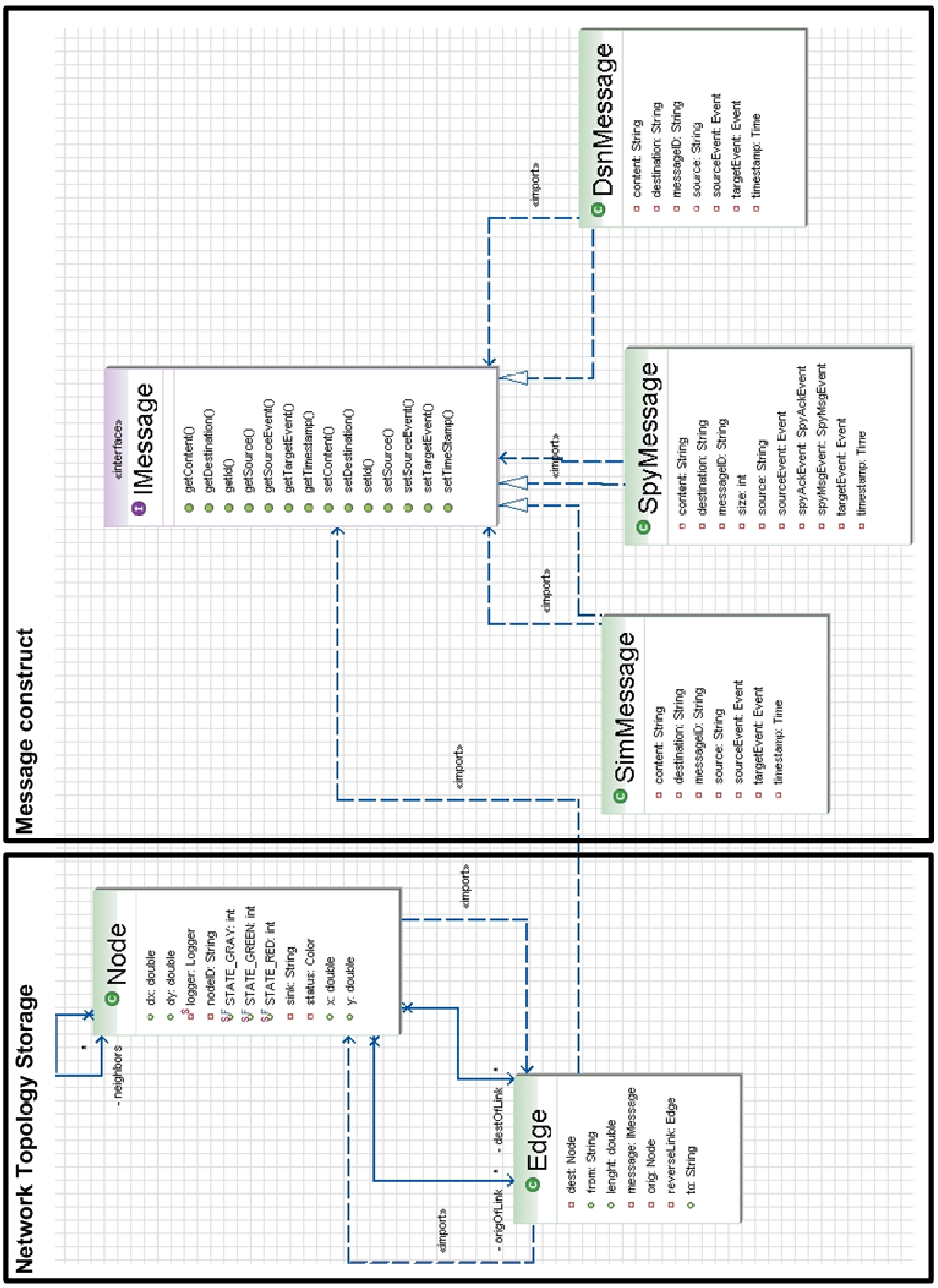


Figure 3-5: **Class Diagram Network Topology Storage and Message Construct**

3.2.2.3 ConnectivityView Package

The package contains classes that are used only in the `ConnectivityView` tab. The package is split into a class for every sub-window and some additional classes for controlling purposes. In the following, a selection containing the most important classes of this package is described in detail.

ConnectivityView
The `ConnectivityView` class is the main class for this tab, holding control over all sub-windows and the connections in between.

NetworkGraph
The largest sub-window in size displays the network topology of the reconstructed network. The `NetworkGraph` acts as an `Observer` of the `InputDataProcessor` and therefore gets notified on new message input. For displaying the network graph, the JGraph library is used which provides functionality to draw the network and to react on user interaction.

The `NetworkGraph` is used to control selected as well as highlighted network elements. We differentiate between selection and highlighting because if a node is selected, not only this node is highlighted, but all neighboring nodes as well as incoming and outgoing edges are highlighted. This allows to examine a part of the network more thoroughly. Highlighted nodes are colored according to the status color set, i.e. red or green (or gray by default). Edges are labeled according to the number of message transmissions.

MsgWindow
The `MsgWindow` controls a text field and reacts on `Observer` notification from the `InputDataProcessor`. The sub-window interacts with the `NetworkGraph` for the selection of network elements. On selecting a certain message, the corresponding edge is selected in the `NetworkGraph` as well, and vice versa.

The `TimeWindowController` is attached to the `MsgWindow` and allows to consider only certain messages within a time frame for the evaluation. This is done by adjusting the visible time frame. The time window can be moved forward and backwards by using a slider button.

InfoWindow
The `InfoWindow` is a passive sub-window and simply provides information about the network, or single nodes and edges depending on the selection of network elements.

PropertyChecker
On initialization the `PropertyChecker` class searches the plugin folder of the WSNSpy for Java classes implementing the `Property` interface. The found `Property` classes are included into the `PropertyChecker` sub-window and are then ready to check a network on fulfillment. The `Property` classes are included by using a class loader, which means that the `Property` classes do not have to be registered anywhere

in the code. So, there is no need to compile the whole WSNSpy tool just for including a new `Property`. However, the `Property` class itself must be compiled, which can be done by adding the jar archive of the WSNSpy to the classpath.

Once the `Property` classes are loaded, the `PropertyChecker` provides the functionality to check a network whether the properties are fulfilled or one or several network elements violate a certain property.

In the next section the purpose of the `Property` plugins are explained. The `Property` classes do not build an individual package, but the files are separated from the rest of the tool and located in a special folder, instead.

3.2.2.4 Property Plugins

A `Property` plugin has to implement the `Property` interface in order to be considered by the `PropertyChecker` and to appear in the sub-window. The `Property` interface defines how the `Property` is accessed and demands, apart from some other methods for configuration purposes, the "eval" method being implemented. This method is called by the `PropertyChecker` in order to evaluate a certain network topology provided as a parameter value to the `Property`. The result of the evaluation consists of all nodes of the network to which "true" or "false", or in case of non-determinism "unknown" is mapped.

Apart from the `Property` plugins explained so far that are used to evaluate a certain network topology, the WSNSpy uses a property file for configuring the spy tool, itself. In the following these properties are explained.

3.2.2.5 Property File

The WSNSpy tool can be configured through a property file in order to specify long-term configurations. The following properties can be specified.

- The path of the DSN-Server is a necessary parameter in order to be able to load data from the server. If this property is not set, events can only be imported from a local file.
- In order to use the debug session functionality where the DSN-Server is polled constantly for database changes, a logging port must be specified.
- The mode property specifies the input source. Possible values are "spy", "dsn", or "sim", which are explained in Section 3.2.2.2.
- The sink node of the network must be specified. This property is used by certain properties that evaluate paths of a network.
- The network layout file contains information on how the nodes should be positioned in order to match to the physical arrangement. If a layout file is specified in the property file nodes are arranged accordingly, otherwise random positions are applied.

- By specifying a map as a background image, the nodes can be placed according to their physical positions.

More detailed instructions concerning the different functionalities can be found in the readme file of the WSNSpy.

4

Evaluation

In this chapter the whole system developed in this thesis is evaluated in order to verify the correct functionality of every single part as well as the complete system.

4.1 Evaluation Cases

As described in Section 2.1, the system consists of four components with individual functionality that are connected building a chain. For the evaluation every component is tested individually, apart from the two components in the middle, i.e. DSN nodes and DSN-Server, which are evaluated in one single test. Therefore, we define the following tests for the evaluation.

1. **Spy Node Evaluation**
   The spy nodes denote the basis for the data acquisition and therefore must be able to capture a high percentage of the WSN traffic.
2. **DSN Node and DSN-Server Evaluation**
   The DSN consisting of the DSN nodes and the DSN-Server is used for collecting the traffic overheard by the spy nodes. The DSN only provides transport functionality and so, as it is used "as is", satisfying performance can be expected, but is tested anyway.
3. **WSNSpy Tool Evaluation**
   The WSNSpy tool is tested on two different properties. On the one hand, the question of how well the real system can be reconstructed, solely based on low-level information is investigated. On the other hand, we evaluate the properties provided by the WSNSpy in order to determine the health state of the network. The evaluation is done by comparing the network obtained by spying with the real network.

By evaluating the three cases explained, we can finally make a statement concerning the functioning of the whole system.

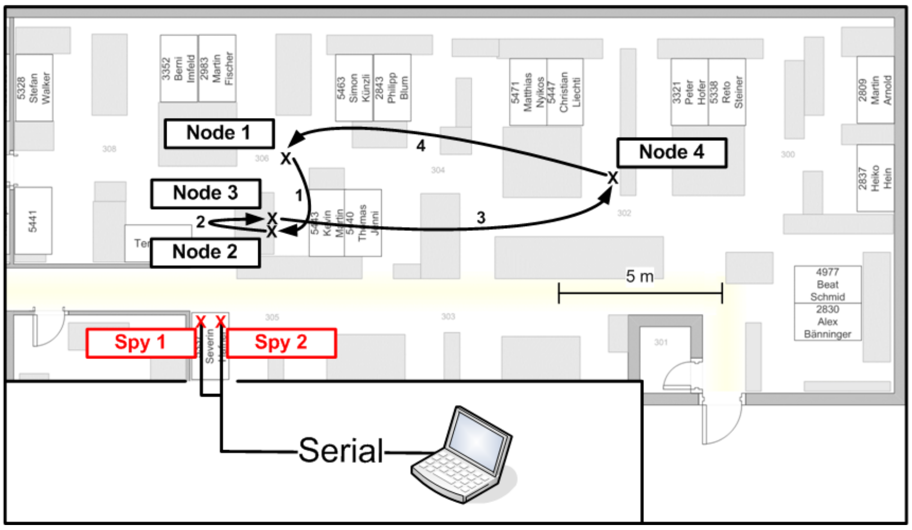


Figure 4-1: **Test Setup for the Spy Node Evaluation**

4.2 Evaluation Tests

In the following, for every part of the system the test performed for the evaluation is explained, the results are presented, and the conclusions are drawn.

4.2.1 Spy Node Evaluation

To evaluate the performance of the spy nodes it is crucial to know the exact number of successful message transmissions to be able to compare the number of sent messages with the number of captured message transmissions. This allows us to determine the percentage of overheard messages compared to the actually sent messages. The application used for the sensor network, as well as the evaluation framework is presented in the next section.

4.2.1.1 Test Setup

For the evaluation of the sensor network we used the "WiseMAC-Test" application, which provides the functionality of simply forwarding every received message without any retransmissions. The system under test shown in Figure 4-1, consists of four sensor nodes which are placed in close distance to each other. So we can state that the reception range is not an issue in this test.

The traffic generated by the test network is captured by two independently operating spy nodes which are placed directly next to each other. The spy nodes are connected via a serial connection to a computer, thus we get a reliable connection to the spy nodes which allows to count the number of overheard message transmissions.

With the test setup explained, we did five tests including precisely 1000 message transmissions. The tests were done in a large office building.

	Spy 1 (Serial)	**Spy 2 (Serial)**
Minimum	99.0%	99.0%
Maximum	100%	100%
Average	99.5%	99.7%

Table 4-1: **Spy Node Evaluation Results**

4.2.1.2 Results

The results obtained for the spy node evaluation is shown in Table 4-1. At least 99.0% of the message transmissions could be overheard by the spy nodes for each test. On the other hand, in at least one of the five tests all message transmissions could be overheard by the spy nodes. On average three to five messages were missed.

4.2.1.3 Conclusions

The test results show that there is almost no difference between the two spy nodes, which allows us to conclude that the software is working correctly. The minor differences of 0.2% on average can be explained by the placement, which was slightly different for the two spy nodes.

So, we can state that the spy nodes of the system provide suitable capturing functionality, which allows us to use the data acquired by the spy node as a basis providing enough information for the analysis and evaluation of the capture data.

By using two redundant spy nodes at the same time, we are able to increase the percentage of the overheard message transmissions even more. If we apply this approach to the tests described above, all messages could be captured, because every message transmission was overheard by at least one of the two spy nodes.

4.2.2 DSN Node and DSN-Server Evaluation

Beutel et al. stated in [3] that packets might get lost on the DSN, depending on the total amount of DSN traffic. Due to this fact it is necessary to test the DSN on packet losses. If we can verify that the DSN, which is not overloaded works reliable, the system provides the functionality to passively capture WSN traffic in a satisfying manner.

4.2.2.1 Test Setup

In comparison with the test explained above for the evaluation of the spy nodes, the system under test and the amount of the sent messages remain the same for the DSN evaluation. However, only one of the two spy nodes is connected directly the computer, using a serial connection as shown in Figure 4-2. Spy number "1" though, is connected via the DSN to the computer, which allows us to investigate the effects of the DSN. As the DSN only contains one single spy node, the loss of data due to an overloaded network can be neglected.

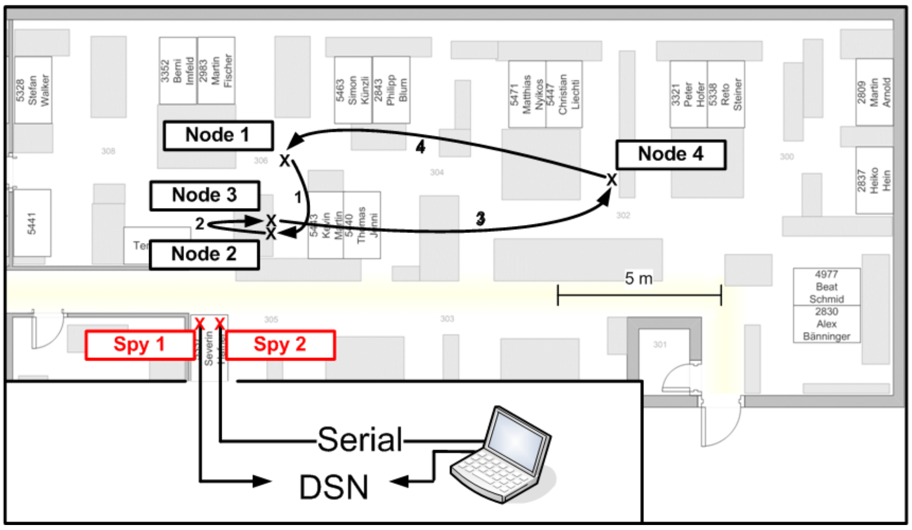


Figure 4-2: **Test Setup for the DSN Evaluation**

4.2.2.2 Results

The results obtained are shown in Table 4-2. The minimal percentage of the overheard message transmissions for any of the five tests is 98.1% for the spy node connected via the DSN to the computer, and 98.2% for the node connected directly via a serial link. For at least one of the five tests both of the spy nodes overheard all 1000 message transmissions. On average 99.4% and 99.5% of the messages could be overheard.

	Spy 1 (DSN)	**Spy 2 (Serial)**
Minimum	98.1%	98.2%
Maximum	100%	100%
Average	99.5%	99.4%

Table 4-2: **DSN Evaluation Results**

4.2.2.3 Conclusions

The evaluation clearly shows that considering the number of lost messages, there is no difference in using the DSN or connecting the spy node directly to the computer using a serial interface. Obviously, for a single node the DSN is not overloaded and so we can state that in this very case, the DSN does not perform worse than by using a serial interface.

We showed that the system's capturing part has good functionality for overhearing WSN traffic. Hence, the captured data provides a reasonable basis for the evaluation of the captured data.

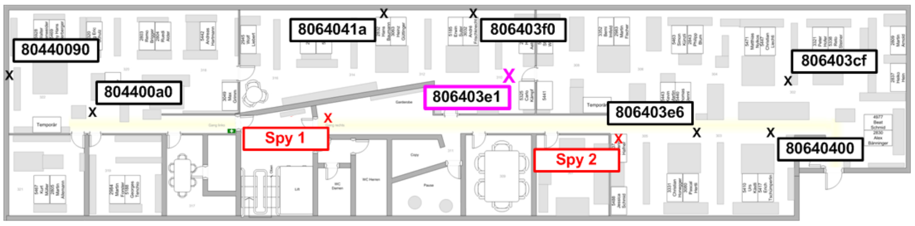


Figure 4-3: **Test Setup for the WSNSpy Evaluation**

4.2.3 *WSNSpy Tool Evaluation*

After the evaluation of the capturing part showing good functionality for overhearing WSN traffic, the WSNSpy tool is evaluated. First, the reconstructed network is verified against the real network topology and afterwards the result of the evaluated properties are compared to the state obtained from the sensor nodes.

4.2.3.1 *Test Setup*

For the evaluation of the WSNSpy tool a different application is used for the sensor network, as sending messages in a circle is too simple for a meaningful property evaluation. Therefore, the *Neighborhood Detection (ND)* algorithm is used as test application.

The ND application was developed for the commissioning of a sensor network in which the determination of a node state only depends on the states learned from direct neighbor nodes. The ND algorithm assigns to every node one out of three possible node state: As long as it does not have a path to the sink, the node state is red. Nodes connected to the sink without having two node-distinct paths are marked yellow, while the state of a node having two node-distinct paths to the sink is green.

The test setup is shown in Figure 4-3. The test network consists of eight sensor nodes including the magenta colored node in the middle (`806403e1`), which denotes the sink. In connection with the sensor network the DSN is used, which allows us to obtain the node states of the sensor network in reality.

The spy network consists of two distributed spy nodes connected via the DSN. In the test, most of the sensor nodes were overheard by both of the spies, which means that we have used two at least partially redundant spy nodes.

For the evaluation the ND algorithm was initiated and every node was switched on after a random delay of at most four minutes. As soon as a node gets switched on, it starts to find neighboring nodes until either a satisfying connectivity is achieved or no new neighbors can be found. The test terminates as soon as no node state changes occur in the whole sensor network for at least three minutes. This is the case after 250 to 300 sent messages.

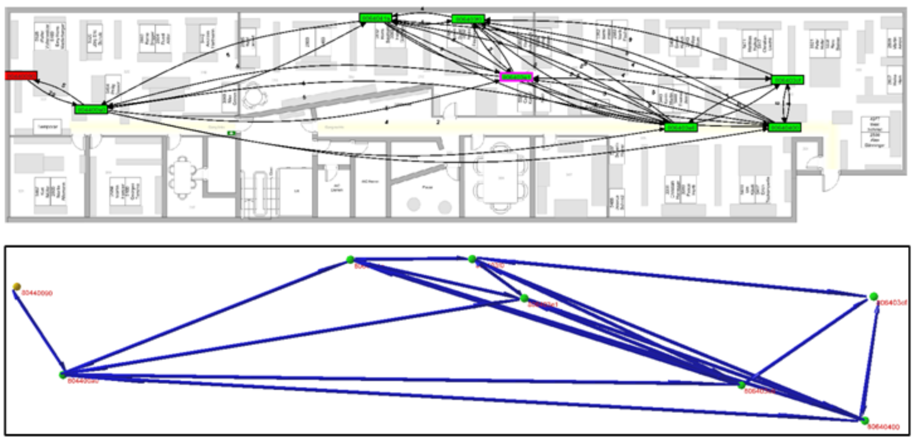


Figure 4-4: **Comparison of the Test Result with Reality**
The upper image shows the reconstructed network evaluated for the two node-distinct paths property by using the WSNSpy tool. The lower image presents the topology of the real network obtained by the DSNAnalyzer.

4.2.3.2 Results

The results presented in this section are derived from nine independent tests. In five tests, a different sink than illustrated in Figure 4-3 is used in order to diversify the network and to get more general results.

For all of the nine tests, the number of neighbors of a node never differs more than by one. The evaluation concerning the number of neighbors yields a percentage of 99.2% of all neighbors that could be found by the system. If a node only receives an initial broadcast message, the sender is registered as a neighbor without an acknowledgment been sent. As this case cannot be detected by the system, some neighbors cannot be determined, which explains the minor difference.

For each test, the reconstructed network is evaluated against the network partition property and the two node-distinct paths property. The evaluation result is verified by considering the topology of the real network. For the nine tests performed, no node state was evaluated falsely. Figure 4-4 above exemplarily shows the comparison between the evaluation of the node states done with the WSNSpy, and below the real network topology. The node to the left is only connected via one single link, which is not sufficient if the condition of two node-distinct paths must hold for the network. Comparing the result obtained by the WSNSpy with the actual topology of the real network, we can verify that the two topologies match and the properties evaluate correctly.

The precise number of sent messages could not be obtained from the test network, as not every single message transmission was logged. However, the capturing performance was investigated in the evaluation of the spy nodes and the DSN, and hence, we rely on these results.

4.2.3.3 Conclusions

From the evaluation of the WSNSpy tool and, as it is based on the data capturing part, from the evaluation of the whole system we can conclude that the system works as intended. The node states evaluated by the WSNSpy do match with reality which leads to the conclusion that the system provides good visibility of the sensor network. This is obtained by only passively capturing and analyzing traffic.

5 Conclusions and Outlook

To finish off, the contributions achieved in this thesis as well as the conclusions that can be drawn out of the evaluation are presented in this chapter. Finally, an outlook on future work is given.

5.1 Summary

In the scope of this thesis we developed a system which allows us to decide in a fast and easy way whether a *Wireless Sensor Network (WSN)* is in good condition. The definition of the term "good condition" is different for every network and therefore must be specified for a certain WSN individually in advance.

The system developed consists of

- a distributed spy for passively capturing WSN traffic, and
- the evaluation tool WSNSpy allowing to analyze a network, based on the data captured by the spy part.

The spy nodes continuously capture WSN traffic and make a log output on every packet received. For practical reasons and in order to provide a sensor network independent framework for collecting the captured messages, the *Deployment Support Network (DSN)* is used for the spy nodes. Overheard messages are stored in a database by the *DSN-Server* providing a basis for further analysis.

In the evaluation tool WSNSpy, the network is reconstructed by simply considering sender and receiver of a captured message transmission. Based on the reconstructed network, the tool provides the functionality to define certain conditional properties which will be checked on fulfillment for every single node. The result is the superposition of all properties evaluated for a node, and is displayed by a red or green icon depending on the property result. By this, the complete network can be checked on user defined conditions, allowing to formulate an individual definition of health for any network.

Additionally, we investigated the realization of a multi-frequency scanner in the scope of a feasibility study, in order to detect channels with ongoing transmissions. We propose to start scanning as a broad-band receiver and to successively narrow the bandwidth on traffic detection until a single channel is identified. However, to be able to detect a transmission on a single channel fast enough, several radio modules running in parallel are needed.

The system developed within this thesis runs completely independent of the sensor network, which allows to spy on the network without any instrumentation of sensor nodes. Hence, the system can be used not only during the development of a WSN, but also for analyzing a fully deployed network.

The evaluation of the system developed was done by considering every single part of the system individually. The following evaluation results could be obtained.

- The spy nodes performed well by at least overhearing 99.5% (98.1%) of all message transmission on average. Redundant spy nodes increase the capturing performance such that hardly any message transmission is missed.
- By using one spy node via the DSN, there is no difference in the percentage of overheard message transmissions compared to a spy node connected directly via a serial link. Hence, the DSN does not reduce the capturing performance as long as it is not overloaded.
- The evaluation of the WSNSpy shows that the topology of the reconstructed network does match with the real network. Basically, all neighbors for each node could be detected (Some neighbors could not be detected by the system due to broadcast messages, which are not considered). Therefore, we can state that the network obtained provides a good representation of reality and can be used for the property evaluation.
- The property for checking the network topology on two node-distinct paths evaluates correctly and the determined node states match with reality.

5.2 Contributions

To finish off, we state the achievements that could be obtained within this thesis.

- With the system developed in this thesis, we achieved good visibility of a WSN without any instrumentation of sensor nodes.
- The evaluation tool is based on low-level information, consisting of sender and receiver of a message transmission only. Such a system is independent of higher-layer protocols and therefore suitable for various applications.
- We can check a sensor network in a fast and easy way on its health state by evaluating properties required for the network.

5.3 Outlook

To reconstruct the network observed, our system simply needs low-level information, i.e. sender and receiver. However, an even more radical approach could be thought of in which no information from the message content is needed at all. By adding to every WSN node a spy, message transmissions could be detected by constantly measuring the signal strength. The evaluation then can be done without the need of any message content information.

The evaluation tool WSNSpy has still room for further development. The following features could be of interest.

- For storing a certain network topology or for saving an image of a specific property evaluation, an export functionality would be useful.
- While continuously importing messages is possible already, the property evaluation can only be done for an unchanging network topology, i.e. continuously evaluating properties is not possible. However, network states can be determined faster if continuous property evaluation would be possible.
- Although the spy nodes run autonomously, the DSNAnalyzer tool is needed for executing commands on the spy nodes, e.g. changing to another channel. Including the functionality for configuring spy nodes directly into the WSNSpy tool would simplify the setup of the spy network.

A

Experimental Equipment

In this part of the appendix we explain the equipment used for the realization of this thesis. On the one hand, we worked with several hardware components either for the development of the system or for its verification and testing. On the other hand, tools used within this thesis are explained.

A.1 A80 Radio Module

The *A80 radio module* shown in Figure A-1, is used for the sensor nodes of the WSN as well as for the spy nodes. The module is based on a MSP430 microprocessor and uses a Chipcon CC1020 UHF transceiver for wireless communication. The MSP430 [13] incorporates a 16-bit RISC CPU and is suitable for demanding mixed-signal applications.

The CC1020 RF chip [2] is a narrow-band low power and low voltage UHF wireless data transceiver in the frequency range from $402\ MHz$ to $470\ MHz$ and from $804\ MHz$ to $940\ MHz$. The CC1020 runs in the *Industrial, Scientific and Medical (ISM)* band using the frequency range from $868\ MHz$ to $870\ MHz$. The chip has a signal sensitivity of $-118\ dBm$, a data rate of $4800\ Kbit/s$ and supports a Frequency Shift Keying (FSK) data modulation.

The A80 contains two orthogonally arranged antennas which allow to switch between them and make use of antenna diversity on receiving.

A.2 BTnode

The *BTnode rev3* [16] is an autonomous wireless communication and computing platform based on an Atmel ATmega 128L microcontroller and a Zeevo ZV4002 Bluetooth radio. The BTnode which is shown in Figure A-2 was developed by the Federal Institute of Technology (ETH) of Zurich.

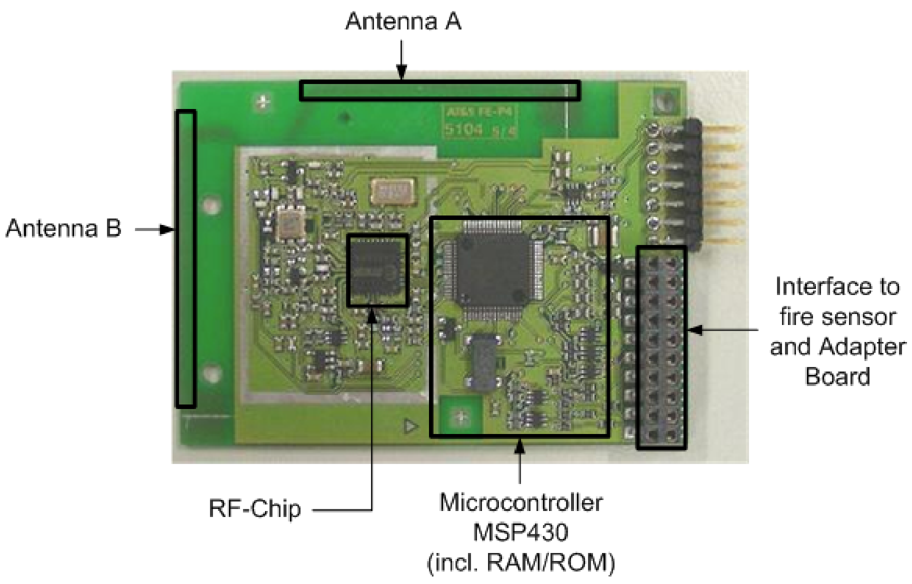

Antenna A
Antenna B
Interface to fire sensor and Adapter Board
RF-Chip
Microcontroller MSP430 (incl. RAM/ROM)

Figure A-1: **A80 Radio Module**

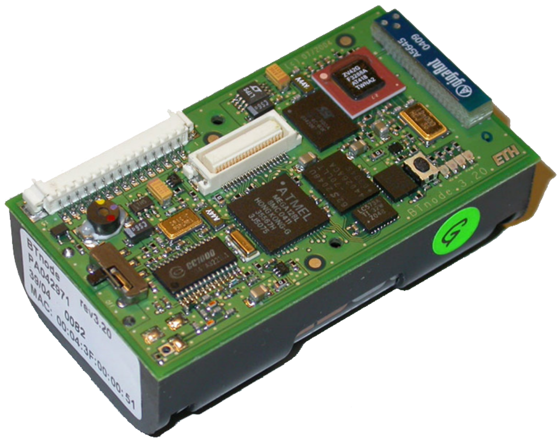


Figure A-2: **BTnode rev3**

For our purposes, the BTnode is used as DSN node in order to forward the logs received from the A80 via the adaptor board using the Bluetooth radio.

A.3 Adaptor Board

The *Adaptor Board* shown in Figure A-3 is a hardware platform acting as a connector between the A80 target and the BTnode. The adaptor board contains a support MSP consisting of a MSP430 providing functionality to measure the current consumed by the A80, and the possibility to spy on the communication between the

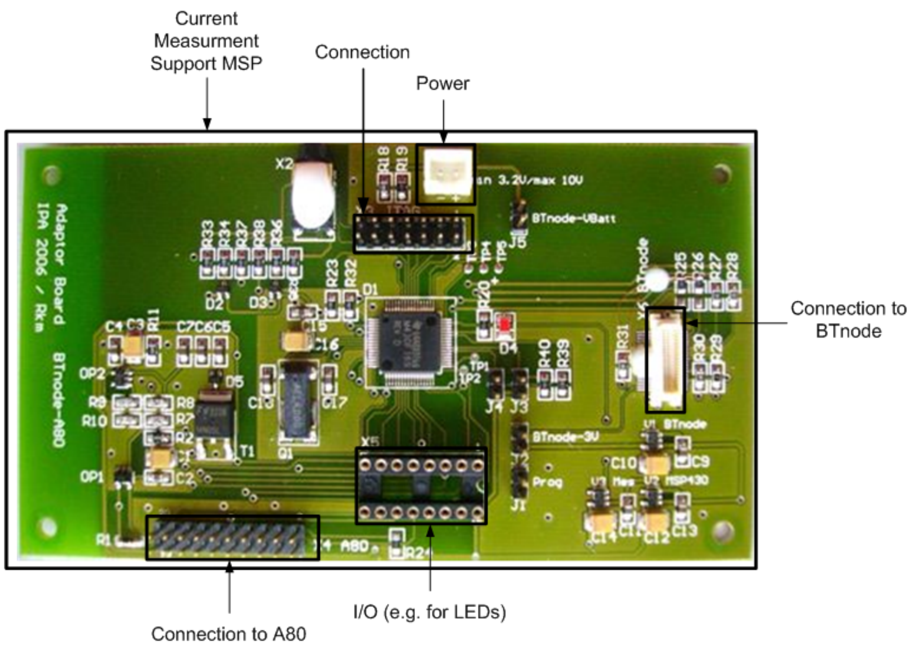


Figure A-3: **Adaptor Board**

BTnode and the A80 radio module. The adaptor board additionally allows direct interaction with the A80 by using a serial interface.

The adaptor board is used for the development of a fire detection sensor network. For practical reasons the board is placed in a so called blue box. A fully applied blue box consists of an adaptor board, A80 module and a BTnode as it is illustrated in Figure A-4.

A.4 Fire Detection Sensor Network

The WSN under development is a fire detection sensor network based on mesh networking functionality. In such a network nodes choose their neighbors if a good connection exists to a node in the vicinity. By this, a stable but possibly over time changing network is obtained, in which a message could be routed via several targets and various paths to its destination. Hence, the network gets more robust against node failures.

A.5 Deployment Support Network (DSN)

Sensor networks are difficult to debug and to configure as the sensors provide hardly any observability due to power and resource constraints. One possibility to improve observability during the development of a WSN is to make use of the *Deployment Support Network (DSN)* as proposed in [3].

The DSN is based on a concept in which a second device is attached to every sensor node building up a second network on top of the sensor network operating

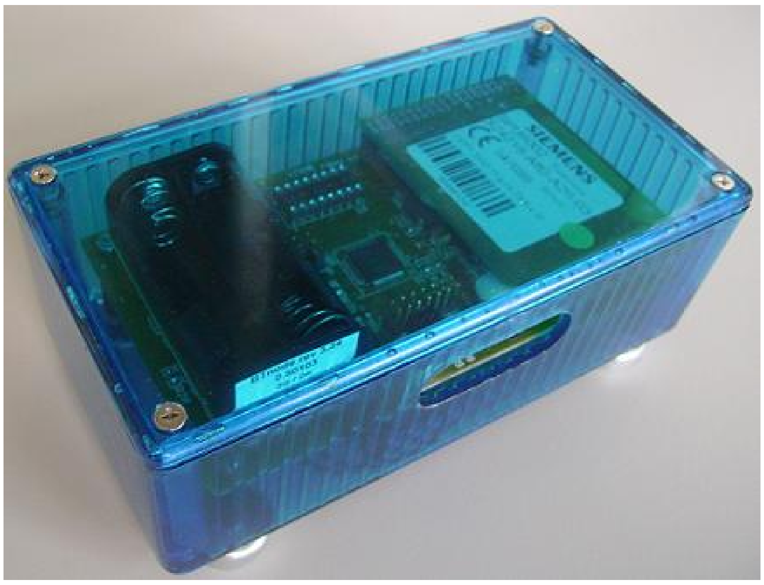


Figure A-4: **BlueBox**

in a different frequency range. This network is used for sending commands to and to collect logging information from the sensor nodes. The DSN used for the WSN development is built out of BTnodes communicating by using a Bluetooth connection.

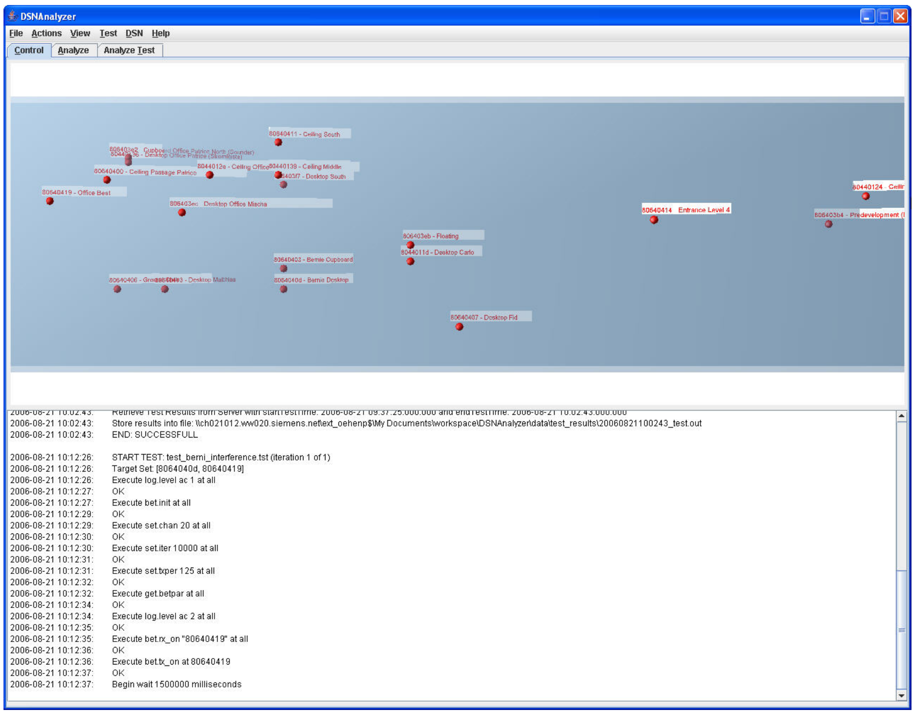
A.6 DSN-Server

The *DSN-Server* [5] provides an interface to the DSN that can be used by many tools in order to access the DSN. On the one hand, the server provides functionality to send commands via the DSN to the targets and on the other hand, logs generated by the spies and collected via the DSN are stores in a database by the server. The database can be accessed by various tools through the DSN-Server in order to get information from the sensor nodes.

A.7 DSNAnalyzer

The *DSNAnalyzer* is a software tool for the analysis and configuration of WSNs using the DSN. The tool was developed within the scope of a master's thesis [8] and the functionality provided consists of the following parts.

- The sensor nodes can be configured through the DSNAnalyzer by providing an interface for executing commands directly on the target. This can be done for every node individually or by commands executed for every node in the network. Commands can also be used to retrieve information about the state of a node.


DSNAnalyzer
File Actions View Test DSN Help
Control Analyze Analyze Test
80640411 - Ceiling South
80640400 - Ceiling Passage Patrice
8064012e - Ceiling Office
80640130 - Ceiling Middle
80640419 - Office Best
806403ec - Desktop Office Mischa
80640414 - Entrance Level 4
806403eb - Floating
8064011d - Desktop Carlo
80640403 - Bernie Cupboard
8064040d - Bernie Desktop
80640407 - Desktop Fid

```
2006-08-21 10:02:43:    Retrieve Test Results from Server with startTestTime: 2006-08-21 09:37:25.000.000 and endTestTime: 2006-08-21 10:02:43.000.000
2006-08-21 10:02:43:    Store results into file: \\ch021012.ww020.siemens.net\ext_oehenp$\My Documents\workspace\DSNAnalyzer\data\test_results\20060821100243_test.out
2006-08-21 10:02:43:    END: SUCCESSFULL

2006-08-21 10:12:26:    START TEST: test_berni_interference.tst (iteration 1 of 1)
2006-08-21 10:12:26:    Target Set: [8064040d, 80640419]
2006-08-21 10:12:26:    Execute log.level ac 1 at all
2006-08-21 10:12:27:    OK
2006-08-21 10:12:27:    Execute bet.init at all
2006-08-21 10:12:29:    OK
2006-08-21 10:12:29:    Execute set.chan 20 at all
2006-08-21 10:12:30:    OK
2006-08-21 10:12:30:    Execute set.iter 10000 at all
2006-08-21 10:12:31:    OK
2006-08-21 10:12:31:    Execute set.txper 125 at all
2006-08-21 10:12:32:    OK
2006-08-21 10:12:32:    Execute get.betpar at all
2006-08-21 10:12:34:    OK
2006-08-21 10:12:34:    Execute log.level ac 2 at all
2006-08-21 10:12:35:    OK
2006-08-21 10:12:35:    Execute bet.rx_on "80640419" at all
2006-08-21 10:12:36:    OK
2006-08-21 10:12:36:    Execute bet.tx_on at 80640419
2006-08-21 10:12:37:    OK
2006-08-21 10:12:37:    Begin wait 1500000 milliseconds
```

Figure A-5: **DSNAnalyzer Screenshot**

- The analyzing part allows the investigation of events collected from the targets. By showing a timeline for each target, the chronological sequence of the events and the relation in between can be examined.
- The third view provides various diagrams for the evaluation of link quality tests.

Figure A-5 shows a screenshot of the DSNAnalyzer tool.

Bibliography

[1] Jan Beutel, Matthias Dyer, and Kevin Martin. Demo abstract: Sensor network maintenance toolkit. In *Proc. 3rd European Workshop on Wireless Sensor Networks (EWSN 2006)*, 2006.

[2] Chipcon AS. *CC1020 Single Chip Very Low Power RF Transceiver Data Sheet*, 2006.

[3] Matthias Dyer, Jan Beutel, Lothar Thiele, Thomas Kalt, Patrice Oehen, Kevin Martin, and Philipp Blum. Deployment support network - a toolkit for the development of wsns. In *Proceedings of the 4th European Conference on Wireless Sensor Networks*, pages 195–211. Springer, Berlin, January 2007.

[4] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *In Proceedings of the 2004 USENIX Technical Conference*, 2004. Also published as CENS Technical Report 0034, December 16, 2003.

[5] Thomas Kalt. Online sensor network analysis tool. Master's thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2006.

[6] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM Press.

[7] K. Martinez, R. Ong, and J. K. Hart. Glacsweb: a sensor network for hostile environments, `http://www.citebase.org/abstract?id=oai:eprints.soton.ac.uk:15604`, 2004.

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

[9] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. Sympathy for the sensor network debugger. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 255–267, New York, NY, USA, 2005. ACM Press.

[10] Matthias Ringwald, Kay Römer, and Andrea Vitaletti. Snif: Sensor network inspection framework. Technical Report 535, Department of Computer Science, ETH Zurich, October 2006.

[11] Kay Römer and Friedemann Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[12] Stanislav Rost and Hari Balakrishnan. Memento: A Health Monitoring System for Wireless Sensor Networks. In *IEEE SECON*, Reston, VA, September 2006.

[13] Texas Instuments. *Microcontroller MSP430 Overview*, `http://focus.ti.com/mcu/docs/mcuprodoverview.tsp?sectionId=95&tabId=140&familyId=342`, 2006.

[14] *WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks*, volume 1, 2004.

[15] Argo - global ocean sensor network, `http://www.argo.uscd.edu`.

[16] Btnode rev3 - product brief, `http://www.btnode.ethz.ch`, April 2005.

[17] Java development environment, `http://www.java.sun.com`.

[18] Eclipse integrated development environment, `http://www.eclipse.org`.

[19] Jgraph library, `http://www.jgraph.com`.

[20] Apache logging service log4j, `http://logging.apache.org/log4j/docs/index.html`.

[21] Bluetooth, `http://www.bluetooth.com`.

[22] Xml-rpc, `http://www.xmlrpc.com`, 2006.

[23] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.