CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Intrusion Detection System Framework for Internet of Things

Construction of an Intrusion Detection System for Wireless Sensor Nodes with Modern Hardware

Master’s thesis in Computer Systems and Networks

Johan Becker
My Vester

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

MASTER'S THESIS 2017

Intrusion Detection System Framework for Internet of Things

Construction of an Intrusion Detection System for Wireless Sensor Nodes with Modern Hardware

Johan Becker
My Vester

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Intrusion Detection System Framework for Internet of Things
Construction of an Intrusion Detection System for Wireless Sensor Nodes with Modern Hardware
JOHAN BECKER
MY VESTER

Supervisor at University: Olaf Landsiedel, Network and Systems division
Supervisor at Company: Stefan Rasmusson, Combitech
Examiner: Magnus Almgren, Networks and Systems division

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Three Texas Instruments SensorTag cc2650stk sensor nodes in casings.

Typeset in LaTeX
Gothenburg, Sweden 2017

Intrusion Detection System Framework for Internet of Things
Construction of an Intrusion Detection System for Wireless Sensor Nodes with Modern Hardware
JOHAN BECKER
MY VESTER
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Today, we see an increasing trend towards connected devices. This trend of connecting devices instead of people is called the Internet of Things (IoT). Some of these devices are sensor nodes that are battery-driven micro controller units that are equipped with sensors and wireless communication capabilities. When they are connected to each other they compose a wireless sensor network (WSN). Historically the sensor nodes have been very limited both in terms of computational power and size of memory. As the nodes have grown more powerful, the WSNs have started to communicate using IP, allowing for communication towards the Internet, which makes the network vulnerable against common attacks against connected devices. This is a problem since the nodes often lack protection due to their hardware limitations. However, a new and more powerful generation of sensor nodes is currently available. Allowing for additional security for the applications because they now have more memory, hence they can store both the intended application and an Intrusion Detection System (IDS).

This thesis presents the design, implementation and evaluation of a novel design of an IDS framework for sensor nodes. The IDS is implemented on top of the Contiki operating system (OS) which is a widely used OS for wireless sensor nodes. The evaluation of the IDS is performed with focus on energy consumption, detection rate, network reliability and latency, which makes the results comparable to other related works in the field. The main contribution of the thesis is a novel design of a detection method for detecting different routing attacks against RPL including sinkhole attacks, wormhole attacks and selective-forwarding attacks. The method is called RoVer which stands for role-based verification.

The IDS framework combines different detection methods for discovering both Denial of Service attacks and routing attacks. The implementation is tested and evaluated on the modern sensor node platform called Texas Instruments SensorTag CC2650STK. Results show that the methods designed and implemented within the thesis are not just feasible but also effective when detecting attacks against the sensor nodes. Evaluation shows that RoVer has a detection rate of 100% while the two detection algorithms for flooding attacks have detection rates on 75%, all while keeping the amount of false alarms to a low number.

Keywords: IoT, IDS, WSN, Contiki, security, 6LoWPAN, IPv6, RPL, DoS, sinkhole.

Acknowledgements

We would like to thank our supervisor Olaf Landsiedel at Chalmers University of Technology for his endless help and support during the thesis. Olaf provided us with the hardware and helped us to form our idea into a real project. It would not have been possible for us to do this without him.

Also a great thank you to our examiner Magnus Almgren at Chalmers University of Technology who has provided us with valuable feedback during the process.

We would also like to thank our supervisor Stefan Rasmusson at Combitech Gothenburg for all of his input and guidance during the thesis.

A special thanks to Combitech who provided us with a workplace and coffee.

Finally, we would like to thank friends and family for their love and support.

Johan Becker, Gothenburg, June 2017
My Vester, Gothenburg, June 2017

Abbreviations

API	Application Programming Interface.
Contiki	A real time operating system for wireless sensor nodes.
Cooja	Wirless sensor networks simulator which simulates nodes running Contiki.
DODAG	Destination Oriented Directed Acyclic Graphs. Graphs built by the routing protocol RPL.
DIO	DODAG Information Object. A routing message sent and received by all nodes in order to select proper routing graphs.
Energest	A module in Contiki for energy estimation of the nodes.
HIN	High intensity network, a network with a high traffic intensity.
IncA	Increasing Average Algorithm, one of the DoS-attack detection algorithms that the thesis proposes.
LIN	Low intensity network, a network with a low traffic intensity.
LiReg	Linear Regression Algorithm, one of the DoS-attack detection algorithms that the thesis proposes.
RoVer	Role-based Verification, a method that the thesis proposes for securing RPL routing by verifying claims from two sides.
RPL	A Routing Protocol for Low power and Lossy Networks.
Servreg-hack	An application in Contiki which is used to register and announce network services.
TCP/IP-module	A module that is part of the Contiki operating system that handles IP-communication.
TUD	Time until detection, the time it takes from an attack is launched until the attack is detected.
WSN	Wireless sensor network.
6LoWPAN	IPv6 over Low power Wireless Personal Area Networks.

Contents

List of Figures

List of Tables

List of Algorithms

1
Introduction

This chapter serves as an introduction to this master thesis. It presents the problem background and the purpose of the project. It also states the aim and discusses previous work. Then it provides a short motivation for the thesis as well as the scope and limitations before briefly describing the implementation, and the hardware and software used. Finally, it provides the reader with an overview of the report.

1.1 Problem Background

Today, we see an ever increasing trend towards connected devices. Earlier, a person connected himself to the Internet via an Internet connection. Today, it is not unusual that a person owns many different devices that are connected to and communicate via the Internet. Many of these devices are small and resource limited, and together they compose what is commonly known as the Internet of Things (IoT). A concept in the IoT is networks of small sensors that communicate with each other wirelessly. These networks of sensors are known as wireless sensor networks (WSNs) and there are many different use cases for them, like monitoring the temperature in a room. Because of the limitations on the devices in WSNs in terms of both computational resources and memory capacity there is a lack of protection against intrusions and attacks [1]. The consequences of an attack against these networks can be very serious. For example, if a small WSN monitoring a forest, looking for forest fires, is attacked in such way that it can not function properly it will not be able to send any warnings. Hence, fires that could have been detected early on and stopped before they could do much damage now have the potential to grow to be a real threat before anyone notices [2].

1.2 Purpose

The purpose of this master thesis is the design, implementation and evaluation of a framework consisting of different intrusion detection mechanisms suitable for nodes that are part of sensor networks in the IoT. The framework has support for different detection methods which can be combined. This also leads to different demands of energy consumption because of the changing complexity of the solutions and the combination of them. The chosen operating system (OS) used for the implementation is Contiki [3]. Contiki is a well known and extensively tested open source OS in the field of WSNs. It is developed by a world-wide community of computer experts and many different companies are contributing to the development of the OS.

1.3 Aim

The overall aim of the thesis is to design and implement an Intrusion Detection System (IDS) for IoT applications running on WSNs with support for different detection methods. Implementation of the IDS is done as an integrated part of the operating system Contiki. The detection methods that this framework implements are both based on previous work in the area of IDSs for the IoT and self-developed algorithms. The framework combines the detection methods, providing the possibility of protection against different attacks at the same time.

The goal of the IDS framework is to detect the following types of attacks against the nodes:

1. Generic DoS/flooding attacks against nodes [4]
2. Sinkhole attacks against 6LoWPAN [5] [6]
3. Selective forwarding attack against routing [6]
4. Wormhole attack against routing [7]

1.4 Previous Work

Previous work in the field has proposed intrusion detection systems for IoT and specifically WSNs. The proposed systems differ not only in architecture and platform but also in terms of the used detection techniques and which attacks they are able to detect.

Riecker *et al.* [4] propose a lightweight intrusion detection system for wireless sensor nodes that is based on energy readings and linear regression. The authors describe an intrusion detection system that is in the form of a moving agent which is sent around the network like a token. The moving agent collects readings of consumed energy of the nodes and makes predictions on how much energy that should have been consumed the next time the agent arrives based on linear regression models. By comparing the expected value with the actual value the aforementioned approach allows for detection of generic Denial-of-Service attacks with a high accuracy.

Cervantes *et al.* [5] describe an IDS that focuses on detecting sinkhole attacks, which are a kind of attack that swallows all packets in the network, for secure routing over 6LoWPAN. They propose an IDS called INTI (Intrusion detection of SiNkhole attacks on 6LoWPAN for InterneT of ThIngs), which combines a watchdog, reputation and strategies that builds on trust between the nodes which are used to analyze the behavior of the network.

SVELTE is a real-time intrusion detection system for WSNs proposed by Raza *et al.* [6]. It is constructed as integrated components in the Contiki OS and targets different routing attacks. The main part of SVELTE is a network mapper, which keeps track of the network topology and uses this information as a map when detecting sinkhole attacks. Evaluation of SVELTE shows that it detects all the attacks that are being launched, however there are some false alarms.

A centralized IDS that focuses on illegal memory writes (so called "buffer overflows") is proposed by Saeed *et al.* [8]. In this case, the IDS is implemented in the sink node and a random neural network is used to train a model of what normal behavior is in the network. Abnormal behavior (or anomalies) such as reception of data that is larger than the normal case will lead to abortion of the operation in order to not compromise the sink.

This section just serves as a brief overview of what previous work has proposed in the field. Chapter 3 describes the approaches mentioned here and other approaches in more detail.

1.5 Motivation

Earlier research in the field of IDSs for IoT devices is carried out on sensor nodes with lower memory and computational power in comparison with sensor nodes available today. This thesis uses a new generation of sensor nodes, which allows for more complex solutions due to an increase in computational power. The chosen hardware platform to use is Texas Instruments SensorTag CC2650STK [9] which supports a processor frequency of up to 48 MHz which is 6 times faster than the popular but older sensor node Tmote Sky which runs at 8 MHz [10]. The platform is further described in Section 2.5.

The framework consists of different approaches that have been implemented and proved to work well as stand-alone applications on specific hardware. This framework provides more functionality. This leads to an IDS-framework with more functionality than the stand-alone solutions. The challenge with the aforementioned approach is to make the different solutions work well together and function properly on modern hardware.

Thus, the process of combining the earlier proposed solutions into a new unique and more powerful system demands in-depth knowledge of existing research prototypes, and understanding the constraints of the problem formulation so that the algorithmic choices in the combination of methods are well founded. Even though the synthesis of the algorithms can sound quite easy, complex trade-offs need to be analyzed and dealt with. Learning outcomes of the thesis is a deeper understanding of the research in the field and also a scientific contribution in the field of security for the IoT. Many research papers are critically analyzed in order to make a plan on how to implement the proposed framework in this thesis. Since the authors lack previous experience of the chosen operating system Contiki, a deeper understanding of the platform is one of the learning outcomes from the thesis.

1.6 Scope and Limitations

The goals which Section 1.3 describes, also describes how the solutions of the framework are implemented and limited to detect certain attacks. The goals are referring to detection methods that are inspired by state-of-the-art techniques, namely in the following papers [4], [5] and [6]. No additional intrusion detection mechanisms are considered beyond the adaption and design of methods that meet the goals.

The design and implementation of the framework should work on wireless sensor nodes. To be more specific, it should work on the hardware mentioned in Section 2.5, no testing or evaluation is performed on other hardware.

1.7 Implementation, Software and Hardware

The design and implementation of the IDS is carried out in the open source OS Contiki. The IDS is implemented and included as a part of Contikis source files. Contiki contains a simulator called Cooja, which has support for simulation of several different sensor nodes. It is convenient to first test the implementation in the simulator to verify the functionality since loading the code into the hardware and test it takes more time. However, the hardware this thesis uses is not supported in the simulator, so preliminary testing in the simulator needs to be carried out on sensor nodes that are less powerful than the actual hardware used within the thesis.

1.8 Organization of the Rest of the Thesis

The rest of the thesis is organized as follows: In Chapter 2, relevant background information introduces the reader to the topic. Then, in Chapter 3, related work in the field is presented before Chapter 4 describes the designed system model of the IDS in this thesis. Chapter 5 further describes how the different detection methods proposed in the system model are implemented and then Chapter 6 describes the evaluation of the system and its parts. Finally, Chapter 7 discusses the results and compares them with results from related work and Chapter 8 presents the conclusions of the thesis.

2
Background

This chapter provides the reader with relevant information about the topic of the thesis. The chapter also introduces the specific hardware and software used in the project, as well as some basics about intrusion detection and relevant attacks.

2.1 Wireless Sensor Networks

A wireless sensor network is a network consisting of a number of sensor nodes, also called motes, that operates together. Generally, a node is a micro controller which consists of a processor, a radio processor and sensors. A node typically contains several different kinds of sensors, for example sensors that can measure sound, temperature and/or light intensity. As IoT as a concept grows bigger, the use for Wireless sensor nodes has increased, and there exist many different vendors that develop a multitude of different nodes. The main contribution of WSNs is that they provide a bridge between the digital and the physical world. There are many different use cases for these networks; typically they are used to monitor systems and gather data. Typical applications for wireless sensor networks include surveillance of industrial plant processes, monitoring of the environment, or as described by Burdakis *et al.* [2], to have safety-critical monitoring of a forest for fast detection of fires to prevent major damages.

The sensor nodes in a WSN communicate with each other and collaborate when routing the data sent to a dedicated node often called the sink node. The sink node is also known in literature as the ”base station” or ”border router” when used as a bridge between low power IPv6 networks and normal IPv6 networks. The responsibility of the sink node is to take care of all the incoming and outgoing communication to and from the WSN [11].

The nature of the wireless sensor networks requires the nodes to be low powered and computationally efficient. Normally, the nodes are battery-driven and certain respect needs to be taken for this constraint. Because of these constraints many routing and network protocols that are commonly used in other systems are too heavy [6]. New protocols have been developed to suit the needs of WSNs, such as the routing protocol RPL and the Internet protocol 6LoWPAN. These are further described later in this chapter in sections 2.4 and 2.3 respectively.

2.2 Contiki

Contiki is a real-time OS for sensor nodes which is written in the C programming language [12]. It is open source and provides support for both the full IPv4 and IPv6 protocol stack, but also contains more recent protocols that are built for low-power and wireless communication, such as 6LoWPAN and RPL [13].

Cooja is a network simulator that is shipped with Contiki. It is built to simulate and test WSNs virtually in order to provide a good overview of a network. The purpose of Cooja is to provide assistance when developing against the IoT-platforms. Testing and verification of solutions is easier to carry out in a simulator than to program physical hardware and take care of the testing on the real hardware. Cooja supports many different sensor nodes that can be fully emulated [14].

The key feature of the Contiki OS is called ”protothreads”. Protothreads are threads that do not have a dedicated stack in memory and are therefore very memory efficient. The protothreads are driven by events and used in order to provide a flow in the programs written in Contiki. At least one thread is started by default, and this thread can then create other threads to provide multithreading functionality to the programs. Since the processes in Contiki are event-driven, the threads can intercommunicate via process events sent to specific processes. This communication can be used in order to command processes to start, stop or perform other actions [15].

In Contiki, another central part of the OS is the usage of timers. There exist many different types of timers which solves different tasks. One usual type of timer is the Event Timer which generates a timer event when it has finished counting. During the time that a process waits for a timer event it can enter a form of deep sleep mode which minimizes the energy usage of the sensor node. The sensors in Contiki are either synchronous or asynchronous. They either generate a sensor event by themselves or they need to be combined with a timer which takes care of fetching values from the sensor on the periodic intervals.

2.3 RPL

Since wireless sensor nodes are limited in their resources, the routing protocol used on the networks connecting these nodes has to be designed with certain constraints. RPL (Routing Protocol for Low power and Lossy Networks) is a routing protocol that, as the name suggests, is designed for low power and lossy networks that communicate over IPv6. These properties make RPL an apt protocol for IPv6 networks with wireless sensor nodes, since they typically have both low power and low storage [11].

RPL is built to keep packet processing/forwarding and the routing part separate from each other in order to have as good routing as possible, i.e. allowing for packets to reach their destination fast and accurately. Since RPL is built on the usage of directed tree graphs, the communication type in RPL is bi-directional communication. This is the case, since in order for the nodes to communicate they need to be able to reply to messages and forward other messages in both directions of the tree. All relevant information about RPL can be found in the Request for Comments (RFC) document about RPL [16].

The graphs that RPL builds over the network are Destination Oriented Directed Acyclic Graphs (DODAGs). It is important to note that one reason for implementing the graph as a DODAG is that there are no cycles in such graphs, so the data will always flow forward in the graph as long as the network is static and therefore the packets will reach the sink eventually and not risk getting stuck in loops. In a dynamic setting however, there is a possibility that nodes will choose each others as parents all the time, making the data loop. The DODAG is optimized to have a good route between the nodes by using an Objective Function (OF). The OF uses at least one metric that defines the rank of the nodes in the network to determine what to look for when constructing the DODAG that best fits the network. The nodes finds the metrics of the other nodes in the network by looking at specific messages received which are further described below, and can for example be the hop count or energy consumption of a node.

A DODAG is built in a tree-like form from one specific node called the root, where the root is the node in the network that acts as the sink or border router (hereafter referred to as the sink) for the network. In order to create, maintain and also exchange information about the graph there exists four different control messages. The DODAG Information Object (DIO) is the control message that contains information about the routing graph such as the IPv6 address of the root and the current ranks (based on the metrics) of all the nodes. Another control message is the Destination Advertisement Object (DOA) which makes it possible for the information to be routed upwards since it enables support for traffic flowing down in the graph. The DODAG Information Solicitation (DIS) control messages can be sent by a node to ask another about a DIO message from that node. The fourth control message, Destination Advertisement Object Acknowledgment (DAO-ACK) is just an acknowledgment message that a node sends when it receives a DAO in order to confirm the reception of the message.

It is the sink that initiates the creation of a new DODAG for the routing of the network. There can only exist one DODAG for each existing sink, except for cases where there are two sink nodes that are somehow connected, e.g. if they share a common backbone, so when the Sink initiates a new DODAG any other DODAG routed from that node is destroyed. The sink creates the new DODAG by choosing itself as the root with the lowest (i.e. best) possible rank and storing the information about the created DODAG which it will send out in DIO messages. When the other nodes in the network get a DIO message they will discover the new DODAG and destroy the old one. New nodes that join the network can easily become a part of the DODAG by listening for DIO messages. Upon receiving a DIO message the node selects an appropriate parent according to the rank of its neighbors in the DODAG and sends out its own DIO messages to the other nodes. Whenever a node receives a new DIO message it will check the properties to make sure that it has the best parent, and therefore the DODAG can be flexible.

2.4 6LoWPAN

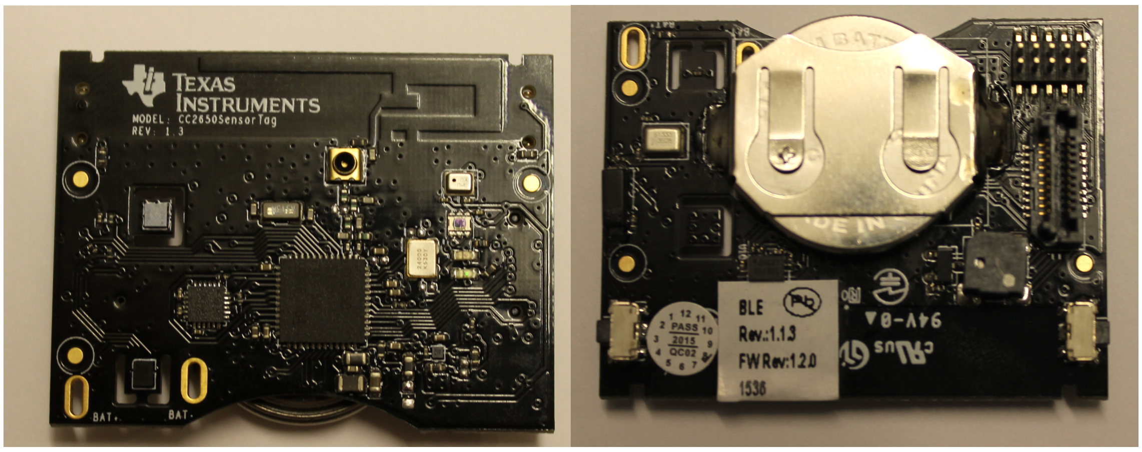
6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) is a developing standard that was originated by the Internet Engineering Task Force (IETF) with the purpose of being used in small networks. IP has come to be a common network protocol for the IoT over the years. However, this was not the initial plan since IP does not meet the specific requirements needed for WSNs since these networks are low powered and lossy while IP is a heavy protocol. There are many advantages with using IP. Some examples of this include that there is no need to start developing something completely new, it will simplify the connectivity model and applications that uses protocols such as TCP and UDP can be used. Because of all the advantages with IP, the development of a thinner version of IP for WSNs called 6LoWPAN was started [17]. As the name suggests, 6LoWPAN is a low power version of IPv6 that works between the ordinary IPv6 protocol and the lossy medium that is the sensor network [13].

SICSlowpan is an implementation of the adaption layer of 6LoWPAN developed by SICS (Swedish Institute of Computer Science). This adaption is now a part of the Contiki operating system. In Contiki, SICSlowpan is used in order to use the radio link layer in the sensor nodes to transmit IPv6 packets. It was a project that aimed to develop the first open source implementation of IPv6 for the low power radio link layer. It resulted not only in SICSlowpan, but also in the implementation of uIPv6, which is the world's smallest IPv6 stack [18].

2.5 The Platform: Texas Instruments SensorTag CC2650STK

Texas Instruments SensorTag CC2650STK is a wireless sensor node that is part of a newer generation of sensor nodes with greater performance. Therefore, it is able to perform heavier computations than the sensor nodes used in many of the referenced papers. The node has a total of ten different sensors, two leds (one green and one red) and two buttons, making the application areas for the node very broad. It contains a micro-controller with a processor that has a frequency of up to 48-MHz Clock Speed, 128KB of In-System Programmable Flash, 8KB of SRAM for Cache and 20KB of SRAM. Figure 2.1 displays the front and the back side of the platform. The left figure shows the front side of the platform where the two leds, some sensors and the CPU is visible. The right figure displays the back side of the platform where the battery and debugging and programming ports are located.

Texas Instruments also develops tools for programing the nodes. These include Code Composer studio (CCS) [19] and UniFlash [20]. CCS is an Integrated Development Environment that is used for programming and debugging the different microcontrollers and processors developed by the company. UniFlash is a standalone software for flashing programs onto the hardware.



(a) Front side. **(b)** Back side.

Figure 2.1: The front and back sides of the platform are displayed. Sensors, leds, CPU, battery and debug ports are visible.

2.6 Attacks against Wireless Sensor Nodes

Since border routers connects wireless sensor nodes to the Internet, they are vulnerable against (adaptions of) common attacks against connected devices. An example type of such attacks is Denial of Service attacks (DoS-attacks), which exhaust the resources on a targeted node [4].

Another common type of attack against WSNs is routing attacks. A routing attack targets the routing information in order to disrupt the routing of the network, e.g. routing information can be spoofed or altered in order to send all traffic through a specific node or create loops in the topology, where packets never reach their destination [21]. Both these types of attacks are further described below.

2.6.1 DoS-Attacks

In a DoS-attack the objective of the attacker is to disrupt the normal functions or services of a network. In a WSN it can be caused by having at least one node in the network send a lot of packets to either all of the other nodes or a chosen node that is particularly important to the system. The nodes that receive the packets need to handle them on the available network services that are open on the nodes, so the nodes will not be able to handle normal traffic or perform its normal tasks at the same time. Kasinathan *et al.* describe in [21] that there exists a wide range of different DoS-attacks, everything from simple variants, like jamming attacks, to more advanced attacks. The paper by Kasinathan *et al.* also discusses that DoS-attacks in WSNs are often targeting 6LoWPAN, since this is a widely used protocol for devices with low capacity. In such an attack the Internet protocol is used to send a large number of IPv6 packets over UDP or TCP.

Further, Kasinathan *et al.* [21], describe the Jamming attack. It is a variant of a DoS attack that focuses on disrupting the radio signal from the nodes by interfering with the radio channel frequency used by the network. It is performed at the link-level by sending a lot of unnecessary packets that will cause the nodes to drop important packets that are used for the normal operation in the network.

2.6.2 Routing Attacks

One common routing attack is the so called Sinkhole attack. It is the main focus of the IDS described by Cervantes *et al.* [5]. They argue that the Sinkhole attack is one of the most destructive attacks against WSNs since it prevents the nodes from communicating properly. A sinkhole attack aims to disrupt the routing in a network by having one node announcing that it knows the shortest path to the sink node. This will make the nearby nodes send much of their traffic in the network through the attacking node. Then, the attacking node will not continue to forward the packets it receives to the other nodes. The node will just receive them and then drop them. The routing protocol designed for and used with 6LoWPAN is RPL, which is further described in Section 2.3. When using this protocol, routing can be disrupted by having an attacking node provide a fake value for the specified metric used to determine how the routing graph, or DODAG, should be constructed [7].

Raza *et al.* [6] focus on a wider range of routing attacks beyond sinkhole attacks. One of these attacks is called the Selective-Forwarding attack. As the name suggest, the main idea behind this attack is to only select certain packets to forward to the rest of the network and disregard all the other packets. An attacker can use this method to let through for example all RPL routing messages but drop messages with other content that the network will send. As an example, if the network is monitoring something the attacking node can choose to drop all the warning messages if something goes wrong, but it can still choose to forward all the RPL routing messages it receives. By doing this it is harder to detect the attack because the nodes will think that there is nothing wrong since the RPL routing messages arrives, making it possible for the DODAG to be constructed and managed as usual [7].

Raza *et al.* [6] also discuss two other routing attacks, which are similar to each other. These are the Sybil attack and the CloneID attack. Both these attacks involves copying identities (IDs). In the Sybil attack, several different IDs are copied on a single node, which makes it look like this node is more than just one to the rest of the network, therefore counting it as several nodes. Whilst in a CloneID attack an ID of a node is copied onto another node in the network. Both attacks have the same goal, to be able to control the network in order to gain majority in voting protocols. In 6LoWPAN and RPL these attacks will be troublesome when a packet has the destination of a cloned node since only the cloned node with the best routing metric will be sent the packet, leaving other nodes with the same ID unreachable [7].

Wallgren *et al.* [7] describe an attack known as the wormhole attack. The wormhole attack is an attack where the attacker creates a virtual tunnel between two nodes. A wormhole itself is not necessary bad and it can be set up intentionally in the system. One potential usage of the wormhole is to send important packets between two nodes that is known to have reliable links while other packets are sent using the normal routing protocol to be sure not to interrupt. However, a wormhole can be set up and used by an attacker to disrupt the normal routing in a network, as an example sending packets in loops or bypassing the sink node in a network.

2.7 Basics about Intrusion Detection

In order to detect attacks in a system, a so called Intrusion Detection System (IDS) can be used. The role of an IDS is to detect attacks in the system and report them. There exists another type of security mechanism that is closely related to an IDS, namely an Intrusion Prevention System (IPS). IPSs aims to prevent an attacker from either continuing with an attack or performing the attack in the first place.

Ghosal *et al.* [22] describe four different kinds of IDSs. These are called signature-based IDS, anomaly-based IDS, specification-based IDS, and hybrid-based IDS. The different kinds of IDSs are further described below.

A signature-based IDS focuses on detecting ”known attacks” by looking at parameters that are known to indicate a specific attack. The known parameters make up the signature of the attack and thus the name of the type of IDS. The limitation of this method is that it will not detect any attacks that are not previously known to the system. A positive thing with the method is that the false positive rate is low.

Anomaly-based IDSs have the advantage of being able to detect attacks that are not previously known to the system if they cause the system to behave different from ”normal” behavior. This means that in order to determine what the normal behavior is in a system, the behavior needs to be somewhat static in its type which does not fit all kinds of systems. This limitation can be a disadvantage for anomaly-based detection techniques. A node can start to behave ”different” than it normally does for natural reasons. One example can be that a node in the network suddenly needs to send a lot of packets at times it usually do not. When a system that is monitoring something have nothing to report due to lack of activity it will not send any data to the border router, but when something happens it will start to send packets. An anomaly based IDS could interpret this as an attack and give a so called ”false positive” alarm [23].

If an IDS is specification-based it means that it combines the best parts of an anomaly- and signature-based IDSs. This is done by having signatures that it knows which can detect known attacks, but it can also look for anomalies. This mean that they do detect anomalies in the network but have fewer false positives since it also combines it with usage of signatures.

Finally, there are hybrid-based IDSs. The main differences between a specification-based and a hybrid-based IDS is that in a system with a hybrid-based IDS there are several explicit IDSs on the WSN that perform different kinds of intrusion detection.

2.8 Detection of DoS-Attacks with Linear Regression

Riecker *et al.* [4] discuss the use of energy consumption as a metric when determining if a system is under attack. The core idea behind this approach is the use of linear regression to be able to predict how much energy that will be used by the node in the following time interval based on the previous energy consumption. Then, by comparing the predicted value with the actual value in the next round, it can be concluded that the node is under attack if the two values deviate too much. This method assumes that the energy consumption in a system is somewhat linear and contains few outliers (data that deviate from the norm) in order to function as expected.

Since energy readings are points of x and y values, where x is the time unit and y is the read energy value, simple linear regression can be used. By fitting a line to the collected x and y values while trying to minimize the distance from the points to the line, the next expected value is predicted to lie on the calculated line. Since the probability that the next energy reading will be exactly on the line is small, there needs to exist a tolerance for how much the real point and the estimated point can deviate from each other without producing alarms that an attack is taking place.

The type of linear regression used by Riecker *et al.* in [4] is simple regression using a variant of the least squares method.

$$f_j(x) = b_j + a_j x = y \tag{2.1}$$

They write the general form of linear regression as seen in Equation 2.1, where $f_j(x)$ is the function for the approximated line in time x at node j, b_j is the y intercept of the linear model at node j, and a_j is the slope of the line at node j. The value that should be calculated, the next expected value, is y. In order to do this, a_j (the slope) and b_j (the intercept) needs to be calculated.

$$a_j = \frac{K \sum_{t=i+1-K}^{i} x_{tj} y_{tj} - \sum_{t=i+1-K}^{i} x_{tj} * \sum_{t=i+1-K}^{i} y_{tj}}{K \sum_{t=i+1-K}^{i} x_{tj}^2 - (\sum_{t=i+1-K}^{i} x_{tj})} \tag{2.2}$$

$$b_j = \frac{\sum_{t=i+1-K}^{i} y_{tj} - \sum_{t=i+1-K}^{i} a_j x_{tj}}{K} \tag{2.3}$$

The slope and the intercept are then calculated by the formulas 2.2 and 2.3 respectively. These formulas are fetched from [4]. Here, x_{tj} is the time value at time t for node j, and y_{tj} the energy reading on node j at time t. This approach uses the last K pairs of time values and energy readings in order to estimate the next value of y. The value of K can be adjusted depending on how accurate the estimation should be. In order to determine a good value for K, there are several things that needs to be taken into consideration. A higher value for K can handle so called ”outliers” better since the other energy readings will be more evenly distributed and lower the slope of the line. As an example, a node will have a higher energy consumption when using the radio, so if an energy reading is taking place shortly after sending or receiving something, the linear regression method will predict the next energy value to be a lot higher than it will be in reality. This can lead the node into thinking that it is under attack when comparing the real energy reading to the estimated one. Therefore K needs to be big enough to prevent this, but having a large K value will have an impact on the memories of the sensor nodes. The scenario can be described as a trade-off between accuracy of the linear regression and the memory footprint of the method on the sensor node's memory.

3
Related Work

This chapter introduces and discusses other works connected to the field of intrusion detection in wireless sensor networks. Section 1.4 gives a brief description of previous works in the field. These are serving as background material for this thesis as some of the algorithms for the detection methods are developed based on approaches found in a selection of those papers. However, many of the algorithms designed and implemented within this thesis are self-developed, and therefore the mentioned papers from Section 1.4 are also listed as related work.

3.1 Energy Consumption as a Metric

As discussed in detail below, monitoring energy consumption as a metric for discovering attacks has previously been tested. The aim with the previous works in the specific field has mainly been to show that energy consumption as a metric for detecting attacks is feasible to begin with. This section introduces two different ways of performing detection with energy consumption as a metric, where the first work is quite early in the research area (2005) and the second is quite new (2015).

3.1.1 An intrusion detection system for wireless sensor networks

Among the earliest research that was carried out on the topic is the work by Onat *et al.* [24]. They propose an architecture of an IDS that takes advantage of an algorithm with low complexity that monitors the received packet signal strength and arrival rates. It is achieved by looking at the last N packets that have been received from each neighbor. These N packets are called the *main packet buffer length.* The packets in the buffer are then used to calculate the statistics for the neighbor node that sent them. Then, each packet that arrives is compared against the calculated values. If the statistics match the information in the arrived packet the newly arrived packet becomes part of the buffer while the oldest packet is discarded. All the packets that are received have their arrival time and receive power saved. A lower and an upper threshold value for the receive power of the packets are updated with each packet that is flagged as normal and inserted into the buffer. If the receive power of an incoming packet is lower or higher than these threshold values the packet is considered anomalous.

However, there are some disadvantages of the method. One of them is that the system requires a so called ”training period”, where the system needs to learn the expected receive signal strength of the packets sent by the neighbors. During the training phase no intrusions can be detected since the system do not know the normal behavior of the network yet. Another drawback is that that the proposed solution only works in a network that has a uniform traffic pattern.

3.1.2 Lightweight energy consumption-based intrusion detection system for wireless sensor networks

Riecker *et al.* [4] propose an architecture of a lightweight intrusion detection system for WSNs that is based on energy readings and linear regression. Their work covers three different aspects: a mobile agent paradigm; the use of this mobile agent to detect intrusions; and the last aspect is that their method shows that it is possible to use the energy consumption as a metric for detecting attacks. As described earlier, they take advantage of a moving agent which moves around from node to node in the network like a token to collect readings of energy consumption by the nodes. The mobile agent uses the energy readings collected to make predictions on how much energy the node should have been consuming the next time the agent arrives based on linear regression models. The approach does not require the nodes to collaborate with each other or monitor their environment and it does not require any data to be transferred to a central point. Their work also serves as a demonstration that power consumption is a suitable metric for detecting denial-of-service-attacks with high accuracy while maintaining low false-positive rates. On the other hand, the implementation of this approach is tested in a simulator and never on real hardware, so the results on real hardware might vary.

3.2 Detection of Sinkhole Attacks in WSN

One attack that can be really devastating in a WSN is the Sinkhole attack. Therefore a number of different works in the field of security for IoT focuses on detecting and mitigating this attack. Some of the earlier research focus on detecting and mitigating not only Sinkhole attacks. Such IDSs with combined detection methods are described in Section 3.3 while this section focuses on works that only target Sinkhole attacks.

3.2.1 Detection of Sinkhole Attacks for Supporting Secure Routing on 6LoWPAN for Internet of Things

One of the works that only focuses on the sinkhole attack is INTI, by Cervantes *et al.* [5]. INTI focuses not only on detecting Sinkhole attacks but also on preventing and isolating the effects that the attacks have on the network. The proposed method is also trying to reduce unfavorable effects on the system. Examples of such unwanted effects can be high resource cost, many false positives and false negatives, and a slow overall system performance. To achieve the goal of detecting and mitigating sinkhole attacks the authors use different strategies. Firstly, a watchdog is used to keep track of the system, and then different reputation and trust strategies are used to detect attacks by analyzing the nodes' behavior. The network topology used is clusters, where the nodes take on different roles. In order to be able to coordinate the network and have cooperation between the nodes the network is self-organized. It is also self-repairing in order to detect suspicious nodes and mitigate them from the network. This IDS also considers the mobility of nodes, something that other works have omitted. The authors conclude that INTI can detect sinkhole attacks in 92% of all cases when the nodes are not mobile and in 75% when they are mobile. This can be achieved while keeping the rate of false positives and negatives low.

3.2.2 Evaluating sinkhole defense techniques in RPL networks

Weekly *et al.* [25] introduce two different techniques for determining if there are any ongoing sinkhole attacks in a network. Their evaluation shows that both the techniques fail to determine if there is an ongoing sinkhole attack on the network most of the times when the techniques are used separately. The authors use a metric called "end-to-end delivery ratio" to evaluate the results of their work. The metric is simply measured by taking number of messages that the sink has received by a specific node divided by the number of sent messages by the same node. The two techniques are named `Parent fail-over` and `Rank authentication`, and when enabled only 16% and 23% of the respective messages are received while an attack is ongoing. This is only slightly higher than what the number of received messages are with no detection method activated at all (10%). However, when the authors combine the two proposed methods the metric goes up to 82%. The first method, the Parent-fail over, is an end-to-end acknowledgment scheme that works by adding a field called the unheard nodes set (UNS) to the DIO message (routing advertisement message) when the sink sends it. The field is then used for storing values in terms of IDs of nodes whose paths may be interrupted by a sinkhole. A node is added to the field if less than a chosen threshold value of messages that it is expected to send are actually received by the sink. If a node receives a DIO message that contains its own ID in the UNS field it will blacklist its parent locally to make sure that it will not become a parent again. The other method, the rank verification, works by making sure that all the nodes in the network are only allowed to lower their announced rank in the routing graph by one unit of routing metric (where a lower rank is better) within a reasonable time. This is achieved by the usage of cryptographic hash values and verification of hash values that are present in the routing advertisement messages.

3.3 Combined IDSs

The related work that has been mentioned up until here only focus on the detection of one kind of attack. This is the case since the nature of many sensor nodes (with their limited resources) often leave little capacity left after the main functionality of the node has been implemented. This thesis takes advantage of modern sensor nodes that are not as limited as the sensor nodes that have been used in previous research and this thesis can therefore focus on detecting several attacks, as described in Section 1.3. However, previous work exist that cover more than just one kind of attack but often with the assumption that the sink node has more resources than the rest of the nodes. With a more powerful sink node the system is therefore able to perform the more heavy parts of the intrusion detection. This section presents and focus on related work that combines different detection methods so that the IDS covers more than one type of attack.

3.3.1 SVELTE: Real-time intrusion detection in the Internet of Things

Raza *et al.* [6] present an IDS for sensor nodes called SVELTE. While SVELTE does detect more than just one kind of attack, it focuses on different kinds of routing attacks, leaving detection of other attacks as a possible extension of the work. SVELTE has a combination of both centralized and distributed architecture over the WSN. The design and implementation of the IDS expects the sink node in the network to be more powerful than the other nodes, and therefore the heavier parts of the IDS are placed in the sink. In SVELTE, the sink has three different modules. The first one is called 6Mapper (6LoWPAN Mapper). This module keeps track of how the RPL routing graph is formed and keeps it as a map in the sink node. The second module uses the information gathered from the 6Mapper in order to detect attacks on the network while the last component is a small implementation off a distributed firewall. By keeping a firewall in the sink malicious traffic can be filtered out before it enters the WSN. In order for the centralized modules to work, all the nodes in the network must also implement a more lightweight sub-module of two of the modules. They all have a module that provides the 6Mapper with routing information so that the sink node can maintain the routing graph correctly. In addition to the 6Mapper component the other nodes also keep a small part of the firewall implementation. Additionally, all the nodes also keep a module for handling end-to-end packet loss.

Evaluation of SVELTE is carried out in the simulator Cooja. The authors choose to evaluate the IDS in two different modes, one mode with packet losses and another mode without any packet losses. In the case with no packet losses the detection rate is almost 100% and no false positives are detected. When the simulated network is allowed to lose packets, the detection rate drops to approximately 90% in the setup used, and the authors note that in a bigger network setup the detection rate decreases even further. After a while, when the 6Mapper gets the time to collect all the data about the network the detection rate will start to rise again as the routing information propagates to the 6Mapper.

3.3.2 Intrusion detection for routing attacks in sensor networks

Loo *et al.* [26] present an IDS that is designed to detect a number of different routing attacks. The authors construct a method that detects abnormal behavior in the network by using a clustering algorithm that builds a model of how normal traffic in the system looks like. This model does not require the nodes to communicate with each other, which is an advantage since the power consumption of the nodes will be smaller. Instead, each node contains its own IDS that will be able to detect attacks on its own, using the routing table information on the node. The authors present twelve different attacks that the system is able to detect, where nine of them are different types of Sinkhole attacks, and the other three are focusing on different Denial of Service attacks. It is important to note that all the testing of the method was carried out in a simulator. The number of false positives depends on the cluster sizes, which can be set in the program. The authors conclude that the system achieve the most reasonable trade-off between the detection rate and the number of false positives when there is a false positive rate of around 5%, so the cluster size w is adjusted for this value when evaluating the system. How much the detection rate for the method varies depends on which attack that is detected, but it varies from 100% (for active sinkhole attacks) to 70% (for passive sinkhole attacks) when w is set according to the trade-off above.

3.4 Our Contribution

The works that this chapter summarizes are similar to the work in this thesis. The biggest difference regarding this thesis is that the detection methods are written for modern hardware, allowing more advanced algorithms and it also allows the code to be larger and contain multiple detection methods at the same time. All of the works this chapter discusses are evaluated completely in simulators while this thesis also takes advantage of real hardware when measuring the detection rates and energy consumption of the already tuned IDS for the first detection method, and the energy consumption for the second detection method. Also, many of the works are detecting one specific type of attack, while this thesis combines two different methods to detect both DoS-attacks and routing attacks. However, other combined IDSs do exist, as presented in Section 3.3. Finally, this thesis builds a framework for different detection methods which makes it easy to extend the IDS with other methods if desired. The framework also makes it possible to easily choose to activate or deactivate one or more detection methods.

4
System Model and Design

This chapter describes the design of the system model of the intrusion detection system framework presented within the thesis. The chapter aims to describe the goals and requirements of the implementation, and thus acts as a template that is realized during the process of implementation. First, the chapter describes the general system overview in a subsection with focus on requirements that need to be fulfilled by the outer parts of the framework. Then, the chapter describes the detection methods in subsections of their own. The subsections for the detection methods focus on requirements on the methods themselves and propose algorithms that solves the problems. Both of the subsections for the detection methods include an adversary model that concretizes what the detection method detects or protects against.

4.1 System Overview

The IDS framework consists of many different components. The design of the IDS framework provides components that are not only security related in terms of algorithms that detects attacks, but also components that runs normal tasks like reading and sending sensor values. One of these components can be described as an outer shell of the framework that acts as a wrapper for the whole application. Another component is an example program that simulates the normal activities of a sensor node in terms of collecting sensor values and sending these over the network. The outer shell component acts as a common ground for the framework and is required to support the activation of different detection methods and turning the example program on and off. Subsections to this section further describes the requirements of the outer shell component and the example program component.

In addition to the outer shell and the example program, the framework includes two detection method components. The detection methods focus on different attacks and adversary models. The first detection method focuses on detection of Denial-of-Service attacks by usage of energy consumption as a metric. The second detection method focus on both detection of and protection against sinkhole attacks (including other routing attacks against RPL) by usage of a strict hierarchy and a verification process of nodes that are responsible for routing and forwarding of messages. Both the receiver side and the sender side of the node verifies that the node behaves as it should and does not attack the network. The remedy includes blacklisting and exclusion of nodes that attack the network.

Figure 4.1 displays an overview of the IDS framework. It can be seen that the implementation of the framework done on top of Contiki OS and that it consists of four different components where three of them are possible to turn on or off.

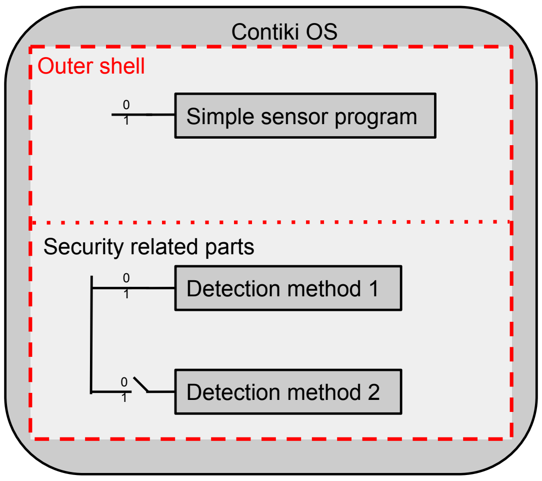


Figure 4.1: An overview of the system model is displayed. The framework consists of an outer shell with support for turning the internal components on or off.

4.1.1 Outer Shell - Requirements

The basic shell of the framework requires support for combinations of different detection methods, where the different combinations can be set by using a parameter called "security level". The chosen security level leads to different effects regarding which detection methods to use. Level 0 corresponds to just the shell itself, level 1 corresponds to outer shell and detection method 1, level 2 corresponds to outer shell and the second detection method, and so forth.

4.1.2 Simple Sensor Program - Requirements

In order to simulate the normal activities of a WSN node a simple program that takes advantage of the sensors on the platform needs to be implemented. The only requirements are that it supports fetching of real sensor values and periodic communication of the sensor values to the sink-node of the wireless sensor network. Normal behaviour is important to simulate since some detection methods rely on the distinction between what is normal and what is abnormal behaviour.

4.2 Detection Method 1: Anomaly-based DoS-Attack Detection

The first detection method that this thesis proposes is denoted Anomaly-based DoS-Attack Detection. It uses the solution that Riecker *et al.* [4] propose as a basis, where energy consumption is used as a metric to determine if a node is under attack. This section describes the requirements of the detection method and presents the adversary model that the method assumes. The adversary model acts as the basis for an attack that is implemented in order to test the detection method.

4.2.1 Requirements

The first detection method focuses on detection of generic Denial of Service attacks. It is based on energy consumption of the radio communication hardware of the node. The reason for this is that any type of attack that targets exhaustion of a node's resources via use of flooding of the radio communication of the node will have an impact of the energy consumption of the radio hardware. The method reads energy consumption of the node and store values of how the energy consumption varies in time in a suitable data structure. An algorithm analyzes the collected values in order to detect attacks. Another requirement of the detection method is that the placement of it needs to be locally on the nodes. The main motivation for this is the improvement in hardware. There are fewer restrictions on the memory footprint of the IDS framework on the sensor nodes that this thesis uses, hence there will be space available for local detection methods.

If the energy consumption of the node is even and somewhat linear, the detection method can take advantage of linear regression and the history of the energy consumption in order to predict what the next-coming energy consumption value should be. Section 4.2.1.1 further elaborates the linear regression-based DoS detection method. Because of uncertainty regarding the power consumption profile of a sensor node, design of an alternative approach that avoids the assumption that the sensor node must have a linear power consumption is also done. Section 4.2.1.2 further describes the alternative approach that avoids assumptions regarding energy consumption profile.

4.2.1.1 Linear Regression Algorithm: LiReg

The linear regression algorithm which this thesis propose is called LiReg. It takes advantage of predictions of the next-coming energy consumption value with the help of the linear regression technique as Section 2.8 describes. To enable this, a data structure needs to store energy values. This enables a history of the node's energy consumption. The algorithm calculates the difference between the next-coming energy value and the predicted value. The difference is then divided with the average energy consumption value for the collected values which gives how many percent larger (or smaller) the difference is in comparison to the average consumption value. The design choice to compare the difference with the average value means that sudden increases in the energy consumption that are unusual in comparison to the history of the node will get noticed. A solution that only compares the difference with the present value or the previous value would be much more sensitive to disturbances between rounds of the program. If the resulting percentage from dividing the difference with the average value is larger than a given threshold the node alerts that it might be the target of a Denial of Service attack. This is a similar approach as in the work by Riecker *et al.* [4]. Algorithm 1 shows the described algorithm. Looking at the pseudo-code, it can be noted that three parameters are required. The three parameters include one threshold limit denoted T that controls how many percent the difference between the predicted value and the actual value is allowed to differ from the average value among the collected energy consumptions. For example, if the value is set at 100, it means that the difference is allowed to be 100 percent larger (or smaller) than what the average consumption value is. The second parameter controls how many elements that are required to be present in the history before linear regression can be performed. This parameter is denoted K, and a last parameter denoted R which controls how often the energy consumptions is read.

Algorithm 1 LiReg

```
Require: $thresholdpercent : T$
Require: $Least\_number\_of\_elemenents\_to\_perform\_linear\_regression : K$
Require: $read\_time\_interval : R$
1: while $Forever$ do
2:     wait($R$)
3:     $energy\_val$ ←read_energy_for_last_period()
4:     insert($energy\_list, energy\_val$)
5:     if length($energy\_list$) $\geq K$ then
6:         if $expected\_val$ is present then
7:             $difference \leftarrow energy\_val - expected\_val$
8:             if $100 \cdot \frac{difference}{energy\_list\_average} > T$ then
9:                 send_alarm()
10:            end if
11:        end if
12:        $expected\_val$ ←linear_regression()
13:    end if
14: end while
```

The linear regression algorithm requires that the energy consumption is somewhat linear. If the behaviour of a system can follow a straight line it is easy to detect outliers by just marking all abnormal values that differs too much from the expected value as an attack. However, if the energy consumption in a system varies a lot because of other behaviors, this can be a problem.

4.2.1.2 Increasing Average Algorithm: IncA

Riecker *et al.* [4] use a simulator to evaluate the implementation of the system design. In such an environment the energy consumption can be expected to be somewhat linear and no strange variances in the energy consumption appears, so linear regression works fine. In this thesis, the methods run on real hardware, so it is hard to predict what kind of energy consumption variations the nodes have.

It could be linear with just a few outliers, in which case the linear regression approach will work well. But, it could also be more diverse, like in Figure 4.2. Real sensor nodes are expected to not have a completely linear energy consumption profile for reasons such as radio announcement messages. As an example, even if the nodes do not run an application that uses the radio a lot, nodes periodically sends beacon messages announcing their presence to the other nodes in the network. These periodic and sometimes unforeseen radio activities cause the radio energy consumption to increase. It will lead to "spikes" in the energy readings and it could cause the linear regression algorithm to predict values that are either much higher or much lower than what the actual values will prove to be. This could lead the node to send a false alarm of an ongoing attack.

Note that Figure 4.2 is just an illustration of how the energy consumption *could* look. In the figure the units are not of importance since it just serves to give visual guidance for the argumentation. On the y-axis we have the unit "real time ticks in the radio hardware" and on the x-axis we have time intervals, where one time interval is 10 seconds and an energy reading is taken at every interval.

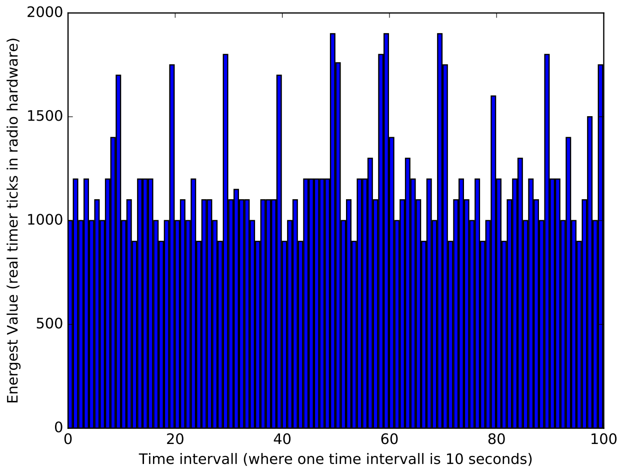


Figure 4.2: Example of how the energy consumption of a node could look like during normal execution. In this example, an energy reading is done every tenth second.

As mentioned earlier, the approach with linear regression as Riecker *et al.* [4] describe has only been tested in a simulator and might thus not be suitable in real-world scenarios without greater changes. As a result of the uncertainty about how well the linear regression approach fits on real hardware, an alternative approach needs to be implemented that avoids the assumption that the power consumption is linear.

The alternative method that this thesis proposes is called IncA. It collects and compares how the average energy consumption varies for the node's history of energy consumption values. The method calculates an average over a specified history size that can be changed as a parameter in order to find the optimal value for history size. If the computed average values increase for a specified consecutive number of rounds the node alerts that it is under attack. By counting the number of rounds that the average value increases, and combining it with a limit for when the node starts comparing average values, the assumption that a node must have linear power consumption is avoided. Algorithm 2 shows the proposed algorithm. In the pseudo-code, it can be noted that three parameters are required. One parameter describes how many consecutive rounds that the average energy consumption value is allowed to increase, denoted I, one describes how many rounds that the algorithm waits before starting to compare average-values, denoted N, and the last parameter describes how often energy values will be read, denoted R. By studying Algorithm 2 one can note that if the average value decreases between two rounds, the counter is reset to value zero.

Algorithm 2 IncA

```
Require: $increasing\_avg\_limit : I$
Require: $rounds\_before\_activation : N$
Require: $read\_time\_interval : R$
 1: $round \leftarrow 0$
 2: $increasing\_avg\_counter \leftarrow 0$
 3: while $Forever$ do
 4:     $round \leftarrow round + 1$
 5:     wait($R$)
 6:     $energy\_val \leftarrow$read_energy_for_last_period()
 7:     insert($energy\_list, energy\_val$)
 8:     $average\_prev \leftarrow average$
 9:     $average \leftarrow$average_list_value()
10:     if $round > N$ then
11:         if $average > average\_prev$ then
12:             $increasing\_avg\_counter \leftarrow increasing\_avg\_counter + 1$
13:             if $increasing\_avg\_counter > I$ then
14:                 send_alarm()
15:             end if
16:         else
17:             $increasing\_avg\_counter \leftarrow 0$
18:         end if
19:     end if
20: end while
```

4.2.2 Adversary Model

There is a possibility that an attack launches from a node within the network itself, and not necessarily from the outside through a border router. Such an attack would avoid eventual filtering at the border router by means of firewall usage or other defense mechanisms. This means that in this adversary model the adversary is considered to be spatially close. The adversary can disrupt the traffic in a WSN by just bringing a node programmed with an application designed to flood the network and launch the application when he is geographically close. Many nodes announce their presence regularly, which means that the new node could potentially be able to join the routing graph and then proceed to flood the network by sending packets at a high frequency.

The adversary model uses an aggressive strategy. It expects that the adversary sends a lot of high intensity traffic to all the nodes it can reach. The motivation for using such an aggressive strategy is that it is likely that an attacker would try to disrupt the network using an aggressive strategy for exhaustion rather than by trying to disrupt the network by flooding the network slowly. There are two main goals for launching this type of attack. The first one is to disrupt the normal activity on the network since the nodes will be busy handling all the incoming packets. The other goal is that the attacker wants to drain the batteries on the sensor nodes, causing the network to go down and stay down after the attack until the batteries can be replaced.

4.3 Detection Method 2: RoVer

The works of Cervantes *et al.* [5] is an inspiration for the second detection method. They propose the use of a hierarchical distributed architecture where nodes keep track of parents of clusters and intermediary nodes between clusters. The use of a hierarchical approach enables the authors to detect sinkhole attacks in the network with a high accuracy. Therefore, this section describes the design and the requirements of a detection method called RoVer which stands for Role-based Verification for securing RPL routing by verifying claims from two sides. The method is based on a hierarchical structure of the network. The method's focus is both on detection and prevention of routing attacks against 6LoWPAN and the routing protocol RPL. The method aims to detect sinkhole attacks, wormhole attacks and selective forwarding attacks. The last subsection presents the adversary model that the detection method assumes and protects against.

4.3.1 Requirements

Focus on both detection and mitigation of routing attacks against WSNs that are communicating using the 6LoWPAN and RPL protocols is a requirement for the second detection method. The approach must function without the usage of RPL for communication between the nodes since the nodes cannot send traffic through a node that potentially does not forward all packets.

4.3.1.1 Roles and Responsibilities

In the hierarchical structure of the network there are three roles present in addition to the sink node, namely child, parent and grandparent. The three roles make the node behave differently depending on role and situation and thus require to have respective sets of operations. A child that is only a child is a leaf in the network graph, i.e. a node that only connects to a parent. A parent is a node that connects to its children and either the sink or another parent that is responsible for forwarding messages to the sink. If the parent connects to another parent the node acts both as a child (to the other parent) and a parent. Parents are nodes that are responsible for forwarding messages from children towards the sink. Grandparents are parent nodes that have at least one child who also acts as a parent and thus have its own set of children. Figure 4.3 displays the hierarchy of the system architecture and its corresponding roles.

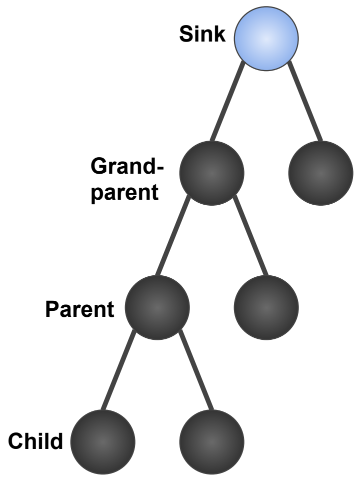


Figure 4.3: The hierarchy of the network is displayed and the three roles (child, parent and grandparent) a node can have are displayed. A node can have several roles in the network; Parents who have children which are parents themselves are actually grandparents and grandparents can also be children in the network.

The different roles have different sets of responsibilities and operations. Children must exchange messages with their siblings about how many packets they have sent to the parent. A requirement on these messages are that they need to reach all the available siblings and must not go via the parent since the parent could potentially be malicious. Children are responsible for the process of monitoring the parent. A requirement on parents are that they need to broadcast how many packets they have sent to the next node in the route to the sink. The same message must be sent via broadcast to both the children and the grandparent of the node (with the message content of how many packets the parent has sent to its own parent, i.e. the grandparent). If a parent has a child that also has children then that parent is also a grandparent and needs to keep track of the child that has children. This way, both the children and the grandparent verify the parents behavior. Children and grandparents verify a parent from two sides. Children verify that the parent's claim of how many packets it has sent is at least as many as all the siblings have sent to the parent. For the grandparent, it verifies that the parent's claim is actually how many packets the grandparent have received from the node.

The different roles of the system can be combined. For example, a parent can also be a child that communicates with its own siblings and verifies the respective parent. In similar fashion, a grandparent could also potentially play the role of a child but is always guaranteed to be a parent of another parent and thus the name of the role in the system.

4.3.1.2 When an Attack is Detected

When the method detects an attack the network tree must be restructured in order to remove the malicious node. The malicious node will not be permitted to become a parent again. Blacklisting of a malicious node must be carried out by the same means as the communication among children, i.e. via communication that avoids using RPL. For example by usage of broadcasts and replaying of broadcasts. This way of communicating the blacklist message helps to avoid situations where some part of the network still believe that the malicious node is a good parent. The network allows a malicious node to be part of the network in the role of a child i.e. as a sender of data, nothing else. If the method detects a malicious node, every other node in the system ignores future routing announcement messages from that node. Thus, the node that performs the attack is denied from ever becoming a parent or a grandparent again since no other node will ever use the evil node as a router.

4.3.1.3 Tolerance of Packet Loss

The design and implementation of the detection method are required to have a tolerance of packet loss since it occurs naturally on this type of network. The tolerance of packet loss is part of the verification of the nodes that are responsible for forwarding messages in the network. It uses hard limits for maximum number of allowed packets to be lost when the total amount of messages is below a certain threshold. If the total number of messages is above the threshold, a soft limit is used instead which indicates how many percent of the traffic that are expected to arrive at the receiver and then routed further on. The proposed strategy for packet loss is shown in Algorithm 3.

Algorithm 3 RoVer: Fault tolerance for loss of messages

```
Require: threshold $T$
Require: numeric below threshold $N$
Require: percent value above threshold $P$
 1: (In parent verification function:)
 2: if messages by self and siblings $\geq T$ then
 3:     if messages by self and siblings $-N \geq$ claim by parent then
 4:         Flag parent node as malicious
 5:     end if
 6: else
 7:     if messages by self and siblings $*P \geq$ claim by parent then
 8:         Flag parent node as malicious
 9:     end if
10: end if
11: (In child verification function:)
12: if claim by child $\geq T$ then
13:     if claim by child $-N \geq$ messages received by child then
14:         Flag child node as malicious
15:     end if
16: else
17:     if claim by child $*P \geq$ messages received by child then
18:         Flag child node as malicious
19:     end if
20: end if
```

4.3.1.4 Algorithm with Four Phases

An algorithm with four phases realizes the sections above. The four phases are a child broadcast-phase, a parent broadcast-phase, a phase for verification of data, and a phase for blacklisting malicious nodes and distribution of the blacklist. Algorithm 4 shows the described algorithm. Looking at the algorithm, it can be seen that the four-phase system can be modeled and implemented using a finite state machine-approach where transitions between states are controlled by timers. Figure 4.4 displays the four different states of the system and the transitions between the states.

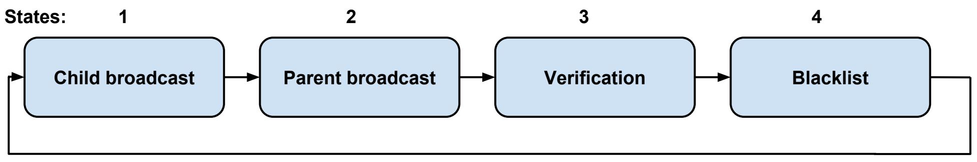


Figure 4.4: The four different states of detection method 2 are shown. Each transition between states are controlled via timers.

4.3.1.5 Fault Tolerance for Control Messages

In addition to the fault tolerance of accepting normal packet losses, the system also requires the implementation to be fault tolerant for the control messages of the detection method. That is, if a node does not receive a broadcast from a parent (whether it is the child of the parent or if it is the parent of the parent) it will not mean automatic flagging of the node as malicious. The motivation for this design choice is the uncertainty involved in broadcasting messages. In normal cases, the sending node signals to the receiving node that it is about to receive a message over the network by performing a process that can be described as flashing a light until the receiver wakes up. For cases where broadcasts are involved, the sending and receiving can be classified as a best effort process. When sending a broadcasts no information about it beforehand is sent and thus it can not be expected to be received by all intended recipients every time. To overcome this obstacle the design of the system takes advantage of fault tolerance when verifying other nodes as can be seen in Algorithm 5.

4.3.2 Adversary Model

The attacks that the second detection method protects against are originating from a node within the network. This is the case since the malicious node needs to be part of the routing graph in order to alter routing information. However, a remote adversary could be possible if a node on the local network is compromised and used as a so called "bot". Just as in the adversary model for the first detection method, a geographically close adversary could bring a node that connects to the network via the routing protocol if it is open for connections.

The attacking node needs to announce a spoofed routing metric, or rank, to the rest of the nodes in order to receive as much traffic as possible to be routed through it. This can be done by fabricating fake information and telling the rest of the network

Algorithm 4 RoVer

```
 1: while $Forever$ do
 2:     wait for phase 1 timer
 3:     if $isChild$ then
 4:         broadcast number of messages sent to parent
 5:         receive messages from siblings
 6:         re-broadcast messages received from siblings once
 7:     end if
 8:     wait for phase 2 timer
 9:     if $isParent$ then
10:         broadcast number of messages sent to parent or sink
11:     end if
12:     if $isChild$ then
13:         receive claim from parent
14:     end if
15:     if $isGrandParent$ then
16:         receive claim from child who is parent
17:     end if
18:     wait for phase 3 timer
19:     if $isChild$ then
20:         $parent\_result \leftarrow$verify($parent$)
21:     end if
22:     if $isGrandParent$ then
23:         $child\_result \leftarrow$verify($child\_who\_is\_also\_a\_parent$)
24:     end if
25:     wait for phase 4 timer
26:     if $parent\_result == evil$ then
27:         add_to_blacklist($parent$)
28:         broadcast($blacklist$)
29:     end if
30:     if $child\_result == evil$ then
31:         add_to_blacklist($child$)
32:         broadcast($blacklist$)
33:     end if
34:     if blacklist update received then
35:         re-broadcast blacklist once
36:     end if
37: end while
```

Algorithm 5 RoVer: Fault tolerance for verification

```
Require: max number of missed broadcasts $N$
1: (In verification function:)
2: if Parent broadcast has not been received then
3:     increase *counter*
4:     if *counter* $\geq N$ then
5:         Flag node as malicious
6:     end if
7: else
8:     Proceed to verify node
9: end if
```

that it has the best metrics for routing the packets in the RPL routing messages. In the case of a sinkhole attack, once the RPL graph is formed and the nodes starts to send packets the adversary starts to drop all the incoming traffic, i.e. it does not forward any packets. The malicious node can perform the dropping process by looking at the destination address of all packets it receives. If the adversary receives any packet not destined for the node itself it will do nothing instead of routing it towards the final destination, thus swallowing the traffic. In the case of a malicious node performing a selective forwarding attack, random messages, a specific type of messages or all messages from a specific node can be dropped. This can be realized by dropping packets according to a timer that is set with a random interval, looking at the message type or looking at the source address of the message Lastly, for the case of the adversary launching a wormhole attack, the received messages can be sent to another node than the actual parent of the malicious node.

5 Implementation

This chapter presents the implementation and realization of the framework. The chapter also describes the different implementation decisions of the components and motivates them with respect to the requirements of the system model as Chapter 4 mentions. Each component in the framework has its own section in this chapter. In relation to the implementation of the component, tests are conducted to verify the functionality of the component. More intensive testing and evaluation of detection methods are carried out in the chapter for evaluation, Chapter 6.

5.1 Outer Shell

The implementation of the basic shell of the framework supports different detection methods by taking advantage of a predefined value for which method or methods to run. Since distributed agreement protocols are out of the scope for this thesis the choice of pre-defining which detection method(s) that should be run is motivated. Another motivation is that since a system probably uses the detection method(s) it needs depending on the application of the sensor node it can be assumed that the node is programmed for the specific application area beforehand. The parameter that defines which detection methods that should be run can be set at the same time as the node gets programmed in order to also configure it security wise. Depending on the value of the parameter, different processes are placed in the auto start-command of the operating system. The different auto-start definitions are possible with the help of conditional groups and C-preprocessors directives. The system can add more detection methods easily due to the nature of the implementation. The basic shell and its corresponding compilation files are placed in a dedicated location while all the detection methods are placed inside another location which is dedicated as a code-base for the framework.

5.2 Simple Sensor Program

The simple sensor program is built from an example program for the chosen platform that was present as an example in Contiki. The program has support for storing sensor values and network support to send the stored values to a sink node. The program reads values from five out of the ten sensors that are present on the node and stores them in a list structure which is designed to only contain the latest sensor readings. Periodically the sensor values are sent to a sink with the help of added network support via IPv6, UDP and an application called `servreg-hack` which registers and announces network services in the WSN. `Servreg-hack` is a built-in application within Contiki used to register an IP-address with an 8-bit service ID which can be used to perform lookups in the WSN for a specific node running a specific service. The specific node is the sink node in the network and the specific service is a simple service to receive sensor values sent over the network.

The simple sensor program is programmed in such a way that it never sends the same value twice. It just sends the latest values read when all the values have been read once again from the sensors (although the value read can be the same). The motivation for this decision is that careful consideration must be taken in order to keep energy consumption down when developing applications for WSNs. The send interval consists of a static part and a random part that is dependent on and smaller than the static part. The send interval is configured in this way to ensure randomization so that multiple nodes do not send at the same time. The static part of the send interval is adjustable to allow for experimentation with more or less frequent network communication.

5.3 Anomaly-based DoS-Attack Detection

The implementation of the first detection method was done locally on the node as specified by the requirements in Section 4.2. Its core idea is to detect generic Denial of Service attacks against wireless sensor nodes with the use of measurements of energy consumption as a metric. The detection method consists of the following components: a process that fetches energy consumption values and inserts these into a list data structure defined in Contiki, an algorithm to determine if the node is under attack combined with a network component to inform the sink-node that the node running the IDS is under attack. The following subsections further describes the components and their important parameters. The subsections also presents two different methods that determines what a normal state is and what an attack state looks like.

5.3.1 Collecting Energy Values

Values for energy consumption can be fetched in Contiki via the activation of a module called `Energest`. `Energest` is a module for software-based online energy estimation of the sensor nodes and is entirely written in software and incorporated into the operating system [27]. The module works by having two lines of code in each hardware driver, one that produces a real-time clock timestamp when the hardware component gets activated and one that produces another real-time clock timestamp when the hardware component gets turned off. The `Energest` module keeps track of the timestamps and is thus able to tell how much time has been spent in a given component. With the help of these time stamp values combined with voltage and current specifications for the specific component, an energy consumption for the component can be approximated.

In the implementation of the first detection method, the `Energest` module gets polled every tenth second as default. The values polled are the time values of the real-time clock that has been spent in radio receive and radio transmit. It is important to mention that the time period for fetching energy values is one of the parameters that can be varied and is evaluated for the different detection strategies. The parameter is called ”Read Interval”. The values for time spent in radio listen and radio transmit are added together into a combined radio energy metric and, since the return value of the module is the total amount of time spent in the components, the old total energy time value gets subtracted from the new value to form the energy consumption metric for the last time period. The energy consumption values are kept in a list to enable a history of energy metric values which enables statistical analysis of the energy consumption. The length of the list, i.e. how many historical elements to store, is the second parameter that can be varied and might have an impact on performance since the list has to be looped through in order to perform linear regression or the average method. The name of this parameter is ”Max Readings”.

5.3.2 Detection with LiReg

Section 4.2.1.1 presents the algorithm called LiReg, which is contained in Algorithm 1. The algorithm describes how the values from linear regression are used to determine if an attack is taking place. The function *linear_regression()* on line **12** uses the equations that Section 2.8 describes in order to perform linear regression on the collected x (time readings) and y (energy readings) values collected in the list. There are two important parameters for the algorithm that can be changed to achieve an optimal setting. The first one, that is present in both Equation 2.2 and 2.3 is called K in the equations. It denotes how many historical values that are required to be present in the list before the algorithm tries to predict what the next value should be. The parameter is called ”Least elements to perform Linear Regression” and it must be smaller than or equal to the maximum length of the list with historical values in order to fit. The second parameter is a parameter that defines how sensitive the algorithm should be for differences between the actual energy value measured and the expected value, denoted y in equation 2.1. The name of this parameter is ”Sensitivity Level” and it is given in how big the number of percent larger or smaller the difference is allowed to be in comparison to the average energy consumption value. The second parameter is not present in the formulas, but set depending on how the nodes behave. It is used in the implementation to check if the difference between the expected and actual value is greater than the sensitivity level multiplied by the average value. Or, if the expression is rephrased, if the difference divided by the average energy consumption value is larger than the percents given by the sensitivity level as specified in Equation 5.1, if so, then the node informs both readers of the serial output and the sink node about it being under attack.

$$100 \cdot \frac{Difference}{Average} > Sensitivity\ Level \tag{5.1}$$

5.3.3 Detection with IncA

Section 4.2.1.2 describes the algorithm called IncA, which is contained in Algorithm 2. The algorithm describes how the average energy consumption values of the historical energy values of the node are used to determine if an attack is taking place. Important parameters for the average algorithm include two parameters as well. The first parameter is for controlling how many rounds of reading and storing historical values that needs to be processed before the algorithm starts to compare the average values. The parameter is denoted ”Least Rounds to Track Average” and it is used to control how many rounds that are needed to be processed as normal rounds before the algorithm tries to find differences. It is needed since sensor nodes have an initial high energy consumption when programs are started and announcement messages are sent to the network.

The second parameter is denoted ”Sensitivity Level” and is used to control how many consecutive algorithm rounds that the average value is allowed to increase (compared to the last average value produced). In the implementation the second parameter is used to control the sending of alarms to both serial output and the sink node. If the average value has increased for ”Sensitivity Level” consecutive rounds, the node produces an alarm. If it has not increased, the counter is reset to zero.

5.4 RoVer

The second detection method called RoVer is focusing on detection and mitigation of routing attacks against the routing protocol RPL which is used in combination with the IPv6 and 6LoWPAN protocols by sensor nodes that are running Contiki. The implementation is done using the system model, algorithms and requirements as described in section 4.3. The attacks which the system is designed to detect are sinkhole attacks, wormhole attacks and selective forwarding attacks. A common denominator for these attacks is that they disrupt normal routing and make packets from other nodes either disappear partly, totally or take other unexpected paths (as in the case of a wormhole attack). To detect such types of attacks the detection method is designed to make nodes monitor each other to detect anomalies in the number of forwarded or sent messages.

To aid in the detection of the aforementioned routing attacks the second implementation of the detection method is using the model of a distributed system built upon the use of three different roles (as specified in Section 4.3.1), namely children, parents and grandparents. The roles in the system can be combined, i.e. a node can be both a child, a parent and a grandparent at the same time but not in relation to the same set of nodes. The roles describe how a specific node is in relation to other nodes in the nearby area. If a node acts as a parent to another node, then the parent node is responsible for forwarding the child’s messages towards the sink. Children that have the same parent are siblings and they need to exchange information with each other when determining how many messages they have sent to their common parent.

The system uses verification of how many messages a node has actually forwarded in comparison to how many messages the node claims that it has forwarded to function as intended. If a node a acts as a parent for two nodes b and c, and at the same time is the child of another node d that acts as both a parent and a grandparent in the system, then the node a can be monitored from two sides by both the children (nodes b and c) and the grandparent denoted d. The verification process is carried out by letting the children and the grandparent verify a claim made by the parent-node a towards both the children and the grandparent. The children or the grandparent will discover an erroneous or fake claim since such a claim cannot be made to satisfy both sides.

For the verification of parents and children to function properly, switching of parents cannot be allowed between some of the phases of the program. The phases where statistical data is gathered, the phases where claims and announcements are being made and finally the phase where the data is verified require a node to have a static parent for the verification process to function properly. Otherwise, the method could erroneously flag nodes as evil if a child switches parent and expects the new parent to have forwarded all the messages during the round of the program. A more complex solution could be implemented that would require more memory (by storing unique information for each recipient of data) but in the end it is a trade-off between memory usage and mobility where memory usage is being prioritized in the implementation. Thus, the implementation disallows the switching of parents during the first three phases of the program and it can be concluded that the design and implementation of the system is not built with mobility in mind.

The rest of this section and its corresponding subsections describe the implementation in more detail. Starting with a description of the implementation of the four phases that were required in the system model, and then continuing with a description of how information is spread through the network without usage of the routing protocol RPL. The following subsection also cover topics related to how storage space is spared in the implementation, a simple algorithm for determining what role a node has in the system, details about which messages that are counted, a description of how the verification process is implemented and lastly details about how the blacklisting, i.e. the actual remedy when an attack has been discovered, is implemented.

5.4.1 Four Phases

The overview of the system design for the second detection method describes a system model consisting of four phases that loop continuously while the detection method is active. The phases are called child broadcast, parent broadcast, verification of data, and blacklist. Each phase consists of different responsibilities for nodes depending on their roles in the network. The outer main control loop that controls the shifting between the four phases is implemented with the help of timers and instructions that tell the process to wait until the timers generate an event that informs the program that the timer has expired. After expiration of the timer, the program can enter the next phase and execute the corresponding actions therein. The timer for each phase of the system is connected to an individual time period parameter. The length of the periods are controllable and shall be evaluated thoroughly since the lengths will be a key factor for how synchronized the system will need to be. The parameters are called ”Period 1”, ”Period 2”, ”Period 3” and ”Period 4”.
A brief description of the phases and how they are implemented follows in the next-coming subsections of this section.

5.4.1.1 Child Broadcast Phase

The first phase, known as the child broadcast phase, is where the children in the system exchange messages with the other children that have the same parent. The children are performing this so that the siblings will be able to agree upon how many messages they have sent to their common parent, and thus, how many messages the parent must have sent further on if the parent behaves as it should.

During the first phase, the siblings communicate by broadcasts in order to avoid the routing protocol RPL (which could be compromised). To ensure that all the siblings receive each other's messages each node re-broadcasts or replays a received message once. Implementation-wise, in the communication the nodes need an identifier for the parent and one identifier for the node itself in addition to the actual number of messages that have been sent to the parent. To conserve space on the nodes and to reduce energy consumption when communicating, the identifiers are constructed by taking the last 16 bits of the IPv6-address of the node and the parent which sums up to 4 bytes in total. 4 bytes is considerably smaller than sending two full length IPv6-addresses on 128-bits each (which would have summed up to 32 bytes). In addition to the four bytes for identifiers, there are two more bytes for sending a 16 bit unsigned integer that contains the number of messages sent to the parent. Thus, the message payload sums up to 6 bytes in total in the implementation of the child broadcast phase.

All nodes in the area close to a sending node will be able to hear a child broadcast message that a node sends. The system has been designed to ignore messages that does not contain the same parent identifier as the parent identifier for the actual parent of the node.

5.4.1.2 Parent Broadcast Phase

The second phase, called the parent broadcast phase, is when nodes that acts as parents in the system announce the number of messages they have sent to the next responsible node, i.e. to the parent's parent. A broadcast is sent which announces the number of messages. Both the children of the parent and the parent of the parent receive the same message and claim in the payload of the message. Implementation-wise, the program performs a check so that only parents send broadcasts during the second phase. The message that is sent is 4 bytes large, where 2 bytes comes from the identifier of the node (the last 16 bits of the IPv6-address) and the remaining 2 bytes represent an unsigned 16 bit integer which contains the number of messages that the parent has sent to its parent.

The children of a parent listen for an incoming parent broadcast and stores the claim made by the parent when it arrives. During the second phase, grandparents also listen for and store claims made by its children that also acts as parents in the system. At the same time, a grandparent also stores how many messages it has actually received from the given child node. The verification of the claims is left until the third phase of the program.

5.4.1.3 Verification Phase

During the third phase, denoted verification phase, no communication is taking place within the program. Instead, the nodes verify each other's activities and store the results. Children verify parents by comparing the number of messages they and their siblings have sent to their parent with the claim made by the parent during the previous phase. Parents verify grandparents in the same way (a parent has a child-parent relation to the grandparent). Grandparents verify its children that also act as parents by comparing the claims made by the children with the number of messages that the grandparents have actually received from the given nodes. The objective is to discover fake or erroneous claims made by parent nodes. The result from the verification, whether a parent or any children are misbehaving, is stored as status bits to be read during the fourth phase. Finally, the flag for allowing possible switching of parent is set before waiting for the next phase, i.e the system allows switching of parent as soon as the verification of the current parent has been carried out.

5.4.1.4 Blacklist Phase

In the fourth phase, called the blacklist phase, nodes check the results from the verification phase, i.e. if any misbehaving nodes have been discovered. If misbehaving nodes have been found, those nodes are added to a local blacklist which is implemented as a part of the RPL-module in Contiki. After the misbehaving nodes have been added to the local blacklist the nodes are sent as blacklist updates by broadcasting to the network. Any node that receives a blacklist update checks if the blacklisted node is already on the local blacklist. If the node is not already on the list, the node adds the blacklist update to its local blacklist and then re-broadcasts the message to allow the blacklist update to continue to be spread throughout the network.

5.4.1.5 Between Last and First Phase

Between the fourth and the first phase all statistical data is reset to prepare for a new round in the program. This include resetting all the siblings and children a node saves during the first phase, resetting the statistics gathered in the `TCP/IP-module` by calling a function implemented in the API and also reset the counters for number of messages received by siblings and the parent claim. After resetting all variables, the node disallows parent switching, checks if it is time to reset the blacklist by checking a timer and then it goes to sleep until it is time to start with the instructions of the first phase again. While the program is sleeping the node is collecting statistical data in the background on the number of messages sent and received.

5.4.2 Broadcasting and Re-broadcasting

One of the requirements in the system model of the second detection method is that the control messages that are sent as a part of the detection method are not to be sent with the help of the routing protocol RPL. The motivation for this requirement is that the there exists a risk that the communication over the protocol could be compromised in case of an attack, and that sending control messages over the routing protocol could break the functionality of the detection method. To overcome this problem, the implementation takes advantage of broadcasting, i.e. sending information to all nodes in the reachable nearby area. All the broadcasts in the system are realized by making the nodes perform a best effort IPv6 link-local multicast to the system. This means that all the nodes on the same local area network will listen to the multicast, hence it will be the same as performing a broadcast. The terms link-local multicast and broadcast are used interchangeably within the report since the resulting effects are the same. All broadcasts except for the re-broadcast (replaying of messages) are performed more than once to increase the chance of other nodes receiving the broadcasted message.

When a node receives a broadcast, the node can optionally re-broadcast the message if the message needs to spread further than to the closest neighbors of the source. Such a strategy allows for messages to spread throughout the network of nodes without using the routing protocol RPL for routing of messages. Additionally, the broadcasting strategy allows for nodes on two different sides of the sender to receive exactly the same message which makes it harder for an adversary to make false claims in the system. An erroneous claim that satisfies one side would dissatisfy the node on the other side and the node would get blacklisted for claiming false information.

5.4.3 Algorithm to Determine Role

All nodes in the system except for the sink-node are children in one way or another. Since the different roles in the system have different responsibilities and tasks to carry out, it is important to determine if a node acts as a parent or a grandparent. One way to discover what role a node has is by making the node listen on the child broadcasts during the first phase of the program. If any of the nodes in the nearby area performs a child-broadcast where the listening node's id is reported as the parent id in the payload of the broadcast, then the listening node knows that it must be a parent in the system. In the case of such a message, the parent node needs to save the id of the child so that the child can be identified later on. The id of all its children are saved in a list, named *children*. When receiving the message the node will first check if the child is already in the list; if not, it will add it. This information is saved in order for the parent node to later on to be able to verify a child that possibly also acts as a parent.

To determine if a node is a grandparent the node needs to listen for parent broadcasts. If one of the children of a parent perform a parent broadcast then it can be concluded that the parent node is in fact a grandparent and that the child of the grandparent is responsible for forwarding messages from other nodes in the system. In the case of the reception of such a parent broadcast from a child (from now on denoted child-parent) the grandparent needs to save two additional things next to the identity of the child-parent. The first is the child-parent's claim of how many messages the child-parent has sent to the grandparent, the second is how many messages the grandparent have actually received from the child-parent node. Since the child-parent is responsible for forwarding messages from other nodes it must also be verified in the system.

5.4.4 Message Statistics

The implementation of the message counting for the detection method is found in the `TCP/IP-module` of Contiki. The node stores both the messages sent and the messages received in two different variables. In order for the detection method to access the statistics in the `TCP/IP-module`, an extended `API` to access the statistical data added to the module is implemented. The extensions include functions for reading the number of messages sent, reading the number of messages received by a specific node, and also a function for resetting all the statistical components to allow for old data to be discarded.

One counter, implemented as a 16 bit integer, stores the number of messages sent from the node. For the case of received messages there needs to be an individual counter for each sender. This functionality makes it possible for a grandparent to be able to verify claims made by its children that acts as parents as well. In both cases, when counting the number of received messages and when counting the number of sent messages, the only messages that are counted are the messages that are destined beyond the nearest neighbor with the exception if the nearest neighbor is the sink. The implementation choice to only count those specific types of messages means that announcement messages and routing information messages are not counted. This behavior is correct as those messages are not supposed to reach further than the closest neighbors of the sending node. In practice it means that only the messages that are sent towards the sink are counted. Implementation-wise the methods perform a check that investigates whether the next-hop address for the routing of the packet to send is the same as the destination address of the packet. The node ignores all those messages except for the case when the next-hop in the routing is the sink. The sink itself requires special code since it does not forward any packets. In the implementation, the sink counts messages as they are received directly in the packet-input handler which has the drawback that also routing messages and announcements are counted as received messages by the sink.

5.4.5 Verification of Statistical Data

During the verification phase of the program, all the claims made by parent nodes needs to be verified and compared with the statistical data acquired for the number of messages that have been sent to or received by the parent nodes. It is the children of a parent that performs the verification. For the children the objective is to find out if the parents have sent at least as many messages as all the siblings in common have sent to the parent for forwarding further to the sink. Grandparents also perform verification of nodes that are their children that also act as parents. For grandparents the objective is to find out if the claim made by the child during the parent broadcast is the same as the number of messages that the grandparent has actually received from the child.

The program implements fault-tolerance in order to be able to handle packet loss (which are common in wireless sensor networks). To achieve fault tolerance there are two parameters for accepted packet loss. The first parameter is a numerical value and acts as the exact number of accepted packet loss when fewer than 10 messages have been sent or received during the round. For example, if 9 messages have been sent and the threshold is to accept 2 lost packets. Even if only 7 messages arrive it is still acceptable and not considered an attack. The second parameter is for when more than ten messages have been sent or received. The second parameter is given in percent of the packets that are expected to arrive. If for example, the parameter is set at 90 and fewer than 90 % of all sent packets arrive, it can be concluded that an attack is ongoing. The two parameters are denoted "Below Ten Limit" and "Over Ten Limit".

The following two formulas are used for verification of parents: If the total number of sent messages (by node and siblings) is greater than 10 Equation 5.2 is used. Multiplication with 100 on the left-hand side makes the number of messages comparable with the threshold multiplication on the right-hand side (where the threshold is given in percent).

$$Messages\ by\ Parent \cdot 100 \geq Total \cdot Over\ Ten\ Limit \tag{5.2}$$

If total number of sent messages is fewer than 10 Equation 5.3 is used,

$$Messages\ by\ Parent \geq Total - Below\ Ten\ Limit \tag{5.3}$$

where $Messages\ by\ Parent$ in both equations is representing the claim made by the parent. If the check in the formula is not valid, the node is flagged as evil.

The following two formulas are used for verification of children's claims (as claimed in their parent broadcasts): If the total number of received messages from the child is more than 10, Equation 5.4 is used. Multiplication with 100 is performed with the same reasoning as in the formulas above.

$$Child\ Claim \cdot Over\ Ten\ Limit > Child\ Received \cdot 100 \tag{5.4}$$

If the total number of received messages is fewer than 10, Equation 5.5 is used.

$$Child\ Claim - Below\ Ten\ Limit > Child\ Received \tag{5.5}$$

For the verification of children, only the children that are acting as parents are verified and only if the checks in the formulas are valid, the children are flagged as evil. (Note that the logic is inverted from the parent verification.)

5.4.6 Blacklisting Nodes

Based on the results from the verification phase, evil nodes can become blacklisted. If the method flags a node as malicious during the verification phase, the node that made the discovery first adds the malicious node to a blacklist and then broadcasts a blacklist update to all its neighbors twice. Nodes that receive a blacklist update adds the blacklisted node to a local blacklist and then re-broadcasts the blacklist update once. Upon reception of a blacklist update the sink node waits for a timer to expire (so that the blacklist update has time to spread) and then it performs a repair on the routing graph (DODAG) of the network by using a function included in the RPL-module in Contiki. The repair function effectively means that the routing graph is reconstructed without the malicious node being able to be a part of it.

The implementation of the blacklist is found in the RPL-module mentioned above. To be blacklisted effectively means that all nodes are ignoring DIO (DODAG Information Object) messages from the blacklisted node. Since all the nodes in the network are ignoring the DIO-messages the rank of the blacklisted node will not be known by any other node and therefore it will not be chosen as a parent by anyone. The blacklisted node is still allowed to send packets to the sink, but it will not have any children and is thus effectively blocked from performing any routing attacks. The blacklisting is not made permanently. It has a timer connected to it for resetting the blacklist so that nodes that are falsely flagged as evil will be able to join the network again. The parameter used to set the time before allowing a node back in the network is called ”Empty Blacklist Period” and the blacklist is emptied periodically using this parameter.

5.5 Implementation of Attacks

This section describes the attacks that were implemented in order to test the detection rate of the IDS. The attacks are following the specifications given in Sections 4.2.2 and 4.3.2 in order to meet the requirement to verify the functionality.

5.5.1 Generic DoS-Attack using IPv6 and UDP

One of the nodes in the WSN runs a program written to launch a DoS-attack. The node starts and does not begin sending packets directly. It just indicates that it is on by having the green led turned on. Once the button on the node is pressed, the green led is turned off and the red led is turned on as the node starts sending packets to all other sensor nodes that it can reach via IPv6 link-local multicast.

Since it is possible to sniff network traffic in a wireless network, it is feasible for an attacker to find out which services and ports that are being used. This information can be used by the attacker of the network and the adversary could start sending multicast packets to the existing services and open ports. In order to send the packets to all the nodes the packets are sent over UDP using multicast for IPv6, where the multicast address is the link-local multicast address for IPv6 as defined by IANA (Internet Assigned Numbers Authority) [28]. Since the nodes are complying to this standard, all nodes in the network have receiving of multicast traffic enabled by default, and hence they receive the sent packets (given that they are in range).

It is easy to change the parameters for the attacking node to simulate different intensity of the attacks. The implementation uses a timer that reads a value for the send interval to determine how often the packets is to be sent to the network. This send interval is defined in the code, and it is this parameter that controls the intensity of the attack. A shorter send interval means more packets and thus a heavier DoS-attack. The implementation sends packets with a small payload of just 4 bytes. It can be argued that larger packets demand more power to be handled by the nodes and thus provide a heavier attack energy and resource-wise. However, if the IDS can detect an attack that only sends small packets, which thus consumes less energy in the radio module, it should also be able to detect attacks that uses larger packets that give a larger energy consumption. This fact acts as a motivation of the choice of using small packets.

5.5.2 Routing Attacks

Three different routing attacks are implemented in order to test the second detection method. This section describes the implementation of a sinkhole attack, a selective-forwarding attack, and a wormhole attack.

5.5.2.1 Sinkhole Attack

The Sinkhole attack needs to fulfill two requirements. The first requirement it needs to fulfill is to announce a good rank to the rest of the network in order to get as many packets as possible routed through it. It also needs to drop the packets received, thus reducing the traffic that actually reaches the sink and by doing so interrupting the routing of the network. The implemented attack fulfills both of these requirements, in two different Contiki modules; the RPL-module and the TCP/IP-module. The implementation of the attacker node takes advantage of a parameter that when activated identifies that the node is a ”sinkhole attack”-node which means that it runs modified code of the aforementioned modules.

In the RPL-module, the attacking node will first announce that it has the lowest (and therefore best) rank in the routing graph (DODAG) by claiming that it has the same rank as the sink. At first, the attacking node will just continue to route the packets received towards the sink as a normal node, waiting for a timer to be set off. The timer goes off after a predefined delay that is given in seconds, and then the attacking node will start to drop all the packets it receives. It will however,continue to send its own packets to the rest of the system.

5.5.2.2 Selective-Forwarding Attack

The selective forwarding attack acts in a similar way as the sinkhole attack. It has its own parameter defining it as a ”selective forwarding”-node and runs modified code added to the source files of Contiki in order to launch the attack. All the implementation is done in the TCP/IP-module of Contiki. A selective forwarding attack can start off by announcing a false rank as the sinkhole does in order to draw in more traffic, however, this is not the case with this implementation. A selective forwarding attack can act very differently depending on what the attacker wants to achieve. In this case it is implemented by having the attacker node dropping all packets from a certain given node (instead of, for example, dropping a certain kind of messages like all the routing messages). In the TCP/IP-module it is defined which node that will be the ”target node”, i.e. the node whose packets the evil node will drop when the attack is launched. The IP-address and last bits of the IP-address, which serves as an identifier, for the target node is saved for later. Then the attacking node will check the source addresses of the packets it receives. If the identifier of source address of the packet is the same as the target node, it will not send it through to the rest of the network; it will just drop the packet.

5.5.2.3 Wormhole Attack

The wormhole attack is implemented similarly to the previously described attacks. The implementation has a parameter that can be set in order to run the modified code in the TCP/IP-module. The wormhole attack is implemented by sending all packets it receives to another pre-defined node instead of routing the messages in the correct way by sending it to the parent of the node. The attack is realized by changing the next-hop address of all messages that the attack node receives. By only changing the next-hop address in the messages, the routing in the network will be disrupted, but the message will eventually reach the sink unless the node that receives the message routes it further away from the final destination. However, there exist cases where it will not reach the sink. An example of this is when the wormhole-node sends all messages back to one of his children. The messages will then be stuck in a loop and never reach their destinations.

6 Evaluation

This chapter describes the evaluation of the IDS framework. The overall aim of the evaluation is first to set the parameters of the algorithms to suitable values and then to demonstrate that the framework functions as intended (i.e. detect attacks) while not being too demanding energy-wise. The chapter first presents an overview of the evaluation which also mentions the aim and the goals of the evaluation of the framework in more detail. The following sections present the evaluation methodology and the evaluation metrics used for evaluating the detection methods. Then, the chapter presents the results from the evaluation of both detection methods in separate sections.

6.1 Evaluation Overview and Goals

Evaluation of the IDS framework focuses on demonstrating that the detection methods work as expected. Therefore, the focus of the evaluation is on the individual methods and the evaluation of them. To aid the demonstration that the IDS functions as expected, recommended parameters for the individual methods need to be chosen. These parameters are not to be seen as the most optimal parameters but rather as guidelines on what parameters to chose in order to have functional detection and protection.

Goals for the evaluation include deciding upon recommended parameters through experiments and tests. With parameters chosen the next goal is to demonstrate that the detection methods work as expected, i.e. detecting attacks. Demonstration is assisted with the help of common metrics which make the results comparable with state of the art research papers in the field. The metrics that are considered are detection rate, time until detection, false alarm rate, energy consumption, latency and reliability. Section 6.3 further explains these metrics and when they apply.

When it comes to measurements of energy consumption, the goal is to compare having the IDS on and off in order to be able to see how large the impact is on the sensor nodes. For other metrics, the goal is to demonstrate that the IDS either detects or as in the case for the second detection method both detects and protects against routing attacks attacks which are launched against the network.

In order to achieve all the aforementioned goals, a scientific approach is needed that fulfills the question on how the evaluation goals are going to be achieved. Section 6.2 describes the methodology that is used to achieve the goals and also discusses the evaluation of the detection methods in more detail. The second detection method is primarily evaluated in the wireless sensor network simulator Cooja using sensor nodes that are has almost as much memory and computational power as the sensor nodes used within this thesis. The methodology section further discusses the subject in more detail.

For the first detection method, two different algorithms that detect DoS-attacks using energy consumption as a metric were designed and implemented. The goal of evaluating the two algorithms is to compare their results in order to see which of the two that performs the best according to the chosen metrics. The evaluation of the first detection method is found in Section 6.4 where both parameterization of the algorithms and evaluation results are presented.

The second detection method is evaluated according to more metrics since it not only detects attacks, it also presents a remedy for the attacks in terms of blacklisting malicious nodes. Section 6.5 presents the parameter choices, network setup and results from the evaluation of the second detection method in the IDS framework. The results demonstrate what impact the protection has for the operation of a network that is the victim for a routing attack.

6.2 Evaluation Methodology

Evaluation is performed iteratively during the thesis to a small degree. After the implementation of each detection method has been finalized, a micro evaluation takes place. This evaluation includes a verification that the method functions as expected and a test to search for suitable parameter choices that should be set when the framework is undergoing the final thorough evaluation. During the micro evaluations, attacks are implemented in order to be able to verify the desired functionality of the detection methods. The requirements of the attacks and the adversary models that they fulfill are described in Sections 4.2.2 and 4.3.2.

The process of evaluation and verification of the implemented components of the intrusion detection system is carried out both in the WSN simulator Cooja and on real hardware using the platform that Section 2.5 describes. When evaluating on real hardware the sensor nodes need to be connected to a computer to inspect how values and results change over time. The sensor nodes are connected in a similar setup as Figure 6.1 displays.

The evaluation of the framework focus on the following metrics: energy consumption, detection rate and time until detection. For the second detection method the RPL reliability (throughput in the network) and network latency are also measured and evaluated. For the first detection method, the metrics are evaluated by comparing the results when having the IDS on and off while running the Simple Sensor program described in Section 5.2. For the second detection method, the results are produced while letting the IDS be on and off while running a Contiki example program called "unicast-sender" which transmits messages using UDP and RPL at a predefined frequency. The metrics are also compared between cases when the net-

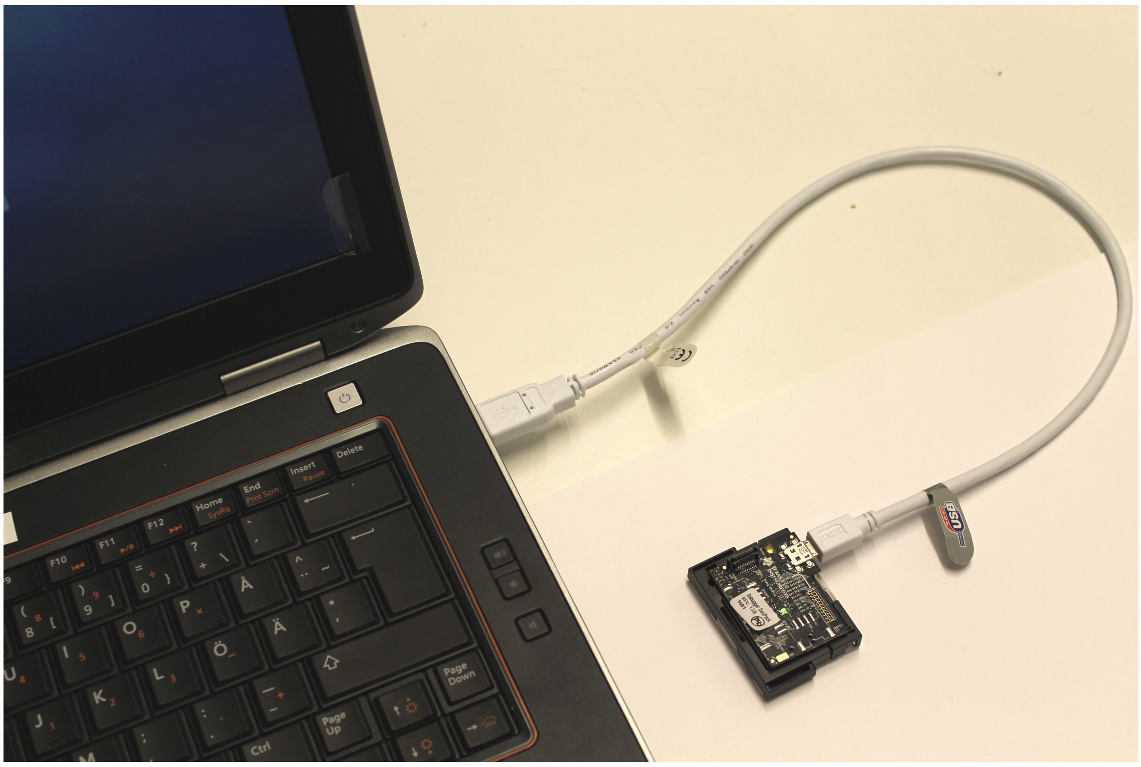


Figure 6.1: The platform is connected to the computer with an USB-cable. Print outs and results on the sensor node can be accessed from the computer host.

work is under attack and when the network is not under attack in order to have a normal case to compare against.

The first detection method is thoroughly parameterized and then evaluated on the hardware specified in Section 2.5. Typically, evaluation is performed using two or three nodes where the setups include a node running the IDS, a node that acts as a sink node and a node that performs the DoS-attacks. It is a known limitation that only three nodes are used during the evaluation. The resulting parameter choices are therefore fit only for setups similar to the one used during the evaluation and other setups would probably require different parameter choices. The choice of having just three nodes is motivated since the aim of the evaluation is to demonstrate that the detection method works as intended on the hardware and to give initial parameter choices, not to demonstrate that the IDS is built to scale to thousands of nodes.

The second detection method focus on routing attacks and therefore needs to be evaluated on larger clusters of sensor nodes than the first detection method. The evaluation of the second detection method is carried out in the simulator Cooja to a large extent in order to be able to evaluate different parameter choices with more determination. In Cooja, the sensor nodes that are programmed with the IDS code and are used for the simulations are of the type Zolertia Z1 since they have nearly as much programmable memory as the platform that the thesis targets [29]. Table 6.1

presents a comparison between the two platforms in terms of programmable memory (ROM) and processor frequency.

Table 6.1: Hardware specifications for Texas Instruments SensorTag and Zolertia Z1. The Z1 mote has almost as much programmable memory (ROM) as the SensorTag.

	CPU frequency (MHz)	**ROM (kB)**	**RAM (kB)**
SensorTag	48	128	28
Z1	16	92	8

Using a simulator is motivated since it would be problematic and hard to evaluate the different roles in the distributed system using real sensor nodes since one node would need one computer in order to be monitored. Within Cooja however, it is easy to setup clusters of sensor nodes and to monitor the nodes individually. Another motivation for evaluating the second detection method in Cooja is that the algorithm is less sensitive for false alarms and hardware dependent values than the algorithms used within the first detection method are.

Both of the detection methods within the IDS framework are evaluated on real hardware in order to find the energy consumption of the designs that the thesis propose. This choice is motivated since it is well within the scope of the thesis and is part of the scientific contribution of the report. It is expected that the results might vary since it is within the nature of real world scenarios that non-determinism is involved when measuring energy consumption.

6.3 Evaluation Metrics

The detection methods of the framework are evaluated according to the following metrics:

Total energy consumption of the implemented framework is measured by calculating energy consumption of both the communication and the computations. The total energy consumption is measured by adding the results from the two components together, as Equation 6.1 describes.

$$E_{tot} = E_{radio} + E_{CPU} \tag{6.1}$$

Calculation of time spent on communication is performed by measuring transmit and listen times of the radio hardware. These times are then used to calculate energy consumption for radio activities according to Equation 6.2,

$$E_{radio} = (T_{Receive} \cdot I_{Radio\ RX} + T_{Transmit} \cdot I_{Radio\ TX}) \cdot V_{System} \tag{6.2}$$

where $T_{Receive}$ is the time spent on receiving radio communication, $T_{Transmit}$ is the transmit time, V_{System} is the operating voltage of the system, $I_{Radio\ RX}$ is the current consumption required for receiving, and $I_{Radio\ TX}$ is the current consumption required for transmitting. $I_{Radio\ RX}$, $I_{Radio\ TX}$ and V_{System} are fetched from the manuals of the specific radio component of the sensor nodes used within the thesis.

The energy consumption of the CPU can be calculated by measuring time spent on computations and time spent in standby mode and then use these values to calculate total energy consumption of CPU according to Equation 6.3,

$$E_{CPU} = (T_{CPU\ active} \cdot I_{CPU\ active} + T_{CPU\ standby} \cdot I_{CPU\ standby}) \cdot V_{System} \tag{6.3}$$

where $T_{CPU\ active}$ is the active time of the CPU, $I_{CPU\ active}$ is the current consumption of the CPU when it is active, $T_{CPU\ standby}$ is the time when the CPU is in standby (sleeping), $I_{CPU\ standby}$ is the current consumption of the CPU when it is in standby and V_{System} is the operating voltage of the system.

Table 6.2 shows the current consumption and voltage for the internal components and states of the hardware which are found in Table 5.4 in the data sheet for the CC2650 platform [30]. There are two clocks that can be used during low power mode of the CPU. During the experiments the RCOSC clock is used during standby mode since it is chosen by default in the Contiki source-code. In the low power mode the platform supports retention of values in the registers of the CPU, value of the real-time clock and contents in RAM.

Table 6.2: Values for current consumption for the different states of the micro controller unit. System voltage is also shown.

State	**Value**
$I_{CPU\ active\ at\ 48\ MHz}$	2.9 mA
$I_{CPU\ standby\ (RCOSC)}$	0.001 mA
$I_{Radio\ RX}$	5.9 mA
$I_{Radio\ TX\ at\ +5dBm}$	9.1 mA
V_{System}	3.0 V

Detection rate, in addition to measuring the energy consumption, the detection rate is an important metric to measure. It is calculated by taking the number of true alarms generated by the IDS and dividing it by the number of launched attacks against the network, as Equation 6.4 shows.

$$Detection\ rate = \frac{\#\ of\ true\ alarms}{\#\ of\ launched\ attacks} \tag{6.4}$$

Number of false alarms, i.e number of erroneous alarms generated by the intrusion detection system.

Time until detection (TUD) measures the time it takes for the IDS to detect an attack from the time that the attack is launched. Equation 6.5 shows how the TUD is calculated.

$$TUD = T_{detected} - T_{launched} \tag{6.5}$$

RPL reliability, i.e. throughput in the network, is also an important metric to measure since if the throughput is low the normal functionality of the network will be disrupted. The formula that calculates the reliability takes number of packets sent divided by number of packets delivered at destination as Equation 6.6 shows.

$$Reliability = \frac{\#\ of\ packets\ received}{\#\ of\ packets\ sent} \tag{6.6}$$

Latency is measured as the latency introduced in the network by the IDS. Latency is the time it takes from sending a message until the time it is received. The latency is measured with the help of the simulator Cooja since Cooja provides global time stamps in the system log of the simulation. Equation 6.7 shows how the latency for a message is computed.

$$Latency = T_{received} - T_{sent} \tag{6.7}$$

6.4 Anomaly-based DoS-Attack Detection

This section presents the results of evaluating the two different detection algorithms within the first detection method. The section presents a study on the energy consumption profile of the radio hardware that is used when setting parameters for the algorithms. The section also presents reasoning on how the different parameters are set and why. Lastly, the section presents the results from the two algorithms in terms of detection rate, false positives, time until detection of attacks and energy consumption of using Contiki with the different components of the detection method.

6.4.1 Setup, Attack Parameters, Energy Profile and Detection Parameters

The evaluation of the first detection method is performed using three nodes where the setup includes one node that acts as a sink, one node that runs the IDS and one node that attacks the network with a DoS-attack. There are two ways to vary the DoS-attack of the network where either the intensity or the launch time of the attack can be varied. Section 6.4.1 describes how the parameters for different attacks are varied during the evaluation.

The energy consumption profile of the sensor nodes is important to determine if anomaly based detection of DoS-attacks is going to function without giving too many false alarms. Section 6.4.1.1 elaborates and determines how the profile looks depending on how often the energy consumption is inspected. When the energy consumption profile has been determined, the parameterization of the detection algorithms is carried out. Section 6.4.1.2 describes how the parameters are determined and also presents recommended parameters to use for the algorithms that are used for detection of DoS-attacks. The attacking node that is implemented to test the first detection method has primarily two different parameters that can be adjusted. The parameters control intensity of the attack and startup delay, i.e. when the attack shall be launched.

Table 6.3 describes the four different attack intensities that are used when evaluating the first detection method. The table describes how often the attacking node tries to send messages given in terms of messages per second. In the code, the intensity is defined in terms of timer ticks. One second is 128 timer ticks, so in the high intensity case, the node will try to send 128 messages every second as the table shows. Note that just because the program tries to send a message during every timer tick in the high intensity attack does not mean that it will succeed in doing so.

Table 6.3: Different attack intensities that describe how often the attacking node will try to send a message over the network.

Intensity name	**Number of messages sent per second**
Low	1.28
Medium low	2.56
Medium high	5.12
High	128

Table 6.4 shows the different attack delays that are used during the evaluation. The value indicates that an attack is started after the given delay value when a delay is used. The attack node continuously sends packets until it is shut down.

Table 6.4: Different attack start up delays that describe when the attack will be launched. Time is given from startup of attack program on attack node.

Startup delay name	**Startup delay value (seconds)**
Short	$1 \cdot 60$
Medium short	$5 \cdot 60$
Medium long	$10 \cdot 60$
Long	$60 \cdot 60$

6.4.1.1 Radio Energy Consumption Profile

Riecker *et al.* [4] assume that the energy consumption of the sensor nodes is fairly even and linear. If the energy consumption values are evenly distributed it allows the usage of linear regression to determine sudden changes of the energy consumption as caused by a DoS-attack. Tests are conducted in order to determine how the energy consumption is distributed in six different modes using a standardized approach of reading how much time that has been spent in the radio hardware once every tenth second. The tests include having two sensor nodes connected to one computer each for reading the output. The first sensor node acts as the sensor node running the IDS and the second sensor node acts as the sink node. The results from the tests are plotted in graphs in order to visually show how the time spent in the radio hardware varied. Energest value, energy consumption and real-time clock value is used interchangeably to describe the tests since the values scale the same and represent the same metric, i.e. energy consumption by radio hardware.

The first test measures the energy consumption of the radio component while running only the first detection method, i.e. the Simple Sensor program is not active at all. The test serves as an indicator whether energy consumption can be read or not. Figure 6.2 presents the results from both the first and the second test which is described below. The data that the graph presents shows that at most of the time periods the energy consumption is linear, with only a few data points that reach a value of over 10k. There are also two time intervals where the y-value goes over 20k. The periods where the energy consumption peaked are most likely due to announcement and beacon messages that the node sent and received.

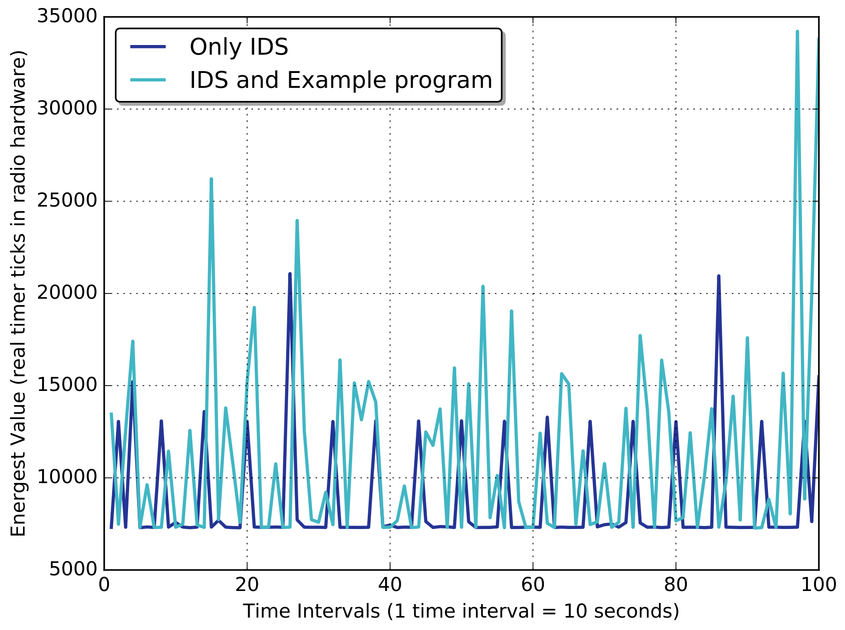


Figure 6.2: Energest value variations for 100 time intervals are shown for two different cases. The difference between the cases are that the first case shows a node that is having just the IDS on and the second case shows a node that has both the IDS and the example program on.

The second test measures the energy consumption of the radio component while the node is running both the first detection method and the Simple Sensor program that periodically sends sensor values to the sink. The second test serves to show what a normal energy consumption of the sensor node looks like and it serves as a basis of comparison against how the energy consumption varies when the node is under attack. It can be noted that the example program sent sensor values to the sink node approximately once every 30th second. Figure 6.2 shows the plotted graph of the results. Looking at the graph, it can be seen that the energy consumption does not follow a linear pattern at all. For most time-periods the y-value varies between just below 10k and 20k. A few values are over 20k and two values go up to around 35k. It is likely that the distribution is due to announcement and beacon messages as well as sending of sensor values to the sink node at the same time interval.

The third, fourth, fifth and sixth tests measures the energy consumption of the radio hardware when the node is running both the example program and the detection method while being under attacks with varying degrees of intensities. Table 6.3 in Section 6.4.1 describes the four different attack intensities that are used during the tests. Figure 6.3 displays the resulting graphs of how the time spent in the radio hardware varies for the four different attack intensities. The attack is undergoing during the whole test interval and the results therefore serves to show the viability of the detection method. Looking at the figure, it can be noted that the energy consumption of the radio hardware increases during attacks just as one expects. For the low intensity, all time periods have an Energest value of over 10k, three time periods reach over 30k and one time period reaches over 50k. For the medium low intensity, it can be noted that the time spent in radio hardware increases so that all values are over 15k and that there are some spikes in the energy consumption. For the medium high intensity attack, it can be noted that all the time periods have an Energest value of at least 25k with some spikes reaching over 30k and a few up to over 40k. For the high intensity attack, it can be noted that the y-values for all the time periods reach over 30k and many reach up to around 35k, there are a few spikes where the largest spike in consumption reach up to 65k in Energest value.

Figure 6.2 shows the normal consumption pattern for a sensor node when it runs both the first detection method and the example program. From the figure it can be concluded that the energy consumption is not smooth and linear for the chosen length of time interval. The energy consumption pattern for the radio hardware when reading the energy consumption once every tenth second has a lot of spikes and variations. This indicates that the linear regression algorithm with 10 seconds as length for the time periods is hard to use for determining what is a state of attack and what is a normal state since predictions of next-coming values will potentially be very far away from what the actual values are.

When using the increasing average algorithm however, one or two spikes do not cause the node to yield an alarm since the average needs to increase for a certain number of consecutive rounds before any alarm is triggered. The two spikes visible in the aforementioned graph causes the average to increase for at least two rounds, but when the node goes back to the ”normal” energy consumption the average sinks again and the counter on how many consecutive previous rounds that the average has been increasing is reset to zero again.

To make the linear regression algorithm predict values more accurately the time period needs to become longer. A longer period will create a smoother graph and

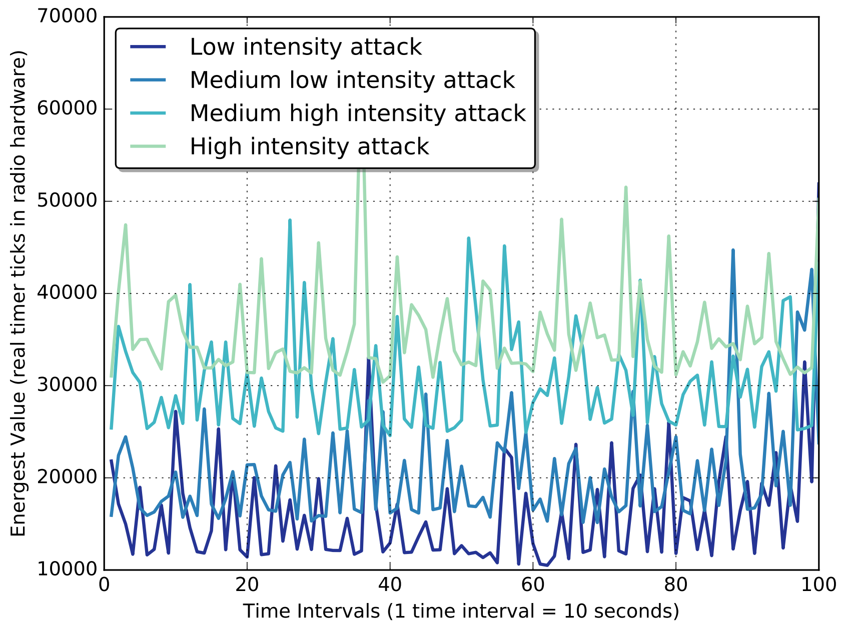


Figure 6.3: Energest value variations for 100 time intervals for four different cases. The graph shows the time spent in radio hardware while being under four different degrees of intensity of a DoS-attack.

energy consumption profile since the sudden bursts will become interpolated over a larger time period. To confirm this, further testing is conducted and Figure 6.4 presents the results from two tests where the period length of the rounds in the method is increased to 20 and 30 seconds instead of 10. Looking at the figure, it can be noted that the energy consumption profile becomes smoother the longer the time period becomes, i.e. the profile is more linear between the first and the last data points and the difference between the highest and the lowest data points is smaller.

6.4.1.2 Parameters for Detection Algorithms

In order to tune the performance of the DoS-attack detection method so that the method avoids giving false alarms, the parameters for how often the energy should be read and how many readings to store needs to be decided on. The parameters are denoted read interval and max readings respectively, and Section 5.3.1 describes them further. To achieve this, tests are performed by running the ”Simple Sensor Program”, which Section 5.2 describes, with the detection method active as an application without any ongoing attacks in the system.

Nine different test cases are designed with varying values for the aforementioned parameters as Table 6.5 shows. The read interval is varied between 10, 20 and 30 seconds. The number of elements to keep, max readings, is varied between 5, 10 and 15 elements in the list.

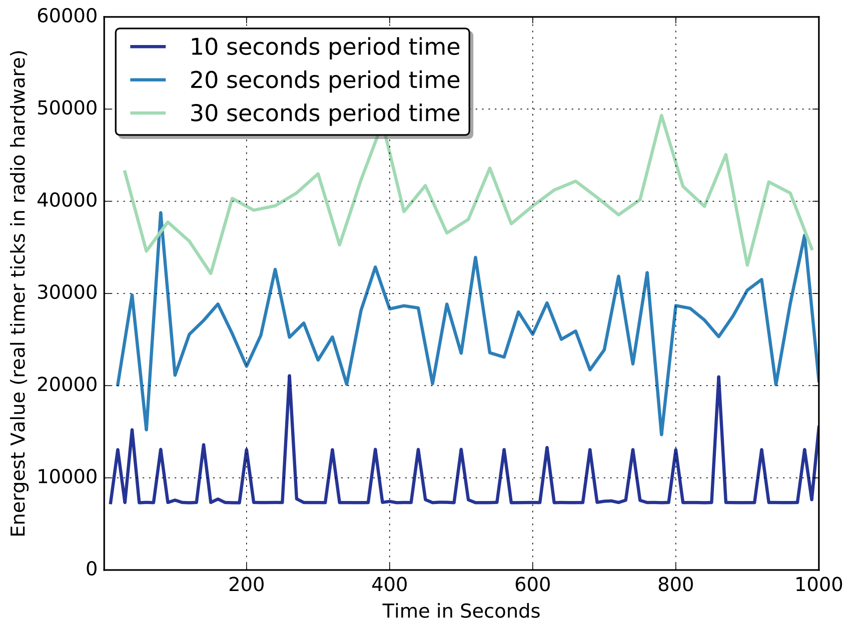


Figure 6.4: Energest value variations over 1000 seconds are displayed for three different period times. The graph shows that a greater value for the time period parameter creates a smoother graph with variations that are not in the order of a 100 % increase or decrease between time periods.

Table 6.5: Test cases with variations of the parameters that control how many elements that are stored in the energy consumption history of a node and how often the energy consumption should be read. Each test is performed for 100 time intervals. The table summarizes how long each test takes.

Test case #	1	2	3	4	5	6	7	8	9
Read interval (seconds)	10			20			30		
Max readings #	5	10	15	5	10	15	5	10	15
Duration (seconds)	1000			2000			3000		

For each test, the program runs for 100 time intervals which makes the duration of the tests different depending on how long the time interval is set to be. All the tests are performed twice, once with the *LiReg* algorithm activated and once with the *IncA* algorithm activated with the overall aim to set the sensitivity parameters of the algorithms, i.e. how sensitive the algorithms shall be for raising alarms.

For the LiReg algorithm, Table 6.6 presents the lowest, highest and the average values for the difference between where the predicted Energest value should be and where it actually is when reading the energy consumption the next round. The values that the table presents are calculated by taking the difference between the predicted value, produced by the linear regression algorithm, and the actual value read with Energest and then dividing by the average value of the earlier energy readings. The obtained value is then multiplied by 100 so the resulting difference can be shown in percents deviation from average. The average and highest values in the table are used to determine how much the expected value and actual value can deviate before assuming that the system is under attack. It is desired to have the "highest" value as low as possible in order to avoid false positives while still detecting as low intensity attacks as possible since a low value for "highest" indicates a high accuracy of the prediction made by the LiReg algorithm.

Table 6.6: The values in percents deviation from the average are obtained when changing the two parameters Read Interval and Max Readings for the LiReg algorithm are shown. It can be seen that the read interval has a great impact on the average and highest values, and that the significance of max readings becomes smaller as the value for read interval increases.

Configuration parameters		**Percents difference from average** $(100 \cdot \frac{diff}{avg})$		
Read Interval (seconds)	**Max Readings (# elements)**	**Lowest**	**Average**	**Highest**
10	5	0	37.31	113
10	10	0	32.36	106
10	15	0	25.86	131
20	5	0	19.73	58
20	10	0	18.70	77
20	15	0	17.53	77
30	5	0	10.74	34
30	10	0	11.06	45
30	15	0	10.02	39

Table 6.6 shows that with a higher value for the read interval, the "average" and "highest" values become lower. However, as the read interval gets larger the impact of having max readings set to a higher value gets smaller. The table also shows that the difference for the average value between having max readings set to 5, 10 or 15 is less than one percent and the values in the highest values column seem to differ independently of the value for max readings. Therefore, in the configuration to be used when testing the system under attacks the read interval is set to 30 seconds and max readings is set to 5 elements. A smaller list will conserve memory, and, since the difference of having a smaller list in comparison to a larger can be considered to be negligible, it provides a good set up. The threshold value for giving alarms for an attack in the system is set to 40 although the highest value obtained for this configuration is 34. This decision is based on the fact that the highest value seem to deviate a bit when the list is of different length even though, as said earlier, the impact of the average is negligible. By having a small margin it is desired to avoid false alarms but still detect real attacks.

Table 6.7 presents the resulting test data for the algorithm that counts consecutive rounds of increasing average. In this case the goal is to decide upon an apt value for the parameter that determines how many rounds the average value for the list is allowed to increase before the algorithm assumes that the system is under attack. The data that the table presents in the columns called ”average” and ”highest” are the number of consecutive rounds that the average value has continued to increase before the counter has been reset (i.e. the average value has decreased again). The data can be used to conclude that the parameters read interval and max readings have a low influence on both the ”average” and the ”highest” values. The maximal value detected for the consecutive rounds counter in all the tests is 5 and the average number of rounds before resetting the counter is roughly 1. Since the parameters for read interval and max readings have little to no impact on the highest value observed the value of the parameter max readings is set to 5 to make the program have a smaller memory footprint and the period time used is 10 since this allows for attacks to potentially be detected faster. If there would be a need to save more battery on the nodes the read interval can be easily changed to a larger one, but the set up for the evaluation within the thesis prioritizes a better detection rate.

Table 6.7: The right half of the table shows the values, in terms of number of rounds before the increasing energy value is reset to zero. The values are obtained from varying the two parameters in the left half of the table, Read Interval and Max Readings, for the IncA algorithm.

Configuration parameters		**Count on consecutive rounds with increasing average value**		
Read Interval (seconds)	**Max Readings (# elements)**	**Lowest**	**Average**	**Highest**
10	5	0	0.91	4
10	10	0	0.85	4
10	15	0	1.08	5
20	5	0	0.87	5
20	10	0	0.70	4
20	15	0	0.78	5
30	5	0	1.02	4
30	10	0	0.96	5
30	15	0	0.96	5

The data that the table presents for the chosen configuration makes it possible to determine the parameter for how many rounds the value can continue to increase before sending an alarm. The chosen configuration (and all other configurations as well) has an observed highest value of 5 consecutive rounds of energy values that increase the average value before emptying the list. This means that the number of rounds that the average can increase to before sending an alarm can safely be set to 6 rounds to eliminate most of the false positives and still detect real attacks.

Table 6.8 presents the two different recommended configurations that are chosen for the two algorithms within the first detection method. It also shows the values for the limits that are chosen based on the output values for the configurations. The table serves as a summary of the tests performed in this section by presenting the most important data.

Table 6.8: The recommended configurations for each algorithm in the first detection method are shown. The values for the chosen sensitivity levels, or limits, are also shown for each of the recommended configurations.

Recommended parameters			
Algorithm	**Read Interval (seconds)**	**Max Readings (# elements)**	**Limit**
LiReg	30	5	40
IncA	10	5	6

6.4.2 Results: LiReg

This section presents the results in terms of false alarms, detection rate, time until detection for the linear regression algorithm called LiReg. The tests runs with the attack parameters set according to Section 6.4.1 and the configuration used in all the test cases are presented in Table 6.8, on the row for LiReg.

6.4.2.1 False Alarms

Table 6.9 presents the number of false alarms in the system. It can be seen that during all the 16 tests only 2 false alarms are raised. Both of the false alarms occur during the long delay of the attack (where the tests are run for a total of 65 minutes). It can also be seen what intensity the attacks are run at, but since the false alarms occur before the attack launches this fact does not matter in the data.

Table 6.9: Number of false alarms for the LiReg algorithm while testing. It can be seen during which delay and intensity the false alarms are generated.

# False alarms				
Delay ↓ \ Intensity →	Low	Medium low	Medium high	High
Short	0	0	0	0
Medium short	0	0	0	0
Medium long	0	0	0	0
Long	1	1	0	0

The obtained data is used to calculate the False Alarm rate, which Equation 6.8 shows. It can be seen that in 12.5% of all test cases a false alarm is generated.

$$False\ Alarm\ Rate = \frac{2}{16} = 0.125 \tag{6.8}$$

6.4.2.2 True Alarms and Time Until Detection

Table 6.10 shows the time until detection for all the attacks that are discovered or not detected which are marked with N/D. It can be noted that none of the Low intensity attacks and only one of the Medium Low intensity attacks are detected. It can be argued that the Low intensity attack does not disrupt the normal function of the system and therefore does not count as an attack. This is further elaborated on and discussed in Chapter 7, but mentioned here as it gives two different results for the detection rate. The formulas below the table present the detection rates.

Table 6.10: Number of attacks that are not detected (N/D) and the time until detection for the attacks detected with the LiReg algorithm.

True alarms and Time Until Detection (seconds)				
Delay ↓ \ Intensity →	Low	Medium low	Medium high	High
Short	N/D	N/D	120.01	120.01
Medium short	N/D	N/D	30.01	30.06
Medium long	N/D	N/D	30.01	30.10
Long	N/D	30.06	30.05	30.23

The table is used to calculate the detection rate according to Equation 6.4. Equation 6.9 shows the detection rate with low intensity attacks included while Equation 6.10 shows the detection rate without low intensity attacks included.

$$Detection\ Rate_{16\ Attacks} = \frac{9}{16} = 0.562 \tag{6.9}$$

$$Detection\ Rate_{12\ Attacks} = \frac{9}{12} = 0.75 \tag{6.10}$$

6.4.3 Results: IncA

This section presents the results in terms of false alarms, detection rate, time until detection for the increasing average algorithm called IncA. The tests runs with the same attack parameters as the LiReg algorithm does, which are set according to Section 6.4.1. The configuration used in all the test cases is presented in Table 6.8, on the row describing the IncA algorithm.

6.4.3.1 False Alarms

As can be seen in Table 6.11, only one false alarm is raised during testing. The false alarm occurs during the Long delay interval before the Medium high attack is launched.

Table 6.11: Number of false alarms for the IncA algorithm while testing. It can be seen during which attack delay and intensity the false alarms are generated.

# False alarms				
Delay ↓ \ Intensity →	Low	Medium low	Medium high	High
Short	0	0	0	0
Medium short	0	0	0	0
Medium long	0	0	0	0
Long	0	0	1	0

The obtained data is used to calculate the False Alarm rate, which Equation 6.11 shows. It can be seen that in 6.25% of all test cases a false alarm is generated.

$$False\ Alarm\ Rate = \frac{1}{16} = 0.0625 \tag{6.11}$$

6.4.3.2 True Alarms and Time Until Detection

Table 6.12 shows the test cases where an attack is detected by showing the time until detection in seconds. It also shows when an attack is not detected (N/D). One of the four Low intensity attacks is detected, which is more than the results for the LiReg algorithm. But on the contrary only half of the Medium high intensity attacks are detected, while the LiReg algorithm detects all of them. The Medium low intensity attack with the Short delay is not detected, but the other Medium low intensity attacks are. The detection rate for the IncA algorithm is presented with and without the low intensity attack included (using the same logic as with the LiReg algorithm) below the table.

Table 6.12: Number of attacks that are not detected (N/D) and the time until detection for the attacks detected with the IncA algorithm.

True alarms and Time Until Detection (seconds)				
Delay ↓ \ Intensity →	Low	Medium low	Medium high	High
Short	N/D	N/D	N/D	60.06
Medium short	N/D	60.01	N/D	60.01
Medium long	80.11	60.04	50.04	50.08
Long	N/D	50.37	50.06	40.53

The table is used to calculate the detection rate according to Equation 6.4. Equation 6.12 shows the detection rate with low intensity attacks included while Equation 6.13 shows the detection rate without low intensity attacks included.

$$Detection\ Rate_{16\ Attacks} = \frac{10}{16} = 0.625 \tag{6.12}$$

$$Detection\ Rate_{12\ Attacks} = \frac{9}{12} = 0.75 \tag{6.13}$$

6.4.4 Results: Energy Consumption

This section presents the results in terms of energy consumption by the sensor nodes. The section is divided into three subsections so that each subsection can hold information about the three components that are used when evaluating the framework; the Simple Sensor Program, the detection method with the LiReg algorithm activated and the detection method with the IncA algorithm activated. The energy consumption evaluation is conducted using six tests, two for each component. The two tests are based on having the system under normal operation and having the system under a high intensity attack. The Simple Sensor Program is evaluated energy-wise in order to have a reference consumption to compare against. Both the evaluation of the LiReg algorithm and the evaluation of the IncA algorithm is conducted using the Simple Sensor Program running in the background in order to simulate a "real world" scenario. All the tests are conducted using a 30 minutes duration in order to provide statistical security on the results. The test that are conducted when the node is not under attack are performed using two sensor nodes, one that runs the IDS and one that acts as a sink node. The tests that are conducted when the node is under attack are performed using three sensor nodes, one running the IDS, one that acts as a sink node and one node that runs the DoS-attack program.

6.4.4.1 Energy Consumption: Simple Sensor Program

The Simple Sensor Program is evaluated in order to have reference values to compare against when running the other tests. The program is evaluated under both normal conditions and under attack conditions using the high intensity DoS-attack.

Table 6.13 presents the results in terms of how much time has been spent in different hardware states while letting the program run for half an hour under normal conditions. It shows that under normal operation a lot of time is spent in active CPU mode. Only about half a minute is spent in CPU standby mode and even less time is spent in the send and receive states of the radio hardware.

Table 6.13: How much time the system has spent in different hardware states while running the Simple Sensor Program for 30 minutes under normal conditions is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	115 925 406	1768.88
$T_{CPU\ standby}$	2 045 657	31.22
$T_{Receive}$	1 637 858	24.99
$T_{Transmit}$	836 293	12.76

The values in the table are used to calculate the energy consumption of the CPU and the radio according to the formulas as specified in Section 6.3. Equation 6.14 shows the energy consumption for the CPU and Equation 6.15 shows the energy consumption for the radio. Finally, Equation 6.16 shows the total energy consumption of the detection method and the Simple Sensor Program.

$$E_{CPU\ 30min} = (1768.88s \cdot 2.9mA + 31.22s \cdot 0.001mA) \cdot 3V = 15.3893497J = 4.27481935mWh \tag{6.14}$$

$$E_{Radio\ 30min} = (24.99 \cdot 5.9mA + 12.76 \cdot 9.1mA) \cdot 3V = 0.790671J = 0.219630833mWh \tag{6.15}$$

$$E_{tot\ 30min} = 15.3893497J + 0.790671J = 16.1800207J = 4.49445019mWh \tag{6.16}$$

Table 6.14 presents the results in terms of how much time has been spent in different hardware states while letting the program run for one hour during a high intensity DoS-attack. The values in the table are then used to calculate the energy consumption of the node during the timeperiod below the table using the same approach as mentioned earlier.

Table 6.14: How much time the system has spent in different hardware states while running the Simple Sensor Program for 30 minutes under a high intensity attack is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	113 335 266	1729.36
$T_{CPU\ standby}$	4 642 352	70.84
$T_{Receive}$	6 654 664	101.54
$T_{Transmit}$	695 866	10.62

Equation 6.17 shows the energy consumption for the CPU and Equation 6.18 shows the energy consumption for the radio. Finally, Equation 6.19 shows the total energy consumption of the Simple Sensor Program.

$$\begin{aligned} E_{CPU\ 30min} = (1729.36s \cdot 2.9mA + 70.84s \cdot 0.001mA) \cdot 3V = \\ 15.0456445J = 4.1793457mWh \end{aligned} \tag{6.17}$$

$$\begin{aligned} E_{Radio\ 30min} = (101.54s \cdot 5.9mA + 10.62s \cdot 9.1mA) \cdot 3V = \\ 2.087184J = 0.579773333mWh \end{aligned} \tag{6.18}$$

$$E_{tot\ 30min} = 2.087184J + 15.0456445J = 17.1328285J = 4.75911903mWh \tag{6.19}$$

6.4.4.2 Energy Consumption: LiReg

The energy consumption of running the Simple Sensor Program and the detection method with the LiReg algorithm in the background is measured during a 30 minutes test. The test is performed twice, once under attack and once under normal conditions.

Table 6.15 presents the results in terms of how much time has been spent in different hardware states while letting the Simple Sensor Program and the detection method with the LiReg algorithm run for one hour under normal conditions. Most of the time is spent in active CPU mode while the time spent in standby CPU mode is significantly small. Even less time is sent in radio, where the receive time exceeds the send time.

Table 6.15: The time that the system has spent in different hardware states while running both the Simple Sensor Program and the first detection method using LiReg algorithm for 30 minutes under normal conditions is shown. The time is shown both in real-time clock (RTC) ticks and in seconds

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	115 929 484	1768.95
$T_{CPU\ standby}$	2 041 580	31.15
$T_{Receive}$	1 621 207	24.74
$T_{Transmit}$	859 471	13.12

Equation 6.20 shows the energy consumption for the CPU and Equation 6.21 shows the energy consumption for the radio. Finally, Equation 6.22 shows the total energy consumption of the detection method and the Simple Sensor Program.

$$\begin{aligned} E_{CPU\ 30min} = (1734.35s \cdot 2.9mA + 65.84s \cdot 0.001mA) \cdot 3V = \\ 15.0890425J = 4.1914007mWh \end{aligned} \tag{6.20}$$

$$\begin{aligned} E_{Radio\ 30min} = (99.56s \cdot 5.9mA + 11.04s \cdot 9.1mA) \cdot 3V = \\ 2.063604J = 0.573223333mWh \end{aligned} \tag{6.21}$$

$$E_{tot\ 30min} = 15.0890425J + 2.063604J = 17.1526465J = 4.76462403mWh \tag{6.22}$$

Table 6.16 presents the results in terms of how much time has been spent in different hardware states while letting the Simple Sensor Program and the detection method with the linear regression algorithm run for one hour while being under attack of high intensity DoS-attack. It shows that the most time is spent in active CPU mode. The node also spends more time in the radio hardware receiving packets than it does in CPU standby mode due to the DoS-attack.

Table 6.16: The amount of time the system has spent in different hardware states while running the Simple Sensor Program and the first detection method using the LiReg algorithm for 30 minutes under a high intensity DoS-attack is shown in both real-time clock (RTC) ticks and in seconds

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	113 662 670	1734.35
$T_{CPU\ standby}$	4 314 660	65.84
$T_{Receive}$	6 524 572	99.56
$T_{Transmit}$	723 508	11.04

Equation 6.23 shows the energy consumption for the CPU and Equation 6.24 shows the energy consumption for the radio. Finally, Equation 6.25 shows the total energy

consumption of the detection method with the LiReg algorithm and the Simple Sensor Program.

$$E_{CPU\ 30min} = (1734.35s \cdot 2.9mA + 65.84s \cdot 0.001mA) \cdot 3V = 15.0890425J = 4.1914007mWh \tag{6.23}$$

$$E_{Radio\ 30min} = (99.56s \cdot 5.9mA + 11.04s \cdot 9.1mA) \cdot 3V = 2.063604J = 0.573223333mWh \tag{6.24}$$

$$E_{tot\ 30min} = 15.0890425J + 2.063604J = 17.1526465J = 4.76462403mWh \tag{6.25}$$

6.4.4.3 Energy Consumption: IncA

The energy consumption of running the Simple Sensor Program and the first detection method using the IncA algorithm is calculated by measuring how much time the system spends in different hardware states. The energy consumption is measured during two cases, one where the system is under a DoS-attack and one where the system is not under attack. The test results are presented in individual tables.

Table 6.17 presents the results in terms of how much time has been spent in different hardware states while letting the Simple Sensor Program program and the first detection method using the IncA algorithm run for one hour under normal conditions. It can be seen that most of the time is spent in active CPU mode, far less time is spent in standby CPU mode. Even less time is spent in the radio module, for both send and receive time.

Table 6.17: The amount of time that the system has spent in different hardware states while running the Simple Sensor Program and the first detection method with the IncA algorithm for 30 minutes under normal conditions is shown. The values are shown in both real-time clock (RTC) ticks and in seconds

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	115 866 919	1767.99
$T_{CPU\ standby}$	2 104 145	32.11
$T_{Receive}$	1 657 257	25.29
$T_{Transmit}$	845 020	12.89

Equation 6.26 shows the energy consumption for the CPU and Equation 6.27 shows the energy consumption for the radio. Finally, Equation 6.28 shows the total energy consumption of the detection method using the IncA algorithm and the Simple Sensor Program.

$$E_{CPU\ 30min} = (1767.99s \cdot 2.9mA + 32.11s \cdot 0.001mA) \cdot 3V = 15.3816093J = 4.27266926mWh \tag{6.26}$$

$$E_{Radio\ 30min} = (25.29s \cdot 5.9mA + 12.89s \cdot 9.1mA) \cdot 3V = 0.79953J = 0.222091667mWh \tag{6.27}$$

$$E_{tot\ 30min} = 0.79953J + 15.3816093J = 16.1811393J = 4.49476092mWh \tag{6.28}$$

Table 6.18 presents the results in terms of how much time has been spent in different hardware states while letting the Simple Sensor Program and the first detection method using the IncA algorithm run for one hour during a high intensity DoS-attack. The energy consumption is presented below the table using the same approach as mentioned above.

Table 6.18: The amount of time that the system has spent in different hardware states while running both the Simple Sensor Program and the first detection method using the IncA algorithm for 30 minutes under a high intensity DoS-attack is shown. The values are shown in both real-time clock (RTC) ticks and in seconds

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	113 597 410	1733.36
$T_{CPU\ standby}$	4 379 886	66.83
$T_{Receive}$	6 433 676	98.17
$T_{Transmit}$	739 900	11.29

Equation 6.29 shows the energy consumption for the CPU and Equation 6.30 shows the energy consumption for the radio. Finally, Equation 6.31 shows the total energy consumption of the detection method and the Simple Sensor Program.

$$E_{CPU\ 30min} = (1733.36s \cdot 2.9mA + 66.83s \cdot 0.001mA) \cdot 3V = 15.0804325J = 4.18900903mWh \tag{6.29}$$

$$E_{Radio\ 30min} = (98.17s \cdot 5.9mA + 11.29s \cdot 9.1mA) \cdot 3V = 2.045826J = 0.568285mWh \tag{6.30}$$

$$E_{tot\ 30min} = 15.0804325J + 2.045826J = 17.1262585J = 4.75729403mWh \tag{6.31}$$

6.4.4.4 Energy Consumption: Summary

This section provides a summary of the results from the energy consumption evaluation. The section does not add any new data from the evaluation but rather acts as an overview of the results from the previous energy consumption measurements of the first detection method and its related components. The section only brings up the total energy consumption of the measured components since those numbers are what matters in the end when draining the battery of the sensor node.

Table 6.19 shows a summary of the measured energy consumptions and also displays the difference between normal state and attack state for each of the component combinations tested. It can be noted that just running the Simple Sensor Program has the lowest energy consumption and that the LiReg algorithm is a little bit more complex than the IncA algorithm. The difference in energy consumption is not very large between a state of attack and a normal state but the difference is definitely measurable and visible in the results.

Table 6.19: Summary of the total energy consumption of the sensor node under both normal state and attack state. The table also presents the difference between the two states in order to show how much the energy consumption increases when the system is under attack.

Component	**Normal state** $E_{tot\ 30min}$ (J)	**Attack state** $E_{tot\ 30min}$ (J)	**Difference** (J)
Simple Sensor Program (SSP)	16.1800207	17.1328285	0.9528078
SSP+LiReg	16.1860324	17.1526465	0.9666141
SSP+IncA	16.1811393	17.1262585	0.9451192

6.5 RoVer

This section presents the results from the evaluation of the second detection method which focuses on detecting routing attacks in WSNs. The section describes two types of network setups including their components and corresponding parameters which are used in the simulator Cooja during the evaluation. The section also discusses which parameters that are tuned and evaluated in order to demonstrate the intended functionality of the design of the method.

6.5.1 Network Setup, Parameters and Attack Description

The evaluation of the second detection method considers two types of networks that are simulated within the wireless sensor network simulator Cooja using the mote type Zolertia Z1 as specified in Section 6.2. One of the networks is a high intensity network and the other is a low intensity network. The nodes in the high intensity network sends a message to the sink every 15-30 seconds. The nodes have a send interval of 15 seconds, and to the send interval a random time of 1-15 seconds is added to make sure that all nodes in the network do not try to send at the same time. The nodes in the low intensity network sends a message every 15-30 minutes. The two setups require different periodic execution times of the IDS in order to properly detect attacks against the network. Both networks use the same network topology which consists of 12 nodes spread randomly across an area that is roughly 100 meters wide and 100 meters long. Figure 6.5 shows the network topology. Among the nodes there is one sink node, one attack node (where the attack mode can be turned off so that the node acts as a normal node) and 10 nodes that tries to send a message at every send interval. The normal nodes and the attack node are running an example program in Contiki called ”unicast-sender” which sends messages to the sink node at a given time interval. Both the network simulations are set to have a 50% packet loss ratio in order to simulate lossy wireless network traffic.

There are three parameters in the second detection method that are interesting to study. Two of the parameters are used for setting the sensitivity of the detection method and the third parameter is used to control how often the program is executed. Section 5.4.5 describes the sensitivity parameters which are called ”Below Ten” and ”Above Ten”. The two parameters are from now on denoted lower limit and upper limit. These are used in order to determine if there is an ongoing attack. The first parameter is given as a numeric value of how many messages that are allowed to be lost during normal operation if 10 or fewer messages has been sent by a node and its siblings. The second parameter is used when more than 10 messages has been sent by a node and its siblings to their common parent. The second parameter is given in percent and controls how many percent of the messages that are expected to arrive at the destination (i.e. the parent). In order to set the two sensitivity parameters to values that do not cause the program to create false alarms, the length of the first phase of the program is important to look at. Section 5.4.1 describes the implementation of the internal phases of the detection method. It is during the first phase of the program that all the statistical data is gathered, and therefore it is the longest and most important phase. The other timers for the remaining three phases could also be set to different values, but these time periods are short and belongs

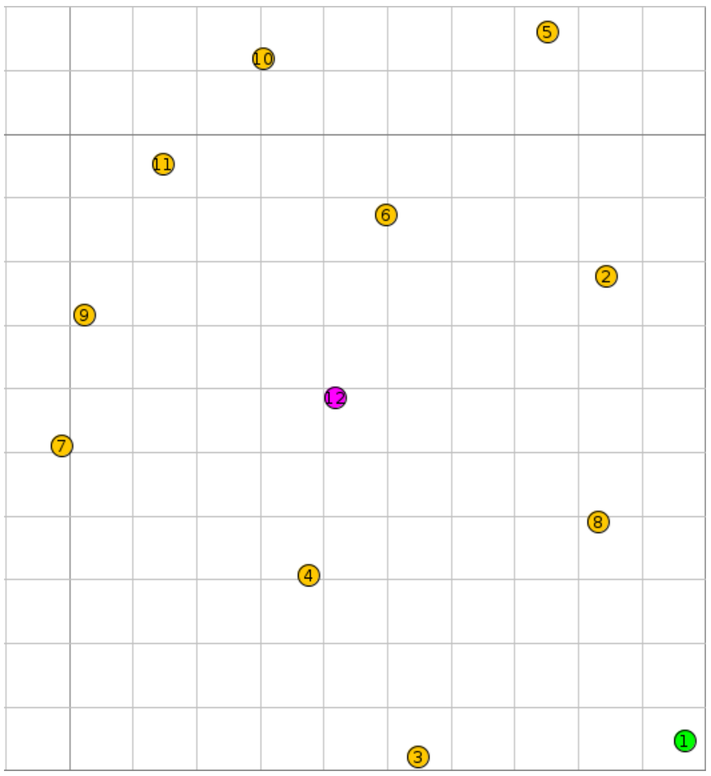


Figure 6.5: Network topology of simulated network of sensor nodes in Cooja. Node marked with number 1 is the sink node, nodes 2-11 are normal nodes and node 12 is the attack node. One square is 10 meters wide and 10 meters high.

to the internal functionality of the program and are therefore not as important to evaluate. Hence, the period length of the first phase will be used with different values to control how often the program is executed for the two different network types.

The two different networks require different lengths of the first phase in order to work properly. For the high intensity network, the time of the first phase is set to 10 minutes. 10 minutes is a reasonable time interval to check and verify the parents and the children in the network with the IDS since the nodes are sending messages at such a high frequency. A too short period would also be inadequate since it would drain the battery to run the IDS too often.

The low intensity network have the period time set to 60 minutes. Due to the long send interval it is necessary to wait such a long time in order to be able to have any messages to count at all. In one hour, the individual nodes are able to send a maximum of 4 messages in total. This amount should be sufficient to detect attacks if the limit value is set to an apt value. It can be argued that an even longer time period could be used in order to save as much battery as possible, but that would mean hours until an attack would be detected.

Table 6.20 gives an overview of the chosen network setups. In the table, both the period time of the first phase of the detection method and the send interval for each of the nodes that are running the ”unicast-sender” program are shown. Note that the send interval is given as an interval since it is composed by a static and a random part. In addition to the information in the table, the maximum number of missed broadcasts from parents is set to five for both networks in order to avoid unnecessary false alarms.

Table 6.20: Description of the two network setups. The parameters that are used in order to set the limits for both networks are shown. Period time indicates how often the detection method is executed, and send interval indicates how often the nodes in the network send a message.

Network setups			
Network type	**Period time (seconds)**	**Send interval (seconds)**	**# Nodes**
High Intensity	10·60	15-30	12
Low Intensity	60·60	15·60-30·60	12

6.5.1.1 Attack Parameters and Test Durations

The evil node launches the attack after different time periods in order to see if the system detects the attack (and also prevents the node from being a parent again). For each of the two network types that are described in the network setup, the test duration is set to go on for two additional program rounds after the launch of the attack. The length of the program rounds differ between the low intensity and the high intensity network, therefore the length of the testing differs between the network types. For each program round there are additionally two minutes of internal time for the program, so for each round of testing two minutes are added to the test duration time. Table 6.21 shows the delays that are used for starting the attack against the network. The value that is given in the table means that the attack starts after the given time. The table also shows the lengths of the tests for the network types.

Table 6.21: Different attack start up delays that describes when the attack will be launched. Time is given from startup of attack program on attack node. Test durations for both high intensity network (HIN) and low intensity network (LIN) are defined as two program execution rounds longer than the attack launch.

Delay name	**Delay value (seconds)**	**HIN test duration (seconds)**	**LIN test duration (seconds)**
Short	$15 \cdot 60$	$36 \cdot 60$	$124 \cdot 60$
Medium short	$45 \cdot 60$	$72 \cdot 60$	$124 \cdot 60$
Medium long	$75 \cdot 60$	$108 \cdot 60$	$186 \cdot 60$
Long	$105 \cdot 60$	$144 \cdot 60$	$186 \cdot 60$

6.5.1.2 Parameters for Detection

In order to determine recommended values for the parameters that control the lower limit and the upper limit so that the program does not produce false alarms, an iterative approach is used which is described as follows. Before testing begins the success ratio of receiving network traffic in Cooja is set to 50% if the receiving node is at the maximum hearing distance from the sending node. This is important since a success rate of 100% would not simulate a real world scenario. When determining the value of the lower limit parameter the upper limit parameter is locked to just accept all packet losses, and vice versa. By locking one of them it is certain that the parameter that is currently not being tested will not generate any false alarms and interfere with the testing. The test starts by setting the parameter currently under evaluation to not accept any packet losses at all and then starting the simulation to simulate normal network traffic, i.e. without launching any attacks. If there are any false alarms the limit is changed to the next value and testing is started again. For the lower limit parameter the next value is the current value increased by one, so it starts with accepting 0 network losses, and the next round it will be set to 1 and so on. For the upper limit parameter the next value is set to reduce the current by five steps, so it starts of being 100 (i.e. expecting that 100% of network traffic to arrive), and in the next round it will be set to 95 and so on. This method repeats until the lowest (or highest for the upper limit parameter) value with no false alarms is found.

Parameters: Low Intensity Network

The low intensity network uses both the lower limit and the upper limit parameters, depending on how many children a parent has. A child which only has one or zero siblings will never use the upper limit since the child and its sibling can not send 10 or more messages together, but a child with at least two siblings might.

In order to set the parameters, each test is run for three detection rounds, meaning it runs for slightly more than three hours. This gives the algorithms current value for the parameter several chances to raise false alarms. Three rounds is enough since each node verifies their parents or children (that acts as parents as well) which gives several chances to report a false positive in each detection round.

Testing shows that no false positives are generated for either of the two parameters when they are set to accept no packet losses. Therefore, only one test for each parameter is run and the parameter values that are chosen for the first testing round of the method are chosen as recommended values. This means that the lower limit is set to zero (indicating that zero message loss is tolerated) and the upper limit is set to 100 (indicating that 100% of messages are required to arrive).

Parameters: High Intensity Network

The high intensity network also needs recommended values for both the packet loss parameters, even though the lower limit seldom (or possibly never) is used due to the high number of messages sent. One node will send a minimum of two messages per minute, making it a total of at least 20 messages when the detection method checks the total number of messages after 10 minutes. This means that even a child with no siblings sends more than 10 messages. Each test is run for six verification phases, meaning that it runs for slightly more than an hour per test.

Testing of the lower limit parameter shows that it can be set to the strictest value (namely zero) since no false alarms are raised. As described previously, this behavior is expected since the low limit is probably not used and the value that is used (the higher) accepts all losses during the test. The upper limit parameter is set to 65% after eight number of testing rounds. Figure 6.6 demonstrates how the number of false positives drops until it reaches zero as the parameter gets less strict.

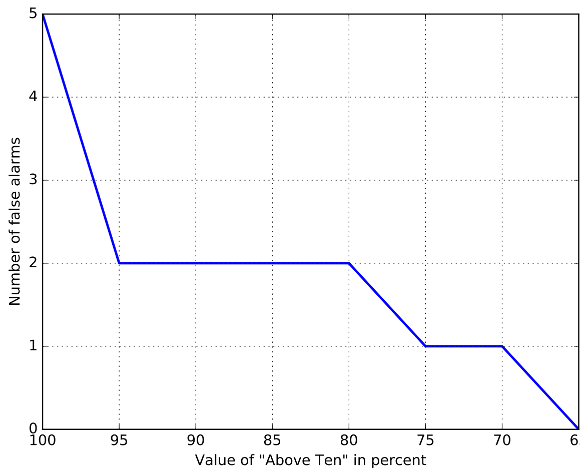


Figure 6.6: Number of false alarms decreasing as the limit value grows less strict. With a a strict limit not allowing any missed messages five false alarms are raised. When the limit reaches 65% no false alarms are detected any more.

Recommended Parameters: Even though that tests demonstrate that the limit parameters could be set to be as strict as possible for three out of four possible parameter choices, it could be dangerous to have such strict limits. Having such strict limits on the network would raise false alarms if just one message gets lost during communication. The recommended parameters are therefore slightly less sensitive for the parameters that could be set as strict as possible. Table 6.22 shows the recommended parameter choices for the two types of network setups that are simulated during the evaluation. The values in the table are the limit values that are used for the rest of the evaluation and are thus the values that are tested when determining if there are any ongoing attacks or not.

Table 6.22: Recommended parameters for the second detection method. The table shows parameter values for both a high intensity and a low intensity network as defined within the chapter. The values are slightly less sensitive than what tests showed in order to avoid false alarms.

Recommended parameters		
Network type	**Lower limit (numeric value)**	**Upper limit (percents)**
Low Intensity	1	95
High Intensity	1	65

6.5.2 Results: Detection

This section presents the results of evaluating the detection method with the parameters for the upper limit and the lower limit set according to the recommended parameter values which the previous section presents. To evaluate the detection rate the Sinkhole attack is used which Section 5.5.2.1 describes the implementation of. The implementation section also describes two other attacks, namely the Selective forwarding attack and the wormhole attack. These two attacks are only used in order to verify the correctness of the method (i.e. that they are detected by the detection method) and are not further evaluated.

6.5.2.1 False Alarms

Table 6.23 presents the number of false alarms for every test in each network type. After a node has been marked as evil and excluded from the network, alarms from the node is ignored since it is not part of the network anymore. The tests on the Low intensity network reports no false alarms at all. Tests on the High intensity network reports a varying number of false alarms, from no false alarms at all to four false alarms in the case of the Medium long delay before launching the attack.

Table 6.23: Number of false alarms when running the tests. It can be seen that no delays are reported for the Low intensity network while a few are reported in the high intensity network.

# False alarms		
Delay ↓ \ Network type →	Low intensity	High intensity
Short	0	0
Medium short	0	1
Medium long	0	4
Long	0	1

The table can be used to calculate the false alarm rate as following; there are eight tests with twelve nodes that have the possibility to send a false alarms and a total of six false alarms. The result is shown by Equation 6.32. The equation shows that false alarms exist in 6.25% of all test cases.

$$False\ Alarm\ Rate = \frac{6}{8 \cdot 12} = 0.0625 \tag{6.32}$$

6.5.2.2 True Alarms and Time Until Detection

Table 6.24 and 6.25 shows the time until detection (TUD) for all launched attacks for the low intensity network and the high intensity network respectively. TUD is the time it takes from the point where the attack starts until one node reports the attacking node as evil. The attack consists of two parts, one that attracts nearby nodes' traffic by announcement of spoofed routing rank and one that actively drops all the received traffic. The latter part is controlled by the delay while the first part of the attack is active from start. In some of the cases, the attacks are detected before the second part is activated. These have been marked as prematurely detected and the TUD value is calculated from start (t=0) until the alarm is raised. The premature detection can be explained by the effects of the first part of the attack. All nodes that can physically reach the attacking node will chose the attacking node as their parent (with the exception of nodes that are in direct contact with the sink). This is an anomalous behavior in the network, and all children are not able to communicate due to the unnatural routing graph that is formed. Because of the lack of communication the children flags the attack node as evil.

By looking at Tables 6.24 and 6.25, it can be seen that all attacks are detected. Therefore, the detection rate is at 100% for the second detection method for both types of network.

Table 6.24: True alarms and how long it takes before the attack is detected for the `low intensity network`. If the attack is detected prematurely, i.e. before the dropping of packets starts, the TUD is calculated from t=0 to time of detection, otherwise it is calculated from the time that the dropping of packets starts until the attack is detected.

True alarms and Time Until Detection (seconds) for LIN		
Delay	**TUD**	**Premature detection**
Short	6 357.864	no
Medium short	4 563.556	no
Medium long	2 780.022	no
Long	4 639.140	no

Table 6.25: True alarms and how long it takes before the attack is detected for the `high intensity network`. If the attack is detected prematurely, i.e. before the dropping of packets starts, the TUD is calculated from t=0 to time of detection, otherwise it is calculated from the time that the dropping of packets starts until the attack is detected.

True alarms and Time Until Detection (seconds) for HIN		
Delay	**TUD**	**Premature detection**
Short	409.464	no
Medium short	1 312.147	yes
Medium long	1 312.147	yes
Long	4 562.241	yes

6.5.3 Results: Latency

When measuring the latency of the packets, success rate for packet transmit and receive is set to 100% in Cooja. The aim when measuring the latency is to compare the latency that the IDS introduces to the network compared to when the IDS is turned off. To achieve this, the success rate of communication is set to 100%. This is due to the fact that when measuring the latency, and also the reliability, it gives the best result to have as many messages as possible to measure upon and that the messages should not be subject of randomization introduced by simulated packet loss. When measuring the latency, the high intensity network is used and when an attack is present, the short delay is used which indicates that the test is run for 36 minutes as Table 6.21 states. The latency for the network is counted on for node #10 or node #5 since those nodes are the furthest away from the sink as can be seen in Figure 6.5.

Figure 6.7 shows the latency for messages sent by node #10 both when the IDS is turned on and when it is turned off when no attacks are launched against the network. It can be seen that there is no significant difference between the two test cases in general. Note that the points that disappears over value six seconds on the y-axis goes to infinity since the messages are never delivered to the sink.

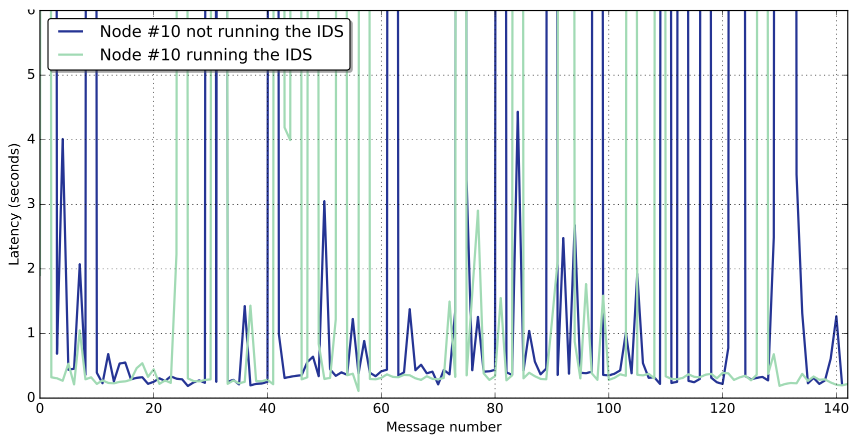


Figure 6.7: The latency for the messages sent from node #10 can be observed both when the node runs the IDS and also when it does not. No direct impact on the latency can be seen between running the IDS or not. Note that lines that are reaching over the y-axis goes towards infinity due to non-delivered messages.

Figure 6.8 shows the latency for node #5 before, during and after the dropping phase of the sinkhole attack has been launched. The chosen node has the attacking node as parent in the routing graph before and during the attack. The figure shows that the node that is running the IDS recovers and switches parent so that the latency decreases to normal values again. It can be seen in the figure that the node not running the IDS never recovers when the attack is launched, all future messages goes to infinity.

6.5.4 Results: Reliability

In order to look at how the reliability changes over time in general and how it changes during an attack scenario, the high intensity network is used in combination with the long attack delay which means that the test is run for 144 minutes as Table 6.21 states. The total number of messages sent by a node are then divided into ten equally large intervals and the rest that are left after the division are part

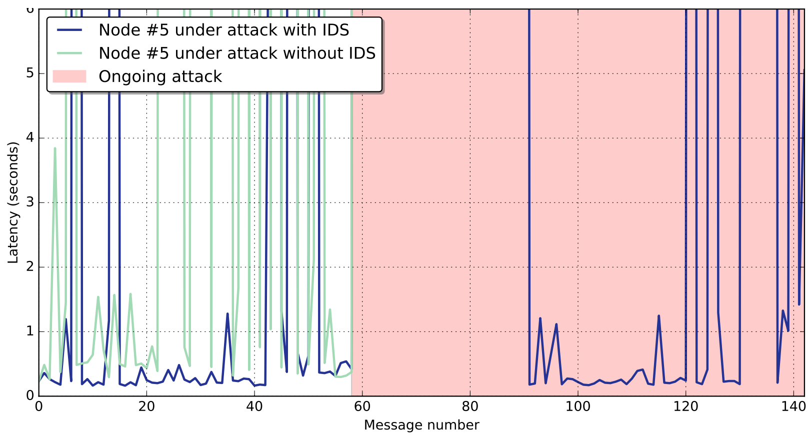


Figure 6.8: The latency for the messages sent from node #5 can be observed, both when the node runs the IDS and also when it does not. It can be seen that in both cases the latency of the messages goes to infinity when the drop phase of the sinkhole attack starts. The messages are being delivered again after a while for the node that runs the IDS while the other node does not show the same result. Note that lines that are reaching over the y-axis goes towards infinity.

of an eleventh interval).

Figure 6.9 shows how the reliability changes over time for nodes #2-11 when the IDS is active while an attack is launched against the system. The sink node is excluded from the figure since it does not send any messages and the malicious node is also excluded since it has an abnormal function in the network. It can be seen that during message intervals 7 and 8 the attack is ongoing, lowering the reliability of the network significantly. In interval 9 the attack is detected and the sinkhole node excluded from the network, effectively stopping the attack and increasing the reliability again.

Figure 6.10 shows the reliability for nodes #2-11 without the IDS active while an attack is launched against the system. It can be seen that when the attack is launched in interval 7 the reliability for the nodes quickly drops to zero and never recovers. Only nodes number #3 and #8 are unaffected, but these are in direct contact with the sink.

Figure 6.11 shows the average reliability and the standard deviation per interval for both having the IDS on and off as shown in Figure 6.9 and Figure 6.10. It can be seen that the reliability for the network with the IDS active recovers again while the reliability for the network not running the IDS stays at 20%. The only thing that keeps the reliability of the network without the IDS from dropping to zero is the fact that nodes number #3 and #8 are unaffected by the attack since these nodes are in direct contact with the sink.

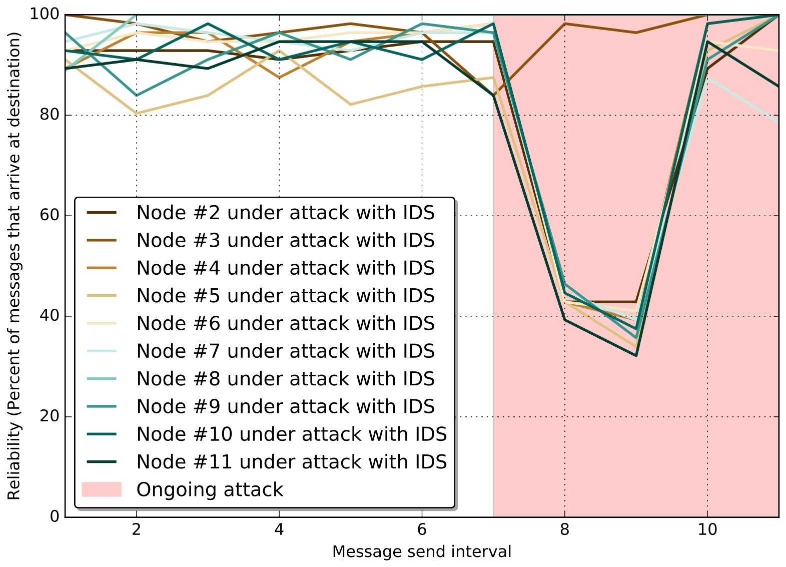


Figure 6.9: The reliability for delivered messages for all nodes in the network is shown except for the attack node and the sink node. All the nodes are running the IDS. It can be seen that after the launch of the drop phase of the attack, the reliability increases again so that messages are able to reach the destination.

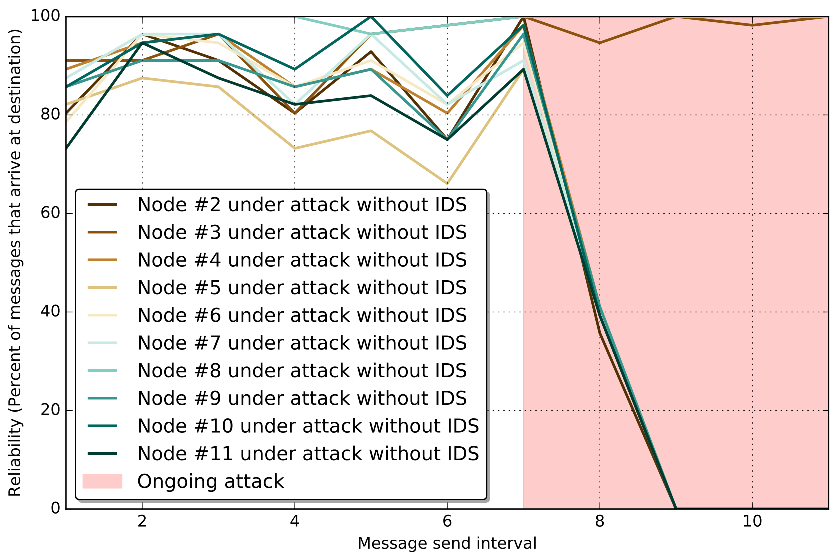


Figure 6.10: The reliability of delivered messages for all nodes in the network is shown except for the attack node and the sink. None of the nodes are running the IDS. It can be seen that after the launch of the attack, the reliability goes towards zero for all the affected nodes.

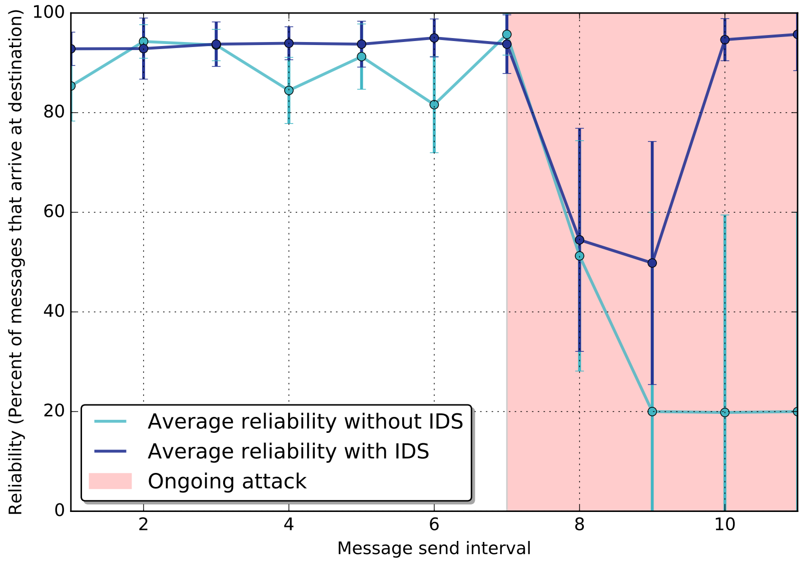


Figure 6.11: The average reliability for delivered messages and the corresponding standard deviation is shown for all nodes shown in Figure 6.9 and 6.10. It can be seen that the IDS restores reliability once the attack is detected and mitigated while the network not running the IDS remains affected.

6.5.5 Results: Energy Consumption

Energy consumption is evaluated both on real hardware and in Cooja since the previous tests have been carried out in Cooja using simulated Z1 motes. Measurements of energy consumption in Cooja serves as a comparison between the simulated Z1 platform and the real hardware used within the thesis. The aim is two-fold, firstly to compare having the IDS on and off on the platform for two different roles; child and parent, and secondly to compare the results of the two individual test cases for the two roles between the simulated and the real platform. To make a fair comparison a similar network setup is used for both simulator and real world tests. The setup consists of three nodes: a sink node and two normal nodes that are placed on such distances away from the sink so that one will act as a parent and the other as a child. Figure 6.12 shows the simulated network in Cooja.

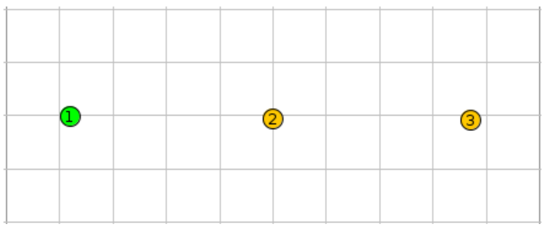


Figure 6.12: Network topology of simulated network of sensor nodes in Cooja that is used when evaluating energy consumption. Node marked with number 1 is the sink node, nodes 2-3 are normal nodes. One square is 10 meters wide and 10 meters high.

6.5.5.1 Energy Consumption on Z1 Motes in Cooja

This section presents the energy consumption of the Zolertia Z1 nodes simulated in Cooja. The results show the energy consumption for both a child and a parent in the system when turning the IDS on and off.

With IDS:
Table 6.26 and 6.27 shows how much time the Cooja-simulated Z1 systems spends in different hardware states during a 30-minutes test while the nodes are running both the Contiki example program called ”unicast-sender” and the IDS. The ”unicast-sender” program is configured as a high intensity network, i.e. sending messages very often, and the IDS is configured to execute verification every tenth minute.

The first table shows the time spent in different states for a parent in the network and the second table shows the time spent in different states for a child. It can be seen that a parent spends more time being in an active CPU-state than a child. This is expected since a parent performs more calculations than a child. The parent also spends more time sending and receiving messages, which is also expected since it forwards the child’s messages as well as the node is sending its own.

Table 6.26: How much time the simulated Zolertia Z1 system, acting as a parent, spends in different hardware states while running both the ”unicast-sender”, configured as a high intensity network, and the IDS for 30 minutes under normal conditions is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=32768 RTC Ticks)**
$T_{CPU\ active}$	894 722	27.30
$T_{CPU\ standby}$	58 087 707	1 772.70
$T_{Receive}$	466 126	14.23
$T_{Transmit}$	281 308	8.58

Table 6.27: How much time the simulated Zolertia Z1 system, acting as a child, spends in different hardware states while running both the ”unicast-sender” configured as a high intensity network and RoVer for 30 minutes under normal conditions is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=32768 RTC Ticks)**
$T_{CPU\ active}$	768 495	23.45
$T_{CPU\ standby}$	58 213 936	1 776.55
$T_{Receive}$	431 203	13.16
$T_{Transmit}$	261 621	7.98

Without IDS:
Table 6.28 and 6.29 shows how much the Cooja-simulated Z1 systems spends in different hardware states during a 30-minutes test while the nodes are running just the Contiki example program called ”unicast-sender” which is configured as a high intensity network, i.e. sending messages very often.

The first table shows the time spent in different states for a parent in the network and the second table shows the time spent in different states for a child. It shows a similar trend as the tables presenting the times spent in different states with the IDS turned on; the active CPU time and the send and receive times are slightly higher for the parent.

Table 6.28: How much time the simulated Zolertia Z1 system, acting as a parent, spends in different hardware states while running just the ”unicast-sender” configured as a high intensity network under normal conditions for 30 minutes is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=32768 RTC Ticks)**
$T_{CPU\ active}$	813 867	24.84
$T_{CPU\ standby}$	58 168 558	1 775.16
$T_{Receive}$	465 485	14.21
$T_{Transmit}$	268 263	8.19

Table 6.29: How much time the simulated Zolertia Z1 system, acting as a child, spends in different hardware states while running just the ”unicast-sender” configured as a high intensity network under normal conditions for 30 minutes is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=32768 RTC Ticks)**
$T_{CPU\ active}$	695 042	21.21
$T_{CPU\ standby}$	58 287 383	1 778.79
$T_{Receive}$	420 438	12.83
$T_{Transmit}$	252 379	7.70

When comparing the tables for the IDS turned on or off it can be seen that there is slightly more time spent in active hardware states and thus higher energy consumption when the network runs the IDS. A parent has 2.46 seconds longer active CPU time and a child has 2.24 seconds longer active CPU time. The increase in time spent in radio for receiving and transmitting packets is even smaller. There is only 0.02 seconds more receive time and 0.39 seconds more send time for a parent and only 0.33 seconds more receive time and 0.28 seconds more send time for a child.

6.5.5.2 Energy Consumption on SensorTag Platform

This section presents the energy consumption for the Texas Instruments cc2650STK SensorTag platform. The tables presents how much time a parent and a child node spends in different hardware states with and without the IDS.

With IDS:

Table 6.30 and 6.31 shows how much time the SensorTag spends in different hardware states during a 30-minutes test while the nodes are running both the Contiki example program called ”unicast-sender” and the IDS. The ”unicast-sender” program is configured for a high intensity network, i.e. sending messages very often and the IDS is configured to execute verification every tenth minute. The first table shows the times for a parent in the system and the second table shows the times for a child.

The tables shows that, on the contrary from the Z1 node, the SensorTag spends most of the time in active CPU mode instead of standby CPU mode. A child spends 11.9 seconds more in CPU standby mode than a parent, but on the other hand a child spend almost half a minute more in the radio receive state than a parent. However, a parent spends 22.9 seconds longer in the radio transmit state.

Table 6.30: How much time the SensorTag, acting as a parent, spends in different hardware states while running both the ”unicast-sender” program configured as a high intensity network and the IDS for 30 minutes under normal conditions is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	114 042 404	1 740.15
$T_{CPU\ standby}$	3 934 732	60.04
$T_{Receive}$	2 259 866	34.48
$T_{Transmit}$	2 713 098	41.40

Table 6.31: How much time the SensorTag, acting as a child, spends in different hardware states while running both the ”unicast-sender” sender program configured as a high intensity network and the IDS for 30 minutes under normal conditions is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	113 262 402	1 728.25
$T_{CPU\ standby}$	4 714 758	71.94
$T_{Receive}$	4 128 754	63.00
$T_{Transmit}$	1 189 694	18.15

Without IDS:
Table 6.32 and 6.33 shows how much time the SensorTag spends in different hardware states during a 30-minutes test while the nodes are running just the Contiki example program called "unicast-sender". The "unicast-sender" program is configured for a high intensity network, i.e. sending messages very often and the IDS is configured to execute verification every tenth minute. The first table shows the times for a parent in the system and the second table shows the times for a child.

The tables shows that the SensorTag platform spends more time in active CPU mode than in CPU standby mode once again also when it is not running the IDS. A child spends some seconds more in standby CPU mode than a parent does, but it spends more time in radio hardware for both sending and receiving. The time that differs is however small. A child only spends 3.4 seconds longer in receive mode and 3.69 longer in transmit mode than a parent does.

Table 6.32: How much time the SensorTag, acting as a parent, spends in different hardware states while running just the "unicast-sender" program configured as a high intensity network for 30 minutes under normal conditions is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	114 908 294	1 753.36
$T_{CPU\ standby}$	3 068 842	46.83
$T_{Receive}$	2 563 762	39.12
$T_{Transmit}$	1 126 240	17.19

Table 6.33: How much time the SensorTag, acting as a child, spends in different hardware states while running just the "unicast-sender" program configured as a high intensity network for 30 minutes under normal conditions is shown in both real-time clock (RTC) ticks and in seconds.

State	**RTC Ticks**	**Seconds (1s=65536 RTC Ticks)**
$T_{CPU\ active}$	114 437 258	1 746.17
$T_{CPU\ standby}$	3 539 904	54.01
$T_{Receive}$	2 786 674	42.52
$T_{Transmit}$	1 368 208	20.88

6.5.5.3 Energy Consumption Summary

This section summarizes the energy consumption in terms of time spent in hardware for both the simulated Z1 platform and the SensorTag platform. For each of the platforms four different measurements are made: IDS on and off for both a child and for a parent in the network. In order to compare the simulated Z1 platform and the SensorTag platform energy-wise, the tables with data that the previous sections present have been summarized into two figures where radio times are placed in one graph and CPU times are placed in another.

Figure 6.13 and 6.14 present the summary for all four individual test cases for both platforms. In the first figure the time spent in the radio hardware is shown, and in the second figure the distribution between active and standby CPU time is shown. In the figures it is visible that the difference of having the IDS on and off is minimal for all cases for the Z1 platform. The difference is also minimal when studying CPU state distribution for the SensorTag but it is visible for the platform that the IDS demands more radio traffic for control messages in the graph for radio times.

It can also be seen that the SensorTag spends more time in high energy consuming states than the simulated Z1 platform. There is more time spent in active radio states, which is costly, and the SensorTag spends almost all of its time in an active CPU state instead of going to sleep in the standby mode. The pattern is visible independent of the on or off status of the IDS and the pattern is not observable on the Z1. This indicates that the problem is not related to the IDS but rather to the platform itself.

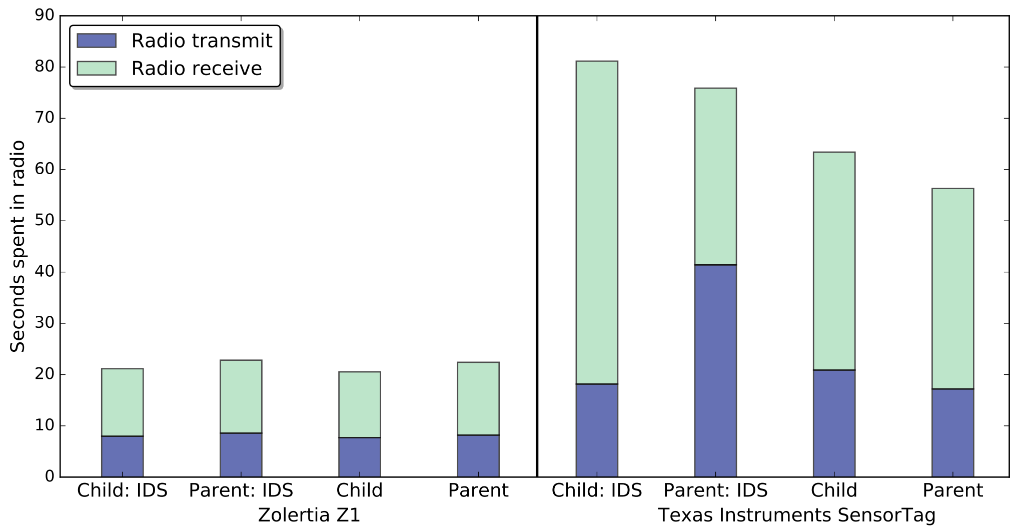


Figure 6.13: The time spent in radio hardware for both parent and child with and without the IDS activated. Both the simulated Z1 and the SensorTag is shown. It can be seen that the SensorTag spends more than double the amount of time in radio hardware compared to the simulated Z1 system. The difference between having the IDS on and off is minimal on the Z1 platform. There is more difference of having the IDS on and off on the SensorTag and it is definitively visible that the IDS demands more radio traffic for communication of control messages.

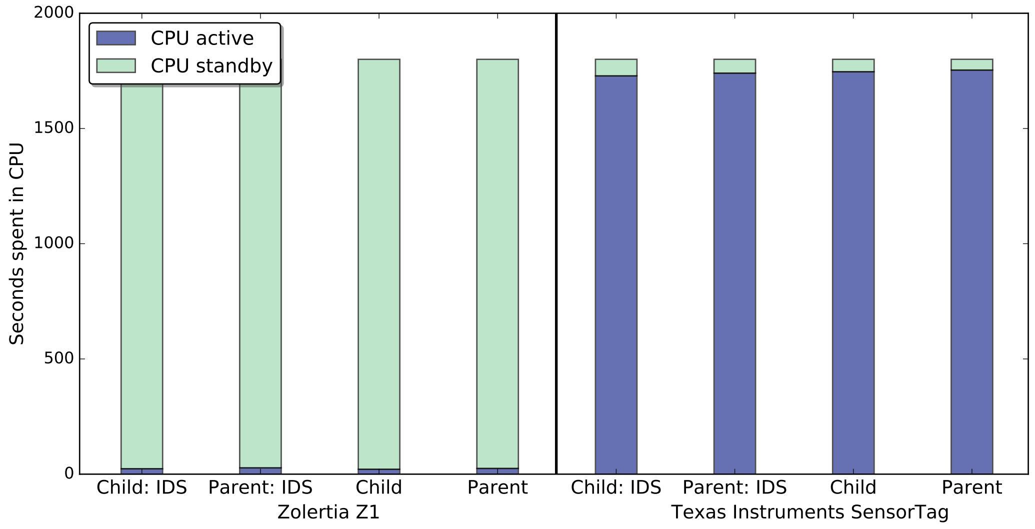


Figure 6.14: The time spent in active and inactive CPU state for both the parent role and the child role for both the Z1 platform and the SensorTag platform is shown. The Z1 platform spends most of it time in the inactive CPU state while the SensorTag spends most of it time in active CPU state. This applies for both the roles in the network. The difference between having the IDS on and off is minimal.

7 Discussion

This chapter discusses the results from the evaluation in Chapter 6. It also mentions the strong and weak points of the detection methods in this thesis and compares the obtained results with results from related work found in Chapter 3 as far as possible. Finally, the chapter introduces ideas for future work that can be done to extend the work done in this thesis.

7.1 Analysis of Evaluation

This section discusses and analyses the findings from the evaluation in Chapter 6. The results and what they implicate for both detection methods are elaborated on and discussed in their respective sections below.

7.1.1 Anomaly-based DoS-Attack Detection

For the first detection method, the first set of parameters used when conducting the tests were set after extensive testing in Cooja. The behavior in a simulator and the real world may differ a bit, and therefore it may be possible that other parameters had been chosen if the initial testing would have been performed on real hardware as well. However, setting the initial parameters on real hardware would have taken a lot more time than what had been feasible for the time scope of the thesis. Therefore the decision to test on real hardware first after the first initial parameters had been set according to experiment results from Cooja is motivated.

7.1.1.1 Detection Rates

For both the algorithms used for detection, only one low intensity attack in total was detected. As Section 6.4.2.2 mentions, it can be discussed whether this counts as an attack or not. While the attacks were run no disruption of the normal function of the network was seen. The nodes were able to send their messages with seemingly no delay, and the sink node received them as usual. It is important to determine when something counts as an attack when evaluating the detection rate of the method for both algorithms. If the threshold for when something counts as an attack is set at the medium low intensity (when the attacking node tries to send a message every 50th timer tick, or roughly 2 messages per second) the detection rate will go up from 56.25% to 75% for the LiReg algorithm and from 62.5% to 75% for the IncA algorithm. This is a significant increase of the result. By comparing the Energest value from the example program and the IDS in Figure 6.2 and the low intensity attack in Figure 6.3 it can be argued that the low intensity attack should be excluded from being counted as an attack. This conclusion originates from the fact that the Energest values for both programs deviates so little that both algorithms would need to set their corresponding parameters so low to detect the attack that they would have high false positive rates.

The number of false alarms is low for both algorithms. There is only one false alarm during the testing of the IncA algorithm and two alarms during the testing of the LiReg algorithm. Tables 6.9 and 6.11 show during which tests the false alarms happens. The intensity of the attack does not matter since the false alarms occur before the attack is launched. However, it is not surprising that the false alarms happen during the Long delay in all cases since the detection method will check the energy values more times and thus the chance of something going wrong increases. The evaluation calculates the rate of false alarms by taking the number of false alarms divided by the total number of launched attacks. This gives a relatively high rate of false positives on 12.5% and 6.5% respectively. But if the total time spent on tests are considered just one and two alarms are low. Table 6.4 shows the time each Delay have before launching the attack. The sum of these values is a total of 76 minutes, and all the delays are run for all the intensities, giving a total run time of roughly 5 hours for each algorithm. Considering this, the false positives is quite low.

7.1.1.2 Comparison to Related Work

Riecker *et al.* [4] test their system by launching a blackhole attack and a flooding attack. They present the results for the detection rate with number of false positives and false negatives for two different configurations of their system, one that is "biased" to achieve a higher detection rate. They achieve a false positive rate of 18.75% and a detection rate of 93.75% in the non-biased configuration in a system with 16 nodes. This can be compared to the two algorithms that this thesis presents, which achieves a false positive rate of 12.5% for the LiReg method and a rate of 6.25% for the IncA algorithm. It can be seen that the algorithms in this thesis perform better in terms of false positive rates. However, Riecker *et al.* achieve a detection rate of 93.75%, which is considerably higher than for both algorithms (at the best for both 75%). While keeping the false positives low is good, it is more important to actually detect attacks and it can be argued whether the algorithms presented in this thesis would have gotten a higher detection rate if the false positive rate had been allowed to be larger or not. It can be noted that Riecker *et al.* evaluate their work completely in a simulator, so the results may have been different on real hardware.

7.1.1.3 Energy Evaluation

Table 6.19 presents a summary of the total energy consumption of a node under three different conditions. When the Simple Sensor Program is run without the detection method, and when it is run with the detection method using LiReg or the IncA algorithm.

The difference in energy consumption between just running the example program and also running the IDS is as low as 6 mJ for the LiReg and 1.1 mJ for the IncA algorithm while the system is not under an attack. The overhead produced by the

IDS is relatively low in this case. The small addition to the energy consumption can be seen as quite low in contrast to gaining protection against attacks that would drain the batteries a lot faster than the IDS would. On the other hand, this is just how much extra energy the IDS uses over 30 minutes. Over time, the overhead will of course grow bigger. This trade off is something one would have to reflect over whether it is more useful to run the IDS and detect attacks or if it is better to leave the IDS out and be vulnerable against attacks, but with batteries that last longer.

The difference in power consumption in the system when it is run in an attack state or not is roughly the same for all components, roughly one joule for each of them. It is interesting to note that while under attack, the power consumption of the LiReg method is larger than the one for only the Simple Sensor Program while the power consumption of the IncA algorithm used is still smaller. This might be due to various reasons. The tests were carried out on real hardware, which makes the results non deterministic. It might just happen that the sensor program tried to send values at the same time as the attack and therefore tried to send values again.

7.1.2 RoVer

As mentioned earlier, in Chapter 6, the evaluation of the second detection method is carried out in the network simulator Cooja (except for the energy consumption). Evaluation of the second detection method on real hardware would require more nodes (and therefore more computers to read the output) and a lot of time to properly test the system and make sure that all nodes chose a parent that is not the sink. This would require a very large geographical area and would not be feasible. Hence, it is decided to only measure the energy consumption on real hardware (which will only require three nodes and therefore is feasible) while the parameterization and testing of detection rates is carried out in Cooja. Of course, by only evaluating in a simulator one might miss some important aspects of how the real system behaves. A simulator, even a really good one, is still deterministic and can not fully replace testing on real hardware. Nevertheless, all of the papers referenced in Chapter 3 use some kind of simulator to evaluate their system, probably due to feasibility. Raza *et al.* [6] claim that Cooja has been proven to produce realistic results, and therefore they are content with only using Cooja for evaluation purposes. Since all these papers have deemed it a good enough evaluation it is supportable to use a simulator to some extent in this thesis. The energy consumption is measured on real hardware in this thesis, and thus the thesis adds even further knowledge about the feasibility of running the actual IDS on real hardware, which others lack for their methods.

7.1.2.1 Detection Rates

The detection rate for the second detection method is as high as 100% since all attacks are detected. However, for the cases with the three highest values in the high intensity network the sinkhole node is detected before it starts dropping packets. A sinkhole attack consists of two stages. The first is to attract as many nodes as possible and the second is the actual dropping of packets. The behavior of just making other nodes choose the malicious node as a parent that would not have chosen the evil node as a parent otherwise causes a disturbance in the network that leads to the reporting of him as being evil, as Section 6.5.2.2 describes. This behavior is observed in three out of four cases for the High intensity network, and not at all in the Low intensity network. It will probably vary with the network setup used, were one of the most important factors probably is how many nodes that can reach the attack node and choose it as a parent. Since this behavior is observed only in the high intensity network another important factor for premature detection is probably the number of messages, but this would have to be investigated further.

In most cases the attack is detected at the end of the same detection round or in the following verification round. Therefore the time until detection is decided depending on how long one round of the detection method is. A shorter detection round will detect the attack earlier, but it will also increase the energy consumption due to the program running more often. Another issue to take into account is that a too short period will not detect the attack at all. If too few messages have been sent the threshold value for lost packets will make sure that the attack node goes undiscovered. Therefore, careful consideration will have to be taken when lowering the duration of the detection round.

The latency that Figure 6.7 presents show no overhead due to the IDS. Most of the times the latency is below 1 second independent of having the IDS activated or not, with some exceptions. In some cases the messages just take a bit longer to arrive, but some of them are lost. The longer latency time probably depends on the fact that some nodes on the route to the sink were busy with other tasks before they could send the message on to the next node.

When comparing the reliability for a network running the IDS with RoVer activated (Figure 6.9) to a network not running the IDS (Figure 6.10) it can easily be seen that all nodes affected by the sinkhole attack never recovers when RoVer is not active. Another interesting note when comparing the figures is that the reliability seems to be better in general (even before the attack is launched) for the network that are running the IDS. The IDS should not influence this for the better (or for worse) so this is probably just a coincidence. Other tests might show a different result but it is interesting to note.

7.1.2.2 Comparison to Others

Raza *et al.* [6] evaluate SVELTE against sinkhole attacks and selective forwarding attacks. They evaluate for both a lossless network in Cooja and a lossy network. For the lossy network, SVELTE achieves approximately 90% detection rate while on a lossless network it achieves almost 100% detection rate. In this thesis, the second detection method is evaluated towards a lossy network in Cooja by launching a sinkhole attack. A detection rate of 100% is achieved, which is a higher result than SVELTE since Cooja is set to a packet receive rate of 50%.

With a lossless network, SVELTE does not have any false positives at all, while they have few false positives in a lossy network. Since no exact numbers are presented it is hard to compare, but RoVer also achieves a low false positive rate of 6.25%.

7.1.2.3 Energy Evaluation

When comparing the energy consumption on the simulated Z1 nodes only a small overhead is seen due to the IDS being run. The biggest difference lies in the time spent in active CPU time for both the parent and the child. The biggest impact of the energy consumption lies in the time spent in active CPU for a parent and this amount of time is still very small. The time spent in active CPU is 1,5% when running the IDS and 1.4% when not running the IDS. Therefore the impact in energy consumption can be seen as almost negligible in comparison to gaining protection against attacks.

Section 6.5.5.3 presents two graphs that shows the time spent in different hardware states for a child and parent node in both the simulated Zolertia Z1 nodes and the Texas Instruments SensorTag. In the first graph it is interesting to note that the SensorTag platform spends a lot more time in the hardware states for sending and receiving. More than double the time in all cases regardless of the IDS is turned on or off. This is probably due to the nature of the different hardware and has probably nothing to do with the program (and the IDS) that is being run on the nodes since they are identical. Therefore further investigation into the difference between the platforms and also how Cooja emulates the Z1 nodes would be needed in order to find out why this happens.

Another interesting fact to note about the first graph is that almost all nodes spends more time receiving packets than sending them, except for the parent running the IDS on the SensorTag platform. Due to how the algorithm for RoVer is written it is expected that a parent would have much to transmit, both to its children and its own parents. The node playing the same role in the network with the Z1 mote does spend more time in the radio transmit state than the rest of the Z1 nodes, but no significant differences. This is probably also hardware specific.

When looking at the time spent in the radio hardware for the Z1 nodes both children spend less time in radio hardware than the parent regardless of the IDS or not, thus showing that the IDS has a very small overhead on the platform. On the other hand, when looking at the time spent in radio hardware for the SensorTag it can easily be seen that the IDS adds significant more time in the radio hardware when comparing the child roles with each other and the parent roles with each other. There can be many reasons for this. Since the Z1 has been on the market for longer than the newer SensorTag there would also had been more time to update and provide new, better drivers that works more effectively. Another reason could possibly be due to the fact that Z1 is emulated in Cooja, which has a more deterministic behavior than the real hardware, which might need to re-send messages several times, as an example.

The other graph also shows an interesting behavior. While the Z1 spends almost all time in CPU standby mode and has very little active time, the SenorTag does the complete opposite. Naturally wireless sensor nodes will require low power and should spend a lot of time in the standby node instead of the active mode since this reduces more energy consumption. No solution to why the SensorTag behaves in this way could be found during the thesis while investigate the source files in Contiki for the specific platform. But again, since both platforms runs the same code, the overhead could not be blamed on the IDS or the unicast-sender program.

7.2 Limitations

This section describes the limitations of the work in this thesis. The limitations are discussed not only for the two detection methods but also for the method used during the evaluation and verification of the framework.

7.2.1 Anomaly-based DoS-Attack Detection

Since the first detection method uses two different algorithms for detecting attacks it will have different strengths and weaknesses depending on which one it uses. A limitation for both of the algorithms is that once an attack is launched the list of energy readings will start to fill up with values that are much bigger than those in a system under normal energy consumption. This means that both algorithms will stop generating alarms that an attack is ongoing after only one or a few alarms, as this behavior will start to become the new "average" that they both use when determining if the system is under attack.

As argued in Section 4.2.1.2 and confirmed in Section 6.4.1.1 a weakness with linear regression is that it needs a fairly flat energy curve in order not to generate a lot of false positives. In real life scenarios, the energy consumption is seldom flat. In fact, as shown in previous chapter, the energy consumption varies a lot depending on what the node is currently doing. This problem can be somewhat overcome by choosing good parameters for the read interval and the length of the list of energy readings. This way, how big the deviation of the actual value and the estimated value can be set to a number in percent that will lower the number of false alarms while still detecting alarms.

Another weakness of both the LiReg and the IncA algorithm is that they will not detect DoS-attacks that start by sending few packets and then increase gradually. This is because if the energy consumption rises slowly enough the new energy value will always be within the difference of what is allowed. Thus, the average value of all energy readings will slowly increase and therefore the read energy consumption value can continue to rise undetected.

7.2.2 RoVer

If a malicious node has only a single child, the second detection method will probably not detect that the parent node is evil. This is due to the fact that when the parent node sends the claim about how many messages it has sent it will include its own messages in the total number of sent messages. The child will not have any knowledge of number of messages sent by the evil node and therefore it will think that the number of messages in the claim includes its own. The implementation includes a threshold value that determines how many messages the parent can lack in its claim (due to the fact that these networks are lossy) and if this value is high, one child will definitely not detect any attack. As an example, a child to a sinkhole node has sent 10 messages to the parent. The parent has sent 8 messages on his own (and dropped all of the child's packets) and will therefore claim 8 sent messages to the child. If the parameter for tolerated packet loss is set to 2, the child will be satisfied and not raise any alarm. This may not be a problem in real life scenarios since nodes in WSNs are often placed in clusters, and therefore one node should have several children, but it is important to note that this weakness exists.

A node that comes from the outside and wants to join the network would be flagged as malicious quite fast with RoVer if it does not run the RoVer code and announces the required control messages. It can be argued whether it is a limitation or not since an administrator that places the sensor nodes at their locations most likely knows which nodes that are present (and thus which nodes that should run the IDS code). The limitation is important to mention however, as it reduces the possibility of having the IDS in a mobile scenario.

7.2.3 Implementation of Attacks

The attacks used to test the detection methods are designed according to the specifications of the adversary models described in Section 4.2.2 and Section 4.3.2 and the implementation can be found in Section 5.5. Since both the adversary models and the implementation are specified and realized as part of this thesis it is known how both the detection methods and the attacks work before testing the system. It can be argued that this is a problem. It is not hard to know how to detect an attack when the exact implementation of it is known. The attacks were designed to be as general versions of the described attacks as possible, but this might not be enough. There exists a lot of different versions of attacks so it might be the case that the attacks can be implemented in such way that they will not be detected by the IDS. A good example of this is that both the LiReg and the IncA algorithms will not detect DoS-attacks that are slowly increasing, as described in Section 7.2.1. To properly test if the system detects attacks the attack should be designed by someone who does not know how the detection method works and the other way around. However, this is out of scope for the thesis since it would have needed someone not involved in the design of methods to design the attacks.

7.3 Future Work

The IDS is designed and implemented as a framework with support for adding of more detection methods to the outer shell (which Section 4.1.1 describes). As of now it can detect two types of attacks, three kinds of routing attacks and DoS-attacks based on network flooding. These are some of the most common and destructive attacks towards WSNs today, as argued earlier in the thesis, but new attacks that are more dangerous can be implemented or discovered at any point. Thus, by making sure that the framework is easy to extend will not make it outdated as soon as new attacks and algorithms for detecting them are invented.

For the first detection method further functionality could be added in order to save the ”normal” state of energy consumption when an attack occurs. This could possibly be done by saving more elements in the list and then, when an attack occurs the latest values will be replaced by the older, lower, ones. This can be done in order to generate continuous alarms that an attack is ongoing. This is just an idea of an extension, and it would need to be designed, implemented and evaluated as a new method in order to determine if it works as intended. Thereof, it is not a part of this thesis.

It might also be possible to further increase the detection rate and lower the number of false positives for the first detection method by combining the two algorithms. By combining the algorithms it might be possible to set one of them to a stricter value but not report anything until the other method agrees. However, this might not be possible to actually implement with a good result. Further research and testing will need to be performed in order to conclude if this would be feasible.

In contrary to the second detection method, the first only focuses on the process of detection and does not have a remedy for the attacks. A possible extension to this is to design a firewall for sensor nodes that keeps track of the senders IP-address and excludes the IP-address of a node that has been detected as a malicious flooding attack node. When such a node then tries to communicate with the node that runs the IDS and the firewall, it will not succeed since the firewall will drop the packet immediately. To make it more advanced, the sink node can be involved in the flagging of a node as malicious and can also be in control of firewall updates regarding adding and excluding nodes from the blacklist in the firewall.

One way that the second detection method could be extended would be to involve the sink more in the blacklisting process. The system could be redesigned to make nodes send alarms to the sink via broadcasts and re-broadcasts until the message reaches the sink. Then, the sink will not blacklist the malicious node until a second node marks the malicious node as evil. When that happens, the sink can send out blacklist updates to all the nodes and recreate the routing graph with the malicious node excluded.

Other future works includes a more thorough evaluation of the second detection method on real hardware. Even though it has been tested and verified to work on the Texas Instrumens CC2650STK platform, the evaluation (aside from the measurements of energy consumptions) was performed in Cooja. To properly test the method would require a lot of time and also computers to read the output of nodes in order to make sure that it actually work as expected.

7.4 Reflections on Ethics and Sustainability

With an intrusion detection system, it is easy to see the ethical sides of the thesis. Without protection of valuable systems, no one would be able to trust them and therefore use them for safety-critical operations. For example, if a wireless sensor network is used to monitor a forest for detection of wild fires, it would be catastrophic if someone attacks the network so that fire alarms do not reach the fire department. It would be unethical to leave such a safety-critical network without protection. The second detection method, RoVer, that this thesis proposes would solve the ethical issue by providing both detection of and remedy for such attacks of the networks. Therefore it is easy to see that this thesis performs well within an ethical perspective.

Regarding the perspective of sustainability, this thesis proposes the use of a distributed system for detection of attacks and intrusions. This means that a network needs no additional nodes to detect attacks in comparison to the use of a centralized approach. Therefore, no additional hardware is needed to protect against attacks and therefore no additional hardware needs to be manufactured which in the end means that sustainability is achieved with the use of the IDS framework. From another perspective, the IDS uses more energy on the nodes which from the perspective of sustainability is a bad thing. On the other hand, security is achieved which indicates that there is a trade-off between the ethical part and the sustainability part. Therefore, we as authors behind this thesis argue that some sustainability has to be set aside in order to achieve the ethical perspective of gained protection against attacks of vulnerable networks.

8
Conclusion

This thesis introduce an IDS framework with three different techniques for detecting attacks against wireless sensor networks. Two of the algorithms are based on energy consumption as a metric for detecting Denial of Service attacks against sensor nodes. The two algorithms fall into the category of anomaly-based detection and evaluation shows that they are not only functional but also effective at their task. The two algorithms are called LiReg and IncA and they both prove to be effective at detecting attacks with a detection rate of 75% each while still maintaining a low number of false positives on 12.5% and 6.25% respectively throughout the evaluation.

One of the greater scientific contributions of this thesis is the second detection method called RoVer which is short for Role-based Verification. RoVer is a method for securing RPL routing by verifying message statistic claims from two sides of the routing graph. RoVer demonstrates superb detection of attacks and has a detection rate of 100% and a false alarm rate of only 6.25%. Through the evaluation it has been shown that RoVer greatly improves the reliability of a network that is under attack by a routing attack by encapsulating and excluding the malicious node from the routing graph. Also, the evaluation shows that the induced latency by RoVer in the network is minimal and not noteworthy.

The results in the thesis can easily be compared to other results from state of the art research papers in the field of security for IoT and Wireless Sensor Networks. The algorithms and methods that this thesis propose perform at least as good as the techniques proposed by others in the field. The results from this thesis and from the other researchers in the field serve as a basis for the conclusion that security for the Internet of Things is no longer an issue and will be a decreasing future problem.

Bibliography

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[2] S. Burdakis and A. Deligiannakis, "Detecting outliers in sensor networks using the geometric approach," in *2012 IEEE 28th International Conference on Data Engineering.* IEEE, 2012, pp. 1108–1119.

[3] Thingsquare. (2017) Contiki: The Open Source OS for the Internet of Things. [Online]. Available: http://www.contiki-os.org/

[4] M. Riecker, S. Biedermann, R. El Bansarkhani, and M. Hollick, "Lightweight energy consumption-based intrusion detection system for wireless sensor networks," *International Journal of Information Security*, vol. 14, no. 2, pp. 155–167, 2015.

[5] C. Cervantes, D. Poplade, M. Nogueira, and A. Santos, "Detection of Sinkhole Attacks for Supporting Secure Routing on 6LoWPAN for Internet of Things," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on.* IEEE, 2015, pp. 606–611.

[6] S. Raza, L. Wallgren, and T. Voigt, "SVELTE: Real-time intrusion detection in the Internet of Things," *Ad hoc networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[7] L. Wallgren, S. Raza, and T. Voigt, "Routing Attacks and Countermeasures in the RPL-based Internet of Things," *International Journal of Distributed Sensor Networks*, 2013.

[8] A. Saeed, A. Ahmadinia, A. Javed, and H. Larijani, "Intelligent Intrusion Detection in Low-Power IoTs," *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 4, p. 27, 2016.

[9] Texas Instruments. (2017) CC13x0, CC26x0 SimpleLink™ Wireless MCU Technical Reference Manual. [Online]. Available: http://www.ti.com/lit/ug/swcu117g/swcu117g.pdf

[10] Moteiv Corporation. (2006) Ultra low power IEEE 802.15.4 compliant wireless sensor module. [Online]. Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

[11] I. El Korbi, M. B. Brahim, C. Adjih, and L. A. Saidane, "Mobility enhanced RPL for wireless sensor networks," in *Network of the Future (NOF), 2012 Third International Conference on the.* IEEE, 2012, pp. 1–8.

[12] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on.* IEEE, 2004, pp. 455–462.

[13] A. L. Colina, A. Vives, A. Bagula, M. Zennaro, and E. Pietrosemoli, *IoT in five Days.* E-Book, june 2016, rev 1.1. [Online]. Available: https://github.com/marcozennaro/IPv6-WSN-book/releases/

[14] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Local computer networks, proceedings 2006 31st IEEE conference on.* IEEE, 2006, pp. 641–648.

[15] A. Dunkels, O. Schmidt, and T. Voigt, "Using protothreads for sensor node programming," in *Proceedings of the REALWSN*, vol. 5, 2005.

[16] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," Internet Requests for Comments, RFC Editor, RFC 6550, March 2012, http://www.rfc-editor.org/rfc/rfc6550.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6550.txt

[17] G. Mulligan, "The 6LoWPAN architecture," in *Proceedings of the 4th workshop on Embedded networked sensors.* ACM, 2007, pp. 78–82.

[18] A. Dunkels, "Sicslowpan-internet-connectivity for low-power radio systems," *SICS*, 2008.

[19] Texas Instruments. (2016) Code Composer Studio (CCS) Integrated Development Environment (IDE). [Online]. Available: http://www.ti.com/tool/CCSTUDIO

[20] ——. (2016) Uniflash Standalone Flash Tool for TI Microcontrollers (MCU), Sitara Processors & SimpleLink devices. [Online]. Available: http://www.ti.com/tool/UNIFLASH

[21] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits, "Denial-of-Service detection in 6LoWPAN based Internet of Things," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on.* IEEE, 2013, pp. 600–607.

[22] A. Ghosal and S. Halder, "A survey on energy efficient intrusion detection in wireless sensor networks," *Journal of Ambient Intelligence and Smart Environments*, vol. 9, no. 2, pp. 239–261, 2017.

[23] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, "Evaluating computer intrusion detection systems: A survey of common practices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 12, 2015.

[24] I. Onat and A. Miri, "An intrusion detection system for wireless sensor networks," in *Wireless And Mobile Computing, Networking And Communications, 2005.(WiMob'2005), IEEE International Conference on*, vol. 3. IEEE, 2005, pp. 253–259.

[25] K. Weekly and K. Pister, "Evaluating sinkhole defense techniques in RPL networks," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on.* IEEE, 2012, pp. 1–6.

[26] C. Eik Loo, M. Yong Ng, C. Leckie, and M. Palaniswami, "Intrusion detection for routing attacks in sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2, no. 4, pp. 313–332, 2006.

[27] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proceedings of the 4th workshop on Embedded networked sensors.* ACM, 2007, pp. 28–32.

[28] R. Droms, "IPv6 Multicast Address Scopes," Internet Requests for Comments, RFC Editor, RFC 7346, August 2014.

[29] Zolertia. (2013) Z1 Features: Quick Hardware Tour. [Online]. Available: http://zolertia.sourceforge.net/wiki/index.php/Z1

[30] Texas Instruments. (2016) CC2650 SimpleLink™ Multistandard Wireless MCUl. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2650.pdf