Passive and Active Inspection of Wireless Sensor Networks: A Practical Use Evaluation

Dik-Jan Wisse

Delft University of Technology

Passive and Active Inspection of Wireless Sensor Networks: A Practical Use Evaluation

Master's Thesis in Computer Science

Embedded Software group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Dik-Jan Wisse

May 14th, 2009

Author
Dik-Jan Wisse

Title
Passive and Active Inspection of Wireless Sensor Networks: A Practical Use Evaluation

MSc presentation
May 28th, 2009

Graduation Committee

prof. dr. K.G. Langendoen (chair)	Delft University of Technology
prof. dr. ir. H.J. Sips	Delft University of Technology
dr. ir. R. Hekmat	Delft University of Technology
ir. G.P. Halkes	Delft University of Technology

Abstract

Sensor nodes are small, autonomous, battery-powered devices that use a wireless network interface to communicate with other sensor nodes. Together these sensor nodes form a Wireless Sensor Network (WSN), which has the purpose to sense information about the environment. Before a WSN is deployed, it is tested using simulators and testbeds. Both simulators and testbeds are not able to fully capture all non-deterministic behaviour of the environment in which the WSNs are deployed. Passive inspection is an approach that detects problems within a WSN during deployment. Passive inspection makes use of the broadcast nature of wireless communication and is realised by overhearing messages sent by sensor nodes. The limitation of passive inspection is that operator of the WSN has to wait until a sensor node transmits a message and that this message contains the specific information is needed. To overcome this limitation active inspection can be used, and is realised by injecting messages into the deployment to force a reaction from a sensor node. The concept of passive inspection is not new and has already been evaluated in a simulator and presented in a demo. In this thesis we will further evaluate the practical use of passive inspection using real hardware and we will try to extend passive inspection with active inspection. In order to evaluate passive and active inspection, we designed and implemented an inspection network and an analysis tool. The evaluation compared the normal operation of a deployment with an operation containing a previously resolved bug. The result of this evaluation is that the analysis tool is clearly capable of detecting the previously resolved bug. In this way we showed that passive inspection would have been able to detect that problem when it occurred in a real-life deployment. The evaluation of active inspection showed that active inspection can be used without negatively effecting the behaviour of the sensor nodes in the deployment.

"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence" – prof. dr. Edsger W. Dijkstra

Contents

Chapter 1

Introduction

Sensor nodes are small, autonomous, battery-powered devices that use a wireless network interface to communicate with other sensor nodes. Together these sensor nodes form a Wireless Sensor Network (WSN), which has the purpose to sense information about the environment and send that information to a central point in the network. The central point is called the sink, and functions as the exit and entry point of the WSN. Most WSNs are self-organising and consist of multi-hop paths from source to sink. Self-organisation is important as a WSN must operate autonomously and without interruption for months or even years. Multi-hop paths are created because the radio that is responsible for the wireless communication has a limited transmission range due the inherent power limitations of radio communication [8].

Once created, a WSN-application is tested before being deployed. On most deployment sites limited resources are available. It is therefore important to resolve every possible problem before deployment. There are two possible means of testing a WSN-application before deployment. It can be tested using a simulator or using a testbed. An advantage of a simulator is that it can be run on any arbitrary computer and extra hardware, such as sensor nodes, is not required. The main reason for the use of a testbed is that it is very difficult for simulators to create realistic models of non-deterministic behaviour. The non-deterministic behaviour in a WSN is caused by its distributed nature, the use of interrupt-driven processors, and radio propagation. We mention that processors are interrupt-driven as interrupts have a large impact on the processors limited processing power. Radio propagation has a non-deterministic behaviour as it is influenced by the environment. The environment has so many variables that it is impossible to accurately model it. A testbed uses, therefore, the actual hardware to test the WSN-application. A testbed consists of a small number of sensor nodes that have, besides a radio, a wired network connection. This wired network connection can be used to retrieve state information of each node in order to reveal problems in the WSN-application. Although a testbed is a valuable asset to the debugging process, it still not able to detect and analyse every problem. The reason for this is that a testbed is susceptible to the probe-effect [14], as sending debugging information over the wired connection can consume a significant amount of processing power of a sensor node. Also, although the actual radio chip is used, a testbed is still not capable of fully capturing all non-deterministic behaviour of the environment. This is because it is

usually located in the controlled environment of a laboratory, whilst a deployment is located, for example, outside or in a factory building.

Bugs that appear due to non-deterministic behaviour mostly manifest themselves in race conditions. In order to detect these race conditions and other problems during a deployment, passive inspection and active inspection can be used. Passive inspection is an approach that inspects the communication between sensor nodes without interfering with their communication, and is realised by overhearing messages sent by sensor nodes [17]. Overhearing messages makes it possible to create knowledge as to which state the WSN is in. In order to overhear messages extra sensors nodes that are called inspection nodes, are placed in the deployment. The information collected by the inspection nodes is sent to a central computer, where it is processed by an analysis tool. The inspection nodes form, with the central computer, an inspection network. The limitation of passive inspection is that the operator has to wait until a sensor node transmits a message and that the message contains the specific information the operator wants. To overcome this limitation active inspection can be used to retrieve more information than the information that has already been sent, and to retrieve information on-demand. Active inspection is realised by injecting messages at the inspection nodes into the deployment.

There are two problems that we address in this thesis. The first problem is that passive inspection is evaluated in a simulator and presented in a demo [17], but not yet further evaluated with real hardware. What we want to know is whether or not passive inspection can offer some added value to the process of debugging a WSN. Further, we want to know what is needed to apply this concept to our specific deployment that is described in Chapter 4. Passive inspection has inherent limitations, and therefore the second problem that we address is, whether or not we can reduce these limitations by extending passive inspection with active inspection. Active inspection forces a reaction from the sensor nodes in the deployment, therefore interfering with the normal operation of the deployment. The main challenge to implementing active inspection is to minimise the side-effects of this interference.

The goal of this thesis is to evaluate the practical use of passive and active inspection in an WSN-deployment. The concepts of passive and active inspection are useful if we can use them to detect problems and if they can provide us with insight into how and why problems occur. In order to evaluate the practical use, an inspection network and an analysis tool are created. The inspection network and the analysis tool are evaluated by re-introducing a previously resolved bug to check whether or not the analysis tool can detect the bug, and by other small experiments.

The remainder of this thesis is organised as follows. Chapter 2 briefly discusses related work. After dealing with the concepts of passive and active inspection in Chapter 3, Chapter 4 describes details of the design and implementation of the inspection network. The focus will be on how data collected by the inspection node is transported back to the central computer and how it is possible to communicate with the deployment via an inspection node. Chapter 5 expands on which information can be deduced from the packets received by the inspection nodes. Chapter 6 will explain how the implementation of the inspection network is used to evaluate the practical use of passive and active inspection. Finally, in Chapter 7 we draw conclusions using the results from the practical use evaluation, where after topics for further research will be presented.

Chapter 2

Related Work

The influence of the environment on the wireless communication of the sensor nodes and the distributed nature of sensor networks in general, makes a successful deployment of a sensor network difficult. Therefore a lot of effort is put into testing and debugging a sensor network before and during a deployment. These efforts can be grouped, and are described by the following categories:

Simulators are the first step in testing the operation of sensor networks. Simulators have the advantage that they can run on any arbitrary computer, provide direct access to the internal state of all sensor nodes, and make it possible to test different scenarios in a controlled manner. Simulators model the environment and hardware into discrete events using different levels of abstraction. TOSSIM [11] uses a low level of abstraction to transmit a packet by modelling radio propagation. Other more general-purpose simulators like OMNeT++ [22] use a higher level of abstraction. They model the radio propagation in events like transmitting a packet and receiving a packet. The advantage of using a low level of abstraction is that the environment and hardware are simulated more realistically, but will need a lot more processing power to do so. The biggest limitation of simulators is that it is impossible to fully capture the environment and the non-deterministic behaviour of hardware in models.

Testbeds like MoteLab [23] and PowerBench [7] use a separate wired connection for each sensor node to send state information that can be used to resolve problems in the WSN-application. Another type of testbed is the Deployment Support Network (DSN) [4] that uses wireless connections instead of wired connections. The wireless connections makes it practical to use a DSN outdoors, exposing the testbed to more realistic environmental conditions than is possible inside a building. Although a DSN is a good step forward, the problem of the probe-effect remains. The impact of the probe-effect will be reduced when sensor node hardware becomes more powerful, as a smaller percentage of the processing power has to be spent on the transmission of the debug information.

In-band Debugging makes it possible to debug a deployment by letting sensor nodes send debug information in-band with normal network traffic. Tools that implement this approach are Nucleus [6], Sympathy [16] and Memento [20]. The

sensor nodes in these tools collect debug information and send it periodically or on-demand, back to the sink. The advantage of these tools is that they can be used during the deployment of a sensor network. The major disadvantage of these tools is that a problem that negatively affects the communication between the sensor nodes will also negatively affect the collection of debug information at the sink. Another tool that makes use of in-band network traffic is Clairvoyant [24]. Clairvoyant is a source-level debugger that makes it possible to debug a single sensor node in a deployment. The disadvantage of all in-band debugging tools is that the collection and transmission of debug information consumes resources and influences the normal behaviour of the sensor nodes.

Packet Sniffing makes it possible to detect problems in a deployment by overhearing the messages sent by the sensor nodes. The advantage of packet sniffing is that it does not use any resources of the deployment and it does not interfere with the normal operation of the deployment (eliminating the probe-effect). The passive inspection tool that we propose in this thesis falls in this category, together with Wit [12], Jigsaw [2], LiveNet [1], SNTS [9], and SNIF [18]. Wit and Jigsaw are WLAN packet sniffers that analyse and infer information based on a detailed model of the 802.11 MAC protocol. The main difference between the two is, that Wit merges all traces offline and Jigsaw merges all traces online. The difference between WSNs and WLANs is that most WLANs are single-hop networks and use standardised MAC protocols, whereas most WSNs are multi-hop networks and use different MAC protocols, making WSNs more difficult to sniff.

LiveNet, SNTS, and SNIF are all packet sniffing tools for WSNs. The sniffers of LiveNet store the overheard packets in local flash memory or send them back to a laptop using a wired connection. LiveNet focuses on network dynamics in the form of network connectivity and routing path inference. Like LiveNet, the sniffers of SNTS also store the overheard packets in local flash memory. SNTS distinguishes itself by providing a flexible manner to specify the format of the overheard packets and by applying rule-based machine learning to infer relations between packets. SNIF makes use of a DSN to provide packet sniffing and uses a root-cause decision tree like Sympathy to detect problems. A root-cause decision tree provides an analysis of the problem on a high level, making it useful for the general user. Detailed statistics would provide more insight into the inner-workings of the deployment, but can only be used by domain experts.

The tool we propose is a combination of Jigsaw, LiveNet and SNIF. It will make use of portable wireless packet sniffers like SNIF, online analysis like Jigsaw and SNIF, MAC-layer analysis like Jigsaw, and network dynamics analysis like LiveNet.

Chapter 3

Passive and Active Inspection

Passive inspection is an approach that inspects the communication between sensor nodes without interfering with their communication. Passive inspection is possible because sensor nodes use wireless communication. In order for a sensor node to communicate with another sensor node, it must transmit a message. This message can be received by every node within the transmission range of the sender. Passive inspection is realised by overhearing messages sent by sensor nodes.

Active inspection is used to force a reaction from one or more sensor nodes, and it is realised by injecting messages into the WSN. Usually the sensor nodes react by transmitting a message back, which is then observed using passive inspection. Another example of a forced reaction can be the switching on or off of the LEDs on the sensor node. Active inspection depends on features built into the software that are running on the sensor nodes. Some features are standardised while others are deployment specific.

In order to overhear and inject messages extra nodes, called inspection nodes, are placed within the WSN. Together the inspection nodes form an inspection network. The concept of an inspection network is described in Section 3.1. The inspection network is used to transport the collected messages to a central computer, where after they are processed by an analysis tool. The concept of an analysis tool is described in Section 3.2. The last section, Section 3.3, describes what kind of information can be expected when collecting messages.

3.1 Inspection Network

An inspection network consists of several inspection nodes. Multiple inspection nodes are needed to ensure that transmissions from a reasonable number of sensor nodes are overheard. A typical WSN consists of a large number of sensor nodes, because the transmission range of each sensor node is limited and the field in that they are deployed covers a large area.

The use of multiple inspection nodes has several consequences. Firstly, all messages collected by inspection nodes have to be merged at a central computer. Merging data at a central computer allows for the filtering out of duplicate messages and the identification of dependencies between messages. A duplicate message is created when multiple inspection nodes overhear the same message. A dependency between messages is, for example, a data message that is transmitted by the

sender and an acknowledgement that is transmitted by the receiver. An acknowledgement is transmitted to confirm the reception of a data message. Rather than using a distributed solution, the use of a central computer also makes it easier to not only analyse and display the collected data using graphical notations, but also to coordinate the injection of messages into the WSN. The second consequence for the use of multiple inspection nodes, is that the clock of each inspection node must be synchronised with each clock of every other inspection node in the inspection network. This is necessary for filtering out duplicate messages and identifying dependencies between messages.

There are several consequences for messages being collected by inspection nodes and transported to a central computer. Firstly, the data throughput of the data transport needs to be higher than the data throughput of the WSN. The reason for this is that not only can data messages be duplicated, but when a message is overheard extra information is collected about this message, creating more data. It is important to record, for instance, at which point in time the message was overheard and what the strength of the signal was at which the message was overheard. Secondly, an inspection network must be created, which is responsible for the data transport. The inspection network can be organised as single-hop or multi-hop network, using either wireless or wired connections. When a single-hop network is created, all inspection nodes communicate with the central computer directly forming a star-topology. When a multi-hop network is created, the inspection nodes communicate with the central computer via other inspection nodes or directly, forming a mesh-topology. If wireless connections are chosen for the inspection network, these must not interfere with the wireless connections between sensor nodes. The hardware of the inspection node must, therefore, have two radios: a front-end radio for collecting messages and a back-end radio for transporting data.

Active inspection is realised by injecting messages into the WSN. Messages can be injected by an operator at the central computer or by the inspection node itself. The operator wants a reaction from one or more sensor nodes and therefore creates a message including meta-data. This message is transported to a specific inspection node, from which the message is injected by the inspection node into the sensor network at a certain time. The meta-data added to the message includes the information about which inspection node to use and the time at which the message should be transmitted. The inspection nodes must be able to transmit messages autonomously in the form of acknowledgements. If a sensor node transmits a data message destined for an inspection node, the inspection node must acknowledge that data message. An acknowledgement must be transmitted shortly after the receipt of a data message. Therefore the inspection node must do this autonomously, as it will take too long to transport a data message to the central computer and an acknowledgement back to the inspection node for injection. This implies that the acknowledgement message must be present in the inspection node at the time the inspection node must transmit an acknowledgement. The acknowledgement can be hard-coded into the software running on the inspection node. However, it is more flexible to create the acknowledgement message at the central computer, from which it is transported once to an inspection node and stored in a buffer. In this way the operator at the central computer can change the contents of the acknowledgement message on-demand, leading to more flexibility.

In order for the inspection nodes to inject messages into the WSN, they must participate with the MAC protocol used in the WSN. This means that the point in

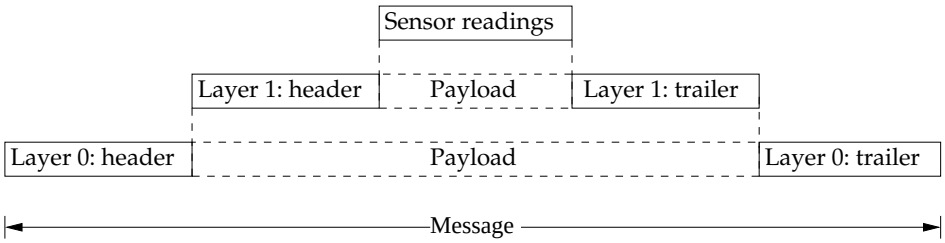


Figure 3.1: Message layout

time, at which the message is injected, is very important. For instance, when a message is injected at a wrong point in time, the message will either not be received by the destination, or it will cause collisions. Collisions disrupt the normal operation of the WSN and result in a crude form of the probe-effect. MAC protocols use carrier sense (CSMA), time division (TDMA) or both, to share the common medium between sensor nodes. CSMA is realised by first listening to the radio, and then if no other node is transmitting, the message is transmitted. TDMA is realised by transmitting in time-slots. Time-slots can be assigned to a sensor node statically or dynamically. When the MAC protocol, used in the WSN, uses a dynamically assigned time-slot algorithm, the inspection nodes must also participate with this algorithm. Using time-slots implies that the sensor nodes must synchronise their clocks with each other. This notion of time is called the "network" time. In order for the inspection nodes to inject messages at the right point in time, the clock of the inspection nodes must be synchronised according to the network time.

3.2 Analysis Tool

The analysis tool operates at the central computer, and is responsible for managing the collected data and coordinating active inspection. The first task of the analysis tool is to store all incoming data to disk. This makes it possible to perform further analysis offline, and replay the events step-by-step in the analysis tool. The second task of the analysis tool is to combine the data traces of all the inspection nodes to form one data trace. Combining the different traces is done based on the time the data was collected and on the contents of the data. The data must be combined online in order to display it online, so that the operator can react immediately. The operator can react, for example, by injecting a message or by moving an inspection node to another location.

3.3 Extracting Information from Messages

The information collected by the inspection nodes are messages transmitted by sensor nodes. These messages contain information that is organised similar to the OSI-model. Each layer is encapsulated by a lower layer as payload (see Figure 3.1). Messages of a typical WSN-application consist of three or four layers, namely:

MAC Layer is responsible for the communication between a sensor node and its neighbouring sensor nodes. The MAC layer uses three types of messages for

the realisation of communication, namely: unicast, acknowledgement and broadcast messages. Each message contains a source and destination address, these can be used to determine which sensor has sent the message and which sensor node should have received the message. When the destination successfully received a unicast message, it transmits an acknowledgement, which can be used by the analysis tool to conclude if the recipient has received the message. However, it is difficult to conclude whether the sender received the acknowledgement or not (Two army problem). We can only partly solve this problem by comparing the data message with the next data message of that specific sender. If that data message contains the same payload, the analysis tool concludes that the second data message is a retry. This partly solves the problem, as the sender will eventually stop sending retries, leaving the analysis tool in the dark as to whether or not the sender has received the acknowledgement. Broadcast messages are not acknowledged making it difficult, or even impossible, to determine which sensor nodes received a broadcast message. The analysis tool has three options to approximate which sensor node received the broadcast message: assume that no sensor node received the broadcast message; assume that all sensor nodes in the neighbour list of the sender have received the broadcast message; or assume all sensor nodes that are covered by the same inspection node as the sender, have received the broadcast message. A neighbour list is a list that contains the link qualities between the sensor node and its neighbours. Whether a neighbour list can be constructed at the analysis tool depends on the information that the network layer exposes. The third option is the least accurate, but is relevant as a neighbour list can not always be constructed at the analysis tool, and not all the sensor nodes that received the message have to be in the neighbour list of the sender due asymmetrical links between the sensor nodes.

Time Synchronisation Layer is an optional sub-layer of the MAC layer. The use of this sub-layer depends on whether a TDMA MAC protocol is used in the WSN, and if its time synchronisation algorithm exposes information in the message sent by the sensor nodes. When this sub-layer is used by the MAC protocol, the information its contains can be used by the analysis tool to check whether or not the clocks of the sensor nodes are properly synchronised with each other.

Network Layer is responsible for the routing of messages towards the sink. The information exposed by this layer are the link qualities between a sensor node and its neighbours, and the distance between the sink and the sensor node. The distance can be expressed in, for example, the number of hops between the sink and the sensor node, or in how many transmissions it will take to transport a message to the sink from the sensor node. This information can be used by the analysis tool to determine whether or not the network-layer is functioning properly and is, therefore, routing the messages as efficiently as possible to the sink.

Application Layer contains, for example, sensor readings. The analysis tool does not use this information, as this information is too deployment specific.

Chapter 4

Inspection Network: Design and Implementation

The starting point for the design and implementation of the inspection network is a deployment, a central computer and the concept of an inspection node. The sensor nodes used in the deployment that we want to overhear are T-Nodes and use the TinyOS operating system. The specifications of the T-Node are listed in Table 4.1a. The central computer is a laptop, at which the collected data is analysed and displayed online. The front-end radio of the inspection node must be the Chipcon CC1000 radio, the same radio that the T-Nodes use.

4.1 Requirements

The requirements for the design and implementation of the inspection network are listed below.

Non-Intrusive means that the inspection network should not change or interfere with the operation of the deployment while debugging the deployment. This requirement excludes, for example, full source level debugging, as breakpoints and stepping through source code interferes with the operation of the deployment. Passive inspection complies with this requirement, as it only uses the messages that are sent by the sensor nodes to debug the deployment. A disadvantage of passive inspection is that the operator has no influence on when and what information is received, and therefore active inspection is used. Active inspection can not fully comply with the non-intrusive requirement as injecting messages is intrusive. To limit the intrusiveness of active inspection no side-effects should occur due to message injection. For example, when the inspection node injects a message destined for a specific sensor node, all other sensor nodes should not be negatively affected by this message injection.

Sufficient coverage of the deployment by the inspection network is needed to distill the dependencies between messages. Only a partial and not a complete coverage of the deployment is needed, as sensor nodes only communicate

Processor

- Type	Atmel ATMega128L
- Clock speed	8 MHz
- RAM	4 KB

Radio

- Type	Chipcon CC1000
- Frequency	868 MHz
- Bitrate	19.2 kbps
- Transmission range	25 - 50 m
- Output data	Stream of bytes 16-bit RSSI value

(a) T-Node Hardware

Software-layers

- Application-layer	LofarAGRO
- Network-layer	LEEP / CTP
- MAC-layer	λMAC Framework / Crankshaft
OS	TinyOS 2.X

(b) Software Stack

Deployment

- Number of nodes	96
- Dimensions	90 m x 50 m

Testbed

- Number of nodes	24
- Dimensions	15 m x 8 m

(c) Network Layout

Table 4.1: Specifications

with their direct neighbours, limiting their energy consumption. A dependency between messages is, for example, a data message and acknowledgement pair. In order to cover any sender-receiver node pair, the minimum transmission range of the back-end radio of the inspection nodes must be at least the maximum transmission range of a sensor node (50 meters). Also, to reach a reasonable area of the deployment from the central computer, the transmission range of the back-end radio of the inspection nodes should be at least 50 meters.

Flexibility and portability are important requirements as the inspection should be easy to place, move and manage. The implications of these requirements are that the inspection network should not consist of too many inspection nodes, and that the inspection nodes should use wireless communication and be battery powered. Being battery powered is not a problem, as the inspection network only has to operate continuously for up to a few days.

Reliable transport of data between inspection nodes and the central computer is needed to ensure that all data collected by the inspection nodes can be used for analysis. Especially the analysis of the dependencies between messages suffers when there is message loss.

Time synchronisation is needed between inspection nodes to enable the combining of messages at the laptop, and to enable message injection between inspection nodes and sensor nodes, depending on the MAC protocol used on the sensor nodes. Time synchronisation between inspection nodes and sensor nodes, must be done passively. In other words, only the inspection nodes should be aware of the time synchronisation and not the sensor nodes, in compliance with the non-intrusive requirement.

A higher data rate of the back-end radio than the front-end radio is needed, as multiple inspection nodes can overhear the same message and all overheard messages are sent to one central point. How much higher the data rate of the back-end radio must be than that of the front-end radio, depends on the rate of messages received by the inspection nodes and on the number of inspection nodes used in the inspection network. The rate of messages received by the inspection node depends on the rate of messages transmitted by the sensor nodes and the number of sensor nodes that an inspection node can overhear.

Specifications of the message layers are needed to be able to distill information from the messages that are sent by the sensor nodes. The specification can be a document that describes the different layers or the source-code. Source-code is preferred, as it contains much more information than a document would.

4.2 Inspection Node

The concept of an inspection node can be realised by different combinations of hardware. We selected three candidates for the choice of inspection node, namely: a BTnode, a Tmote Sky + T-Node, and a PDA + T-Node (see Figure 4.1). Both Tmote Sky and PDA do not have a CC1000 radio in contrast to a BTnode and a

(a) BTnode (b) Tmote Sky

Inspection Node	Back-end radio	Frequency	Data Rate	Outdoor Range
BTnode	Zeevo ZV4002 Bluetooth	2.4 GHz	723.2 kbps	50m (100m)
Tmote Sky + T-Node	Chipcon CC2420	2.4 GHz	250 kbps	125 m
PDA + T-Node	WiFi IEEE 802.11b	2.4 GHz	11 Mbps	100 m

(c) Specifications

Figure 4.1: Candidate Inspection Nodes

T-Node. Therefore both Tmote Sky and PDA will need a T-Node to supply the required CC1000 radio. The requirements that all three candidates meet more or less equally are listed below:

- The wireless connections of the inspection network must not interfere with the connections between the sensor nodes. This requirement is met by using different frequency bands; the CC1000 radio operates at a frequency band of 868 MHz while the back-end radios of all three candidates operate at a frequency band of 2.4 GHz.
- The requirements for the bare minimum and the preferred minimum transmission ranges are met by all three back-end radios. The Zeevo ZV4002 Bluetooth radio is a class 1 radio, implying that the Bluetooth radio could achieve a transmission range of 100 meters. However, without an extra power supply connected to the BTnode, the Bluetooth radio has only power to achieve a transmission range of 50 meters.
- The data rate of the back-end radio rate must be much higher than the data rate of the front-end radio, and is met by all three back-end radios. The data rate of the back-end radios are 250 kbps or higher, while the front-end radio has a data rate of 19.2 kbps.

The PDA is the weakest of the three candidates. Its battery life-time is considerably less than that of the other candidates. Also, the PDA is more expensive, and is therefore too valuable to use in an outdoor environment (moisture). The BTnode has two advantages over the Tmote Sky. There is no need for a custom connection between the BTnode and a T-Node, unlike the connection between the Tmote Sky and a T-Node, as the BTnode has two integrated radios. The other advantage is that Bluetooth radios create more reliable links than CC2420 radios, by

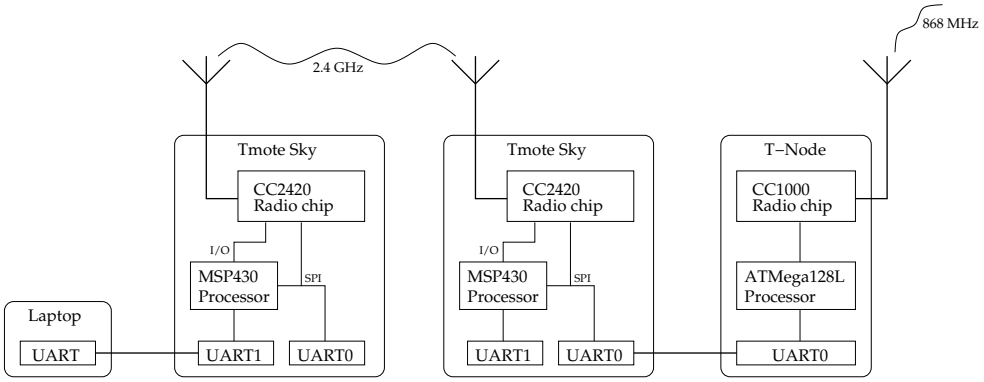


Figure 4.3: Structural Layout Inspection Network

means of frequency hopping techniques. When comparing the available software, the Tmote Sky has a slight advantage over the BTnode. This is because the T-Node connected to the Tmote Sky can use the same TinyOS CC1000 software components used by the T-Nodes in the testbeds and deployments. The BTnode has a different operating system (BTnut) with its own software to control the CC1000 radio. The software of the BTnode must be able to handle TinyOS messages, which could mean that more time is needed during the development phase. A more general advantage to use TinyOS instead of BTnut is that the TinyOS community is much larger than that of BTnut, meaning that there are more resources available in the form of software and support.

In summary, the best candidates for the realisation of an inspection node are the Tmote Sky/T-Node combination and the BTnode. For reasons of convenience, we have chosen the Tmote Sky as it was already available, and the connection between the Tmote Sky and the T-Node was easy to fabricate. Figure 4.2 and 4.3 give an idea as to how a Tmote Sky is connected to a T-Node. Messages are collected by the CC1000 radio of the T-Node. This data is sent over the serial connection (UART0-to-UART0) to the Tmote Sky. The Tmote Sky transmits this data to the Tmote Sky connected to the laptop. The Tmote Sky connected to the laptop, sends the data over the USB-port to the laptop where it is analysed.

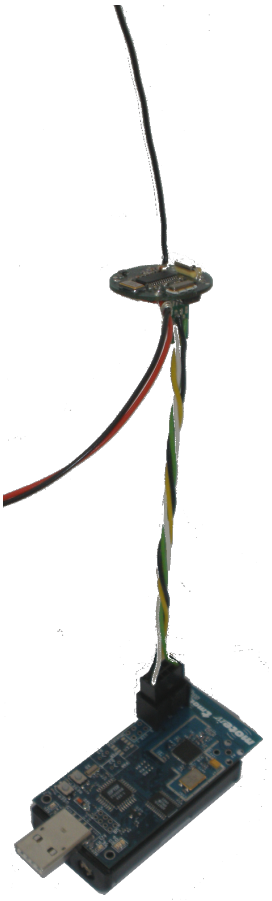


Figure 4.2: Tmote Sky connected to a T-Node

4.3 Packet Detection

The CC1000 radio of the T-Node part of the inspection node samples the ether and converts the radio signal into a continuous stream of bytes. This stream of bytes contains not only noise but also messages in the form of radio packets. These radio packets can be detected and filtered from the stream. It is up to the designer to decide where and how the detection and filtering is done. The location of the detection and the detection itself is described in the following subsections.

4.3.1 Location of the Detection

The detection can be done at one of the four locations of the structural layout that is shown in Figure 4.3. Detecting packets at either one of the two Tmote Skys has

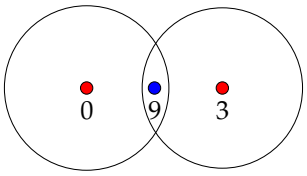


Figure 4.4: Hidden node problem: node 0 and node 3 cannot hear each other and both start sending, resulting in a packets collision at node 9

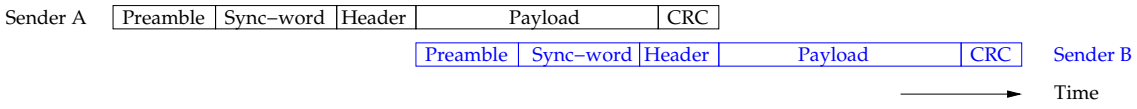
no added value. A Tmote Sky has nearly the same processing power as a T-Node. A better choice for location would be at the T-Node or at the laptop where there is lots of processing power. Detecting packets at the T-Node results in less information being transmitted to the laptop, reducing the load on the inspection network and making it possible to use more inspection nodes. Detecting packets at the laptop implies that all the raw data is collected at the laptop. This makes it possible to use different, stronger, detection algorithms, there less chance of bugs occurring in the inspection nodes, and there is a constant data load on the inspection network, provided that the number of inspection nodes does not change. Debugging software on a laptop is easier than debugging on a node. Also, it is more difficult to locate a bug in the detection algorithm if one part of the detection is done at the T-Node and the other part at the laptop. A constant load is a good property, as the worst-case is the same as the best-case. With a variable load it is more difficult to find the worst-case load while testing. Testing with the worst-case load makes it more likely that the inspection network will stay functioning during operation. Nevertheless, the best location to detect packets is the T-Node, as more inspection nodes can be used, and the collected data can be timestamped straight after collection, which results in a timestamp that is as accurate as possible.

4.3.2 Extended Packet and Collision Detection

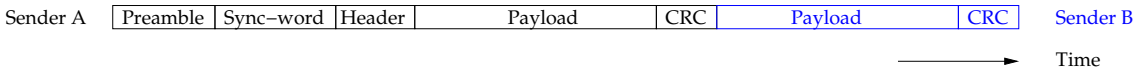
When a radio packet is transmitted, it is preceded by a preamble and a synchronisation sequence. The preamble is used by the receiving radio hardware to tune to the right frequency and phase of the signal. In the case of the CC1000 radio, the preamble is also used in software to detect contention and packets. The synchronisation sequence used by the CC1000 radio is a 2 byte sequence (sync-word). While the radio of the receiver is tuning to the signal of the sender, the received preamble bytes can contain bit-errors. When the length of the preamble is too short to contain a bit-error free preamble byte the packet will be lost. The solution to this problem is to extend the packet detection by making the detection algorithm tolerant of a bit-error occurring in each preamble byte and in the sync-word.

The packet detection algorithm can also be extended by making use of the capture effect. The capture effect makes it possible, in certain cases, to detected packets that were otherwise lost, and to detect the occurrence of collisions. Lee et al. [10] describe the capture effect as the ability of some radios to correctly receive one of several concurrently transmitted messages, even if the received strengths of the messages are almost the same.

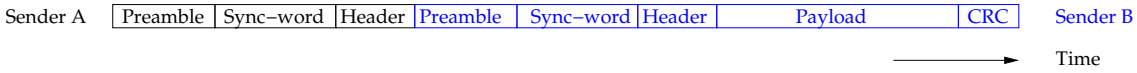
Collisions occur when two sensor nodes transmit a message concurrently. There are two main reasons for collisions: the use of CSMA-based MAC protocols, and the hidden node problem. When using a CSMA-based MAC protocol a collision



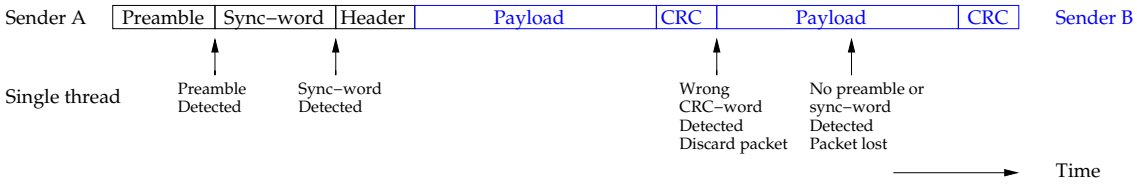
(a) Scenario of two colliding packets



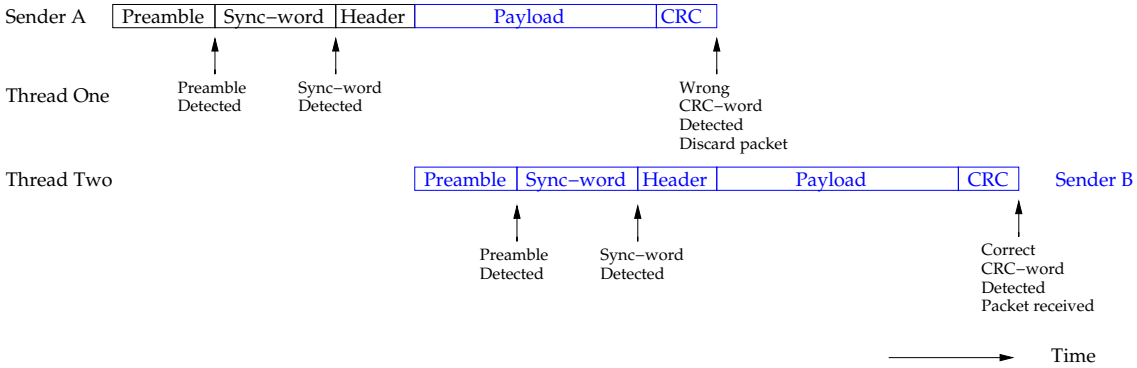
(b) Signal first sender (Sender A) is stronger than signal second sender (Sender B)



(c) Signal second sender (Sender B) is stronger than signal first sender (Sender A)



(d) Standard packet detection where the second packet is lost



(e) Dual thread packet detection where the second packet is correctly received

Figure 4.5: Packet transmission

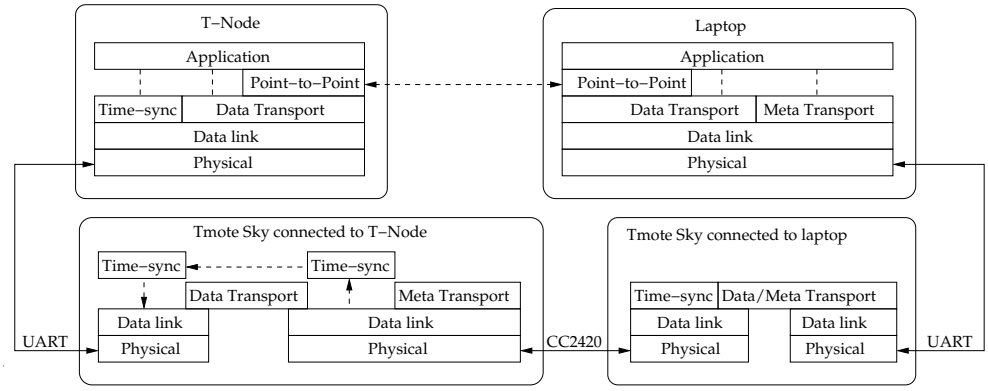


Figure 4.6: Network Architecture Layout

can occur when two sensor nodes sense the absence of a signal and start transmitting concurrently, resulting in a collision. The hidden-node problem occurs when two sensor nodes are not in each others transmission range but there is a destination sensor node that is located in both their transmission ranges. When both sensor nodes start transmitting, a collision can occur (see Figure 4.4). Collision detection is limited, as we can only detect a collision when the start of a second packet is detected during the reception of the first packet. In other words, to be able to detect a collision the sync-word of the second packet must be detected precisely between the sync-word of the first packet and the end of the first packet. In order to detect the sync-word of the second packet, the second packet must have a stronger signal than the first packet.

As mentioned before, the capture effect makes it possible to detect one of several colliding packets and is realised as follows. If the first packet sent has the strongest signal, we will never know that a second packet was sent (Figure 4.5b). By contrast, when the second packet sent has a stronger signal, the second packet can be detected successfully (Figure 4.5c). However, when the standard packet detection is collecting the payload of the first packet while a second packet with a stronger signal is transmitted, the second packet is missed as the packet detection is still busy with the first corrupt packet. This scenario is displayed in Figure 4.5d. To solve this problem, a second detection thread is executed to try to detect the second packet while the first detection thread tries to collect the first packet, as shown in Figure 4.5e.

4.4 Network Architecture

The network architecture (see Figure 4.6) describes the different layers needed to transport the data collected by the T-Node to the laptop, and to transport the messages that need to be injected into the WSN from the laptop to the T-Node. In this section we will give a short description of the different layers, whereafter a more detailed description of the layers is given that are specific for this implementation.

4.4.1 Network Layers

The network architecture consists of six software layers. A short description is given of each software layer.

Application Layer is responsible for timestamping and assigning a type to the collected data. Different types of data are needed as not only received messages are collected, but also, for example, when the channel was busy while trying to inject a message into the WSN.

Point-to-Point Layer is responsible for using the transport layer as efficiently as possible and to avoid a dependency of the application layer on a specific transport layer. Efficiency is achieved by putting the application layer data into a frame and packing as many frames or parts of frames into a transport packet creating a high data throughput. Framing makes it possible to reconstruct the application layer data in its original form at the laptop. A frame consists of a delimiter and a length field. The delimiter is used to detect the start of a frame and the length to detect whether or not the frame is complete. Making the application layer independent of the transport layer has the advantage that the transport layer can be changed or tuned without having to change the application layer. Changing the application layer implies that many parts of the analysis tool have to be changed too. The point-to-point layer is only used for transporting data from the T-Node to the laptop and not the other way around. The reason for this is that de-framing data at the T-Node costs too many resources and not much data has to be transported to the T-Node so a high data throughput is not necessary.

Time Synchronisation Layer is responsible for synchronising the clocks of the Tmote Sky connected to the T-Node with the clock of the Tmote Sky connected to the laptop and for synchronising the clock of the T-Node with the clock of the connected Tmote Sky.

Data Transport Layer is responsible for identifying the source inspection node and the destination inspection node, and to detect packet loss and duplicate packets.

Meta Transport Layer is responsible for the transport of commands and meta data between the laptop and the Tmote Sky connected to the T-Node. A command can be, for example, to blink the LEDs of the Tmote Sky connected to the T-Node. Meta data can be, for example, statistics about the connection between the Tmote Sky and the T-Node.

Data Link Layer is responsible for creating reliable connections between nodes, and between node and laptop. Reliability is achieved by detecting corrupt packets and by retransmitting unacknowledged packets.

4.4.2 Implementation Details of the Connection between T-Node and Tmote Sky

The connection between the T-Node and the Tmote Sky is a custom connection that also needs custom software. The basis for this connection is the TinyOS UART component that is available for both nodes. This UART component is designed for

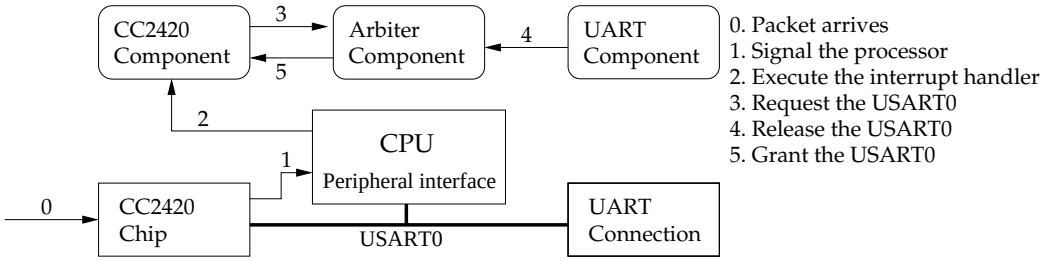


Figure 4.7: Sharing the Peripheral Interface

a connection between a personal computer (pc) and a sensor node. This connection is asymmetrical; from pc to sensor node a stop-and-wait protocol is used, and from sensor node to pc a non-reliable protocol. The developers of TinyOS wanted to use a non-reliable protocol for both ways, as a non-reliable protocol is less complex and smaller in size than a stop-and-wait protocol. However, the connection between the pc and the sensor node is too unreliable to use a non-reliable protocol. The reason for this is that the size of the receiving buffer at the sensor node is too small to cope with a high interrupt load, resulting in bytes in the buffer being overwritten. The pc does not have this problem as more memory is available for the receiving buffer. As the UART connection is asymmetrical, when a T-Node is connected to a Tmote Sky, both nodes expect a pc on the other side of the connection and will receive an acknowledgement for each message that they send, while not expecting any. The UART connection had to be modified to use a two-way stop-and-wait protocol, as both nodes have limited size buffers and reliability is an important requirement.

Due to hardware limitations of the Tmote Sky, the UART connection with the T-Node has to share a peripheral interface of the processor with the radio chip. The radio chip uses this peripheral interface for the transport of radio packets between the processor and the radio chip. The radio chip is also connected to the processor with dedicated control lines. With these control lines the processor coordinates with the radio chip when the peripheral interface is needed. In software, coordination of sharing a resource, in this case the peripheral interface, is done through an arbiter component. To give an idea how the sharing of the peripheral interface works, a scenario is sketched in Figure 4.7, where the UART component is owner of the peripheral interface and the radio receives a radio packet. In this scenario the radio chip signals the processor that it needs the resource.

When the T-Node wants to send data to the Tmote Sky it can not signal the processor of the Tmote Sky for the resource, as there are no dedicated control lines between the two nodes. To solve this problem flow control is incorporated in the UART component at the HDLC-like framing [21] layer. Framing is used to create a reliable connection over the UART's byte-oriented connection. The HDLC-like framing protocol reserves special bytes for the use of flow control commands. Two flow control commands are used, Xstart and Xfinished. They are escaped the same way as a frame-delimiter. The use of flow control is illustrated in Figure 4.8, where the UART component periodically requests the use of the peripheral interface. When granted the UART-component starts the transfer with the T-Node of one UART unit each. The UART unit can either be a data frame or an acknowledgement frame. This implementation of flow control differs from the standard

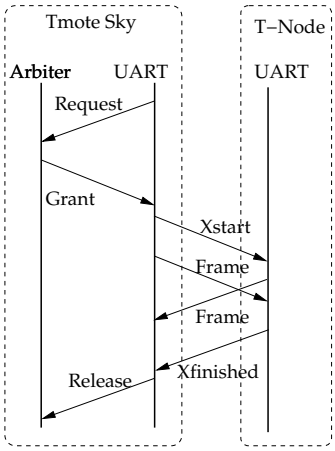


Figure 4.8: Flow Control

implementation, as time-slicing is used and the Xfinished signal transmits in the opposite direction.

Time-slicing ensures that the peripheral interface is shared in the most optimal way possible to support both the needs of the radio chip and the T-Node. The needs of the radio chips differs from that of the T-Node. When packets arrive, the radio chip needs the interface as quickly as possible, but only for a short amount of time. As the T-Node has more memory than the radio chip, it can wait longer for the interface to become available. However, the T-Node needs the interface for a longer period of time, as the UART connection is much slower than the connection between the radio chip and the peripheral interface.

The Xfinished signal transmits in the opposite direction to ensure that all data sent by the T-Node is received. When the interface is granted to the radio chip, all data sent by the T-Node is lost and the transmission of data by the T-Node will probably interfere with the transmission between the radio chip and the processor, creating corrupt packets.

Complete frames are chosen as UART unit to assist the stop-and-wait protocol. When the stop-and-wait protocol receives a corrupt acknowledgement frame or does not receive an acknowledgement frame at all in the time slice after the transmission of a data frame, it retransmits the data frame.

4.4.3 Implementation Details of the Time Synchronisation

The clocks of the inspection nodes need to be synchronised with each other. Time synchronisation can be realised by either using a centralised or decentralised approach. In this implementation we have chosen for a centralised approach, as we already have a central point and a centralised approach is less complex than a decentralised approach. The clock of the Tmote Sky connected to the laptop is used as the central clock instead of the clock of the laptop. This removes the need for an additional synchronisation stage, which reduces the inaccuracy of the synchronisation.

The time synchronisation consists of two stages. In the first stage the current time is broadcasted by the Tmote Sky connected to the laptop and received by the

Tmote Skys connected to T-Nodes. This stage is realised by an implementation of "Packet-level time synchronization" [13] for CC2420 radios. In order to make this implementation work we had to resolve two problems both related to the "start of frame delimiter" (SFD) pin of the CC2420 radio [3]. The SFD pin is connected to an interrupt pin of the processor. The moment the CC2420 is going to transmit a packet the SFD pin becomes active generating an interrupt. In the interrupt handler of the SFD pin a timestamp is taken and transferred to the transmit buffer in the CC2420, to update the timestamp in the packet. For some reason, it takes too long between the activation of the SFD pin and the transfer of the updated timestamp, causing the packet to be transmitted with an old timestamp. We resolved this problem by first putting a marker in the packet to indicate an "invalid" timestamp and overwriting the "invalid" timestamp in the interrupt handler. When the packet is transmitted before the timestamp is updated the receiver will receive the marker "invalid" and discards the timestamp. The rate at which the timestamp packets contain an "invalid" marker depends on the rate of messages received at the Tmote Sky connected to the laptop. At a rate of one message per second, 0.3% of the time synchronisation packets contain an "invalid" marker, and at a rate of 20 messages per second, 3%.

The second problem occurred while a packet is being received and hardware address recognition is enabled. In receive mode the SFD pin becomes active when the start of a packet is detected and a timestamp is taken in the interrupt handler. When the hardware address recognition discards the packet, the SFD pin becomes inactive straight after the receipt of the destination address. The interrupt handler is fired again and the timestamp is discarded. When the SFD pin becomes active again before the interrupt handler has been able to discard the timestamp, the timestamp queue will still contain an incorrect timestamp. We resolved this problem by increasing the packet size so that only one packet can fit in the receive buffer, and reduce the size of the timestamp queue to one item. When the receive buffer can contain only one packet, only one timestamp needs to be saved and an old timestamp can be safely overwritten.

The second synchronisation stage is between the Tmote Sky and the T-Node. We implemented this stage using the same principles as the "Packet-level time synchronization" [13].

Chapter 5

Analysis Tool: Analysis of the received information

The previous chapter focused on how the collected data is transported back to the central computer while in this chapter, Section 5.1, will concentrate on how the transported data is used for analysis. The second section, Section 5.2 describes how active inspection is coordinated from the analysis tool and how the messages are injected at the inspection nodes.

5.1 Passive Inspection

Subsection 5.1.1 describes how the radio packets of multiple inspection nodes are combined into one stream. The following subsections describe what kind of data can be extracted from the different layers that form a radio packet.

5.1.1 Combining Radio Packets of Multiple Inspection Nodes

The stream of radio packets supplied by each inspection node need to be combined online into one stream (see Section 3.1). This combination process is realised using the following steps:

1. Radio packets of each inspection node will arrive at the central computer in FIFO-order, and are added to a queue per inspection node. The timestamp of a new packet is checked against the timestamp of the oldest packet in the queue. When the timestamp of the new packet is older than the timestamp of the oldest packet, a clock skew is detected. The timestamp of the new packet is not compared to the timestamp of a previously arrived packet, as a small time synchronisation inaccuracy would be identified as a significant clock skew.

2. The packet with the oldest timestamp of all the packets at the head of all the queues is selected. The timestamp of this packet must be older than the timestamp of the latest received data from each inspection node. This ensures that no packets will arrive that are older than the selected packet.

3. Packets at the head of each queue with the same contents as the selected packet are removed. This filters out duplicate packets.

4. The selected packet is dispatched, so that it can be analysed further.

5.1.2 Statistics of the MAC-Layer

The packets that are dispatched by the combination process are used by the MAC-layer statistics process. When a packet arrives, it is sorted into the correct node queue by identifying the source address of a data packet and the destination address of an ack packet. A node queue is a queue of packets associated with a particular sensor node in the deployment. When a node queue contains two data packets, dependencies are deduced based on the payload and the number of acknowledgements. The payload consists of data from the network and application layer (see Section 3.3). In order to deduce dependencies between two data packets the payload of the two data packets are compared with each other:

- **Equal payloads**: the second data packet is marked as a retry. If the queue contains an ack between the two data packets, the source node did not receive the ack, otherwise the destination node did not receive the first data packet. As links between sensor nodes are asymmetrical, it is important to make a distinction between the two directions.

- **Unequal payloads**: if the there is an ack packet between the two data packets, the first data packets is marked as acknowledged. If there was no ack packet between the two data packets, either the inspection nodes have missed the ack or the sensor node has given up retrying as the maximum number of retries had been reached.

Depending on the number of radio packets missed by the inspection nodes, uncertainties appear in the statistics. It is obvious that when radio packets are missed, the statistics are incomplete. A bigger problem is, for instance, when a data packet is marked as acknowledged whereas it has actually been dropped, or when the recipient is blamed for not receiving the data packet when actually the sender did not receive the acknowledgement.

In order to reduce these uncertainties the software running on the sensor nodes can be modified to include sequence numbers in the radio packets that they transmit. There are two sequence numbers used to reduce the uncertainty to a minimum:

- **Transmit index:** an index that increases with every transmitted packet. (tidx)

- **Payload index:** an index that increases every time the payload of the packet changes (pidx)

Δtidx is the difference between the transmit index values of the two data packets, and Δpidx is the difference between the payload index values of the two data packets. Δtidx is used to determine the number of missed packets. Δpidx is used to determine the number of missed packets with a unique payload. When the payload of the two data packets are equal, all missed packets are marked as retries. When the payload of the two data packets are not equal, we assume that all missed packets with a unique payload are acknowledged, and that all missed packets represented by Δtidx - Δpidx are retries.

5.1.3 Network Time

The packets that are dispatched by the combination process are used by the network time statistics process. This process supplies statistics about the time synchronisation protocol that is used to synchronise the clocks between the sensor nodes. The synchronisation protocol used by the sensor nodes is part of the λMAC framework [15], and works as follows. Each radio packet contains a network-timestamp, that represents the network time of the sensor node at the moment the packet was sent. When a sensor node receives a packet with a network time further in the future than its own network time, the sensor node will adjust its network time accordingly.

In order to check whether each sensor node has the same network time, the network time timestamp in the packet is compared to the timestamp of receipt by the inspection node. The difference between these two timestamps, hereafter called delta, should be more or less constant for all sensor nodes at a specific point in time, during the life-time of the deployment. The delta can be used to check whether or not two sensor nodes can communicate with each other and whether time-partitions have formed in the deployment. When two nodes have different deltas it is very likely that the destination node's radio is turned off while the source is transmitting. Time-partitions can be detected by comparing the delta of different sensor nodes. Sensor nodes with the same delta form a group. If there are multiple groups, time-partitions have been formed in the deployment.

5.1.4 Statistics of the Network-Layer

The packets that are dispatched by the combination process are used by the network-layer statistics process. The network protocol used in the deployment is the Collection Tree Protocol (CTP) [5]. CTP is a proactive routing protocol that creates a spanning tree by letting each node choose one parent to send its data packets to. This tree can be re-constructed and visualised using the parent field of a CTP routing packet and the destination field of a CTP data packet. The re-construction of the routing tree is used to detect routing loops, to detect partitions within the network, to check the routing tree against the link qualities broadcasted by the sensor nodes, and to track CTP data packets that are routed through the network. The visualisation of the routing tree can be used to aid in the detection of problems effecting the routing tree. A CTP routing packet contains a sequence number that is used by the link estimation protocol to count the number of missing CTP routing packets. The analysis tool uses this sequence number to detect sensor nodes that have been rebooted.

5.2 Active Inspection

Injecting messages into a deployment is coordinated using the analysis tool at the central computer. In order to do this successfully, inspection nodes must co-operate with the MAC protocol used in the deployment (see Section 3.1). Both CSMA-based and TDMA-based MAC protocols, with the additional use of the time-synchronisation layer (see Section 3.3), are supported. However, the current implementation for the support of CSMA-based MAC protocol does not include advanced features like channel reservation techniques (RTS/CTS) and collision

#	Action	TDMA-based MAC protocol	CSMA-based MAC protocol
1.	Move the inspection node timestamp to the present if it was in the past.	Increase the inspection node timestamp and network timestamp with steps the size of the frame length.	Move the inspection node timestamp and network timestamp to the present by adding the difference between the present and the past to both timestamps.
2.	Inject the message.	Activate a timer at which the packet must be injected. When the inspection node is receiving a message, the network timestamp is adjusted to the next time-frame and the timer is reactivated to try and inject the message in the next time-frame.	Check whether or not a message is being received. If so, the inspection node will adjust the network timestamp and inject the packet once the message has been received, otherwise the packet is injected directly.

Table 5.1: The process of injecting a message into the network at the T-Node part of the inspection node

avoidance techniques (random back-off timer). The following parameters are used to inject packets at the correct moment in time:

- **Inspection node timestamp**: a timestamp according to the clock of the inspection node at which the packet should be injected into the network.
- **Frame length**: the length in milliseconds of the time-frame used by the TDMA-based MAC protocols. A frame length of zero indicates the use of carrier sense with a CSMA-based MAC protocol.
- **Network time offset**: the offset within the packet where the network time is located. A network time offset of zero indicates the absence of the time-synchronisation layer in the packet.

When the operator wants to inject a packet, the message is set up at the analysis tool and the above parameters are added to the packet. The packet is transported to the selected inspection node and is processed as described in Table 5.1. The first action listed in Table 5.1 is to move the inspection node timestamp to the present if it was in the past. An inspection node timestamp can be in the past due to a delay in the transport of the packet from the analysis tool to the T-Node part of the inspection node, and for convenience. It is convenient to re-use a packet received from the destination. This enables the re-use of the time-delta between the inspection node timestamp and the network timestamp. Re-using the time-delta ensures that the clock of the inspection node is synchronised to the clock of the destination, ensuring that the packet is injected in the correct time-slot of the destination.

When injecting acknowledgements into the deployment, special functionalities are needed. This is because acknowledgements must be injected immediately after the receipt of a data packet. It would take too long to transport the data message

all the way to the central computer where after the acknowledgement would have to be transported back to the inspection node. The acknowledgement packet is set up at the central computer and transported once to inspection node where it is stored in its memory to be used when needed. The acknowledgement packets, that are transported, contain the following parameters:

- **Source address**: is the address that the inspection node will assume. For example, when the inspection node assumes the address 40, the inspection node will inject an acknowledgement with source address 40 when it has received a data packet with destination address 40. If the source address is the same as the broadcast address, the injection of acknowledgements is stopped.
- **Network time offset**: the offset within the packet where the network time is located. A network time offset of zero indicates the absence of the time-synchronisation layer.

After the receipt of a data packet, the destination address is compared to the source address of the acknowledgement that is stored in memory. When they are equal, the network time of the acknowledgement is updated and injected into the deployment.

Chapter 6

Evaluation

In this chapter we evaluate the passive and active inspection capabilities of the implemented analysis tool and the inspection network. First, in Section 6.1 we describe how the experiments, using the testbed, are set up. In Section 6.2, we evaluate the accuracy of the time synchronisation between the inspection nodes. Section 6.3 and 6.4 describe the passive and active inspection experiments that were performed, and the results of these experiments are evaluated. The last section, Section 6.5, describes what influence the temperature of the environment has on the clocks of the sensor nodes.

6.1 Setup of the Experiments

The experiments were performed using a testbed that consists of 24 T-Nodes. The layout of the testbed is shown in Figure 6.1. The blue dots represent the locations of the inspection nodes, and the red dots that of the sensor nodes. The T-Nodes have a transmission range of 25 meters and the dimension of the testbed is approximately 8 x 15 meters. This means that all nodes lie within in each other's trans-

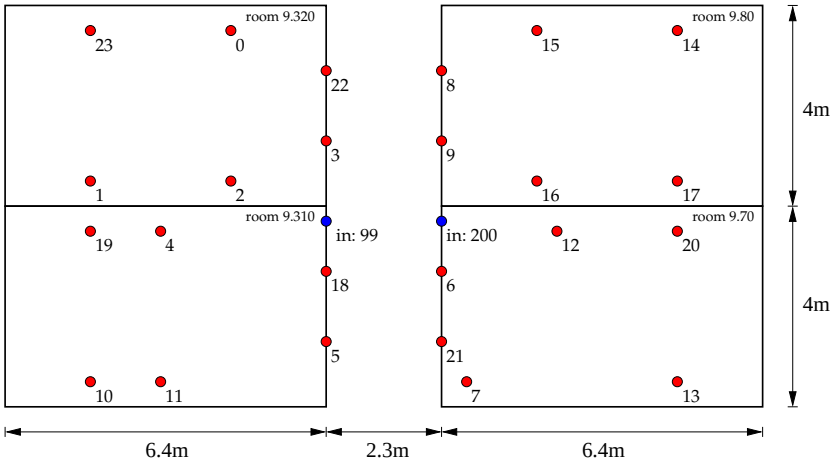
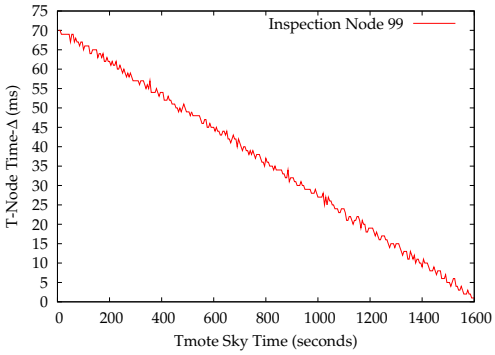
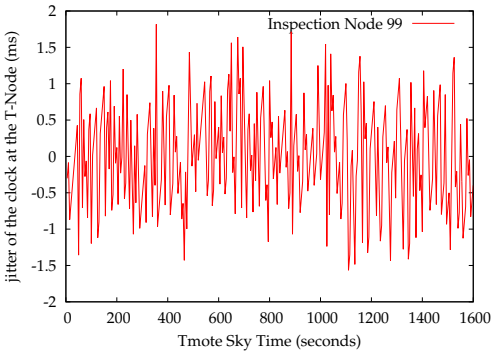


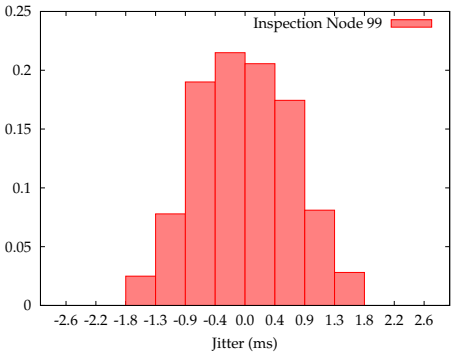
Figure 6.1: Testbed Layout



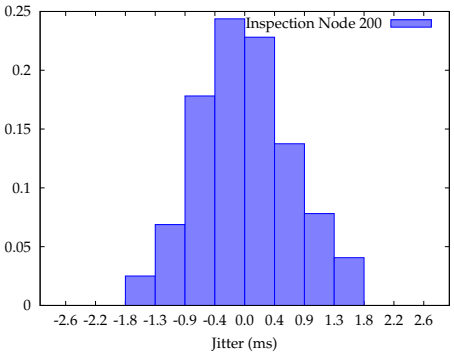
(a) Clock-drift of the T-Node with respect to the clock of the Tmote Sky



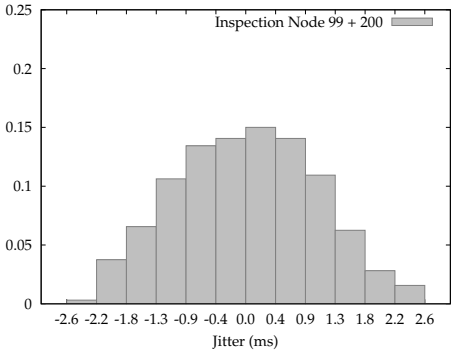
(b) Jitter of the clock of the T-Node



(c) Probability Density Function of the jitter of the clock at the T-Node of inspection node 99



(d) Probability Density Function of the jitter of the clock at the T-Node of inspection node 200



(e) Probability Density Function of the jitter between the T-Nodes of the two inspection nodes

Figure 6.2: Accuracy of the Time-Synchronisation between the Inspection Nodes

mission range, and they will, therefore, form a single-hop network. This is not a realistic representation of a deployment, and therefore the transmission power of the T-Nodes is reduced, so that a multi-hop network will be formed. Each sensor node will generate a data message every 2 seconds and will transmit that message towards the sink (sensor node 0). The laptop with a connected Tmote Sky is placed 5 meters from the inspection nodes.

6.2 Accuracy of the Time Synchronisation between the Inspection Nodes

The need for time synchronisation between the inspection nodes is described in Section 3.1 and Section 4.1. Time synchronisation is an important requirement and therefore its accuracy is also important. The accuracy is measured by the amount of jitter of the clock at the T-Node part of the inspection node.

The clock of the T-Node is synchronised every 5 seconds to the clock of the Tmote Sky connected to the laptop by recording the difference between the two clocks (time-Δ), at the T-Node. When a synchronised timestamp is needed, a timestamp of the local clock of the T-Node is taken and added to the time-Δ. The slope in Figure 6.2a, obtained by linear regression, represents a 1 millisecond clock-drift of the T-Node every 22.8 seconds with respect to the clock of the Tmote Sky connected to the laptop. Time synchronisation is able to compensate for the clock-drift, but not for the jitter. When the clock-drift is removed from Figure 6.2a, the jitter remains and is shown in Figure 6.2b. The probability density function of the jitter of inspection nodes 99 and 200 are shown in Figure 6.2c and Figure 6.2d. From these figures we conclude that there is a 84.4% and 84.6% probability that the error of the time synchronisation of the inspection nodes lies around 1 millisecond. The synchronisation processes in both inspection nodes are independent of each other, and both jitter functions have a normal distribution, therefore both jitter functions can be added to each other to form one combined probability density function. This function is shown in Figure 6.2e. From this figure we conclude that the difference in time between two inspection nodes lies within 3 milliseconds of each other with a probability of 85.6%.

The radio of the T-Node can only send 7 bytes within 3 milliseconds, as its bit-rate is 19.2kbps. The header of a CC1000 radio message, alone, is 7 bytes, meaning that the radio message is long enough to span more than 3 milliseconds. This makes it impossible to receive multiple radio messages within 3 milliseconds of each other. Therefore when the message traces of the inspection nodes are combined at the central computer, the logical time order of the collected messages will be preserved. This implies that the time synchronisation is accurate enough to combine the message traces of the inspection nodes and to correctly filter-out duplicate messages at the central computer.

6.3 Passive Inspection

In order to evaluate the passive inspection capabilities of the implemented analysis tool and the inspection network, we compare two cases. The first case will function as a reference experiment for the second case. In the second case we re-introduce a previously resolved bug to check whether or not the analysis tool can detect the

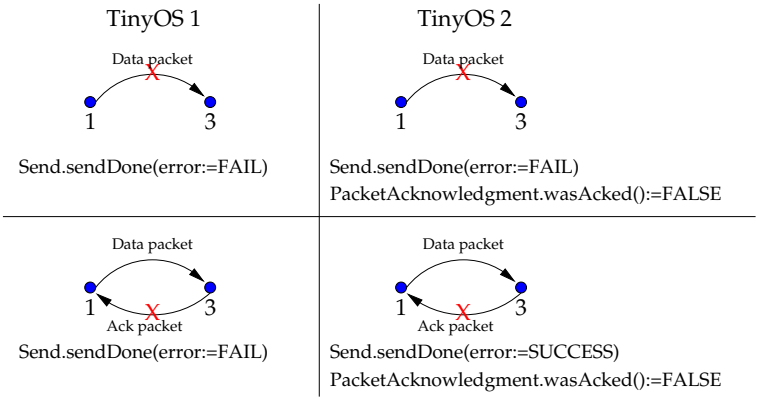


Figure 6.3: Port-bug: changed `Send.sendDone` semantics

bug. We have chosen a previously resolved bug for two reasons. The first reason is time constraints. In the case of observation you have to wait until an anomaly occurs and that can take a long time. The second reason is that it is very difficult to detect an anomaly during the development phase of the inspection network and analysis tool. Especially in the initial development phase, where an anomaly is more likely to be a bug in the tool than in the subject under observation.

The previously resolved bug used in the experiment was introduced while porting software from TinyOS 1 to TinyOS 2. This bug will be referred to hereafter as the port-bug and is described as follows. The process of sending a data packet consists of two events; transmitting a data packet and receiving an acknowledgement packet. When both events are successful, the data packet is successfully received by the destination. In TinyOS 1, if either event failed the parameter `error` of the signal `Send.sendDone` would indicate FAIL, as shown in the TinyOS 1 column in Figure 6.3. In TinyOS 2 there was a need to make a distinction between the two events. The parameter `error` of the signal `Send.sendDone` now only indicates FAIL when the data packet was not successfully sent. Whether or not the acknowledgement packet is successfully received is indicated with the result of the function `PacketAcknowledgment.wasAcked()`, as shown in the TinyOS 2 column in Figure 6.3. Due to the change of the semantic meaning of the `error` parameter, the network-layer assumes that the packet was not successfully transmitted while it actually was but no acknowledgement was received. The network-layer will assume that the process of sending a packet went wrong, and therefore not accounting for the fact that the link was too weak between the node and its destination. This results in the network-layer repeatedly retransmitting the packet, as long as the packet is not successfully being acknowledged. Added to this, the packet is never marked as retry and its link statistics are never updated.

The impact of this bug can be visualised using the MAC-layer statistics and the routing tree. The MAC-layer statistics of the reference experiment are shown in Figure 6.4. Data packets can either be collected or deduced. If a data packet is collected, it is actually overheard by an inspection node. If a data packet is deduced, it is deduced from the extra sequence numbers of the data packets that are overheard by the inspection nodes. From this figure we conclude that most data packets are acknowledged, except for those of node 14. The benefit of the

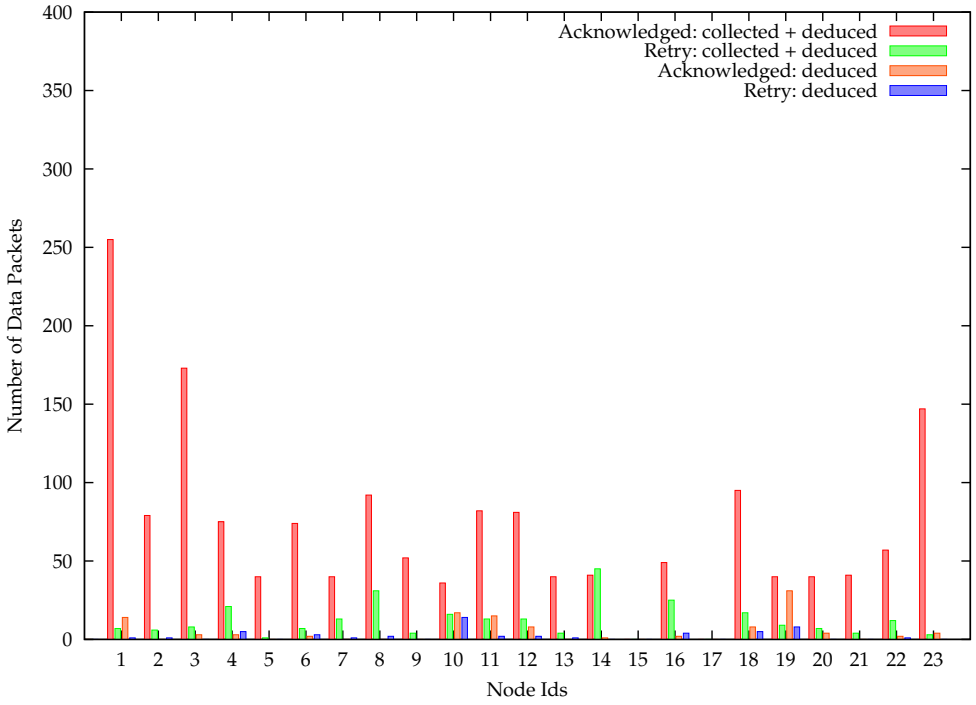


Figure 6.4: MAC-layer statistics of the reference experiment

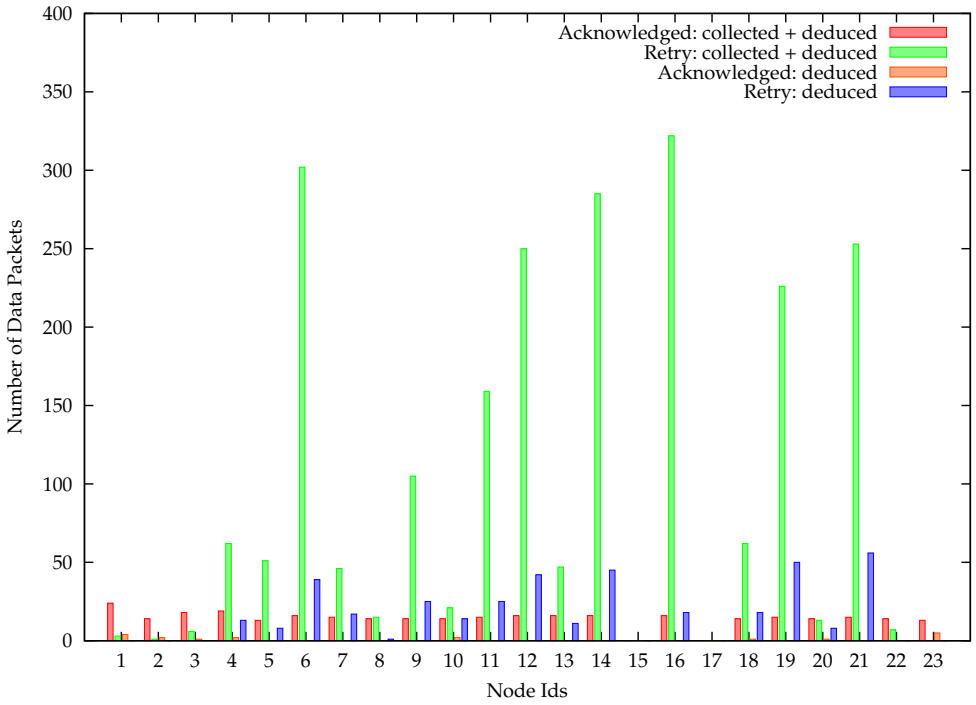


Figure 6.5: MAC-layer statistics of the port-bug experiment

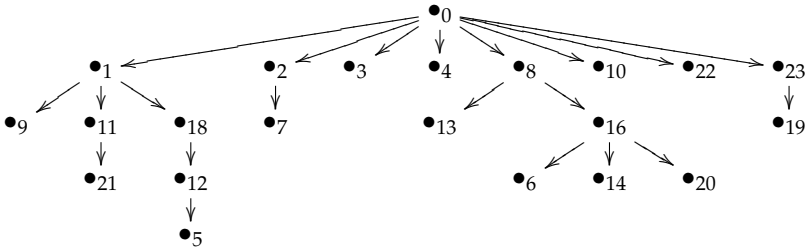


Figure 6.6: Routing Tree Layout of the reference experiment

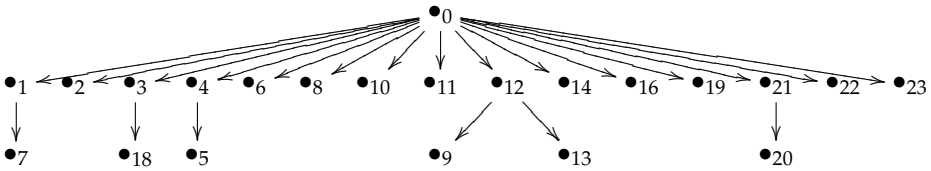


Figure 6.7: Routing Tree Layout of the port-bug experiment

extra sequence numbers (see Section 5.1.2) is evident at nodes 10 and 19. Without the extra sequence numbers, 31 of the 40 (78%) acknowledged packets and 8 of the 9 (89%) retry data packets would not have been detected from the collected data packets of node 19. Figure 6.4 also shows that nodes 15 and 17 have sent no data packets at all. The reason for this is that the links are asymmetrical; nodes 15 and 17 are not able to hear the other nodes, while the other nodes can hear nodes 15 and 17. This conclusion is based on the routing tables of the nodes, collected by the network-layer statistics.

The MAC-layer statistics of the port-bug experiment are shown in Figure 6.5, and shows that almost 90% of all transmitted data packets are retry data packets. This is a clear indication of a problem. The benefit of the extra sequence numbers in the port-bug experiment is less apparent than in the reference experiment. For example, in the port-bug experiment 65 of the 253 retries of node 21 are not detected. Although 65 is a significant number, it is only 26% of the total number of retry data packets.

Figure 6.5 also shows that not all sensor nodes are affected by the port-bug. The sensor nodes that have sent many retries are affected by the port-bug while the other ones are not. Sensor nodes that are closer together are more likely to have stronger links than nodes that are situated further apart from each other. Due to this there is less chance of an acknowledgement not being received when sent between two nodes that are close together. Therefore, there is less chance of an instance of the port-bug occurring between two nodes that are situated close together. The sensor nodes that have sent many retries have chosen the sink as parent. Due to asymmetrical links, they are able to receive packets transmitted by the sink but not the other way around. The reason why the sensor nodes have chosen the sink as parent, is because the routing layer wants to create the shortest path towards the sink, and this is achieved by communicating with the sink directly. The asymmetrical links, together with the link statistics not being updated, causes the sensor nodes not to switch to a parent closer by.

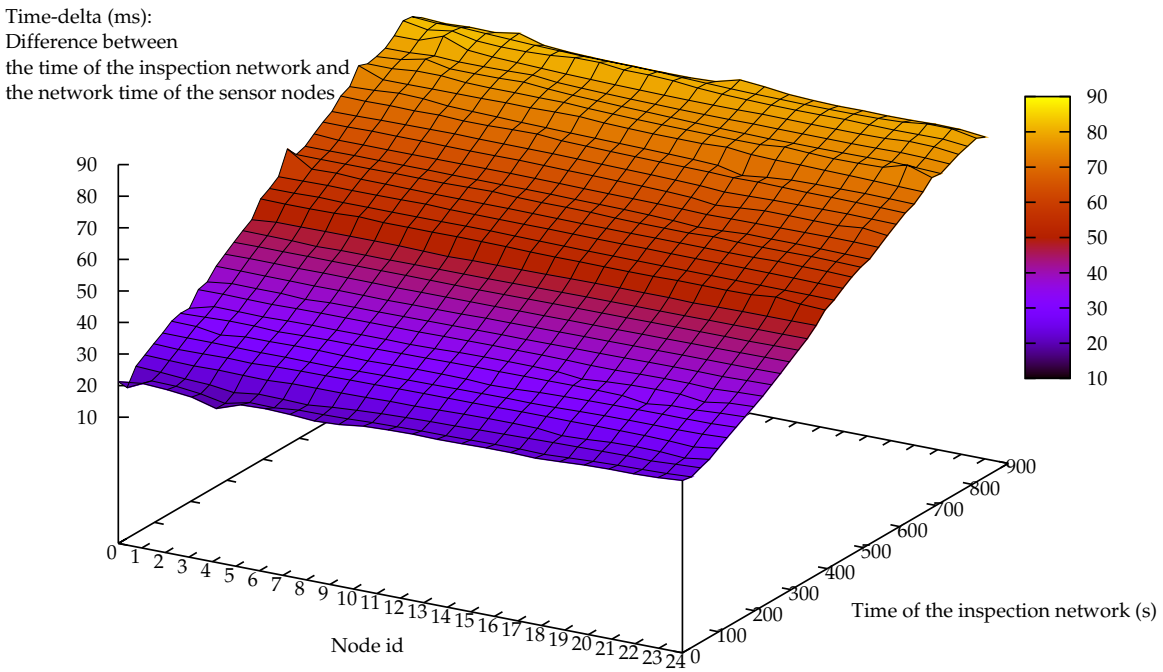


Figure 6.8: The elapse of the network time during active inspection

That the sensor nodes affected by the port-bug are located too far from the sink is visualised by comparing the routing-tree layout of the reference experiment (Figure 6.6) with that of the port-bug experiment (Figure 6.7). In Figure 6.7 the sensor nodes affected by the port-bug are connected to the sink directly, whereas in Figure 6.6 they are not.

6.4 Active Inspection

Active inspection consists of the ability to inject radio packets and acknowledge data packets. In order to test the ability to inject radio packets, we have chosen to use data packets, as it is a requirement that data packets are acknowledged by the recipient. An acknowledgement also shows whether or not the network time of the destination node is unaffected by the injection. This is done by comparing the timestamp in the acknowledgement with the timestamp of the inspection node, and by comparing the timestamp in the acknowledgement with the timestamps of previously received packets from the destination node and other nodes (see Section 5.1.3).

	Inspection Node 156		Inspection Node 200	
Run	Temperature	Clock-drift	Temperature	Clock-drift
1	-2 °C (outdoors)	47.1 ppm	20 °C (indoors)	42.6 ppm
2	20 °C (indoors)	49.9 ppm	20 °C (indoors)	42.6 ppm
3	-6 °C (outdoors)	44.5 ppm	-6 °C (outdoors)	29.9 ppm

Table 6.1: The clock-drift of the T-Node with respect to the clock of the Tmote Sky connected to the laptop.

The experiments of injecting radio packets uses the same setup as the experiments used for passive inspection. In the experiment performed, inspection node 99 sends each sensor node a data packet with source address 40 and CtpOriginId 40. Data packets with CtpOriginId 40 are tracked by the analysis tool to visualise the complete path from the source to the sink. Inspection node 99 was able to successfully send a data packet to each sensor node in the testbed, as its send power was not reduced. Nodes 15 and 17 also acknowledged the receipt of the data packet, but were not able to send the data packets towards the sink, as they both do not have a parent. Figure 6.8 shows the elapse of the network time during the experiment. For nodes 15 and 17, the inspection network only collected the acknowledgements for the data packets that were injected. The rest of the data collected about these nodes are interpolated. The slope of the line in the delta/time plane represents the drift between the clocks of the T-Nodes in the deployment and the clocks of the inspection nodes. In the delta/node id plane the ideal scenario would be that the lines are straight and horizontal, meaning that all the clocks of the sensor nodes are always synchronised with each other. Although this is not completely the case, the lines are straight enough to conclude that the network time was not affected by the injection of data packets.

The setup of the experiment of injecting acknowledgements consists of two sensor nodes and two inspection nodes. The idea behind this experiment is to first let the two sensor nodes setup a link between themselves, where after the destination sensor node is replaced by an inspection node. In this experiment we want to test the ability of an inspection node to successfully acknowledge any data packets that the sensor node sends. Node A generates data packets and sends them to node B, the sink. After node A has successfully sent multiple data packets to node B, node B is disabled by removing its battery pack. At the analysis tool, inspection node 99 is set to reply as node B. Inspection node 200 collects data messages from node A and collects acknowledgements from inspection node 99. From the MAC-layer statistics at the analysis tool we were able to conclude that node A successfully received the acknowledgements sent by inspection node 99, as its payload immediately changed after the transmission of an acknowledgement.

6.5 Influence of Temperature Differences on the Time Synchronisation between Sensor Nodes

In early January, we inspected a deployment of roughly 30 sensor nodes with a dimension of approximately 15 by 15 meters located in two back gardens. The sink and two sensor nodes where placed inside one of the houses. A number of

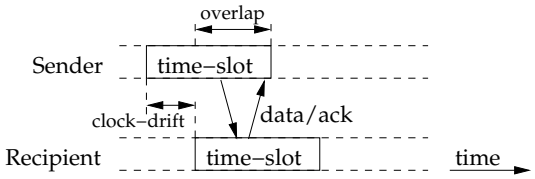


Figure 6.9: Time-slot miss-alignment due to clock-drift

sensor nodes where placed inside the garages at the end of the gardens, while the remaining sensor nodes were placed in the two gardens. Different temperatures were experienced by sensor nodes placed in different locations. During the inspection of this deployment we detected that the network time of certain sensor nodes drifted away from other sensor nodes. After brainstorming we suspected that the difference in temperature could have an effect on the synchronisation of the network time. In order to check whether this premise was valid, we used the built-in facilities of the inspection network to monitor the time synchronisation between the inspection nodes. The influence of the temperature difference is measured in the amount of clock-drift of the T-Node with respect to the clock of the Tmote Sky connected to the laptop. The duration of each run of the experiment lies around 50 minutes, and the results are summarised in Table 6.1. From these results we can conclude that as the temperature drops the clock-drift between the T-Node and the Tmote Sky becomes smaller. This means that the clock of the T-Node runs slower as the temperature drops.

The variation of clock-drift due to temperature difference has the following effect on the operation of the deployment. The clock-drift between two sensor nodes causes the miss-alignment of the time-slots between the sensor node that sends the message and the sensor node that receives the message (see Figure 6.9). Due to the extreme drift between the clocks of these two sensor nodes, the overlap of the two time-slots becomes too small for the destination to receive the message sent by the source. When the destination does not receive any messages it can not synchronise its clock with the clocks of the other sensor nodes, as it needs the network timestamp in the message to do so. The solution to this problem is to increase the frequency of broadcasting synchronisation messages. The result is that before the time-slots do not overlap any more, the clocks of the sensor nodes are synchronised to ensure the alignment of the time-slots.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

It is impossible to resolve all problems before deployment, as it is impossible to accurately model an environment due to its many variables. Passive inspection is proposed by Ringwald et al. [18] to detect problems in WSN-application during deployment by overhearing the communication between sensor nodes. Our research attempted to further evaluate the concept of passive inspection and to answer whether we can extend passive inspection with active inspection. To perform our evaluation, we first had to design and implement an inspection network.

Before we could evaluate the passive and active inspection capabilities of our inspection network we had to evaluate the accuracy of the time synchronisation between the sensor nodes (see Section 4.1). The results of this evaluation are very positive as the time synchronisation manages to achieve that the drift between clocks of the inspection nodes stays within 3 milliseconds of each other with a probability of 85.6%. This implies that the time synchronisation is accurate enough to combine the message traces of the inspection nodes and to correctly filter-out duplicate messages at the central computer.

The passive inspection capabilities of our inspection network were evaluated by comparing a normal operation of the deployment on our testbed with an operation where a previously resolved bug, the port-bug, was inserted. The aim of this was to check whether or not the port-bug could be detected by the analysis tool of the inspection network. The results show that we would have been able to detect the port-bug based on the MAC-layer statistics and the routing tree layout. The MAC-layer statistics revealed that almost 90% of all data packets were retries in the port-bug experiment whereas less than 10% of all data packets were retries in the reference experiment. When comparing the routing tree layout with the MAC-layer statistics we were able to deduce why some sensor nodes were less affected by the port-bug than others. The results also show that without extra sequence numbers (see Section 5.1.2) a significant number of messages would not have been detected. This would negatively effect the MAC-layer statistics, but not so much that we would be unable to detect the occurrence of the port-bug. From these results we make the following generalisations:

Completeness vs. coverage: In order to detect problems within the deployment it is more important that the limited number of inspection nodes are placed in

such way that the part of the deployment that they cover delivers complete information rather than trying to cover the whole deployment at once. It is not necessary that the inspection nodes cover the whole deployment at once, as sensor nodes only communicate with their direct neighbours. However, it is important that the data collected by the inspection nodes is complete, meaning that there are no holes in the collected information. For instance a hole can be a missed acknowledgement packet. Suppose that the maximum number of retries is set to 10 data packets and only 9 data packets are collected then the question arises if we missed the 10th packet; meaning that the maximum number of retries has been reached; or that the data packet was acknowledged after the 9th transmission. It is impossible to make this distinction, because of the incompleteness of the collected data.

Significance: The effect of a bug must be measurable. If the effects are small, the bug is difficult to detect. The effect of the port-bug on the MAC-layer statistics was so big that it was easy to detect that there was a problem.

The conclusion of this MSc thesis is that with our further evaluation of passive inspection, we were able to detect real-life bugs using real hardware. Added to this, it is possible to detect problems at a lower level than it is possible with a root-cause tree, by using detailed statistics about the inner-workings of the deployment. Further, we conclude that active inspection can be used without negatively effecting the behaviour of the sensor nodes in the deployment. Future research into new applications of active inspection must evaluate its usefulness.

7.2 Future Work

Our suggestions for future work are grouped by the following topics:

Inspection Network: The inspection network can be improved by modifying or replacing the hardware of the inspection node. The peripheral interface of the processor at the Tmote Sky is responsible for the bottleneck of the transport of data between the inspection node and the laptop (see Section 4.4.2). By modifying the hardware of the Tmote Sky, the UART connection should be able to use the second peripheral interface (USART1) of the processor. A more rigorous solution would be to increase the data throughput by replacing the radio chip and the processor with more powerful ones.

Passive Inspection: The most common concept in order to increase the practical use of passive inspection, is to let the sensor nodes in the deployment broadcast debug information, so that the inspection network can capture this information. This removes the need to send debug information in-band with the actual sensor network traffic. An example of a realisation of this concept has been proposed by Römer et al. [19] where sensor nodes emit an assertion message when a predefined condition is met. This increases the visibility of the internal state of the sensor nodes.

Active Inspection: There are several concepts that could make use of active inspection. The most basic concept would be to alter settings within the sensor

nodes. For example, to enable or disable the broadcast of assertion messages. A more advanced concept would be to inject messages in order to execute predefined test-cases. Passive inspection can be used to evaluate whether the test-case passed or failed. Active inspection can also be used to spread new software images in order to re-program sensor nodes.

Bibliography

[1] B. Chen, G. Peterson, G. Mainland, and M. Welsh. Livenet: Using passive monitoring to reconstruct sensor network dynamics. In *Proceedings of the 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2008)*, pages 79–98, 2008.

[2] Y. Cheng, Bellardo J, P. Benkö, A.C. Snoeren, G.M. Voelker, and S. Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2006)*, pages 39–50, 2006.

[3] Chipcon. CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. Technical report, Texas Instruments, 2007.

[4] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, and P. Blum. Deployment support network - a toolkit for the development of WSNs. In *Proceedings of the 4th European conference on Wireless Sensor Networks (EWSN 2007)*, volume 4373/2007, pages 195–211. Springer Berlin / Heidelberg, 2007.

[5] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo. The collection tree protocol (CTP). Technical Report TinyOS: TEP 123, TinyOS Core Working Group, 2007.

[6] G.Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN 2005)*, pages 121–135, 2005.

[7] I. Haratcherev, G. Halkes, T. Parker, O. Visser, and K. Langendoen. Powerbench: A scalable testbed infrastructure for benchmarking power consumption. In *International Workshop on Sensor Network Engineering (IWSNE 2008).*, 2008.

[8] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Network*, chapter Network Architecture, pages 60–61. Wiley-Interscience, 2005.

[9] M.M.H. Khan, L. Luo, C. Huang, and T. Abdelzaher. Snts: Sensor network troubleshooting suite. In *Proceedings of the 3rd International Conference on Distributed Computing in Sensor Systems (DCOSS 2007)*, 2007.

[10] J. Lee, W. Kim, S. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi. An experimental study on the capture effect in 802.11a networks. In *Proceedings of the the 2nd ACM international workshop on Wireless network testbeds, experimental evaluation*

and characterization (WiNTECH 2007), pages 19–26. ACM New York, NY, USA, 2007.

[11] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003)*, pages 126–137. ACM New York, NY, USA, 2003.

[12] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Analyzing the mac-level behavior of wireless networks. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2006)*, pages 75–86, 2006.

[13] M. Maroti and J. Sallai. Packet-level time synchronization. Technical Report TinyOS: TEP 133, TinyOS Core Working Group, 2008.

[14] C.E. McDowell and D.P. Helmbold. Debugging concurrent programs. In *ACM Computing Surveys (CSUR)*, volume 31, pages 593–622. ACM New York, NY, USA, 1989.

[15] T. Parker, M. Bezemer, and K. Langendoen. The λMAC framework: redefining MAC protocols. PDS-2007-004, Delft University of Technology, 2007.

[16] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd international conference on Embedded Networked Sensor Systems (SenSys 2005)*, pages 255–267, New York, NY, USA, 2005. ACM.

[17] M. Ringwald. *Reducing Uncertainty in Wireless Sensor Networks: Network Inspection and Collision-Free Medium Access*. PhD thesis, ETH Zurich, 2009.

[18] M. Ringwald, K. Römer, and A. Vitaletti. Passive Inspection of Sensor Networks. In *Proceedings of the 3rd IEEE international conference on Distributed Computing in Sensor Systems (DCOSS 2007)*, volume 4549, pages 205–222. Springer, 2007.

[19] K. Römer and M. Ringwald. Increasing the visibility of sensor networks with passive distributed assertions. In *Workshop on Real-World Wireless Sensor Networks (REALWSN 2008)*, 2008.

[20] S. Rost and H. Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *Proceedings of the 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON 2006)*, pages 577–584, 2006.

[21] W. Simpson. PPP in HDLC-like Framing. RFC 1662, Daydreamer, 1994.

[22] A. Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM 2001)*, pages 319–324, 2001.

[23] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: a wireless sensor network testbed. In *Proceedings of 4th international symposium on Information Processing in Sensor Networks (IPSN 2005)*, pages 483–488, 2005.

[24] J. Yang, M.L. Soffa, L. Selavo, and K. Whitehouse. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys 2007)*, pages 189–203, New York, NY, USA, 2007. ACM.