NTNU
Norwegian University of
Science and Technology

Intrusion Detection System In IoT

Frederik Nygaard

Master of Science in Communication Technology
Submission date: June 2017
Supervisor: Peter Herrmann, IIK
Co-supervisor: Zeeshan A. Khan, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: Trust-Based Intrusion Detection System in IoT

Student: Frederik Nygaard

Problem description:

IoT is a hybrid network of tiny devices that often have limited memory, storage and is often battery powered. Therefore, IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) is a standardized routing protocol for IoT. RPL networks are connected as a Destination-Oriented Directed Acyclic Graph (DODAG), where the system has a border router that is connected to the Internet and nodes that are connected to the Internet through the border router.

Because IoT systems have vulnerabilities that may cause an attacker to get control over IoT devices, this is an important topic to research. Since most of the devices in IoT, have limited memory, storage and energy limitations, IoT becomes vulnerable to routing attacks. Therefore, IoT networks need a way to detect these type of attacks. For that reason, trust-based intrusion detector in RPL networks is interesting to investigate.

A trust-based intrusion detection system is a type of reputation system where a node rates other nodes, compared to how they experience the other node. Every node in the network is being observed and evaluated by the nodes that are within reach of the node that is evaluated. These nodes make an attempt to decide if the node is trustworthy. When using trust-based systems, we need trust variables at a centralized node. In this project three variables, belief, disbelief and uncertainty, which Jøsang introduces will be used.

This projects goal is to implement trust-based intrusion detection system, and test it practically. Therefore, I have to evaluate different trust-based IDS, decide which I want to implement in my project, implement them in Contiki OS and test them both with real devices and in simulation.

Responsible professor: Peter Herrmann, IIK

Supervisor: Zeeshan Ali Khan, IIK

Abstract

Intrusion detection detects misbehaving nodes in a network. In Internet of Things(IoT), IPv6 Routing for Low-Power and Lossy Networks (RPL) is the standard routing protocol. In IoT, devices commonly have low energy, storage and memory, which is why the implemented intrusion algorithm in this thesis will try to minimize the usage of these resources. IDS for RPL-networks have been implemented before, but the use of resources or the number of packets sent was too high to be successful when finding malicious nodes.

In this thesis, Trust-based Intrusion Detection System (TIDS), a trust based intrusion detection system was implemented. Each node in the network should observe and evaluate its neighbors based on whether or not they were acting according to the RPL protocol. These observation were sent to a centralized node where these observations were analyzed. The trust values used are based on Subjective Logic; values for belief, disbelief and uncertainty are used to analyze the observations received from the normal nodes. The number of detected nodes, false positives, false negatives, energy usage, memory and number of nodes were measured. An attack's detection time is also measured.

Seven different experiments were conducted based on this IDS. They were chosen to investigate the different number of nodes and attackers in the network, when the observations should be sent to the border router and whether the border router is always in listening mode. This showed that the implemented IDS captures between 50-100% of the attackers based on what type of experiment was executed and it also introduces between 2 and 15 false positives. Concerning the resource usage, the IDS uses between 5000-6400 bytes of storage and 700-1000 bytes of memory. The energy consumption in the border router increases with 488 mW when the IDS is implemented, while the normal nodes does not see an increase in the energy consumption.

The results of the thesis are promising, but needs more work to be extended to the RPL protocol. Regarding energy consumption, the border router with IDS is using more energy than without. The normal nodes however, do not have an increased energy consumption because of the IDS. The proposed IDS detects every attacker, but does also introduce false positives. The number of false positives can be reduced by improving the way sinkhole attack is captured.

Sammendrag

Inntrengingsdeteksjon er en måte å oppdage noder som oppfører seg mistenksomt i et nettverk. I tingenes internet, er Pv6 Routing for Low-Power and Lossy Networks(RPL) brukt som standard rutingsprotokollen. I IoT har de fleste enheter lav energi, lagringsplass og minne, og derfor må inntrengingsalgoritmen bruke lite ressurser. Folk har laget IDS for RPL-nettverk før, men de bruker enten for mye ressurser eller for mange pakker for å finne skadelige noder.

I denne masteren skal et tillitsbasert inntrengingsdeteksjonssystem (IDS) brukes. Hvert knutepunkt i nettverket skal observere og evaluere sine naboer basert på om de handler i henhold til RPL-protokollen. Disse verdiene vil bli sendt til en sentralisert node hvor disse observasjonene vil bli analysert. Tillitsverdiene som brukes, er basert på Subjective Logic, hvor verdier for belief, disbelief og uncertainty blir brukt til å evaluere observasjonene mottatt fra de normale noder. I denne masteroppgaven må antall registrerte noder, falske positiver, falsk negativ, energiforbruk, CPU, minne og antall noder måles. Hvor lang tid det tar føre et angrep er oppdaget vil også bli målt.

Syv forskjellige eksperimenter ble utført basert på denne IDS. De ble valgt for å undersøke forskjellig antall noder og angripere i nettverket, når observasjonene skulle sendes til border routeren, og om border routeren alltid er i lyttemodus. Dette viste at det implementerte IDS fanger mellom 50-100 % av angriperne basert på hvilket eksperiment som ble utført. Det introduserer også mellom 2 og 15 falske positiver. Når det gjelder ressursbruk, bruker IDS mellom 5000-6400 byte lagring og 700-1000 byte minne. Energiforbruket i border routeren øker med 488 mW når IDS er implementert, mens de normale noder ikke ser en økning i energiforbruket.

Resultatene av masteren er lovende, men trenger mer arbeid for å bli utvidet til RPL-protokollen. IDS bruker noen ressurser, men det vil alltid være en trade off mellom sikkerhet og brukervennlighet. Når det gjelder energiforbruk, bruker border routeren mer energi, som forventet. De normale nodene har imidlertid ikke økt energiforbruk på grunn av IDS. Det foreslåtte IDS registrerer alle angripere, men presenterer også falske positiver.

Preface

This thesis is conducted at the Department of Information Security and Communication Technology at Norwegian University of Science and Technology (NTNU). The thesis is the final project for the MSc program in Communication Technology with specialization in Information Security. The study was performed over 20 weeks, Spring 2017.

I would like to thank my supervisor, Zeeshan Ali Khan, and my responsible professor, Peter Herrmann, for guidance, good discussions and good advise during this period.

I would also like to thank Agnes Nygaard, Henrik Nygaard and Anna Li Rasmussen for proofreading my thesis, and giving me advice on how to make it better. Finally, I would like to thank everybody at the office for making this year unforgettable.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1
Introduction

Today, everything is supposed to be connected. It can be your phone connected to your kitchen supplies or your garage connected to a sensor that registers if your car runs up the driveway. It can also be medical devices communicating information that a person depends on. All these devices are going to be connected through the Internet. This is the reason why it is called the Internet of Things (IoT). IoT is a hybrid network of tiny devices that often have limited storage, memory and is often battery powered. Therefore, IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) is a standardized routing protocol for IoT. RPL networks are connected as a DODAG, where the system has a border router that is connected to the Internet and nodes are connected to the Internet through the border router. In this thesis, these tiny devices used in IoT are called a node.

There has been a lot of interest from researchers across the globe about IoT security, and most IoT systems have vulnerabilities that may cause an attacker to get control over IoT devices. Since most of the devices in IoT, have limited memory, storage and other limitations, IoT becomes vulnerable to routing attacks. In [GMS15], they discuss that intrusion detection is a way to detect misbehaving nodes, hence, a way remove some of the consequences with these attacks.

This thesis implements Trust-based Intrusion Detection System (TIDS), a trust-based intrusion detection system where the each node observe their neighbor and try to detect whether or not the neighbor are acting according to the RPL protocol.

1.1 Motivation

IoT uses a routing protocol called RPL which is designed for tiny devices with limited resources. Because of the resource limitation, some routing attacks have been introduced. Some of these attacks (selective forward attack and sinkhole attack) are discussed later in the thesis. If these attacks are not detected, there can be huge consequences: Billions of devices will be connected to the internet, and the RPL

protocol is used in applications and services where these applications are dependent of sensors. The attacks might cause this information to be dropped. In the worst case scenario, the loss of human life might be at stake, because for instance medical devices that monitor a heart rate of a human are connected to the RPL network. If these devices are disconnected to the internet, the data from the heart monitor are not able to send its data to other applications. Examples like this are the reason for why these attacks need to be detected.

1.2 Objective

The goal of this thesis is to implement TIDS, a trust based Intrusion Detection System (IDS) that is capable of working with tiny devices with limited resources. Therefore, this thesis creates an IDS, and attempt to make the system

- use limited resources
- have an acceptable detection rate, that detects the intended attacks and does not introduce false positives and negatives.
- limit the affect on speed in the RPL routing protocol.
- capture correct packets.

TIDS is inspired and developed based on the simulations of TIDI by Herrmann and Khan [KH17]. TIDI is a trust-based IDS for IoT and Khan and Herrmann simulate this IDS in MATLAB. In this thesis an IDS inspired by TIDI is implemented, and tested physically in Cooja.

1.3 Limitation

This thesis gives a good basis for further research. However, the complexity and lack of technical documentation in Contiki are reasons why there may not be enough time to finish all the objectives. The authors of [RS13] state in their paper that it is likely that every developer faces problems with developing system-level code in Contiki.

1.4 Approach

Each node in a DODAG observes the node that is within the transmission rage of a node, and evaluates whether or not these "neighbors" are acting according to the RPL protocol. For each attack the IDS is going to detect, an evaluation test is created to determine if a node is acting malicious or not. When an observation has

occurred, the IDS at some point send the observation to the border router. Based on the observations from every node in the network, the border router decides whether or not a node is malicious.

The implementation of TIDS consists of several steps: First, the attack needs to be developed, to test whether or not the attacks are detected. Second, a test to evaluate whether or not the node is acting according to the RPL protocol is needed for each attacks. The next step is to implement a way to send and receive the observations from a normal node to the border router. Finally, the last thing to implement is the update of trust in the border router. Here, the border router receives the trust-observations and based on the received data, and the previous observations, the border router decide whether or not a node is malicious.

1.5 Personal Motivation

Information security becomes more and more relevant, and more data is shared over the internet. May 12 2017, WannaCry spread across the world [Res17]. This became the largest ransomware outbreak in history. Attacks like this make me interested in information security. I also am interested in IoT, and it is known that IoT is going to be more and more involved in our life. However, because of the size of the devices, there are a lot of problems with security in IoT. Therefore, I decided to write about IoT security, hence, my topic became about routing security in IoT.

1.6 Structure of the Thesis

The rest of the thesis is structured like this. Chapter 2 introduces the research question of this thesis. Chapter 3 explains relevant theory in order to understand the experiments of this thesis. This chapter is followed by section 4 where the state of the art is explained. Chapter 5 is the methodology chapter, where the methods used are being discussed. After the methodology chapter, the implementation of the thesis is explained in detail. The implementation chapter is shown in chapter 6. An explanation of the experiments conducted is shown in chapter 7. Chapter 8 shows the results of the experiments from this thesis. These results are discussed in chapter 9. Finally, in chapter 10 a future work are presented and a conclusion is shown.

Chapter 2
Research Questions

Based on the objective of this master thesis, research questions are presented and explained. The first resource question takes resource usage of the IDS into consideration. The second question addresses the detection rate of the IDS, while the third research question addresses how often the normal nodes sends their evaluation of the neighbors to the border router. The third question is addressed because it might affect both the detection rate and the resource usage of the IDS.

RQ1: How much will the performance of the RPL protocol be reduced with the IDS?
The devices that will use this IDS might have little resources. Therefore, it is important for an IDS that is going to be used with the RPL protocol to use few resources. Examples of the limited resources are storage, memory and energy. Because of this, the resources that are analyzed are energy consumption, file size and memory usage. To answer this question, the resource usage without the IDS is measured. This is used as a reference when comparing the variables after implementing the IDS. After implementing the IDS, the same resources are measured and the data analyzed.

RQ2: Does the IDS detect the intended attack?
An intrusion detection system is evaluated based on its performance. It is important that the IDS detects the intended attacks. Hence, this thesis keeps track of how many attacks are being performed, and how many attacks the IDS detect. The attacks must be detected at an acceptable time so that the attacks can not do damage over a longer time period. To answer this research question, clear instructions of each experiment will be given:

- How many attackers will be introduced in this attack.
- How many nodes will be introduced.
- When each attack will be introduced.

- Which attack is being introduced.
- When each attack is being detected.

Based on these instructions, it is possible to quantify the attacks, the number of false positives, the number of false negatives and how long it takes for the IDS to detect each attack.

RQ3: When should observed data be sent from the normal nodes to the border router?
The intrusion detection system that is going to be implemented is a trust-based IDS. The nodes observe their neighbors' behavior and send the observations to a centralized node which decides whether or not a node is malicious. How often this data is sent from a normal node to the centralized node might potentially have huge impact on the detection rate and resource usage. Because of this, how often observations are sent to the centralized node is compared. Should the data be sent at once after an observation is done, or should the data be stored at every node and sent to the centralized node periodically?

Chapter 3
Background

In this chapter, an introduction to the theory behind the IDS is presented. This chapter explains the concept of IoT, RPL, trust-based IDS, Contiki OS and Cooja. These explanations are necessary to understand the IDS.

3.1 Internet of Things

The Institute of Electrical and Electronics Engineers (IEEE) has written a paper on trying to define the IoT[Ini12]. They propose this definition for IoT in a small environment scenario: " IoT is a network that connects uniquely identifiable " Things" to the internet. The "Things" have sensing/actuation and potential programmability capabilites. Through the exploitation of unique identifiaction and sensing, information about the "Thing" can be collected and the state of the 'Thing' can be changed from anywhere, anytime by anything." This means that IoT is a network of several small computers' communication over the internet to form a network where everything (from cars to lightbulb) can connect. Since an IoT device can be almost anything, the number of potential devices that can be connected to the IoT are estimated to be over 50 billions by 2020. Therefore, a new version of Ip has been created – IPv6 – which has a huge address space [RWV13].

3.2 RPL

IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) is a standardized distance vector routing protocol for Low Power and Lossy Networks (LLNs) that make use of IPv6. Because of the way that the devices are connected, the protocol does not have any cycles, hence no loops [For12]. DODAG prevents the cycles, by having a border router that is connected to the internet. All the devices connected to the DODAG are connected to the internet through that border router. By computing a node's position relative to the root node, the protocol avoids loops. This relative position is called a rank, and the rank increases the further away you get from the

border router. Loops are avoided by ignoring messages from a child node travelling downwards. A node has a parent, that forwards the nodes data to the border router, and might have several children. The node is responsible for forwarding the children's packets to the border router.

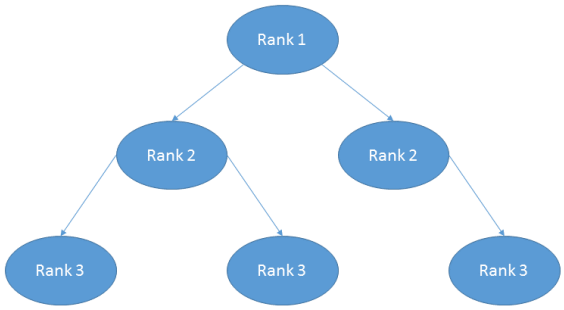

Figure 3.1: A figure of a DODAG. Here the node with rank 1 is the border router that connects the DODAG to the internet. Each node has a rank that increases the further away you get from the border router.

3.2.1 RPL Rank

In Figure 3.1, an example of a DODAG is shown. Here, the node with rank 1 is the border router connected to the internet. Also, the nodes in the RPL-network are assigned with a rank, depending on where in the network you are. The rank is used to detect loops in the network. Each node in the DODAG has a set of candidate neighbors, which is a subset of the nodes that can be reached via link-local multicast. The parent set is a restricted subset of this candidate neighbor set, and the preferred parent is a member of the parent set[For12]. A node must not advertise a rank lower than any of the members of the parent set. The parent of a node is decided based on the rank of the parent, and other parameters like for instance the energy transmission. There exist some special cases where the node's parent is not the node with the lowest rank in the parent set. However, a common way to determine a node's parent is to choose the node with the minimum rank as its appropriate parent, because this means that it is closer to the border router. A node calculates its rank frequently and takes actions based on the new rank.

3.2.2 RPL Control Messages

There are different types of control messages in RPL for information exchange and topology maintenance[For12]. All these messages are sent as an Internet Control Message Protocol version 6 (ICMPv6) type 155 (RPL Control Message) request. ICMPv6 is a supporting protocol for the internet protocol suite.

- DODAG Information Object (DIO), which is mostly used for routing control information. It for instance stores information about the roots IPv6 address. DIO messages carries information which allows a node to discover an other nodes a RPL instance, a nodes rank, information about its configuration parameters, and information about maintaining the DODAG.
- DODAG Information Solicitation (DIS) is used to enable a node to require the DIO message from a neighbour within reach. DIS messages is also used by nodes to probe its neighboorhood to find nearby DODAGs.
- The Destination Advertisement Object (DAO) messages is used to propagate information about the destination upwards.
- DAO messages can also be acknowledged by a DAO-ACK, which are sent by the DAO recipient, as a response to the DAO received. A DAO recipient is a DODAG parent or a DODAG root.

It is also possible to add new control messages to the RPL protocol. Figure 3.2 shows a RPL control message. A RPL control message consists of a type field, code field, a checksum field, a base field and a Options field. The type will always be 155 as stated above. The code field identifies what type of RPL-message it is. In section 6 of [For12] they specify what type of messages already exists. The new RPL control message types need to be specified, because RPL discard messages with an unknown Code field. The rest of the fields are ICMP type, a checksum, the actual message (base) and some options.

3.2.3 Attacks

There are several attacks that can be executed in a RPL-network. This project focuses on two attacks; selective forward attack and sinkhole attack.

Selective-Forward Attacks is an attack where the malicious node decides which packets should be forwarded or not. More specifically for RPL, a type of Selective Forward Attack, is where the malicious node drop all data-packets (for instance UDP-packets), and forwards all the RPL control packets. A potential consequence of this attack is that it could disconnect all the children of the malicious node to

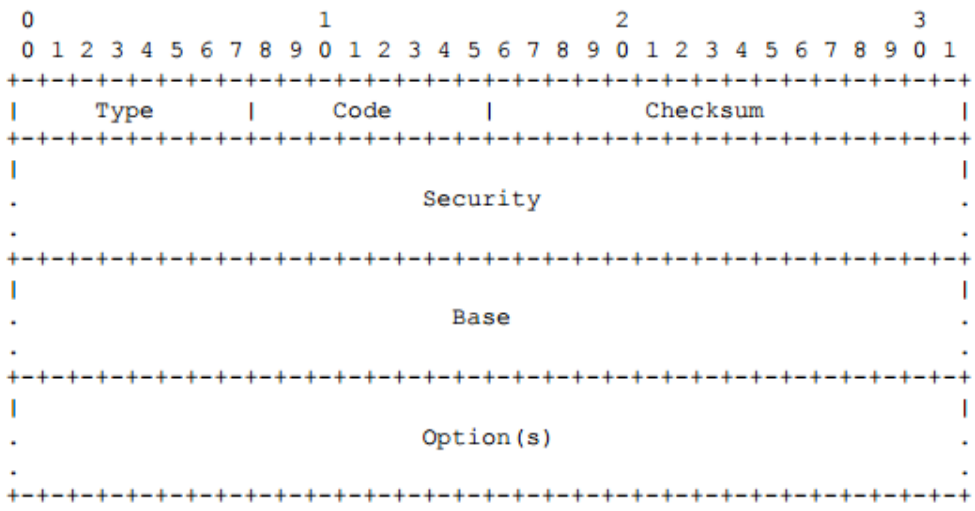

Figure 3.2: A figure that shows a RPL control message. The code field tells us which type of RPL control message it is. The figure is found at [For12]

the internet. If the data packets are not forwarded to the border router, it will not be sent to the internet. Therefore, every data packet that is sent from a child node of the attacker is not received. This attack could therefore disconnect important sensors, leading to for instance missing data, causing huge economic losses or in worst case, it can cause human lives.

A sinkhole Attack is an attack where the malicious node makes its neighbours send their traffic through the malicious node, by convincing them that through that node is the shortest path. A way to perform sinkhole attack in RPL would be for the malicious node to advertise a lower rank than the node originally should have. This would cause each neighbor to choose the malicious node as their parent, and all traffic from nodes in the range of the malicious node will send its traffic to the node controlled by the attacker. An example of consequences this attack can have is that it can be combined with the selective forward attack, causing even more nodes disconnect to the internet.

Version Number Attack , is an attack that takes advantage of the version number in the RPL. The DODAG is the only node that can change the version number for all the nodes, however, there are no mechanisms that protect the version nyumber for illegal changes. If this number gets modified by an attacker, all the nodes need to send each other control messages. This can use up most of the network.

3.3 Trust-Based IDS and Trust Computation

In this section, trust-based IDS is discussed. First an explanation of what an trust-based IDS is. Later in this section, different types of trust is introduced.

3.3.1 Trust-Based Intrusion Detection System

A reputation system is a system where actions of a node is observed and evaluated by every node that is close enough to receive its signal [PTPB10]. These nodes evaluate the node and try to determine if the node is acting according to the RPL protocol. There are several challenges with this, for instance, since the malicious node is a part of the reputation system, it can lie to the system and degrade the quality. Therefore, the system needs to be able to filter these messages. Another challenge is that non-malicious nodes might have errors, which should not be considered as a malicious node.

3.3.2 Trust Computation

Gua et al.[GCT17], shows the five design dimensions of trust computation. These five dimensions are trust composition, trust propagation, trust aggregation, trust update and trust formation. These dimensions are essential to the trust computation because it affects how to decide on trust, who stores the trust, how to combine the trust from different nodes, how often to update the trust and how each attribute affect the system. Each of these choices affects the whole IDS, both considering detection rate and how much resources is being used.

Trust Composition: This includes Quality of Service (QoS) trust and Social trust, and refers to what components to consider in trust computation [GCT17]. Social trust comes from the social relationship between owners of the IoT devices, while QoS trust refers to the belief that an IoT devise provide a responds that is of high quality. Service here is referred to as performance, cooperativeness, reliability etc.

Trust Propagation: In general, either distributed or centralized trust propagation is used to distribute trust observations to peers. This is called trust propagation. Distributed trust propagation is when IoT devices distribute trust observation autonomously to its neighbors without any central entry. In the centralized trust propagation, the trust observations is distributed to a centralized entry. For instance a border router.

Trust Aggregation: Trust aggregation refers to collecting and putting together trust observations from either its neighbors or its own observations. Section 3.4 explains one of the most investigated techniques, the subjective logic. There are

also several others, like weighted sum and Bayesian interface with belief discounting history. Weighted sum is a technique where the nodes with higher reputation, have a higher weight when the ratings are summed together. Bayesian interface with belief discounting history, is a technique where trust is a random variable, following a probability distribution where each time a new observation is observed, the model parameter is updated.

Trust Update: Trust update is about how the trust is updated. The two most common trust updates are event-trust or time-trust update. Event-trust update is when trust data of a node is updated each time an observation happens. When using time-trust update, trust data of a node is updated periodically, and trust is updated by applying a trust aggregation technique, as described above.

Trust formation: How to form the overall trust from several trust properties, is what is referred to as trust formation [GCT17]. Here, either consider single-trust, where only one trust-property is considered in a trust protocol, or, multi-trust, where there are several trust properties that should be considered. Single-trust is when all the trust properties are evaluated individually, and define a minimum threshold for each property. Examples properties could be selective forward test and sinkhole test. Another way to do trust formation is to combine individual trust properties using for instance subjective logic. This would create an overall trust metric where observations from selective forward tests and sinkhole tests is aggregated and form a combined trust. If this combined trust exceeds a minimum threshold, the node would be considered malicious.

3.4 Jøsang's Subjective Logic

Trust-Based Systems need some variables to determine other nodes trust. Subjective Logic [Jøs01] is a smart way of doing it. This does not only consider trust or distrust, but also uncertainty.

A system can be honest, dishonest, straight or crooked. If a system is honest, it behaves as it is supposed to behave, and dishonest if not. A system is straight if it follows the rules and crooked if it does not. The two most relevant combinations are honest and straight, which are called benevolent, and dishonest and crooked which are called malicious.

A node is expected to be either benevolent or malicious and one can never be a hundred percent sure that something is benevolence. Therefore, trust cannot be more than belief. The total lack of trust would signify the whole system has been compromised. However, if malicious behavior did not exist, trust would no longer be a useful measurement, because nothing is acting malicious – everything could

be trusted, no matter what. Because of this, the relevance of trust depends on the uncertainty of whether or not something is malicious. The uncertainty is a variable that tells the system whether or not a node has been observed enough to be evaluated.

Three variables are introduced, and are based on negative and positive trust valuations; belief (b), disbelief (d) and uncertainty (u)[Kna98]:

a) $b = \frac{p}{p+n+k}$ b) $d = \frac{n}{p+n+k}$ c) $u = \frac{k}{p+n+k}$

p is the positive observations, n is the negative observation and k is a constant (usually 1 or 2).

A requirement for using this technique is that the nodes should use transceivers that support idle listening mode of 1-hop neighbours data traffic. The neighbours are rated positive if they behave according to the RPL protocol and negative if they do not. These values can be sent to the border router, which decides if nodes should be removed from the network.

3.5 Contiki OS

Contiki OS is an open source operating system for IoT[Con16]. It is built for tiny devices which only have very small memory. It is a great tool for connecting these devices to the internet and it's a powerful tool to build wireless network. The image you download has Cooja already installed, which is used for simulation of a network. Contiki supports RPL and 6lowpan network, full IP networking and it is power aware.

3.5.1 RPL in Contiki

Roussel and Song [RS13] conclude that Contiki lacks of technical documentation, and that most of the developers have problems with this. This is because RPL and IP are separated from where the examples and code are written. Developers have to understand exactly how Contiki OS works.

Network stack Layer separation is a principle that the Contiki netstack is designed around, because Contiki separates between radio transceivers, routing protocols etc. The Contiki netstack has a lot of layers. The RADIO layer, is the physical driver that controls the radio transceiver and is the bottom layer. The netstack also contains the FRAMER layer, that parses and generates formatted packets. The Radio Duty Cycle (RDC) layer has control over when the radio transceiver is turned off or on during duty cycles. This can be used as an energy-saving strategy. The layer that has responsibility of ordering and sequencing packet transmission, is the MAC layer while

the top layer is the network layer where the network-level protocol implementation is. As seen in figure 3.3, each layer in Contiki communicates with a lot of other

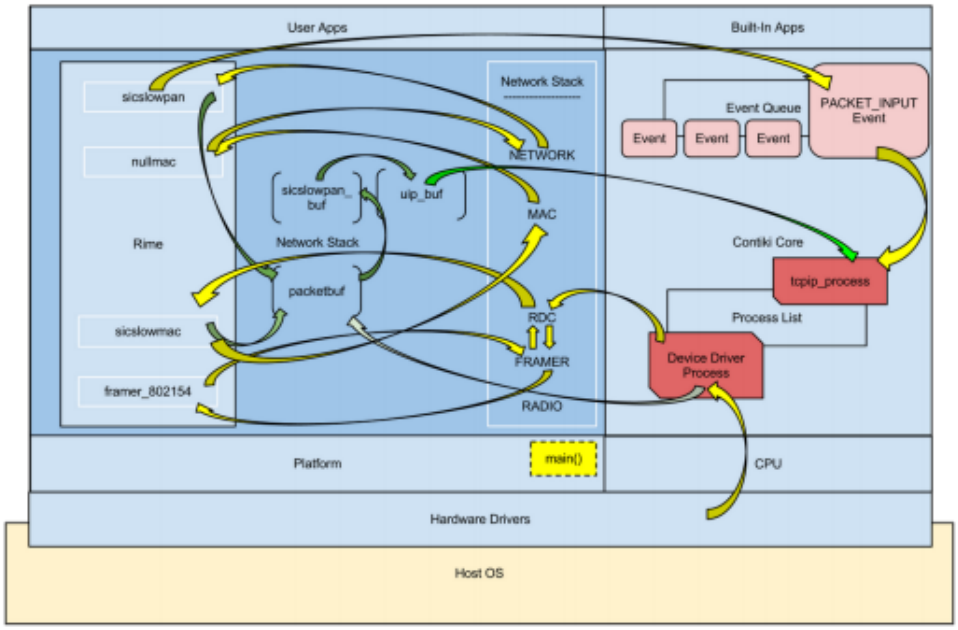

Figure 3.3: This shows a graphical illustration of the functional representation of the Contiki netstack.It shows the flow of control within the OS during a network input. Due to the complexity of Contiki OS and the lack of technical documentation, every developer is likely to get an understanding of most of these components, even though they for instance only are working on receiving RPL control-packets. Figure found at [Udi12]

components in Contiki OS. In this thesis, not every component in the figure is important. However, due to the complexity of Contiki OS and the lack of technical documentation, every developer is likely to get an understanding of most of these components. As mentioned above, the complexity of this layer structure, causes a lot of problems for developers [RS13].

MAC-layer The layer takes care of addressing and retransmitting of packets. In Contiki, there exists two different MAC-implementations: `csma.c` or `nullmac.c` [Gro17]. The nullmac acts as a pass-through protocol where it calls the correct RDC-functions. The CSMA-MAC implementation however, also keeps track of packets from neighbors, and decides whether or not to retransmit etc. CSMA is short for Carrier-Sense Medium Access. The name indicates that the implementation

of CSMA-MAC should rely on carrier sensing, however, the media access is performed by the RDC layer.

RDC-layer The RDC-layer is responsible for the nodes' sleep-periods [Gro17]. This is an important layer, since the nodes need to be awake when the node receives packets, and decides exactly when packets are received. Several different implementations of the RDC-layer are found in Contiki. Two examples of this are the `nullrdc.c` and `contikimac.c`. Nullrdc uses Framer function for creating or parsing the header. It does not put the nodes to sleep to save energy, and as the nullmac, it works as a pass-through protocol. This means that nullrdc only transmits the packet and returns the result of such an transmission.

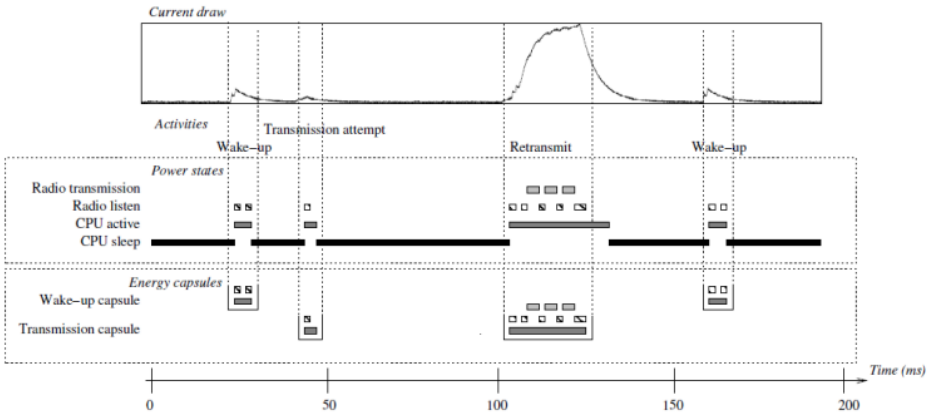

Figure 3.4: A graphical illustration of the capsule for the wake-up and transmission is shown. The transmission is tried around 40 ms, this is sensed by an other transmission and the transmission is backed off. At around 100 ms, the transmission is re-transmissioned, before it is going to be transmitted, the process needs to be woken up. In the power states, the CPU is asleep if the node is not transmitting or listening. Figure found at [GD13].

In figure 3.4, it is seen how the node is asleep, if it is not transmitting, or receiving. The capsule for the wake-up and transmission is shown, the transmission is tried around 40 ms. This is sensed by another transmission and the transmission is backed off. At around 100 ms, the transmission is retransmissioned. Before it is going to be transmitted, the process needs to be woken up. In the power states, one can see that the CPU is asleep if the node is not transmitting or listening.

Framer-layer This layer contains several functions used for creating a Frame with data. This is done for Transmission and parsing purposes. The implementation has

two main functions. One for creating a frame that is going to be transmitted and one is a parse function that takes care of the packets received.

Radio-layer Data arrives in the radio layer and it is the lowest layer in the Contiki network stack [Gro17]. The data that arrives at the node is placed into a buffer called packetbuf, which is used at every layer of the stack. After the data is read into the buffer, a polling process will cause the process to be sent into a special event and sent to the upper layers.

3.6 Cooja

Cooja is a Contiki network emulator where the code to be executed by the node is the same software uploaded to the physical nodes [BE15]. This is a tool that allows large and small networks of nodes to be simulated and tested by developers before running it on the actual physical device. This is used to debug and verify the behavior of software. To open Cooja in Contiki, go to the Cooja directory `cd contiki/tools/cooja` and run the command `ant run`.

In figure 3.5, a printscreen of a Cooja window is shown. Every window is marked with a number, that shows different views found in Cooja. The first view marked with a one in the figure, shows all the nodes in the network. View number two in the figure is where the simulation is started, stopped and reloaded. The third view in the figure is a timeline. A grey line appears if a node´s radio is on, a green line is shown when a node is receiving a packet, a blue line is shown when a packet is sent and red line if there is interfered radio. Cooja also contains a place to store notes for a simulations, as shown in view four. A useful tool in Cooja is the Radio Message view (window number five in the figure). This view shows every packet sent in the network. It also includes a network analyzer where you can store all the packets to pcap-files. Cooja also includes a view that shows the output of each node. This tool is used by developers for debugging and to verify the behavior of their software. Finally, the last view, marked as 7 in the figure, shows the energy used during the simulation.

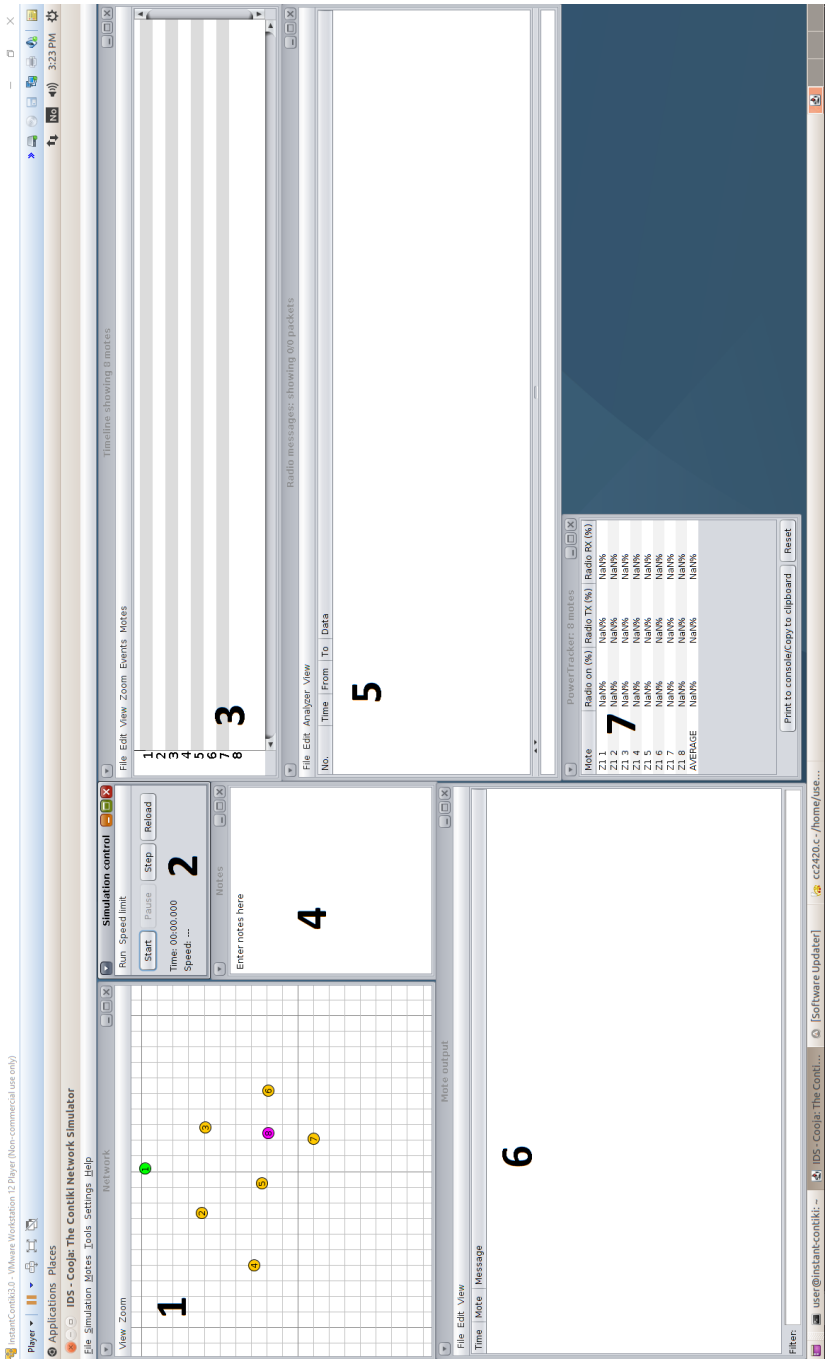

Figure 3.5: A printscreen of Cooja. Cooja is a simulation tool used in this thesis for debug and to verify the behavior of the software.

Chapter 4
State Of The Art

Intrusion detection in Internet of Things based network is well known. Already in 2013, SVELTE[RWV13] was introduced. SVELTE is an IDS in a RPL network. However, this method uses a lot of packets to detect few intruders. In 2014, Matsunaga et al. [MTS14] showed that because of the way the root compares its underlying nodes' rank, SVELTE had high false detection rate.

Cervantes et al. [CPNS15] introduced Intrusion detection of SiNkhole attacks on 6LoWPAN for InterneT of ThIngs (INTI) that detect and isolate the effects of a sinkhole attack by using a combination of watchdog, trust and repetition. However, this article mainly focuses on mobile devices and introduces false positives.

The authors of Intrusion Detection and Repsonse System (InDReS) [SU16] claimed that both INTI and SVELTE had high false positives and that these IDS overlooked resource usages. This paper shows improved resource usage, however, they do not present their detection rates.

There have been several trust and/or reputation systems in Wireless Sensor Networks (WSN) and Mobile Ad-hoc Networks (MANET). In [PTPB10] for instance, a reputation system is proposed where the root node/leader node analyzes the data given by clusters to find malicious events. However, for this to be used in IoT, it needs to be used together with other approaches. Distributed Cooperative Trust Based Intrusion Detection Architecture for MANETS (DICOTIDS) [MY11], created a trust based IDS for MANET. As for the WSN, MANET is a lot different than typical networks used in IoT. Therefore, another approach might be useful.

4.1 Intrusion detection in RPL-networks

In this section, the IDS already created for RPL networks will be addressed. This section will look at SVELTE, INTI and InDReS.

4.1.1 SVELTE

An Intrusion detection system that is already created is SVELTE, an IDS for Internet of Things. To filter out malicious traffic before it reaches the resource constrained nodes, SVELTE includes a distributed mini-firewall. SVELTE uses a hybrid, centralized and distributed approach and places the IDS module in both the border router and the nodes which are resource constrained.

There are three main modules in the border router. A 6LowPAN Mapper (6Mapper), which reconstructs the network in the border router periodically and gathers information about the RPL network by getting information about nodes neighbors. The second module in the border router is the intrusion detection component. This component analyzes the mapped data from the 6Mapper, and detects the intrusion. The last component is the mini firewall that filters out the traffic that is unwanted before it reaches the nodes with limited resources. The firewall can block the external malicious host specified in real time, by the node inside a RPL network. Each of these nodes has two centralized modules; a module that provides mapping information, and a module that works with the firewall.

In order to detect incorrect information and make sure that the information is consistent in the network, each edge in the network is checked if both nodes agree about their rank. The faulty information is corrected if a node has a large inconsistency towards a node, by changing the rank known to 6Mapper and substituting it with the information reported by some of its neighbors. If a node is detected as faulty several time, it is removed from the whitelist. SVELTE detects most sinkhole attacks by analyzing the network topology. If the routing graph is inconsistent — a child has a better rank than its parent— it is likely an attack.

SVELTE considers two attacks; Selective forwards attack and sinkhole attacks. SVELTE has good results (90% true positive rate) with sinkhole attacks for small networks, but this decreases down to 70- 80% in a larger network with 32 nodes. SVELTE has acceptable true positive rates when considering selective forwards attack, where it almost had a 100% detection rate. The network-wide energy usage shows that for network with a small number of nodes, the SVELTE overhead is very small. However, if more and more nodes connect to the border router, the overhead grows.

In 2014, Matsunaga et al. [MTS14] claims that SVELTE has a high false detection rate. The paper states false positives occur when the border router sends request between the updating of the rank, and then each node broadcasts their DIO message. Another reason for SVELTE to have high false detection rates is because neighbor nodes might not receive DIO messages due to packet loss. Due to these timing inconsistencies, SVELTE has high false detection rate.

4.1.2 INTI

INTI [CPNS15] is an intrusion detection system for sinkhole attacks. It extends the RPL protocol to be able to detect these attacks. INTI consists of four modules. It configures clusters, monitor routing, detects attacks and isolates attacks.

The clusters are created to extend the lifetime of the network and it ensures scalability. Monitoring of routing is when it counts the transmission number of input and output performed by a node that is responsible for forwarding messages. If the number of incoming messages and outcoming streams is not equal, the nodes are assumed not to function according to normal operations.

INTI uses trust and reputation to determine whether or not the node is an attacker. The last module is to isolate the module. The node that has detected the sinkhole attack generates and propagates an alarm message to alert its neighbors.

Like SVELTE, INTI is implemented in Cooja. It looks at fixed situations, where nodes are stationary. However, INTI also looks at node-mobility. Result-wise, INTI shows good result, where they almost get the same detection rate as SVELTE, and they also get detection rate over 70% for node mobility. INTI can also show low rate in false positives and negatives. However, as stated earlier, this paper focuses on mobile nodes and it introduces false positives.

4.1.3 InDRes

InDReS [SU16] develops an Intrusion Detection and Response system which relies on constraint based specification to detect sinkhole attack. It detects the node based on whether the rank of the current node summed together with the minimum increase in rank is less than the rank of the parent node. If it like that, it isolates the node and sends alert messages to the network.

InDReS has a higher delivery ratio than INTI, a higher throughput than INTI and lower normalized overhead than INTI. The energy consumption in InDRes is also lower than INTI. The paper does however, not show the detection rate of their experiment.

4.2 Trust based Intrusion Detection Systems in Mobile Ad Hoc Network

Because trust-based IDS is not researched enough, it is also important to look at how trust-based IDS is done in other networks. This section looks at how trust-based IDS is done in Mobile Ad Hoc Networks and more specifically: DICOTIDS.

4.2.1 DICOTIDS

DICOTIDS [MY11] proposes an IDS based on trust where each node runs a Local Intrusion Detection engine. Once a node detects a compromised node, they should send an IDS alert message. These messages can be dropped by fake nodes and therefore, the system needs to have a trust mechanism in the network. A distributed IDS algorithm is started when a node detects misbehavior in a neighbor node. The start of the algorithm is to broadcast IDS alert messages. The neighbor nodes share their trust data and start a diagnostic phase. After all the diagnostic data is collected, an evaluation phase starts.

The system also has an IDS analyze service that outputs Local IDS engine and the IDS alert messages. If the system detects an intrusion, this analyze service will put intrusion prevention mechanisms into effect and forward the related information to the IDS alert distribution service. This is done because the system wants to alert the other nodes in the system.

The IDS alert delivery ratio is decreased proportionally with the number of malicious nodes in the network. It states however, that its ratio satisfies the requirement of the whole system. Based on the fact that resource size of the devices in IoT however, there has to be a lot of changes for this to be used in IoT.

4.3 Trust based Intrusion Detection System in Wireless Sensor Network

In this section, a trust-based IDS in Wireless Sensor Network is explained.

4.3.1 RDAS

RDAS [PTPB10], is an IDS for Wireless Sensor Network, which is based on trust. Each node maintains its own trust variables for all of its neighbors that the node communicates with. To represent the reputation in the sensor network, they follow a Bayesian framework using the beta distribution.

To generate reputation, they use the cluster head to generate the trust, based on information received from the nodes in fixed intervals. The results are good, and simulations show that RDAS quickly detects nodes that are reporting misleading data, even when the network is becoming more compromised.

Nevertheless — as stated in the beginning of the section — this approach needs to be used together with other approaches.

Chapter 5
Methodology

In this chapter, the methods and tools used will be discussed. The topic covered in this chapter is: trust-based IDS, the trust policy, Subjective Logic, the different type of trust computations used, how the system is developed, what to measure and the research methodology.

5.1 Trust based IDS

Most of the other IDS systems in section 4 either use to much of energy, storage or memory on the IoT devices. Reputation based systems are used everywhere in the world, e.g. Ebay and Über uses it to rate their costumers. It can be interesting to test trust in an IDS system, and see how much energy, storage and memory it uses. Khan and Herrmann [KH17] simulated their IDS, but did not implement it.

5.2 Trust-Policy

There are several ways to decide whether or not a node is malicious, based on the belief, disbelief and uncertainty variables. The new trust-values could for instance be weighted higher than the old once. In TIDS, every trust-value is weighted the same and a node is considered malicious if the disbelief is higher than the belief:

TP1: If a node's disbelief-variable has a higher value than the belief variable, the node is considered malicious.

5.3 Subjective Logic

Subjective Logic is a good trust value, because it both has variables for trust and uncertainty, so it can decide if a node is trustworthy. It is also a good choice because it is easy to set trust in each of the attack referenced in section 3.2. These are only

example attacks, and if subjective logic is used, TIDS can be extended to take several other attacks into account. This is because the IDS is flexible, and if other attacks should be implemented, the IDS can easily be extended. These attacks are the most relevant routing attacks for RPL-networks to the best of the author's knowledge. Hence, these are the attacks that are taken into consideration in this project.

Forwarding Check: After node x sends a packet to node y, it puts itself to idle listening mode. This is necessary because ContikiOS has a layer called RDC-layer described in section 3.5.1, that puts nodes to sleep if they are not receiving data. This is done to save energy. If node y forwards the packet to node z, it follows the RPL protocol, and the trust should go up. Node x notices if node y forwards the packets, because even though the data is unicast, the datapacket is broadcasted at the physical layer. Therefore, if node x receives the packet, node y has forwarded it.

Sinkhole Check: Node x checks if packets from a parent resp. child has the correct rank. The node, which sends the pack cannot tell that it has a shorter path, without being detected. So depending on whether or not the rank is allowed, x increases y's p or n values respectively. One of the ways to test whether or not the child has a rank that is not correct, is to look at incoming DIO-messages. If these DIO-messages are from a direct child — the child has the node as a parent — look at the suggested rank in the DIO-message. If this rank is lower than the parents rank, there is an indication that this is a sinkhole attack. This method has a drawback in that it only works with static nodes. However, this is also a normal assumption to take.

5.4 Trust Computation

In section 3.3.2, five design dimensions of trust computations are introduced. This section explains how trust computation is done in this master thesis. The *trust composition* that is used is Quality of Service (QoS) trust. The trust is the belief that a node has good cooperativeness. For instance the node is expected to forward the packets correctly and expect that a node advertises a correct rank.

Concerning *trust propagation*, TIDS has a centralized trust propagation, where each node forwards their trust-observations to the border router as control messages. The border router then decides whether or not the node is malicious, by keeping track of all the nodes in the DODAG, and their respective trust-variables. Each time the border router receives trust-control packets, it calculates a new trust for each node, and decides whether or not to act on the new information. The reason for having the main part of IDS at the border router, is that it is assumed that this node has more resources than other nodes.

For the *trust aggregation*, this project uses subjective logic, which is described in section 3.4. Earlier in this chapter, it was described that this technique is used because it takes uncertainty into consideration, while also being flexible and having potential to be extended to include several other attacks.

As mentioned above, the *trust update* happens at the border router when it receives trust control packets. These control packets can be sent from a node when a positive or negative behavior is detected at the node. However, this could cause the network to be flooded. Another possibility is to store observation at the nodes that are not the border router and send the observations to the border router periodically. This would require more storage at the nodes, but might give better detection rate. Therefore, this is also something that is tested.

Lastly, the *trust formation* in this project is multi-trust, which means that multiple trust properties should be considered. These properties should be combined at the border router and then analyzed to check whether or not it goes below a threshold. Examples of properties in this project are Selective Forward Attacks and Sinkhole attacks. This is chosen to keep the resource usage to a minimum. If single-trust would be used, observations from each attack would have needed to be separated and more resources would be needed.

5.5 System Development

There are several steps to develop TIDS. First of all, it is important to plan how the system is going to work. This IDS detects the attacks at each node, and sends trust data to the border router. The border router will then calculate trust for each node, and based on the trust values, decide whether or not a node is malicious.

It is also crucial to develop the attacks that the IDS are going to detect. In this project, sinkhole attack is the first attack to be developed. To make sure TIDS gets developed, the IDS is created before implementing other attacks. Based on the time left after implementing the IDS, selective forward attack is added to TIDS.

To send the trust from the nodes to the border router, new RPL control messages needs to be introduced. Because the IDS is going to detect routing attacks, and some routing attacks drop data packets, the trust needs to be sent with control packets. So new RPL control packets called *Trust Information* (TRU) are introduced. These messages contain a trust variable and the IP-address of the observed node.

The border router receives the TRU-messages and based on the received information, it decides if the observed node is malicious.

5.6 Algorithms

5.6.1 Discussion about algorithms

As mentioned in section 4, there exists IDS for RPL network already, however, they have some shortcomings. The same section also mentions IDS for WSN and MANETS that uses trust to detect attacks. These are not ready for RPL networks. There is a need to find other algorithms. Herrmann and Khan [KH17] introduced algorithms with methods to stop the routing attack mentioned earlier in the thesis (section 3.2.3). Since one of the goals of this thesis is to test their result in practise, a modified version of these algorithms is used.

5.6.2 Storing Trust Observation at the Border Router

As mentioned in section 5.4, the border router is storing the trust values for each node in the network. The IDS needs to implement an algorithm for detecting attacks and sending the *TRU-Messages* to the border router. It also needs an algorithm for receiving the *TRU-Messages* at the border router, and reacting on the information in the *TRU-Messages*.

5.6.3 Methods for managing reputation

In this section, different possibilities and algorithms for how to manage reputation are discussed.

Neighbor Based Trust Dissemination (NBTD) This method makes the border router the only node that is calculating a combined trust. Each node in the DODAG sends their neighbor's trust values to the border router. Then the border router calculates the complete trust and decides on whether or not to block nodes based on the new ratings.

Clustered Neighbor Based Trust Dissemination (CNTD) assumes that the DODAG is separated into several clusters where each cluster has a cluster head that takes the role as a the border router. This structure can be useful if there are many nodes in the network. The cluster head receives the same information as the border router does in NBTD. Therefore, the cluster head gathers and combines the trust values from the cluster nodes. The cluster head also determines whether or not a node should be removed from the network. If this is the case, it notifies the border router it is connected to.

Tree Based Trust Dissemination (TTD) TTD has the same structure as CNTD, but in order to reduce monitoring overhead, child node checks the trust values for parent only. Basically, a child only evaluates the trust for parent nodes.

TIDS uses Neighbor Based Trust Dissemination because of time constraint.

5.6.4 Trust computing within a node

Sinkhole To compute trust in each node, the first step is to figure out where it is possible to detect each attack. Considering sinkhole attack, there are several different ways to detect the attack. One way is to look at the incoming DIO-messages to determine if a direct child is allowed to have the advertised rank.

If the advertised rank is not allowed, a new trust-control packet to the border router. The control packet contains the IP-address of the evaluated node and the trust-value (0 if negative trust, and 1 if positive trust). If one wants to allow different weighing of each attack, the trust value could be the positive or negative trust of that attack. If the rank is correct, a trust-control packet is sent. This method of detecting a sinkhole attack requires static nodes.

Selective Forward To detect the selective forward attack, one needs to put the node in idle-listening mode after sending a data-packet. The RDC-layer is responsible for when the node is sleeping or not. After the node is put in idle-listening mode, a timer is started.

If the parent does not forward the packet to the border router within this time, it is assumed that the parent-node is performing a selective forward attack. So if the timer goes out, a trust-control packet is sent to the border router with negative trust-value. On the other hand, if the received packet is sent from the parent, a trust-control packet with positive trust-value is sent.

When sending datapackets to the internet, the packets are sent as unicast upwards in the DODAG. The packets need to be filtered at each node. This is due to the packets are broadcasted at the physical layer, so every packet in range of the node receives the packet, as long as the node's radio is on. In order to detect selective forward attacks, the test needs to be done before the filtering of packets.

What Should be sent? An important question in this IDS is "How much data will be sent"? Another question is "What will be sent". If each node is going to store the belief, disbelief and uncertainty for each of its neighbors, including the number of positive and negative observations of each neighbor, each node uses a lot of the limited resources on storing trust data. One of the goals of the thesis is to limit the use of resources. In this thesis, only the trust observations are forwarded to the border router and the border router calculates belief, disbelief and uncertainty.

Event-trust update vs Time-trust update Event-trust update, where the data is sent to the border router once an observation has happened, might flood the

internet. Time-trust update, where the data is stored at each node and sent to the border router periodically, uses extra storage. Because both of these have drawbacks, this thesis tests both event-trust update and time-trust update compare them

5.6.5 Trust computing in the Border Router

When the border router receives the trust variables from nodes, it receives an IP-address of the node that is being evaluated. It also receives this particular node's new trust-value. The border router stores positive and negative values of each node in the DODAG. When a new trust-control packet is received, it adds one extra positive or negative trust observation based on what the trust-value contains. The new number of observation is stored and the border router starts the process of evaluating the observed node.

The first step is to calculate the belief, disbelief and uncertainty. Based on the values of these three variables, the IDS can determine if the observed node is malicious. Trust Policy 1 says that a node is malicious if the disbelief is higher than the belief. The next step is to compare the disbelief and belief as discussed above.

5.6.6 Border Router Always On?

Section 3.5.1, explains that the nodes can save energy, by going to sleep when they are not receiving or sending packets. The border router receives packets from every node, so keeping the border router on always instead of turning on and off, will minimize time processing time. However, this will introduce a lot more energy consumption in the border router. Because of this trade-off, whether the border router should stay in listening mode all the time, or not will be tested.

5.7 Measurements

This section discusses which measurements that are used, and why.

The number of detected nodes is important to measure, because it indicates how good the intrusion detection algorithms is. If the algorithm catches nearly all of the attackers, it is an indication that it is a good algorithm. However, if only a few of the attacks are caught, other IDSs should be considered.

False Positives In this system it means that a node's reputation increases negatively, even though the node is not an attacker. To measure all the false positives is important to minimize false alarms. If a system has few false positives, it is an indication of a good IDS. On the other hand if the system catches almost all the attacks, a lot of false positives can mean that the algorithm is not good.

False Negatives is an attacker that is not detected because the IDS rates the node positive. When an algorithm has a lot of fake negatives, it should not be considered a good algorithm.

Energy and Memory usage should be measured, when implementing algorithms developed for IoT devices. IoT devices are often running on battery, where the energy consumption is indirectly proportional to the lifetime of the network, and have limited memory. Hence, it is important that TIDS do not use a lot of these resources. A good algorithm uses little energy and does not store much on the devices. A fair assumption is that the border router will use most energy and memory, since it gets all of the packets from lower ranked nodes because the border router is connecting the DODAG to the internet. It should not matter that the border router uses more resources considering the border router often in general could be assumed to be a computer that is not resource constrained [RWV13].

The number of nodes is measured to determine if the system can scale. For this reason, the experiments are conducted with few nodes in simulation with several more nodes. The number of nodes should not change the measured values above significantly.

The time before an attack is detected should also be measured. Considering an IDS is implemented because the attacks can harm the system. Therefore, it is interesting to measure how long the attack actually can harm the system before it is detected. Hence, we measure the time before an attack is detected. This is the time the attack is introduced subtracted with the time the attack was introduced.

Trust update should be tested, because there is a trade off between storage at the nodes and the detection rate of the IDS. Because of this, experiments will test if the network actually is flooding the internet, so that there is no need for storing the data at the nodes that is not the border router.

5.8 Research Methodology

To ensure both repeatability and validity, this research is done in a systematically way. The research consists of three parts. The first part is a literature study determine which IDS already exists for RPL to learn which Intrusion Detection methods work. The second part of the research is to implement and run the experiment, and since this study is empirical, several simulations are executed. This is done in the Cooja simulator. This simulator is also used when developing TIDS to ensure that the attack and the IDS works. The third part of my research is to analyze the results of

the experiment and discuss and conclude if TIDS is a good IDS compared to how RPL works without an IDS.

5.8.1 Literature Study

A Literature Study has been conducted in order to identify, evaluate and interpret the relevant research in the area [KC07]. Research has little value if the literature study is not done correctly. The purpose is to learn about RPL and ContikiOS and to to identify gaps in current research. The main source for literature has been the online research database IEEEXPLORE and RFC6550 [For12] and section 4, is a result of the literature study.

5.8.2 Implementation

Contiki OS is a complex program. As stated in section 3.5, Contiki has its own implementation of RPL. To be able to complete this research, learning how this implementation works, and also to learn how each layer explained in the same section is implemented was necessary. The complete implementation and set-up phase can be found in section 6. Attacks and detection of this attack were implemented. Here, Cooja was an important tool where it is possible to see the packets sent from nodes.

5.8.3 Analyzing

In the analyzing phase, an experiment without the IDS was conducted to have measurements of the resource usage without the IDS. This is done to compare the results from the other experiments. In this way, the correct size of the IDS is measured and values to compare energy consumption with are gathered. After conducting the experiment, the results are analyzed, discussed and based on the discussion, a conclusion is presented.

Chapter 6
Implementation

This section will discusses the setup and implementation of the IDS. First, how the to collect the measurements will are explained. This is followed by the implementation of the complete IDS.

6.1 Setup

The main part of this thesis is the the simulation part, where Cooja used to simulate Zoleria Z1 nodes behavior and the programs run on the Z1. Because Cooja simulates the exact way a Z1 would, one can use the same programs in Cooja, as the real physical devices.

6.1.1 Zolertia Z1

Zolertia Z1 is the devices used in this thesis. The device's core architecture is based upon the MSP430 + CC2420 family of microcontrollers and radio tranceivers by Texas Instruments. This makes it compatible with nodes that have the same architechture [Adv10]. The Zolertia Z1 have 92 KB ROM and 10KB RAM.

In table 6.1 the approximate current consumption of Z1 circuits is seen. This is used to calculate power used in each experiment.

Table 6.1: Approximate Current Consumption of Z1 circuits [Adv10].

Notes	Operating Range	Current Consumption
IDLE mode	3V	426µA
Power down	3V	20µA
RX mode	3V	18.8mA
TX mode	3V	17.4mA

6.1.2 Measurements

In this subsection formulas for how to measure the measurement are given. The measurement given is memory, storage and energy usage

Energy To estimate the power consumption, the authors of [HNBH11] claim that using a three state model to measure energy consumption in sensors is a good way to reduce estimation errors. The consumed energy E is the sum of the total time spent in the receive state, multiplied by the respective power-level $T_{rcv} * T_{rcv}$ and the respective terms for transmit and sleep/idle. This leads to this formula:

$$E = T_{rx} * P_{rx} + T_{tx} * P_{tx} + T_{idle} * P_{idle} + T_{off} * P_{off}$$

This causes the energy consumption to be equal to:

$$E = I_{rx} * V_{rx} * T_{tx} + I_{rx} * V_{tx} * T_{tx} + I_{idle} * V_{idle} * T_{idle} + I_{off} * V_{off} * T_{off}$$

18.8 mA, 17.4mA, 426µA and 20µA is taken from table 6.1 and is I_{rx},I_{tx}, I_{idle} and I_{off} respectively and 3V is the approximated operational voltage of z1 nodes [Adv10]. Therefore $V_{rx} = V_{tx} = V_{idle} = 3V$. The time spent in the respective states, is measured in each experiment and total energy consumption is calculated. This data is printed by adding this line to the program `powertrace_start(CLOCK_SECOND * 10);`

Memory After compiling the program, the code size in the ROM and the initialized RAM can be found. This can be found by using this commmand:
`msp430-size <nameOfFile>`, where nameOfFile is the name of the file you want the find the size of. The data and bss section of this command show the statically allocated RAM. The data field consists of data that is initialized. An example of this is a variable that is not a constant. The bss field (Block Started by Symbol) [Mak] contains all uninitialized data. For instance, a variable that is declared but not initialized. Because Contiki does not really use dynamic memory allocation, this is a good indication to determine the run-time usage (excluding stack usage). In order to find the difference in initialized RAM, take the total initialized RAM with implemented IDS - the total initialized RAM without the IDS implemented.

Size For each program that is used (border-router, benevolence-client, malicious client) the size is compared to the size without the IDS. The size of the program is shown in the response from the command mentioned above (under memory). Under the text-field, which indicates all the data that ends up in Read-Only-Memory (ROM). Total size of the IDS in that file is total size of the IDS-program - total size

of the program without IDS. The size of Contiki and the program is represented in the text-field of the command-respond.

6.2 Implementation

In this section, how the IDS is implemented is explained. The code for each part of the program as well as a description of the code is given.

6.2.1 Implementing Sinkhole Attack

In order to implement a sinkhole, one needs to figure out where the rank of a node is calculated. This is done in the file located at `/home/usr/contiki/core/net/rpl/rpl-mrhof.c`. In this file, there is a method called `calculate_rank`. In algorithm 6.1, a section of this method is modified to contain the sinkhole attack.

The interesting part of this algorithm is after the else. Here, the method is checking for a flag called `button_is_pressed`. This flag is set to 1 if the malicious nodes button is pressed. If this flag is up, the rank is reduced to a rank lower than every node that is not the border router, but a little bit higher than the border router. This causes every node except for the direct children of the border router to send their traffic to the malicious node. If the attack is not initiated, the router should calculate the rank in the correct way.

Algorithm 6.1 Pseudocode for implementing sinkhole attack

```
if(INFINITE_RANK - base_rank < rank_increase) {
    /* Reached the maximum rank. */
    new_rank = INFINITE_RANK;
} else {
   /* Calculate the rank based on the new rank information
       from DIO or stored otherwise. */
    if(button_is_pressed) {
        /* If sinkhole attack is activated, calculate new rank
         * less than all nodes, but higher than the
         * border router(256) */
       new_rank = (int) (p->rank * 7  / 20);
        if(new_rank<256) {
                new_rank=256+20;
        }
    } else {
        // If there is no sinkhole attack, act as a normal node.
        new_rank = base_rank + rank_increase;
    }
}
```

6.2.2 Detecting Sinkhole Attack

As stated earlier, to detect the sinkhole attack, this project detect the attack by looking at the DIO messages. The DIO-messages are received and sent in the file located `/home/usr/contiki/core/net/ rpl/rpl-icmp6.c`. The DIO messages are received by the method called `dio_input`. Algorithm 6.2 contains the pseudocode for implementing the sinkhole detection. When the rank is copied from the `UIP_ICMP_BUFFER`, before it is processed by the method, the algorithm tests the rank of the DIO message.

The first step is to test whether or not the DIO comes from a child. This is done by going through the routing entries. `uip_ds_route_head()` returns the routes from the current node. Due to lack of documentation, the author assumes that this method returns all the routes to nodes that is its children (direct or indirect). The algorithm loops through all these routes. If the IP-address of the nexthop in the route is the same as the ip-address of the sender of the DIO, the sender of the DIO is a direct child. This direct child can not have lower rank (The rank increases the further away you get from the border router) than the current node. If this is true, we have detected a potential sinkhole attack and a *TRU-message* is sent to the border router. If the rank is correct, a *TRU-message* is also sent, however, this time a positive observation is sent to the border router.

6.2.3 Main Programs

The main programs of the border router and the normal nodes are shown in algorithm 6.3 and algorithm 6.4. The main program sends a packet to the border router in periods of SEND_INTERVAL. The send_interval period is 60 seconds. When the timer resets, the program sends data to the border router.

In the border router, there is an active waiting mechanism where a periodically timer, that each period (0.1sec) tests whether or not new data have arrived. If new trust data are arrived at the border router, it reacts to the new observation.

6.2.4 Implementing Selective Forward Attack

To perform a selective forward attack, one needs to figure out where the data-packet that is not destined for the current node is being forwarded. In contiki, this is done in the file `/home/usr/contiki/core/net/ ipv6/uip6.c`. In this file there is a section where the program prints out "Forwarding packet to ipaddr". This is where the attack is implemented. Algorithm 6.5 contains the pseudocode for how to implement a selective forward attack.

The `selective_forward_attack_flag` is 1 if the malicious node presses the button on the Zolertia device. If the attack is initiated, every packet is dropped.

Algorithm 6.2 Pseudocode for implementing sinkhole detection

```
route = uip_ds6_route_head();
while(route != NULL) {
    nexthop = uip_ds6_route_nexthop(route);
    if(uip_ipaddr_cmp(&from, nexthop)) {
        /* The sender of the DIO is a direct child
         * Test the rank of the node */
       if(currentNodesRank > DIOrank){
            /* The current node have a rank that is not
             * Allowed. Send negative observation to
             * the border router.*/
             tru_output(server_ipaddr, from, 0);
        } else {
            /*The current rank is correct
             * Send positive observation to border router*/
            tru_output(server_ipaddr, from, 1);

        }
    }
    route = uip_ds6_route_next(route);
}
```

Algorithm 6.3 Pseudocode for Main Program – Normal Nodes

```
  etimer_set(&periodic, SEND_INTERVAL);
  while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&periodic)) {
      etimer_reset(&periodic);
      ctimer_set(&backoff_timer, SEND_TIME, send_packet, NULL);
    }
  }
```

The program only checks if the flag is up, and not if the packets are for the node, or RPL control packets, because it is not possible that any of these situations can happen. Earlier in the code, a check for who the packet is destined for is performed. Hence, it is not possible to get to this part of the code if the packet is destined for this node. RPL control packets will never come to this part of the program, because this is not where these packets are filtered. If the attack is not initiated, every node should forward the packets like it normally would.

Algorithm 6.4 Pseudocode for Main Program – Border Router

```
etimer_set(&periodic, 0.1);
while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      print_data_received();
    }
    if(!(new_data < 0)) {
         isNodeMalicious(trustAddress, negative_value, positive_value);
         new_data = -1;
     }
     if(etimer_expired(&periodic)) {
          etimer_reset(&periodic);
     }
}
```

Algorithm 6.5 Pseudocode for implementing selective forward

```
if(selective_forward_attack_flag) {

    /* go to the section where packets are dropped
     * if the attack is initiated, the packet is not for me
     * and it is not a RPL_control_packet. */
    goto drop;
} else {

    /* else, forward the packet. */
    UIP_IP_BUF->ttl = UIP_IP_BUF->ttl -1;
    goto send;

}
```

6.2.5 Detecting Selective Forward Attack

The goal for detecting Selective Forward Attack was to capture unicast messages sent from a parent to its parents to confirm that the parent actually forwards the packet it sends. How this detection was supposed to work, is shown in section 5.6.4. However, after going to idle-listening mode after sending a packet, the node does not detect the packet at the physical layer. Therefore, due to time constraint, detecting

selective forward attack was not prioritized. More of this problem can be found in section 6.2.10

6.2.6 IDS implementation at the Border Router

The border router's job in the IDS is to receive *TRU-message*, and act on the information given. When receiving a *TRU-message*, the border router collects the information of the *TRU-message*. Based on this information, it runs an algorithm to decide whether or not the new observations cause the observed node to be labeled as malicious.

Algorithm 6.6 is the algorithm mentioned above. The border router contains information of all the nodes in the DODAG, and its trust observations called trustValue. Part 1 of the algorithm goes through this trustValue-list, checks if the observed nodes IP is in the trustValue-list, and save that nodes index.

Part 2 acts on the index received. If the IP is not in the list, the index is NULL, and the newly received data is added to the list. If the list already is in the list, the trust observation data is updated. Part 3 of the algorithm is where the border router decides whether or not to act on the information given in the *TRU-message*. It calculates the trust-variables defined in section 3.4. Based on these variables, the border router decides whether the node is malicious or not.

6.2.7 Developing new RPL control packets

Implementation of the *TRU-messages* happens in the same file as the DIO-messages are received and sent. Two new methods are introduced; `tru_input` and `tru_output`. The `tru_input` is the method for receiving the *TRU-messages* and `tru_output` is the method for sending the *TRU-messages*.

Algorithm 6.7 contains the pseudocode for sending a *TRU-message*. The `UIP_ICMP_PAYLOAD` is the buffer that contains the data that is sent over the internet. The first byte of the buffer contains the trustValue, and the next 16 bytes contain the observed nodes IP-address. The `uip_icmp6_send` method takes in the IP-address of the receiver, the type of the ICMP message, the code of the RPL-message and how many bytes are being sent. ICMP6_RPL and RPL_CODE_TRU are decleared in the file `rpl-private.h` located at `/home/user/contiki/core/net/rpl/rpl-private.h`.

The pseudocode for receiving *TRU-messages* are shown in algorithm 6.8. The data is received from the buffer. As in algorithm 6.7, the first byte is the trustValue, and the second to 17th byte is the IP-address of the observered node. If the trustValue

Algorithm 6.6 Pseudocode for deciding if a node is malicious.

```
isNodeMalicious(trustAddress, negative_value, positive_value) {
/*  PART 1 */
    // trustData contains an array of array where each array
    // has data like this: [ip, negative_value, positive_value].
    for(i = 0; i < trustData.length; i++ ){
        if(trustAddress == trustData[i][0]) {
            // The ip-address is already in the trustData
            index = i;
            break;
        }
    }
    /* PART 2 */
    // If index == Null, the ip is not in the trustData. Add it.
    if(index == Null) {
        newData = [trustAddress, negative_value, positive_value];
        trustData.add(newData);
    } else {
        // Store new data in trustData
        // trustData[index][1] contains the nodes neg observations
        // trustData[index][2] contains the nodes pos observations
        trustData[index][1] = trustData[index][1] + negative_value;
        trustData[index][2] = trustData[index][2] + positive_value;
    }
    /* PART 3 */
    n = trustData[index][1];
    p = trustData[index][2];
    k = 1;
    // Calculate Trust-variables:
    b = p / (p + n + k);
    d = n / (p + n + k);
    u = k / (p + n + k);
    // If d > b and u < 0.05, the node is malicious!
    if ((b < d) && (u < 0.05)) {
        print("This node is an Attacker.");
    }
}
```

is zero or one, add a negative or a positive observation, respectively, to the observed node.

Algorithm 6.7 Pseudocode for sending TRU-messages

```
void tru_output(uip_ipaddr_t *addr,
     uip_ipaddr_t *trustAddr, int trustValue)
{

  buffer = UIP_ICMP_PAYLOAD;

  // Assign the parameter values to the buffer
  buffer[0] = trustValue;
  memcpy(buffer + 1, trustAddr, 16);

  // Send the icmp message.
  uip_icmp6_send(addr, ICMP6_RPL, RPL_CODE_TRU, 17);
}
```

6.2.8 Changes to Support Time-Trust Update

Changing the IDS to be time-trust update, rather than event-trust update, require several code changes. In this subsection, these changes are discussed. Changes are necessary in the nodes that are not border routers main program, when detecting attacks, when sending and receiving *TRU-Messages* and when updating the trust.

The changes in the main program are shown in algorithm 6.9. Here, we introduce three variables that are keeping track of the neighbors and their respective positive and negative observation. These are updated when attacks are detected. Another change is the timer, because the nodes should send the trust during different time. Here, the same timer as before is used, however, the period is increased with a random time between 0 and 120 seconds. The last change in the non-malicious nodes main program, is that when the timer expire, instead of only sending a data packet, it also sends a *TRU-Message* that includes every node observed.

Where we detected the attack, several changes were needed. Algorithm 6.10 shows the changes. Before, we only sent the observation at once, using `tru_output`. This method is now called in the main program, as stated above. Now, if an observation is happening, the node updates the variables set in the main program. The program first needs to filter out the border router, because the border router should only update the trust at once instead of sending trust. This is done in the `tru_output` method. The next thing to do, is to figure out whether or not the node that sends the DIO has been observed before. Update the negative or positive value of the node if it has been observed before. If the nodes have not been observed before, add it to

Algorithm 6.8 Pseudocode for receive TRU-messages

```
static void tru_input(void)
{
    buffer = UIP_ICMP_PAYLOAD;

    // receive the values from tru-messages
    trustValue = buffer[0];
    memcpy(trustAddr, buffer + 1, 16);


    // If the trustValue is zero, add a negative observation
    // to the observed node. Else, add a positive observation
    // to the observed node.
     if(trustValue == 0) {
         negative_value = 1;
         positive_value = 0;
     } else {
         positive_value = 1;
         negative_value = 0;
     }
     new_data = 1;
}
```

the list, and update the values, based on whether or not the first observation was positive or negative.

Algorithm 6.11 shows how the node now needs to send the trust over the network. The first part is to handle the border router so that it does not send *TRU-Messages* to itself. The second part of the program is to add every observation to the *TRU-Message*. Each node keep a count of how many nodes it has observed. For each observation, add it to the *TRU-Message*. The `uip_icmp6_send` needs to change the size of the message. The size is the number of observed nodes times the size of the ipaddress and two integers.

Changes are also needed when receiving the *TRU-Messages*, to receive every observation. These changes are shown in algorithm 6.12. However, these changes are similar to the changes in sending the *TRU-Messages*. It has to capture each observation, and stores them in a temporary lists. After the data is stored at the temporary lists, the `new_data` flag is set, and the main program knows that new data are received.

In the main program, small changes are needed. This is because it needs to

update the trust for each of the observations received. Algorithm 6.13, shows these changes. Here, instead of only updating one value, loop through the observations, and react on each observation node. The border router runs with a 100% duty cycle `NETSTACK_MAC.off(1)` to ensure high packet reception rate.

Algorithm 6.9 Pseudocode for changing to time-trust update: Variables and Times

```
/* -------   In the main program:  ------- */
extern int negative_values[9];
extern int positive_values[9];
extern uip_ipaddr_t neighbors[9];
int r = rand() % 120;
etimer_set(&periodic, (r*CLOCK_SECOND) + SEND_INTERVAL);
while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&periodic)) {
      etimer_reset(&periodic);
     tru_output(server_ipaddr);
      ctimer_set(&backoff_timer, SEND_TIME, send_packet, NULL);

    }
  }
```

Algorithm 6.10 Pseudocode for changing to time-trust update: Detecting sinkhole

```
/* -------   Changes where we detect the Sinkhole ------- */
if(currentNodesRank > DIOrank){
    // tru_output(&server_ipaddr, &from, 0);
    trustAddress = from;
    if( uip_ipaddr_cmp(&server_ipaddr, &addr)) {
         trustValue = 0;
         tru_output(&server_ipaddr);
         break;
    }
    index = -1;
    for(j = 0; j< 10; j++) {
         if(uip_ipaddr_cmp(&neighbors[j], &from)) {
             negative_values[j] = negative_values[j] + 1;
             index = j;
         }
    }
    if(index < 0) {
         neighbors[count] = from;
         positive_values[count] = 0;
         negative_values[count] = 1;
         count++;
         printf("Count: %i", count);
    }
    break;
} else {
     // tru_output(&server_ipaddr, &from, 1);
     trustAddress = from;
     index = -1;
     if( uip_ipaddr_cmp(&server_ipaddr, &addr)) {
         trustValue = 1;
         tru_output(&server_ipaddr);
     }
     for(j= 0; j< 10; j++) {
         if(uip_ipaddr_cmp(&neighbors[j],&from)) {
             positive_values[j] = positive_values[j] + 1;
             index = j;
         }
     }
     if(index < 0) {
         neighbors[count] = from;
         positive_values[count] = 1;
         negative_values[count] = 0;
         count++;
     }
     break;
     }
}
route = uip_ds6_route_next(route);
```

Algorithm 6.11 Pseudocode for changing to time-trust update: Sending Trust

```
/* -------   Sending TRU-Message:  ------- */
void tru_output(uip_ipaddr_t *addr)
{
  int k;
  uip_ipaddr_t currentNodesAddr;
  get_global_addr(&currentNodesAddr);
  int pos = 0;
  //If border router: Do not send. Update trust at once.
  if(uip_ipaddr_cmp(addr, &currentNodesAddr)) {
         br_curr_neighbors[0] = trustAddr;
         if(trustValue == 0) {
             br_curr_negative_values[0] = 1;
             br_curr_positive_values[0] = 0;
         } else {
             br_curr_negative_values[0] = 0;
             br_curr_positive_values[0] = 1;
         }
         new_data = 1;
  } else {
        // Not border router.
        if(count > 0) {
        // If no nodes are observed, do nothing.
             unsigned char *buffer;
             buffer = UIP_ICMP_PAYLOAD;
            // Get the number of nodes evaluated
             set16(buffer, pos, count);
             pos = pos + 2;
             for(k = 0; k< count; k++) {
             // For each node observed, send its ip, negative and
             // positive observations.
             memcpy(buffer + pos, &neighbors[k], 16);
             pos = pos + 16;
             set16(buffer, pos, positive_values[k]);
             pos = pos + 2;
             set16(buffer, pos, negative_values[k]);
             pos = pos + 2;

             positive_values[k] = 0;
             negative_values[k] = 0;
        }
    uip_icmp6_send(addr, ICMP6_RPL,
        RPL_CODE_TRU, 2 + (count*(16 + sizeof(int)*2)));
    }
    }
}
```

Algorithm 6.12 Pseudocode for changing to time-trust update: Receive Trust

```
/* -------   Receiving TRU-Message:  ------- */
static void tru_input(void)
{
//Initialize data
uip_ipaddr_t trustAddr;
unsigned char *buffer;
int k;
buffer = UIP_ICMP_PAYLOAD;
int pos = 0;

//The number of observed nodes
incomingCount = get16(buffer, pos);
pos = pos + 2;
for(k = 0; k< incomingCount; k++) {
//Put each received observation into a temp. list.
     memcpy(&trustAddr, buffer + pos, 16);
     pos = pos + 16;
     posObs = get16(buffer, pos);
     pos = pos + 2;
     negObs = get16(buffer, pos);
     pos = pos + 2;

     br_curr_negative_values[k] = negObs;
     br_curr_positive_values[k] = posObs;
     br_curr_neighbors[k] = trustAddr;
    }
new_data = 1;
}
```

Algorithm 6.13 Pseudocode for changing to time-trust update: Update Trust

```
/* -------   Receiving TRU-Message:  ------- */
NETSTACK_MAC.off(1);
etimer_set(&periodic, 0.1);
while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    }
    if(!(new_data < 0)) {
        for(i = 0; i < count ; i++) {
        //For each observed node:
             if(br_curr_neighbors[i] != NULL) {
             //Update trust
                 reactOnTrustValue(br_curr_negative_values[i],
                 br_curr_positive_values[i], br_curr_neighbors[i]);
         }
         br_curr_negative_values[i] = 0;
         br_curr_positive_values[i] = 0;
         br_curr_neighbors[i] = 0;

    }
     new_data = -1;
    }
    if(etimer_expired(&periodic)) {
      etimer_reset(&periodic);
    }
  }
```

6.2.9 Build program to devices

This subsection informs about how to transfer the programs over to the Zolertia Z1, connect the device and build it to the device. The border router builds the `udp-server.c`, the benevolent node will build `udp-client.c`, and the different malicious attackers build their own program, based on the attack. To build the programs to the node, the program needs to have a *Makefile*, which is used to compile code with different parameters. Then build and upload the program to the device. This is done by the line `make udp-server.upload TARGET=z1`.

6.2.10 Implementation Problems

As stated in section 6.2.4 there were some problems with implementing the detection of the selective forward attack. The idea was to figure out whether or not the sender of a data packet is a parent by looking at unicasted messages at the physical layer,

where the packets are broadcasted. Programming the nodes to be in idle listening mode after sending a data packet was not a problem. However, the packets were not received at the physical layer for other packets than the one it is unicasted to.

In the physical layer, debugging messages were added to figure out who receives the messages. In the physical layer, the only node which printed out that it received a packet was the node that the data packet was unicasted to. This could indicate that the packet is filtered out another place. Figuring out this problem is left for future work.

6.2.11 The complete code

The complete code can be found at https://github.com/freddyny/Master---IDS

Chapter 7
Experiments

In this section, the experiment of this thesis is introduced. For each experiment, a table similar to table 7.2 will be introduced. This is in order to make it easy to determine what will happen. A figure of the network setup will also be given for each experiment.

The experiments conducted in this master thesis are shown below. Here, "listening mode on" means that the border router always is in listening mode, while "listening mode off" means that the border router is not always on.

- **BENCH**: Introduction experiment to benchmark the resource usage without the IDS installed.
- **Ten nodes Event-trust istening OFf (TEOF)**: One attacker, ten nodes, event-trust update, listening mode off.
- **Fifty nodes Event-trust listening OFf (FEOF)**: Ten attackers, fifty nodes, event-trust update, listening mode off.
- **Fifty nodes Time-trust listening ON (FTON)**: Ten attackers, fifty nodes, time-trust update, listening mode on.
- **Ten nodes Time-trust listening ON (TTON)**: One attacker, ten nodes, time-trust update, listening mode on.
- **Ten nodes Event-trust listening ON (TEON)**: One attacker, ten nodes, event-trust update, listening mode on.
- **Fifty nodes Time-trust listening ON (FEON)**: Ten attackers, fifty nodes, event-trust update, listening mode on.

Table 7.1 shows an overview of the experiments conducted in this thesis. The table shows whether an experiment is event-trust update or time-trust update, if the

Table 7.1: A table showing an overview of the experiments executed in this thesis. Here, "listening mode on" means that the border router always is in listening mode, while "listening mode off" means that the border router is not always on.

Experiment	Attackers	Nodes	Trust Update	Listening Mode
BENCH	0	10	None	Off
TEOF	1	10	Event-Trust	Off
FEOF	10	50	Event-Trust	Off
FTON	10	50	Time-Trust	On
TTON	1	10	Time-Trust	On
TEON	1	10	Event-Trust	On
FEON	10	50	Event-Trust	On

border router is always in listening mode and how many nodes and attackers there are in the experiment.

The abbreviations are created based on how many nodes there are in the network, whether event-trust or time-trust is used, and if the border router is in listening mode all the time, or not. If the experiment has fifty nodes in the network, the experiment's abbreviation starts with an F. If it contains ten nodes, it starts with T. The second letter in the acronym is based on the trust update. If event-trust is being used, the second letter is E, while it is T if time is used. The last two letters are based on whether or not the border router is in listening mode. The last two letters will be ON or OFF respectively.

7.1 Experiment 1: Getting results without the IDS for comparing

BENCH: Introduction of the experiment to benchmark the resource usage without the IDS installed.
This experiment is executed to compare how much resources the IDS exactly uses. The experiment consists of 10 nodes, where one node is a border router and the rest of the nodes are non-malicious nodes. The network does not contain any attackers, because the goal is to see how many resources the RPL-network uses without this project's IDS.

Table 7.2: Experiment 1 setup: The experiment consists of 10 nodes, where one node is a border router and the rest of the nodes are non-malicious nodes. The network does not to contain any attackers, because the goal is to see how many resources the RPL-network uses without this project's IDS.

Number of Attackers	0
Number of Nodes in the network	10
Experiment time	4 hours and 30 min
Border Router Listening Mode	Always on
Type of attack	None
How often *TRU-Messages* are sent	None sent
Time when attacks are introduced	[]

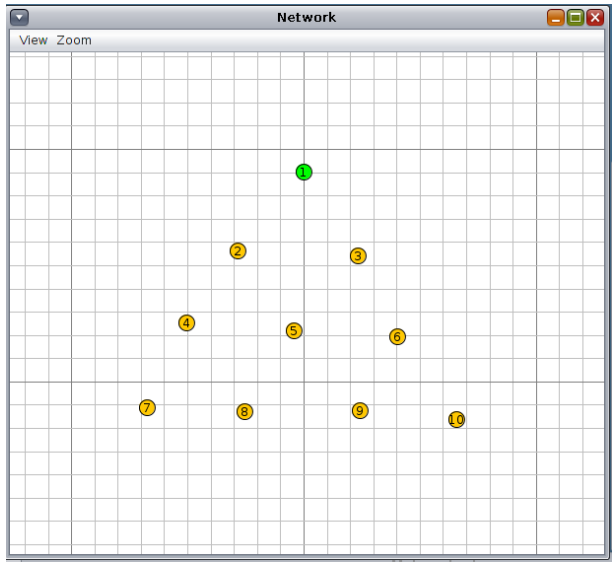

Figure 7.1: Experiment 1: Network Setup: This figure shows the network setup for experiment 1. The border router is the green node and the rest of the node are non-malicious nodes.

7.2 Experiment 2: Energy Consumption with Event-Trust Update

Ten nodes Event-trust listening OFf (TEOF): One attacker, ten nodes, event-trust update and listening mode off.
The second experiment to be conducted is an attack to prove that the IDS works, and detects attacks. Table 7.3 shows the experiment setup for this experiment. There will

be one attacker in this experiment, since the goal of this experiment is to prove that the IDS works. The number of nodes including attackers is 10, to keep the network small. The expected time is less than four and a half hours, thus the experiment time is set to four and a half hours. In this experiment, the *TRU-Messages* (the trust information messages sent from a node to the border router) are sent right after the node makes an observation about a neighbor node. The attack is introduced after five minutes to allow the network to be correctly set up.

Figure 7.2 shows how the network setup for this experiment works. Here, node nr 1 is the border router, node 2-9 are benevolent nodes, while node 10 is the attacker. In this experiment, node 3 is the most likely parent for node 10, which will be the node to detect the sinkhole attack. This is because only the parent checks whether the rank of a direct child is allowed.

This experiment is executed because it shows how many resources the IDS uses. The energy consumption, memory and filesize usage are measured. Hence, this experiment is used to answer RQ1. This experiment will give an indication on how much extra resources the IDS gives to the RPL protocol.

Table 7.3: Experiment 2 setup: There is one attacker in this experiment, since the goal of this experiment is to prove that the IDS works. The number of nodes including attackers is 10 to keep the network small. The expected time is less than four and a half hours, thus the experiment time is set to four and a half hours. In this experiment, the *TRU-Messages* are sent right after the node makes an observation about a neighbor node. The attack is introduced after five minutes to allow the network to be correctly set up.

Number of Attackers	1
Number of Nodes in the network	10
Trust Update	Event-based
Border Router Listening Mode	Varies
Experiment time	4 hours and 30 min
Type of attack	Sinkhole attack
How often *TRU-Messages* are sent	Right after detection
Time when attacks are introduced	[5 min]

7.3 Experiment 3: More nodes, and more attackers

Fifty Nodes Event-Trust Listening off (FEOF): Ten attackers, fifty nodes, event-trust update and listening mode off.
The third experiment tests how the IDS works with several nodes and several attackers. Will the IDS detect every attacker? Questions like this is why this experiment is

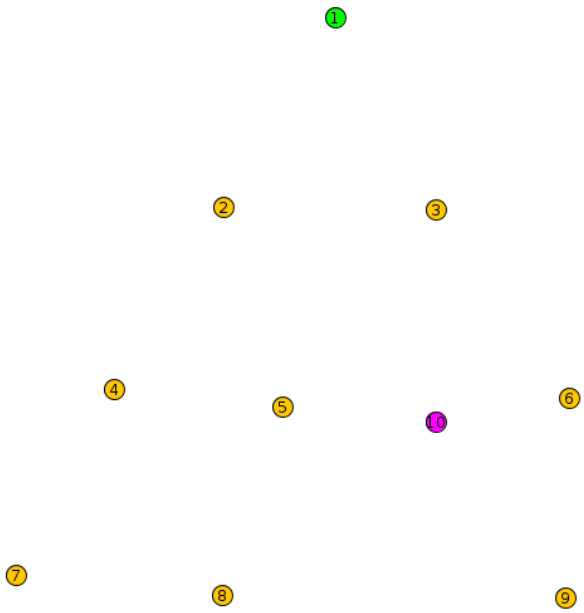

Figure 7.2: Experiment 2: Network Setup: Here, the border router is shown in green, the non-malicious nodes are shown in yellow and the malicious sinkhole attack is shown in purple. In this experiment, there are 10 nodes, one border router and one malicious sinkhole node.

performed. The experiment setup is shown in table 7.4. In the experiment, there are 50 nodes where 10 of the nodes are attackers. Each attacker performs a sinkhole attack. The experiment time is 4 hours and 30 min and each attack is introduced after five minutes.

In figure 7.3, the network setup for experiment 3 is shown. Here, the green node with ID:1 is the border router. The yellow nodes with ID:2-ID:40 are the benevolent nodes and the purple nodes with ID:41 to ID:50 are the malicious nodes that introduces the sinkhole attacks.

As with experiment 2, this experiment is performed because its result can help determine answers to RQ1 and RQ2. The difference between experiment 2 and this experiment is the number of nodes. Therefore, this experiment shows whether or not more nodes in the network will affect the performance of the IDS.

Table 7.4: Experiment 3: setup: In the experiment, there are 50 nodes where 10 of the nodes are attackers. Each attacker performs a sinkhole attack. The experiment time is 4 hours and 30 min and each attack is introduced after five minutes.

Number of Attackers	10
Number of Nodes in the network	50
Trust Update	Event-based
Border Router Listening Mode	Varies
Experiment time	4 hours and 30 min
Type of attack	Sinkhole attack
How often *TRU-Messages* are sent	Right after detection
Time when attacks are introduced	Every attack: after 5 minutes

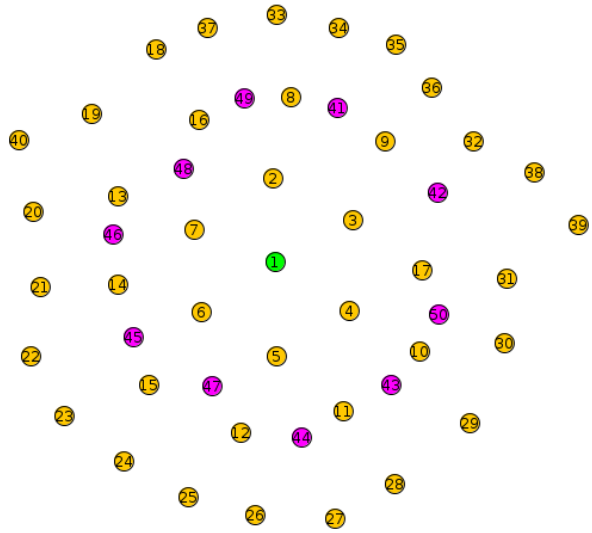

Figure 7.3: Experiment 3: Network Setup: Here, the border router is shown in green, the non-malicious nodes are shown in yellow and the malicious sinkhole attack is shown in purple. In this experiment, there are 50 nodes, one border router and ten malicious sinkhole nodes and the rest is non-malicious.

7.4 Experiment 4: Will sending data periodically improve the detection rate?

Fifty Nodes Time-Trust Listening on (FTON): Ten attackers, fifty nodes, time-trust update and listening mode on.
This experiment is set up in the same way as experiment 3. It has fifty nodes in the

network where ten of the nodes are malicious and perform a sinkhole attack. The experiment setup is shown in figure 7.5. The only difference between experiment three and four is that this time, the trust update is time-based, rather than event-based.

In figure 7.4, the network setup for experiment 3 is shown. Here, the green node with ID:1 is the border router. The yellow nodes with ID:2-ID:40 are the benevolent nodes and the purple nodes with ID:41 to ID:50 are the malicious nodes that introduces the sinkhole attacks.

This experiment is executed because if the network ends up being flooded with trust messages, there is a risk that not all the messages are received at the border router. Because of this, the detection rate might be lower because not every packet is received at the border router. This could improve the IDS, and affect RQ2.

Table 7.5: Experiment 4: setup: In the experiment, there are 50 nodes where 10 of the nodes are attackers. Each attacker performs a sinkhole attack. The experiment time is 4 hours and 30 min and each attack is introduced after five minutes. In this experiment, the observations are sent to the border router periodically every fourth to sixth minute.

Number of Attackers	10
Number of Nodes in the network	50
Trust Update	Time-based
Border Router Listening Mode	Always on
Experiment time	8 hours
Type of attack	Sinkhole attack
How often *TRU-Messages* is sent	Every fifth ±one minute
Time when attacks are introduced	Every attack: after five minutes

7.5 Experiment 5: Energy Consumption with Time-Trust Update

Ten Nodes Time-Trust Listening on (TTON): One attacker, ten nodes, time-trust update and listening mode on

The fifth experiment is very similar to experiment two, but here, the nodes' update the trust using time-trust update. Table 7.6 shows the setup of this experiment. The experiment introduces one attacker, and test whether or not the attack actually detects the intended attacker. There are 10 nodes in the network, in order to keep the network small. The expected time for the attacker to be detected is under 4 and a half hours. Hence, the experiment lasts for 4 and a half hours. In this experiment, the *TRU-Messages* are sent periodically with a period of 5 minutes ±one minute.

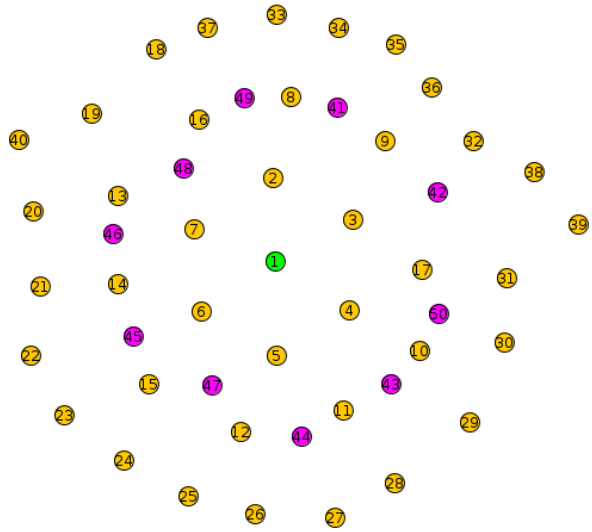

Figure 7.4: Experiment 4: Network Setup: Here, the border router is shown in green, the non-malicious nodes are shown in yellow and the malicious sinkhole attack is shown in purple. In this experiment, there are 50 nodes, one border router and ten malicious sinkhole nodes and the rest is non-malicious.

The attack is introduced after five minutes, to allow the network to be correctly set up before the attack is introduced.

Figure 7.5 how the network setup for this experiment. Here, node nr 1 is the border router, node 2-9 are benevolent nodes, while node 10 is the attacker. In this experiment, node 3 is the most likely parent for node 10, which is the node to detect the sinkhole attack. This is because only the parent checks its direct child if their rank is allowed.

This experiment is executed because it shows how many resources the IDS is using. The energy consumption, memory usage and filesize are measured. Hence, this experiment will be used to answer RQ1. This experiment gives an indication on how much extra resources the IDS introduces to the RPL protocol.

7.6 Experiment 6: Energy Consumption with Event-Trust Update and Border Router always in Listening mode

Ten Nodes Event-Trust Listening on (TEON): One attacker, ten nodes, event-trust update and listening mode on.
The sixth experiment is similar to experiment two and five, but here the nodes update

Table 7.6: Experiment 5 setup: There is one attacker in this experiment, since the goal of this experiment is to prove that the IDS works. The number of nodes including attackers is 10, to keep the network small. The expected time is less than four and a half hours, so the experiment time is set to four and a half hours. In this experiment, the *TRU-Messages* are sent to the border router periodically. The border router is always in listening mode and the attack are introduced after five minutes to allow the network to be correctly set up.

Number of Attackers	1
Number of Nodes in the network	10
Trust Update	Event-based
Border Router Listening Mode	Always on
Experiment time	4 hours and 30 min
Type of attack	Sinkhole attack
How often *TRU-Messages* is sent	Right after detection
Time when attacks are introduced	[5 min]

the trust using event-trust update and the border router is always in listening mode. Table 7.7 shows the setup of this experiment. The experiment introduces one attacker, and test whether or not the attack actually detects the intended attacker. There are 10 nodes in the network, in order to keep the network small. The expected time for the attacker to be detected is under 4 and a half hours. Hence, the experiment lasts for 4 and a half hours. In this experiment, the *TRU-Messages* are sent right after the node makes an observation about a neighbor node. The attack is introduced after five minutes to allow the network to be correctly set up. In this experiment, the border router is always in listening mode. This is the difference between this experiment and experiment two.

Figure 7.6 shows the network setup of this experiment. Here, node nr 1 is the border router, nodes 2-9 are benevolent nodes, while node 10 is the attacker. In this experiment, node 3 is the most likely parent for node 10, which is the node to detect the sinkhole attack. This is because only the parent checks its direct child if their rank is allowed.

This experiment is executed because it shows how much resources the IDS uses. The energy consumption, memory and filesize usage are measured. Hence, this experiment is used to answer RQ1. This experiment gives an indication on how much extra resources the IDS introduces to the RPL protocol.

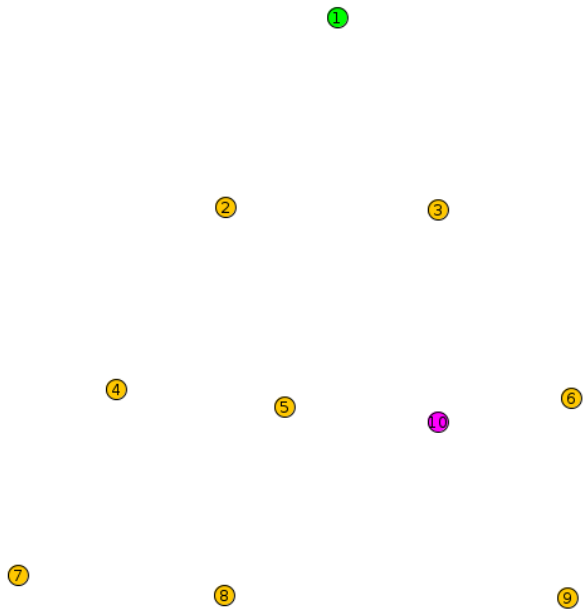

Figure 7.5: Experiment 5: Network Setup: Here, the border router is shown in green, the non-malicious nodes are shown in yellow and the malicious sinkhole attack is shown in purple. In this experiment, there are 10 nodes, one border router and one malicious sinkhole node.

7.7 Experiment 7: More Nodes: Energy Consumption with Event-Trust Update and Border Router always in Listening mode

Fifty Nodes Time-Trust Listening off (FEON): Ten attackers, fifty nodes, event-trust update and listening mode on.
The seventh experiment setup is shown in table 7.8. In the experiment, there are 50 nodes where 10 of the nodes are attackers. Each attacker performs a sinkhole attack. The experiment time is 4 hours and 30 min and each attack is introduced at five minutes. This experiment is using event-trust update with the border router always in listening mode.

In figure 7.7, the network setup for experiment 7 is shown. Here, the green node with ID:1 is the border router. The yellow nodes with ID:2-ID:40 are the benevolent nodes and the purple nodes with ID:41 to ID:50 are the malicious nodes that introduces the sinkhole attacks.

This experiment is performed because its result can help determine answers to

Table 7.7: Experiment 6 setup: There is one attacker in this experiment, since the goal of this experiment is to prove that the IDS works. The number of nodes including attackers is 10, to keep the network small. The expected time is less than four and a half hours, so the experiment time is set to four and a half hours. In this experiment, the *TRU-Messages* are sent to the border router at once. The border router is always in listening mode and the attack is introduced after five minutes to allow the network to be correctly set up

Number of Attackers	1
Number of Nodes in the network	10
Trust Update	Event-based
Border Router Listening Mode	Always on
Experiment time	4 hours and 30 min
Type of attack	Sinkhole attack
How often *TRU-Messages* is sent	Right after detection
Time when attacks are introduced	[5 min]

RQ1, RQ2 and RQ3. The difference between experiment 3, experiment 4 and this experiment is that this experiment uses event-trust update with the border router always on.

Table 7.8: Experiment 7: setup: In the experiment, there are 50 nodes where 10 of the nodes are attackers. Each attacker performs a sinkhole attack. The observations are sent to the border router at once and the border router is always in listening mode. The experiment time is 4 hours and 30 min and each attack is introduced at five minutes.

Number of Attackers	10
Number of Nodes in the network	50
Trust Update	Event-based
Border Router Listening Mode	Always on
Experiment time	4 hours and 30 min
Type of attack	Sinkhole attack
How often *TRU-Messages* are sent	Right after detection
Time when attacks are introduced	Every attack: Every 5th min

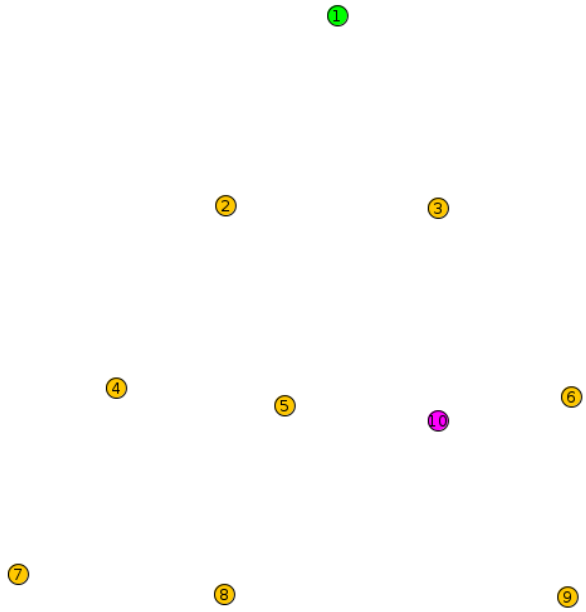

Figure 7.6: Experiment 6: Network Setup: Here, the border router is shown in green, the non-malicious nodes are shown in yellow and the malicious sinkhole attack is shown in purple. In this experiment, there are 10 nodes, one border router and one malicious sinkhole node.

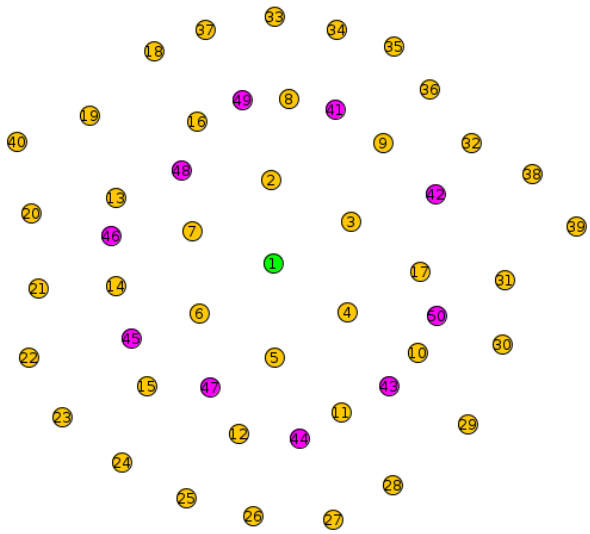

Figure 7.7: Experiment 3: Network Setup: Here, the border router is shown in green, the non-malicious nodes are shown in yellow and the malicious sinkhole attack is shown in purple. In this experiment, there are 50 nodes, one border router and ten malicious sinkhole nodes and the rest is non-malicious.

Chapter 8
Results

In this section, the results of the experiments are presented. The results are presented for each experiment. While the experiment with 10 nodes is meant for proving that the intrusion detection system works, and to measure resources used, while the experiments with 50 nodes are conducted to test the intrusion detection system.

8.1 Experiment 1

Table 8.1: Experiment 1: Results. The total average energy consumption at benevolent nodes is 243 mW after 4 and a half hour, while the total energy consumption in the border router is 140 mW

Average energy consumption in benevolent nodes	191,138 mW
Total energy consumption in the broder router	22841,834 mW

Figure 8.1: A printscreen from experiment 1 that shows the ROM and RAM size of the non-malicious node and the border router, represented as "udp-client.c" and "udp-server.c" respectively

Figure 8.1 shows the memory and filesize of the benevolent node and the border router. The border router's file name is `udp-server.z1` and the non-malicious filename is `udp-client.z1`. For the border router saved in ROM is 45864 bytes, while for the border router stored $328 + 4902 = 5230$ bytes. The regular nodes stored 45830 bytes, and $328 + 4958$ bytes.

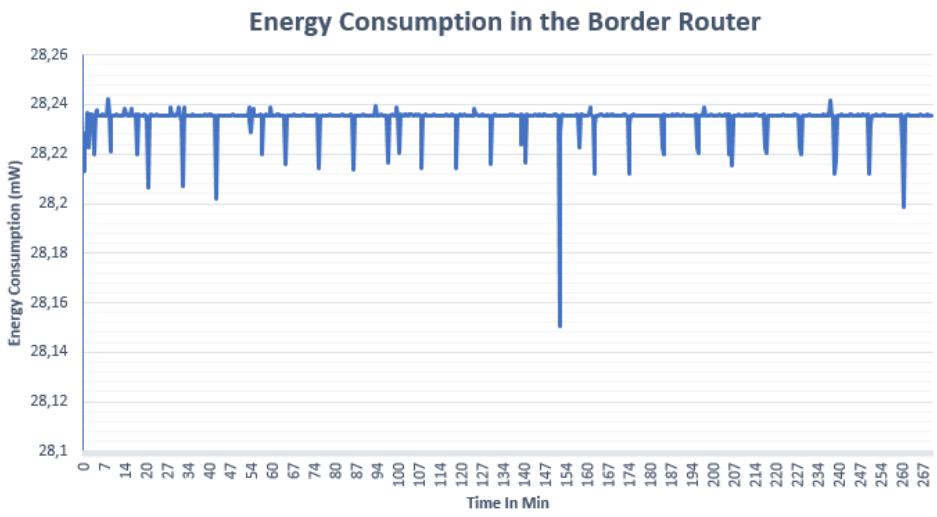

Figure 8.2: A graph that shows energy consumption in the border router during experiment 1.

Concerning energy consumption, figure 8.2 shows the energy consumption of the border router during this experiment. In the graph, you see that when the border router is consuming average 28.23 mW per 20 seconds. This is because the border router always is in listening mode. As stated in table 8.1, the total energy consumption of the border router is 22841,834 mW over a 4 hours and 30 minutes period of time.

Figure 8.3, shows a randomly chosen benevolent nodes energy consumption, after comparing it to the other nodes, the author has decided that it is representative for the other nodes. The figure shows that when a non-malicious node is not receiving or transmitting, the node uses around 0.1mW. We also see that the nodes spend around 0,3 mW and have peaks between 0,5 to 1,5mW. The average total energy consumption in a non-malicious node is shown in table 8.1: 191,138 mW.

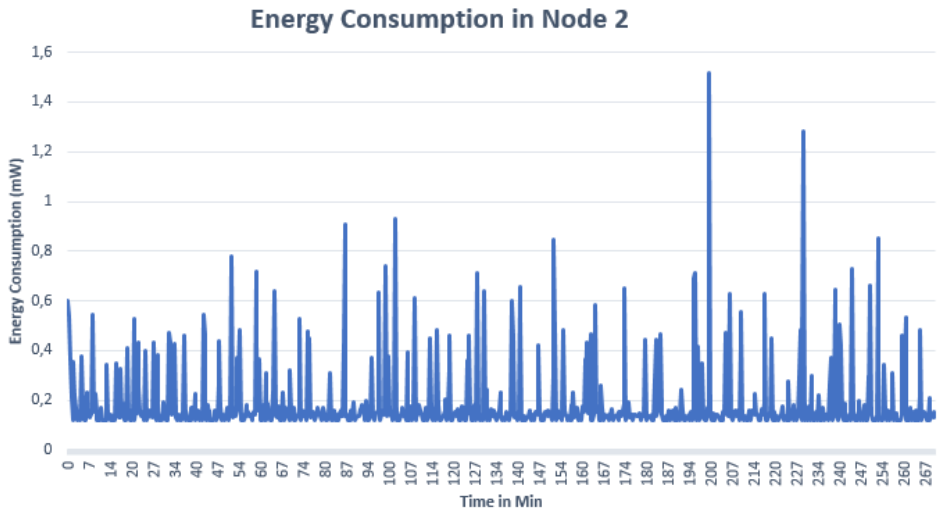

Figure 8.3: A graph that shows energy consumption in node 2 from figure 7.1 during experiment 1

8.2 Experiment 2

Table 8.2: Experiment 2: Results. The attacker got captured, with 0 false negative and 0 false positives. Energy consumption at the border router was high, while the average energy consumption at the non-malicious nodes was fair. The table also shows that the IDS have raised the total file size and the memory usage of both the border router and the benevolent nodes.

Average energy consumption in benevolent nodes	265,409 mW
Total energy consumption in the broder router	628,011 mW
Attackers detected	1/1
False Positives	0
False Negatives	0
Total size of the IDS in border router	5480 bytes
Total size of the IDS in the non-malicious node	5366 bytes
Memory usage of the IDS in the border router	786 bytes
Memory usage of the IDS in the normal nodes	748 bytes

Figure 8.4 shows the memory and filesize of the benevolent node and the border router. The border router's filename is `udp-server.z1` and the benevolent nodes

```
user@instant-contiki:~/Documents/IDS$ msp430-size udp-server.z1
   text    data     bss     dec     hex filename
  51344     298    5718   57360    e010 udp-server.z1
user@instant-contiki:~/Documents/IDS$ msp430-size udp-client.z1
   text    data     bss     dec     hex filename
  51196     298    5736   57230    df8e udp-client.z1
```

Figure 8.4: A printscreen from experiment 2, that shows the ROM and RAM size of the non-malicious node and the border router, represented as "udp-client.c" and "udp-server.c" respectively

filename is `udp-client.z1`. The figure shows that the file size of the border router and the normal nodes are 51344 and 51196 bytes, respectively. The memory usage of the system is $298 + 5718$ bytes for the border router and $298 + 5736$ bytes for the non-malicious nodes. The total memory usage and filezise of the IDS are shown in 8.2.

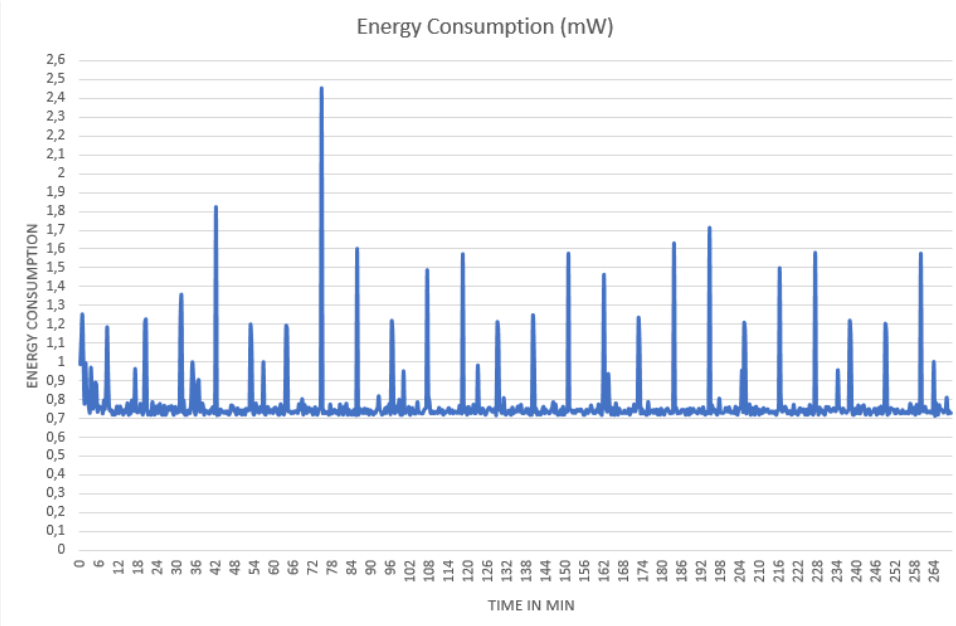

Figure 8.5: A graph that shows energy consumption in the border router during experiment 2.

Figure 8.5 shows the energy consumption of the border router in experiment two. During this experiment, the border router does not use the low power mode of the CPU. When the border router is not sending or receiving, the border router uses

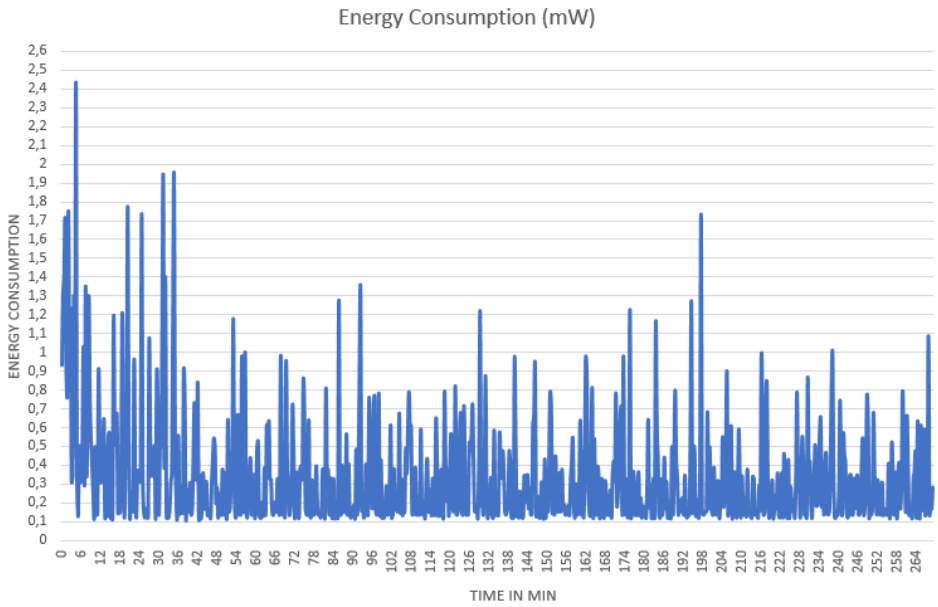

Figure 8.6: A graph that shows energy consumption in node 2 from figure 7.2 during experiment 2

Table 8.3: Experiment 2: Time of captured purposed malicious nodes for the first time

Node 10	04 hours, 3 minutes and 32 seconds

around 0.7mW. The peaks in figure 8.5 are when the border router is receiving and transmitting at the same time(for instance at 75 min). The total energy consumption of the border router is 628 mW after four and a half hours.

For the non-malicious nodes, figure 8.6 shows the energy consumption of a arbitrary non-malicious node. This graph is similar to the rest of the non-malicious nodes energy consumption, and therefore, it is representative for them. As you can see, when the node is in idle, it spends 0,1mW. It has peaks when the node is sending and receiving packets. The highest energy consumption comes after four and a half minute. The energy consumption is at 2,4 mW at this time. The average total energy consumption of the non-malicious nodes are 265 mW after four and a half hours.

Table 8.3 shows the time of detecting potential attackers. The only entry of this experiment is node 10, which, as stated in the experiment section, is the attacker. This attacker was detected by the IDS at 4 hours, 3 minutes and 32 seconds. Because

EnergyConsumption CPU	EnergyConsumption LPM	EC Trans	EC List	Total mW
0,639967236	0	0,21585388	0,13382263	0,98964375
0,639	0	0,21561493	0,26532166	1,11993658
0,63899025	0	0,37714691	0,23847107	1,25460823
0,6390039	0	0,16224884	0,21162048	1,01287322
0,639	0	0	0,14001892	0,77901892
0,6390039	0	0,21585388	0,14242859	0,99728637
0,63899805	0	0	0,19871155	0,8377096
0,6389922	0	0	0,17013977	0,80913197
0,63900195	0	0	0,12418396	0,76318591
0,63899805	0	0	0,09500977	0,73400782
0,6390039	0	0,21601318	0,11867615	0,97369323
0,63900585	0	0	0,15051819	0,78952404
0,63899025	0	0	0,11454529	0,75353554
0,63900585	0	0	0,19699036	0,83599621
0,63899805	0	0,16216919	0,0930304	0,89419763
0,6390039	0	0,16105408	0,08519897	0,88525695
0,6389961	0	0	0,1002594	0,7392555
0,63900195	0	0	0,13089661	0,76989856
0,63899415	0	0	0,11721313	0,75620728
0,6390078	0	0	0,1240979	0,7631057
0,6389961	0	0	0,11772949	0,75672559
0,6389961	0	0	0,08993225	0,72892835
0,63900975	0	0	0,15921021	0,79821996
0,6389961	0	0	0,11385681	0,75285291

Figure 8.7: A printscreen of the first few minutes of Energy Consumpiton data collection at the border router. This set of data is representative for the rest of the data.

this is the only entry in table 8.3, this is the only attacker the IDS identified. Hence, there are no false positive or false negative.

In figure 8.7, a print screen of the data of the first few minutes of energy consumption at the border router. The data represents the four hours, because the data is similar to the presented data. The data shows that the border router do not spend time in IDLE mode.

8.3 Experiment 3

Table 8.4: Experiment 3: Results. The figure shows the number of detected attackers, false positives and false negatives

Attackers detected	5
False Positives	5
False Negatives	5

Table 8.4 shows the number of detected attackers, false positives and the number of false negatives. Here, the number of correct detected attackers is five. The number of detected non-malicious (false positives) nodes is also five. The number of false negatives is the number of attackers in the system subtracted with the number of detected attacker nodes: $10 - 5 =$ five false negatives.

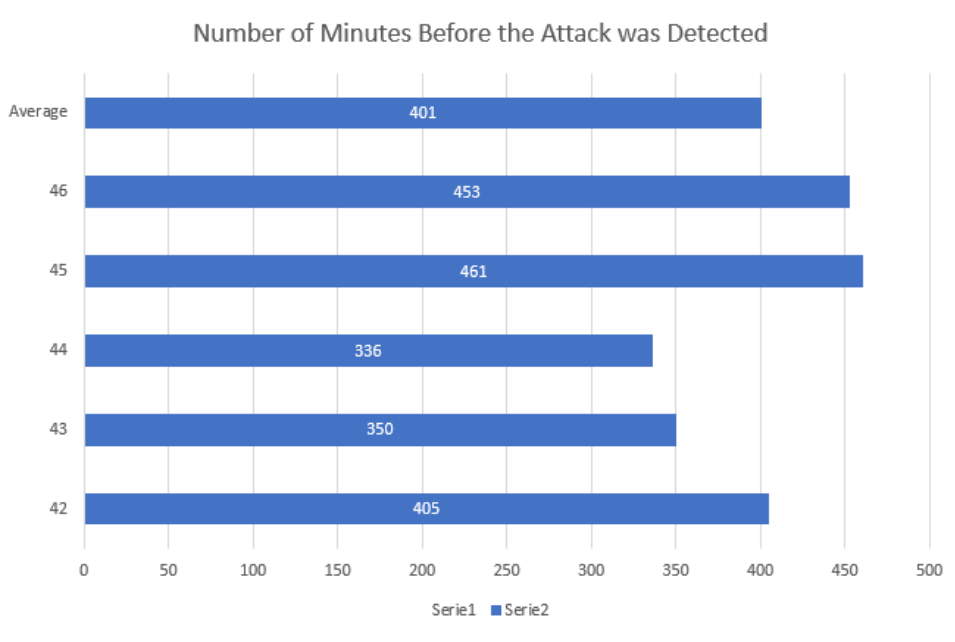

Figure 8.8: A figure that shows the number of minutes before each of the five attackers that were detected actually were detected. The average number of minutes before the attacks were detected was six hours and 41 minutes. The first attack that was detected was detected 5 hours and 36 minutes after the attack was introduced. The last attack that was detected was 7 hours and 41 minutes after the attack was introduced.

When each of the attacks detected, is detected is shown in figure 8.8. Here, you can see that the average time it takes to detect an attack is six hours and 41 minutes. The first attack was detected at 5 hours and 36 minutes, while the last attack detected was detected 7 hours and 41 minutes after the attack got introduced.

```
5:47:11.482   ID:30   Sending received trust about node 21 to BR.
5:47:15.214   ID:20   Sending received trust about node 28 to BR.
5:47:40.236   ID:30   Sending received trust about node 21 to BR.
5:47:48.144   ID:21   Sending received trust about node 29 to BR.
5:48:02.732   ID:30   Sending received trust about node 21 to BR.
5:48:24.704   ID:46   Sending received trust about node 22 to BR.
5:48:35.896   ID:3    Sending received trust about node 2 to BR.
5:48:48.483   ID:30   Sending received trust about node 21 to BR.
5:49:24.068   ID:18   Sending received trust about node 27 to BR.
5:49:32.119   ID:1    Negative trust received about node 27. nTrust: 3, pTrust: 3
5:49:57.216   ID:32   Sending received trust about node 23 to BR.
```

Figure 8.9: A figure that shows that several nodes is sending trust to the border router, but only one out of nine arrives at the border router.

```
85: 15.4 D C1:0C:00:00:00:00:00:05 C1:0C:00:00:00:00:00:2E|IPv6|ICMPv6 RPL 4|01FE8000 00000000 00C30C...
76: 15.4 D C1:0C:00:00:00:00:00:2C C1:0C:00:00:00:00:00:1C|IPHC|IPv6|ICMPv6 RPL DAO|1E40003E AAAA0000...
 5: 15.4 A
73: 15.4 D C1:0C:00:00:00:00:00:0F C1:0C:00:00:00:00:00:31|IPHC|IPv6|ICMPv6 RPL 4|00FE8000 00000000 0...
85: 15.4 D C1:0C:00:00:00:00:00:08 C1:0C:00:00:00:00:00:30|IPv6|ICMPv6 RPL 4|00FE8000 00000000 00C30C...
85: 15.4 D C1:0C:00:00:00:00:00:05 C1:0C:00:00:00:00:00:2E|IPv6|ICMPv6 RPL 4|01FE8000 00000000 00C30C...
76: 15.4 D C1:0C:00:00:00:00:00:2C C1:0C:00:00:00:00:00:1C|IPHC|IPv6|ICMPv6 RPL DAO|1E40003E AAAA0000...
73: 15.4 D C1:0C:00:00:00:00:00:0F C1:0C:00:00:00:00:00:31|IPHC|IPv6|ICMPv6 RPL 4|00FE8000 00000000 0...
85: 15.4 D C1:0C:00:00:00:00:00:08 C1:0C:00:00:00:00:00:30|IPv6|ICMPv6 RPL 4|00FE8000 00000000 00C30C...
85: 15.4 D C1:0C:00:00:00:00:00:05 C1:0C:00:00:00:00:00:2E|IPv6|ICMPv6 RPL 4|01FE8000 00000000 00C30C...
76: 15.4 D C1:0C:00:00:00:00:00:2C C1:0C:00:00:00:00:00:1C|IPHC|IPv6|ICMPv6 RPL DAO|1E40003E AAAA0000...
73: 15.4 D C1:0C:00:00:00:00:00:0F C1:0C:00:00:00:00:00:31|IPHC|IPv6|ICMPv6 RPL 4|00FE8000 00000000 0...
85: 15.4 D C1:0C:00:00:00:00:00:08 C1:0C:00:00:00:00:00:30|IPv6|ICMPv6 RPL 4|00FE8000 00000000 00C30C...
85: 15.4 D C1:0C:00:00:00:00:00:05 C1:0C:00:00:00:00:00:2E|IPv6|ICMPv6 RPL 4|01FE8000 00000000 00C30C...
76: 15.4 D C1:0C:00:00:00:00:00:2C C1:0C:00:00:00:00:00:1C|IPHC|IPv6|ICMPv6 RPL DAO|1E40003E AAAA0000...
73: 15.4 D C1:0C:00:00:00:00:00:0F C1:0C:00:00:00:00:00:31|IPHC|IPv6|ICMPv6 RPL 4|00FE8000 00000000 0...
76: 15.4 D C1:0C:00:00:00:00:00:2C C1:0C:00:00:00:00:00:1C|IPHC|IPv6|ICMPv6 RPL DAO|1E40003E AAAA0000...
85: 15.4 D C1:0C:00:00:00:00:00:08 C1:0C:00:00:00:00:00:30|IPv6|ICMPv6 RPL 4|00FE8000 00000000 00C30C...
85: 15.4 D C1:0C:00:00:00:00:00:05 C1:0C:00:00:00:00:00:2E|IPv6|ICMPv6 RPL 4|01FE8000 00000000 00C30C...
73: 15.4 D C1:0C:00:00:00:00:00:0F C1:0C:00:00:00:00:00:31|IPHC|IPv6|ICMPv6 RPL 4|00FE8000 00000000 0...
76: 15.4 D C1:0C:00:00:00:00:00:2C C1:0C:00:00:00:00:00:1C|IPHC|IPv6|ICMPv6 RPL DAO|1E40003E AAAA0000...
```

Figure 8.10: A figure that shows how many packets are sent in less than 0.1 seconds. Out of the 48 messages, 35 of the messages are *TRU-Messages*.

Not every *TRU-Message* sent from the nodes that is not a border router was arrived at the border router. Figure 8.9 shows that only one out of nine messages sent arrive at the border router. This is however, only an example from the figure. There are significantly less messages that arrive at the border router. During 1 second, up to a thousand messages were sent. Figure 8.10 shows packets that are sent during a time period of 0.1 seconds. Out of the 48 messages seen in the figure, 35 of these are *TRU-Messages*. It is also worth noticing that

a lot of the messages are sent from the same node, to the same destinations (For instance node five sends a *TRU-Message* to node 46 represented in the figure as: `C1:0C:00:00:00:00:00:05 C1:0C:00:00:00:00:00:2E`). Another result from this experiment is that several nodes can have routes to one node. Therefore, based on my assumption that `uip_ds_route_head()` contains a nodes children, a node has several parents.

8.4 Experiment 4

Table 8.5: Experiment 4: Results. The figure shows the number of detected attackers, false positives and false negatives. In experiment 4, all the attackers were detected, all though 15 different non-malicious nodes were marked as malicious.

Attackers detected	10
False Positives	15
False Negatives	0

Table 8.5 shows the number of detected attackers, false positives and the number of false negatives. Here, the number of correct detected attackers is ten. The number of non-malicious (false positives) nodes claimed to be malicious is fifthteen. The number of false negatives is the number of attackers in the system subtracted with the number of detected attacker nodes: $10 - 0 =$ zero false negatives.

Figure 8.11 shows when each of the ten detected attackers is discovered. Here, you can see that the average time it takes to detect an attack, is 32 minutes. The first attack was detected already after one minute, while the last attack detected was detected 84 minutes after the attack got introduced.

In this experiment, after a node sends a *TRU-Message*, it is received at the border router. Figure 8.12 shows that node three sends a *TRU-Message*, to the border router, containing observations about node 17, node 42 and node 9. In the fourth line, you see that node 17 have been observed four times before node three sent the *TRU-Message*, while node 42 and node 9 have been observed one time. Node 42 has not received negative observations, 23 seconds after the attack have been initialized.

This experiment does not flood the internet, which can be seen in figure 8.13. The *TRU-Message* is sent from node nineteen to node 1 through node 13, node 14 and node 6. It takes under a half a second from the message is sent from node 19 to the border router. During the time span of figure 8.13 — which is 1.5 seconds — this *TRU-Message* is the only *TRU-Message* sent.

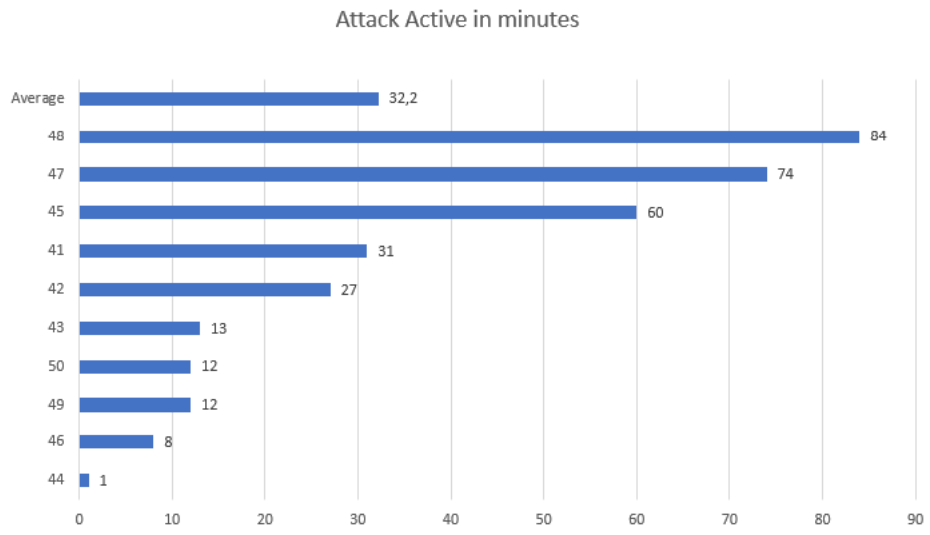

Figure 8.11: A figure that shows the number of minutes before each of the ten attackers that were detected, actually was detected. The average number of minutes before the attacks were detected, was 32 minutes. The first attack that was detected, was detected already one minute after the attack was introduced. The last attack that was detected was after 84 minutes.

```
05:23.714  ID:3   Sending received trust
05:23.718  ID:21   20585 P 193.12 15 195856 5047361 77707 67201 0 0 4311 323357 0 1169 0 0 (radio 2.76% / 0...
05:23.742  ID:1   Received true messages about:
05:23.747  ID:1   IP: 17, pos: 4, neg: 0
05:23.756  ID:1   Updated trust received about node 17. pTrust: 8. nTrust: 0
05:23.756  ID:23   20585 P 193.12 15 137761 5105202 49412 48078 0 0 3617 324061 0 960 0 0 (radio 1.85% / 0....
05:23.760  ID:1   IP: 42, pos: 1, neg: 0
05:23.768  ID:1   Updated trust received about node 42. pTrust: 2. nTrust: 0
05:23.772  ID:1   IP: 9, pos: 1, neg: 0
```

Figure 8.12: A figure that shows the border router receives the *TRU-Messages* right after it is sent. In this experiment, the *TRU-Messages* contain the trust of every neighbor of the node that sent the messages.

8.5 Experiment 5

Figure 8.14 shows the memory and filesize of the benevolent node and the border router. The border router's filename is `udp-server.z1` and the benevolent nodes filename is `udp-client.z1`. The figure shows that the file size of the border router and the normal nodes are 52254 and 51974 bytes, respectively. The memory usage of the system is $298 + 5944$ bytes for the border router and $298 + 5980$ bytes for the

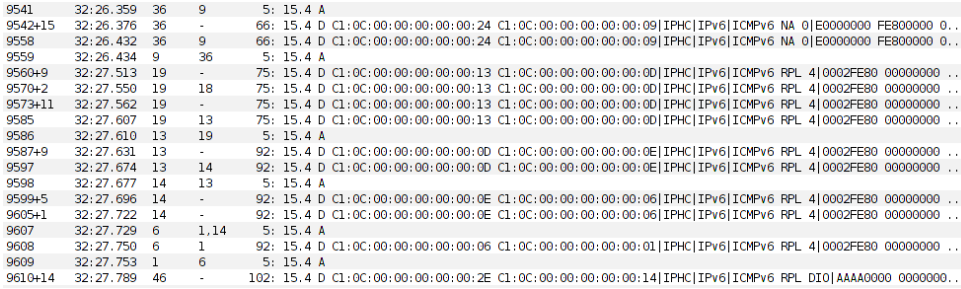

Figure 8.13: A figure that shows that the network is not flooded in experiment 4, and only one node's observations are sent at the time.

Table 8.6: Experiment 5: Results. The attacker got captured, with 0 false negative and 0 false positives. Energy consumption at the border router is high, while the average energy consumption at the non-malicious nodes is fair. The table also shows that the IDS has raised the total file size and the memory usage of both the border router and the benevolent nodes.

Average energy consumption in benevolent nodes	171,783 mW
Total energy consumption in the broder router	23329,471 mW
Attackers detected	1/1
False Positives	0
False Negatives	0
Total size of the IDS in border router	6390 bytes
Total size of the IDS in the non-malicious node	6144 bytes
Memory usage of the IDS in the border router	1012 bytes
Memory usage of the IDS in the normal nodes	992 bytes

non-malicious nodes. The total memory usage and filezise of the IDS is shown in 8.6.

Table 8.7: Experiment 5: Time of captured purposed malicious nodes for the first time

Node 10	2 hours, 50 minutes and 32 seconds

Figure 8.15 shows the energy consumption of the border router in experiment five. As mentioned in the implementation section, the border router is running on 100% duty cycle (listening all the time) to ensure that every packet is received. This causes the energy consumption to become very high. Every 20 second, the border router uses over 28mW. Over four hours and 30 min, the border router has a total energy consumption of 23329mW as shown in table 8.6.

```
user@instant-contiki:~/Documents/IDS_Git/Master---IDS$ msp430-size udp-server.z1
   text    data     bss     dec     hex filename
  52254     298    5944   58496    e480 udp-server.z1
user@instant-contiki:~/Documents/IDS_Git/Master---IDS$ msp430-size udp-client.z1
   text    data     bss     dec     hex filename
  51974     298    5980   58252    e38c udp-client.z1
```

Figure 8.14: A printscreen from experiment 5, that shows the ROM and RAM size of the non-malicious node and the border router, represented as "udp-client.c" and "udp-server.c" respectively.

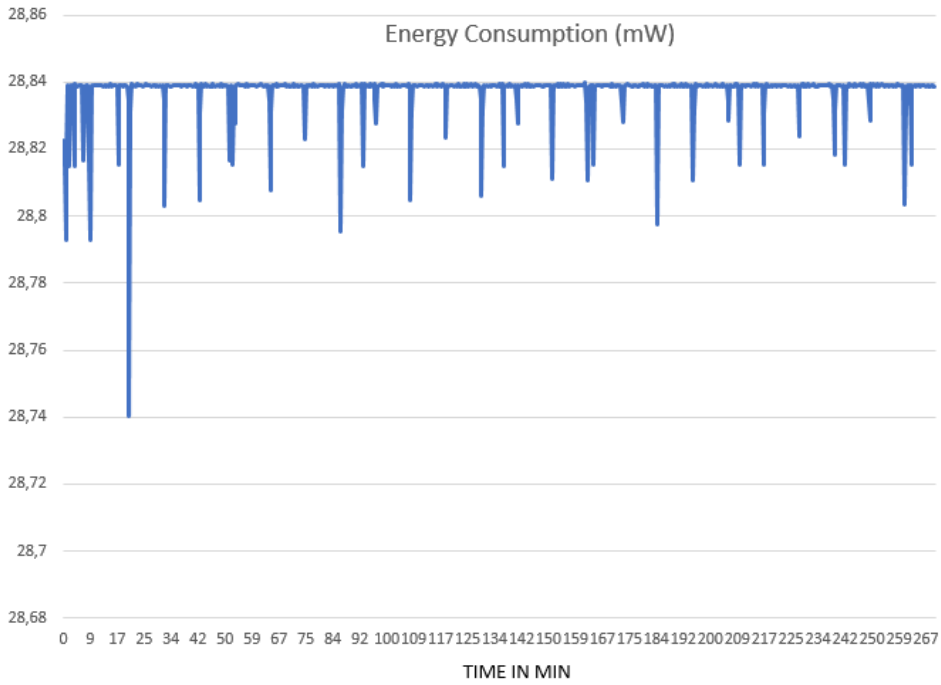

Figure 8.15: A graph that shows energy consumption in the border router during experiment 5.

For the non-malicious nodes, figure 8.16 shows the energy consumption of a arbitrary non-malicious node. This graph is similar to the rest of the non-malicious nodes energy consumption, and therefore, it is representative for them. As you can see, when the node was in idle, it spends 0,1mW. It has peaks when the node is sending and receiving packets. The highest energy consumption comes at time 4 and a half minute. The energy consumption is at 1,2 mW at this time. The average total energy consumption of the non-malicious nodes is 171,78 mW after four and a half hours.

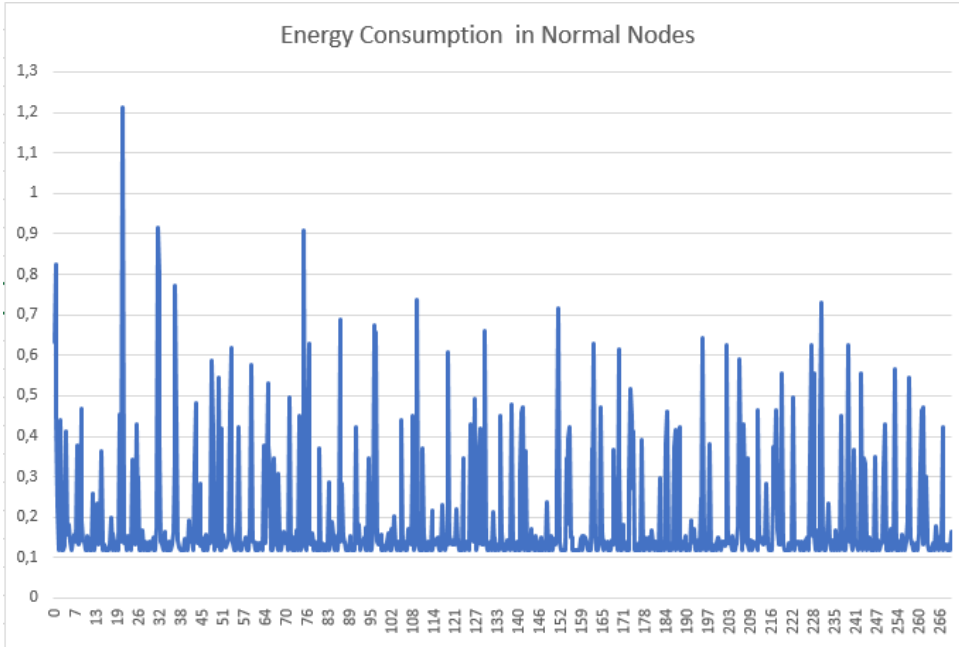

Figure 8.16: A graph that shows energy consumption in node 2 from figure 7.2 during experiment 5

Table 8.7 shows the time of detecting potential attackers. The only entry of this experiment is node 10, which, as stated in the experiment section, is the attacker. This attacker was detected by the IDS at 2 hours, 50 minutes and 32 seconds. Because this is the only entry in table 8.3, this is the only attacker the IDS identified. Hence, there are no false positive or false negative.

In figure 8.17, a print screen of the data of the first few minutes of energy consumption at the border router. The data represents the four hours, because the data is similar to the presented data. The data shows that the border router always uses energy in listening mode.

8.6 Experiment 6

Figure 8.18 shows the memory and filesize of the benevolent node and the border router. The border router's filename is `udp-server.z1` and the benevolent node's filename is `udp-client.z1`. The figure shows that the file size of the border router and the normal nodes is 51344 and 51196 bytes, respectively. The memory usage of the system is $298 + 5718$ bytes for the border router and $298 + 5736$ bytes for the

Table 8.8: Experiment 6: Results. The attacker got captured, with 0 false negative and 0 false positives. Energy consumption at the border router is high, while the average energy consumption at the non-malicious nodes are fair. The table also shows that the IDS have raised the total file size and the memory usage of both the border router and the benevolent nodes.

Average energy consumption in benevolent nodes	185,291 mW
Total energy consumption in the broder router	23329,049 mW
Attackers detected	1/1
False Positives	0
False Negatives	0
Total size of the IDS in border router	5480 bytes
Total size of the IDS in the non-malicious node	5366 bytes
Memory usage of the IDS in the border router	786 bytes
Memory usage of the IDS in the normal nodes	748 bytes

non-malicious nodes. The total memory usage and filezise of the IDS are shown in 8.8.

Table 8.9: Experiment 6: Time of captured purposed malicious nodes for the first time

Node 10	01 hours, 36 minutes and 29 seconds

Figure 8.19 shows the energy consumption of the border router in experiment seven. During this experiment, the border router is always in listening mode. This causes the border router to always use at least 28.75 mW. The total energy consumption of the border router is 23329,05 mW after four and a half hours.

For the non-malicious nodes, figure 8.20 shows the energy consumption of an arbitrary non-malicious node. This graph is similar to the rest of the non-malicious nodes energy consumption, and therefore, it is representative for them. As you can see, when the node is in idle, it spends 0,1mW. It has peaks when the node is sending and receiving packets. The highest energy consumption comes after 141 minutes where the energy consumption is at 1,6 mW. The average total energy consumption of the non-malicious nodes is 185 mW after four and a half hours.

Table 8.3 shows the time of detecting potential attackers. The only entry of this experiment is node 10, which, as stated in the experiment section, is the attacker. This attacker was detected by the IDS at 1 hour 36 minutes and 29 seconds. Because this is the only entry in table 8.3, this is the only attacker the IDS identified. Hence, there are no false positives or false negatives.

8.7 Experiment 7

Table 8.10: Experiment 7: Results. The figure shows the number of detected attackers, false positives and false negatives

Attackers detected	9
False Positives	2
False Negatives	1

Table 8.10 shows the number of detected attackers, false positives and the number of false negatives. Here, the number of correct detected attackers is nine. The number of detected non-malicious (false positives) nodes is also two and the number of false negatives is the number of attackers in the system subtracted with the number of detected attacker nodes: $10 - 9 =$ one false negative.

When each of the attacks detected are detected is shown in figure 8.21. The average number of minutes before the attacks was detected was one hour and 2 minutes. The first attack that was detected was detected four minutes after the attack was introduced. The last attack detected was detected after two hours and 58 minutes.

Not every *TRU-Message* sent from the nodes that is not a border router was arrived at the border router. Figure 8.22 shows packets that are sent during a time period of 0.1 seconds. Out of the 46 messages seen in the figure, 24 of these are *TRU-Messages*. It is also worth noticing that a lot of the messages are sent from the same node, to the same destinations (For instance node 48 sends a *TRU-Message* to node 7 represented in the figure as: `C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07`). Another result from this experiment is that several nodes can have routes to one node. Therefore, based on my assumption that `uip_ds_route_head()` contains a nodes children, a node has several parents.

EnergyConsumption CPU	EnergyConsumption LPM	EC Trans	EC List	Total mW	time in min
0,639965286	0	0,2356073	27,9469849	28,8225574	0,33333333
0,63899025	0	0,235527649	27,9401862	28,8147041	0,66666667
0,63900585	0	0,419839783	27,7341595	28,7930052	1
0,63900195	0	0,184391785	27,9931128	28,8165065	1,33333333
0,63899805	0	0	28,2	28,838998	1,66666667
0,6389961	0	0,2356073	27,9403583	28,8149617	2
0,63900585	0	0	28,2002582	28,839264	2,33333333
0,6389961	0	0	28,1998279	28,838824	2,66666667
0,639	0	0	28,1999139	28,8389139	3
0,63900195	0	0	28,2001721	28,8391741	3,33333333
0,63899805	0	0,235527649	27,9405304	28,8150561	3,66666667
0,63900585	0	0	28,2003442	28,8393501	4
0,63900195	0	0	28,2	28,839002	4,33333333
0,63899025	0	0	28,1996558	28,838646	4,66666667
0,63900195	0	0	28,2000861	28,839088	5
0,6390039	0	0	28,2000861	28,83909	5,33333333
0,63899805	0	0	28,2	28,838998	5,66666667
0,639	0	0	28,2	28,839	6
0,63900195	0	0,184391785	27,9931989	28,8165926	6,33333333
0,639	0	0,184073181	27,993371	28,8164442	6,66666667
0,6389922	0	0	28,1996558	28,838648	7
0,6390039	0	0	28,2000861	28,83909	7,33333333
0,6390039	0	0	28,2001721	28,839176	7,66666667
0,6390039	0	0	28,2002582	28,8392621	8
0,63899415	0	0,184551086	27,9927686	28,8163138	8,33333333
0,6390039	0	0,419680481	27,7340735	28,7927579	8,66666667
0,639	0	0	28,2000861	28,8390861	9
0,6389961	0	0	28,1997418	28,8387379	9,33333333
0,6389961	0	0	28,1999139	28,83891	9,66666667
0,6390078	0	0	28,2002582	28,839266	10
0,63899805	0	0	28,2	28,838998	10,3333333
0,639	0	0	28,1999139	28,8389139	10,6666667
0,639	0	0	28,2	28,839	11
0,63899805	0	0	28,1999139	28,838912	11,3333333
0,63900195	0	0	28,2001721	28,8391741	11,6666667
0,63899805	0	0	28,1999139	28,838912	12
0,639	0	0	28,2	28,839	12,3333333
0,6389961	0	0	28,1998279	28,838824	12,6666667
0,63900195	0	0	28,2000861	28,839088	13
0,63900585	0	0	28,2002582	28,839264	13,3333333
0,6389922	0	0	28,1996558	28,838648	13,6666667
0,63900585	0	0	28,2002582	28,839264	14
0,63899415	0	0	28,1997418	28,838736	14,3333333

Figure 8.17: A printscreen of the first few minutes of Energy Consumpiton data collection at the border router in experiment 5. This set of data is representative for the rest of the data.

```
user@instant-contiki:~/Documents/IDS$ msp430-size udp-server.z1
   text    data     bss     dec     hex filename
  51344     298    5718   57360    e010 udp-server.z1
user@instant-contiki:~/Documents/IDS$ msp430-size udp-client.z1
   text    data     bss     dec     hex filename
  51196     298    5736   57230    df8e udp-client.z1
```

Figure 8.18: A printscreen from experiment 6 that shows the ROM and RAM size of the non-malicious node and the border router. In the printscreen, the filenames of the normal nodes and the border router are represented as "udp-client.c" and "udp-server.c" respectively

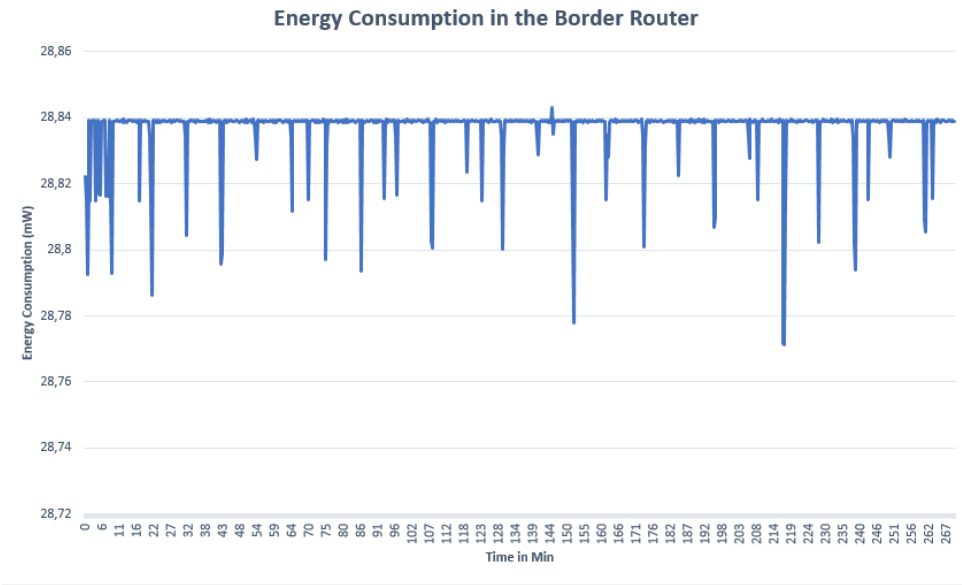

Figure 8.19: A graph that shows energy consumption in the border router during experiment 6.

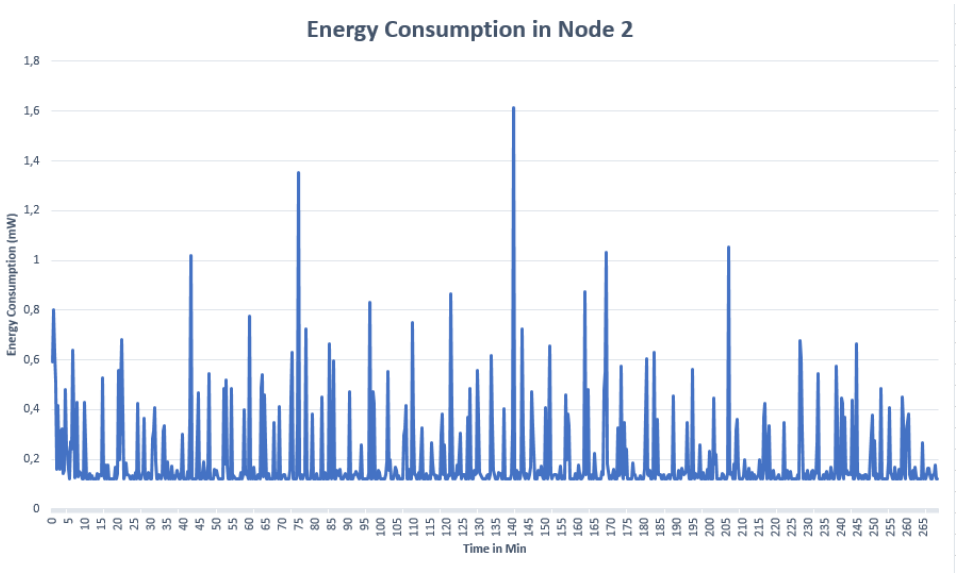

Figure 8.20: A graph that shows energy consumption in node 2 from figure 7.2 during experiment 6

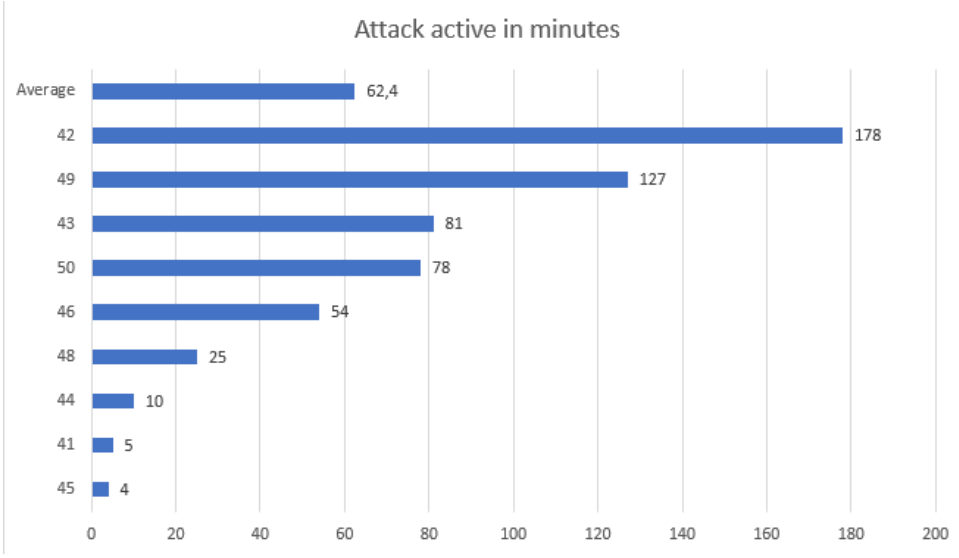

Figure 8.21: A figure that shows the number of minutes before each of the nine attackers that were detected actually were detected. The average number of minutes before the attacks were detected was one hour and 2 minutes. The first attack that was detected was detected four minutes after the attack was introduced. The last attack detected was detected after two hours and 58 minutes.

```
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  17  76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  17  76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  4   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  4   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  4   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  -   76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
10  50  76: 15.4 D C1:0C:00:00:00:00:00:0A C1:0C:00:00:00:00:00:32|IPHC|IPv6|ICMPv6 RPL DAO|1E40000
50  10   5: 15.4 A
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
48  13  85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
48  -   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
48  7   85: 15.4 D C1:0C:00:00:00:00:00:30 C1:0C:00:00:00:00:00:07|IPv6|ICMPv6 RPL 4|01FE8000 00000
```

Figure 8.22: A figure that shows how many packets are sent in less than 0.1 seconds. Out of the 46 messages, 24 of the messages are *TRU-Messages*.

Chapter 9
Discussion

In this section, the results of the thesis are discussed. The topics considered in this chapter is storage, energy consumption, false positives, trust update, border router in listening mode, length before introducing attacks and different attacks.

Table 9.1: A table showing the different experiments discussed in this chapter. The abbreviations are created based on how many nodes that are in the network, Whether event-trust or time-trust is used, and if the border router is in listening mode all the time, or not.

Experiment	Attackers	Nodes	Trust Update	Listening Mode
BENCH	0	10	None	Off
TEOF	1	10	Event-Trust	Off
FEOF	10	50	Event-Trust	Off
FTON	10	50	Time-Trust	On
TTON	1	10	Time-Trust	On
TEON	1	10	Event-Trust	On
FEON	10	50	Event-Trust	On

Table 9.1 shows the different experiments discussed in this chapter. The abbreviations are created based on how many nodes that are in the network, Whether event-trust or time-trust is used, and if the border router is in listening mode all the time, or not. If the experiment has fifty nodes in the network, the experiment abbreviation start with an F. If it contains ten nodes, it starts with T. The second letter in the acronym is based on the trust update. If event-trust is being used, the second letter is E, while it is T if time is used. The last two letters are based on whether or not the border router is in listening mode. The last two letters will be ON or OF respectively.

9.1 Storage

As stated in the results of experiment TEOF, the file size of the IDS is 5.35 KB and 5.24 KB at the border router and the non-malicious nodes, respectively. Both these files include sending and receiving *TRU-Messages* and the detection of the sinkhole attack. These also contain the implementation of the sinkhole attack because the attack is needed to detect the intended attack. Hence, some of the file size can be reduced.

The Zolertia Z1 has 92 KB available storage. Because the file size of the IDS at the border router is 5.35 KB, the IDS occupies almost 6% of the available storage of the border router, while the IDS occupies around 5.5% of the available ROM at the non-malicious nodes.

The amount of memory being utilized by the IDS is also shown in the results of TEOF. The IDS uses 786 bytes of memory in the border router, while it uses 748 bytes in the nodes that are not a border router. As for the file size, the results contain both sending and receiving *TRU-message* and the detection of the sinkhole attack along with the actual implementation of the sinkhole attack. Because of this, the size of the memory can be reduced.

Considering that the memory of the Zolertia Z1 is 10 KB, the IDS takes up 7.6% of the available memory of the border router, and 7.3% of the available memory in the remaining nodes.

As seen in the results of experiment BENCH, the setup of RPL, together with a small application takes up 44.8 KB of the 92 KB available at the border router. The memory usage at the border router without the IDS uses 5.1 KB of the 10 KB available. The small application in the non-malicious nodes sends data packets to the border router, while the application in the border router prints out payload sent to the border router. The total size of the setup of the RPL together with the IDS and the small application is around 50 KB, and the total memory is around 5.87 KB. This size is more than half of the available storage.

The time-trust based IDS uses more memory and more storage on the devices. This is because the nodes need to have a list of each node observed. The border router now has a file size of 6.24 KB, and the common nodes have a file size of 6 KB. The memory usage now is around 1 KB for both the border router and the normal nodes.

Figure 9.1 shows a comparison of the memory usage and storage usage in time-trust update and event-trust update. As seen in figure 9.1, the time-trust update uses more memory and storage compared to the event-trust update. As mentioned, this

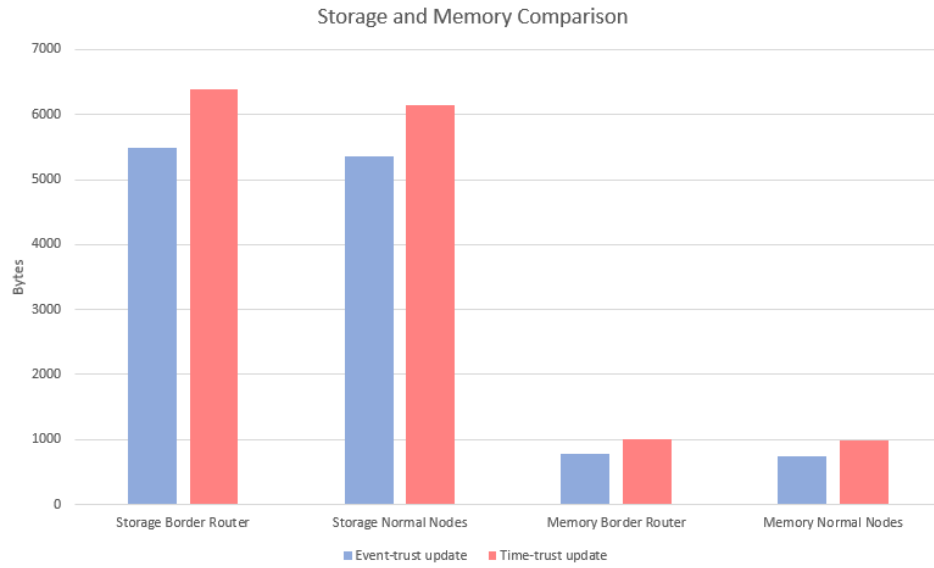

Figure 9.1: A figure which compares the storage and memory usage in event-trust and time-trust update. The figure shows that the time-trust update uses more memory and storage in both the border router and the normal nodes.

is because each node needs to keep track of the observations before the observations are sent to the border router.

The resource usage is significantly higher with time-trust update rather than the event-trust update. The increase in the file size is expected because the trust observed needs to be stored from the moment the observation is made until the node sends the trust to the border router.

Based on these results, the trade-off concerning security is raised: The more security in a system, the more it affects the usability. Security will always affect the usability. However, it is ideal to reduce this affect, because every system wants both good security and good usability. The attacks that are possible to perform without this IDS can potentially have extreme consequences, such as, massive economic losses and in worst case — human life. Based on the implications, additional resources are a necessary cost to detect these attacks. Considering this, using 10% of the whole memory is a significant amount of memory.

One of the research question presented was how much the performance of the RPL protocol would be reduced with the IDS. The total file size of the IDS is relevant

to this question because it affects the number of bytes a program uses with RPL. Hence, it affects the performance of the RPL protocol. The results discussed above are an indication that the IDS does influence the performance of the RPL protocol significantly. This resource usage could be less, because the author has not written code in C before. Hence, some unnecessary storage and memory usage might be removed.

9.2 Energy Consumption

As discussed in the methodology chapter, energy consumption is necessary to consider, since the devices that will be using the IDS might be running on battery. Concerning the nodes that are not the border router, it was not expected that these nodes would have a significant increase in energy consumption. The average total energy consumption at the benevolent node without the IDS was implemented was 191,138 mW, while the same nodes used 265 mW with the IDS implemented. This is because the extra time these nodes are on is when a node receives DIO. While it does extra calculations, it does not take significantly longer time to process the DIO message, and the energy consumption reflects this.

If the selective forward attack had been implemented, the result would have been different, because it would put itself into listening mode after sending a packet, instead of turning off. This would cause the energy consumption to increase at the normal nodes. Due to the problems concerning implementation of the selective forward attack, mentioned in section 6.2.10, how much of an increase in the energy consumption is not available.

In the border router, the results are different. The total energy consumption in the border router increases with 488 mW compared to before the IDS was implemented. Considering the main part of the IDS is in the border router, an increase of the energy consumption was expected.

Figure 9.2 shows the difference between the energy consumption in the border router when the IDS is implemented, using time-trust and event-trust update, and when the IDS is not implemented. There is not a significant difference between time-trust update and event-trust update concerning the energy consumption. The experiments with the IDS implemented use 0,600 mW extra compared to the experiment where the IDS is not implemented.

This difference can be explained by the fact that the border router uses active waiting to see whether or not a *TRU-message* is received. This means each 0.1 second, the border router test this condition. If it receives the *TRU-message* it processes the message, while if it does not, it waits one more period (0.1s). Another way to do this

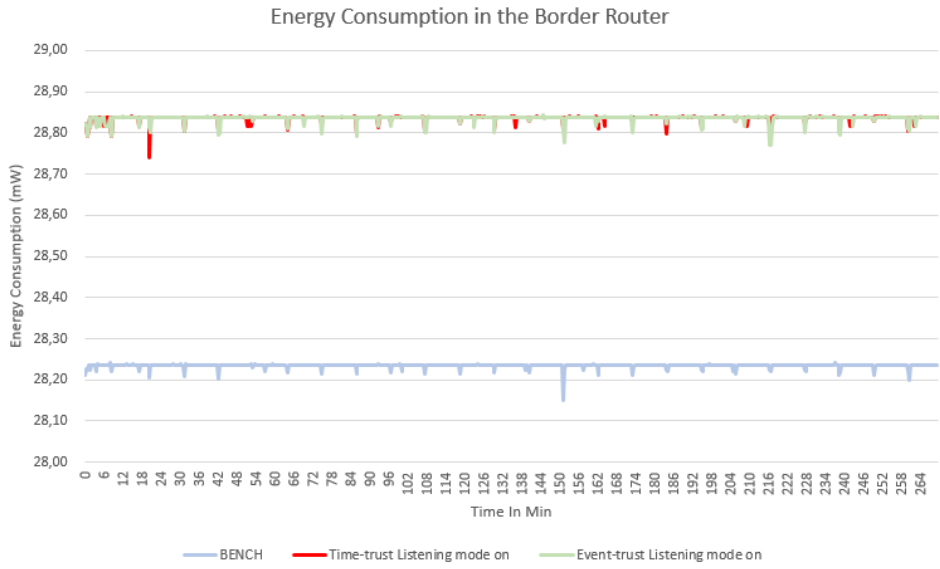

Figure 9.2: A figure that compares the energy consumption in the border router. In the benchmark experiment, the border router has an energy consumption where it uses 28.23 mW, while both experiments with the IDS uses 28.83 mW.

that should reduce the energy consumption at the border router is an interrupt-based method. When a *TRU-message* is received at the border router, it interrupts the current process and processes the *TRU-message*.

In the non-malicious nodes, the energy consumption is less when using time-trust update compared to event-trust update. This is mainly because the application now is sending data packets every 5th minute rather than every minute.

Comparing the results regarding energy consumption from experiment TEOF with experiment TTON and experiment TEON, shows that the border router uses significantly more energy when it always is in listening mode. This is shown in figure 9.2 and 8.5. These figures show that the border router uses 28 mW extra when the node always is in listening mode. Based on the assumption that the border router is not resource constrained, this should not matter too much. Concerning the other nodes, an increase of 265 - 191 = 74 mW in four hours and 30 minutes is acceptable considering the detection rates discussed below.

9.3 False positives

As expected, the network got flooded (a lot of packets sent all the time) if *TRU-Messages* are sent right after observing a negative or positive observation. Almost a thousand messages were sent during a second. Several of the messages sent are *TRU-Messages*. A consequence of this is that not every observation is received at the border router. The IDS may give negative observations to nodes that are benevolent. However, the number of positive observations should be large enough so that these nodes are not marked as malicious. The reason for why a non-malicious node is observed as malicious is discussed next.

Because the benevolent nodes can receive negative feedback, it is important that the border router is receiving every positive feedback. The worst case scenario is that the border router might receive more negative feedback than positive because the positive observations get dropped, causing the border router to claim that a non-malicious node is malicious. This could be one of the reasons that FEOF received a lot of false positives.

The difference between the two experiments using event-trust update with 50 nodes in the network (FEOF and FEON) is major. Without the listening mode on, the IDS only detected five malicious nodes and introduces five false positives, while the experiment with the listening mode on identified nine malicious nodes and introduced only two false positives.

Three other reasons for the IDS to introduce false positives are:

- small differences in the rank of two non-malicious nodes.
- non-malicious nodes that calculate their rank based on the malicious nodes rank. After calculating an incorrect rank, the node is observed by a node with correct rank, hence receives a negative observation.
- implementation could be wrong.

As explained in the background chapter, the rank of a node changes frequently. Consider two non-malicious nodes that have similar rank. A possible scenario is that the node with the highest rank of the two nodes changes to a rank lower than the other node. This can cause the nodes to evaluate each other with negative observations. This should not be a problem, considering the nodes should only observe the rank of its parents, and nodes are not allowed to claim a rank lower than their parent. It is however mentioned because of the third reason, which is implementation error.

Another reason for false positives is that the non-malicious node calculates their rank, based on their malicious neighbor. This causes the node to assume that it is closer to the border router. The problem with this is that it claims a lower rank than the nodes that calculate their rank based on the actual distance from the border router. This should not be a problem either because only the malicious node should observe the node that has been claimed to be malicious because it is the parent. The third reason might cause this to be a problem.

In section 6.2.2, an assumption is that `uip_ds_route_head()` returns routes to the children of the node. However, it seems that this is not the case, considering that several nodes evaluated one DIO-message. If a node is not a parent node, the rank test is not valid because a neighbor can have the same rank and even lower than the current node. If `uip_ds_route_head()` returns routes to nodes that is not the child, the rank test may not be valid. This is discussed in future work.

FTON shows that even though the packets are received at the border router, the IDS still has a lot of false positives. This means that the non-malicious nodes that are claimed to be attackers by the border router are receiving more negative observations than positive. Based on the number of false positives in the IDS, the strength of this intrusion detection system is questioned. RQ2 asks whether or not the IDS detects the correct nodes. The number of false positives states that TIDS does not only catch the malcious nodes, but also nodes that is not malicious. This questions the way sinkhole attack is detected, but not the full IDS, because a better way of detecting sinkhole attack would send less incorrect observations, which would cause the IDS to have fewer false positives.

9.4 Detection Rates

Considering event-trust update sends the data at once when an observation is done, while the time-trust update waits a certain time before the trust data is sent, the data needs to be stored so that all data is sent to the border router. This difference in storage and memory are shown in figure 9.1.

Figure 9.3 shows a comparison of experiment FEOF, FTON and FEON. As you can see, the experiment using time-trust introduces the most false positives but also captures the most attackers. The experiment where event-trust is used as trust update and the border router is always on is the experiment that introduces the least false positives, but the experiment did not detect every attacker. The experiment using event-trust as trust update, and where the border router is not always in listening mode only captured 50% of the attackers, even though the experiment was running four hours more than the other experiments. It also introduced five false positives.

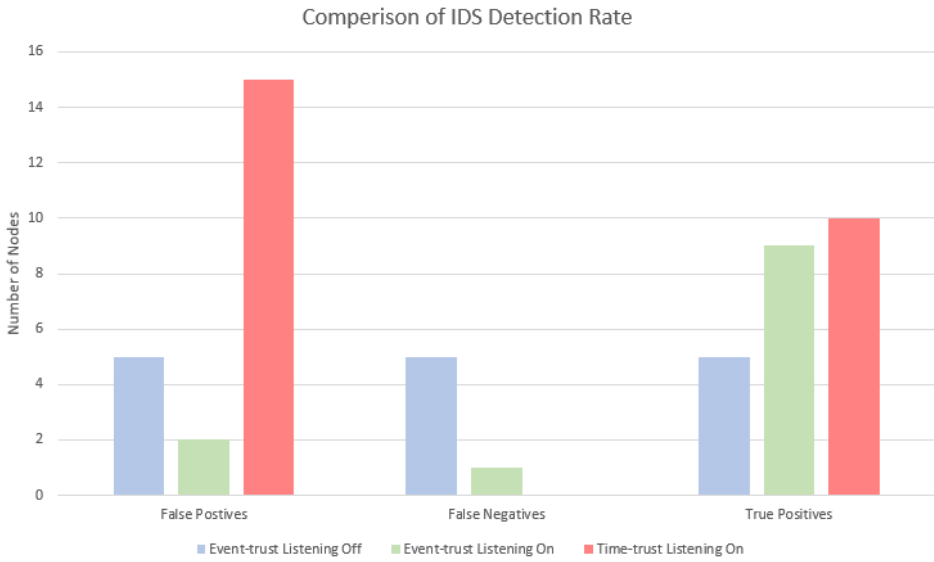

Figure 9.3: A figure that compares the number of detected nodes, the false positives, and false negatives in the experiment where event-trust was used with listening mode on and off, and the experiment where time-trust was used with listening mode on.

TEOF captured the one attacker and introduced zero false positives or negatives, where event-trust update was used (TEOF, FEOF, TEON and FEON). Nevertheless, when adding more nodes to the system, the event-trust update struggled. The network was flooded with *TRU-Messages* and several of the attackers were not captured, because the border router did not receive all the messages. Allowing the border router to be in listening mode all the time, improved the results of FEOF. The border router is using a lot more energy when it is in listening mode all the time. However, the results also improve a lot. Experiment FEON (Event, Listening on) captured nine attackers, while FEOF (Event, Listening off) captured five. FEON also introduced less false positives than FEOF. This could be because several more *TRU-Messages* were received at the border router. This causes several more positive observations about non-malicious nodes.

When the time-trust update was being used (TTON and FTON), the same detection rate was shown as in TTON as in TEOF. Both experiments captured the attacker and introduced 0 false positives and false negatives. In FTON, every attacker was detected. Compared to FEOF (where event-trust update was used), it is a major improvement. FEOF detected 50% of the attackers, while FTON detected

100% of the attacks. In experiment FEON, the detection rate also improved.

FTON did, however, introduce several more false positives than both FEOF and FEON. While FEOF introduced five false positives, FTON introduced fifteen false positives. FEON only introduced two false positives. A reason for this is the result of FTON is more correct, considering more *TRU-Messages* are received at the border router. Because the results from the IDS have shown that it rates non-malicious nodes negative, an assumption is that more negative observation of non-malicious nodes will arrive at the border router in FTON. This is because every node's observations are sent to the border router. If a node starts sending negative observations to a node, it is probable that it will keep evaluating this node negatively several times. For instance, if the false positive reason is that the nodes calculate their rank based on the malicious nodes rank, it will continue to do so. Hence, the border router will end up claiming the observed node malicious.

The time it takes before an attack is detected is important to an IDS because if the attack spends a lot of time in the system, it can be crucial for the system, because of the damages it can do. Figure 8.8 and figure 8.11 display how long it took to detect attacks in FEOF and FTON respectively. While FTON spends 32 minutes in average to detect the attacks, FEOF and FEON spends 400 minutes and 62 minutes, respectively, in order to detect an attack. The reason for this is the border router does not receive every negative observation, and it takes longer before a negative observation is received.

By comparing figure 8.13 and figure 8.10 it is clear to see that in FEOF, the network got flooded with messages. While the *TRU-Messages* in FTON are sent directly to the border router, using nine messages. The messages in FEOF, shown in figure 8.10 shows node five (`C1:0C:00:00:00:00:00:05`) uses five tries at least to send a *TRU-Message* to node 46 (`C1:0C:00:00:00:00:00:2E`). Figure 8.10 also shows several RPL messages are sent at the same time. A delay in the whole network might happen, due to re transmissions caused because the network is occupied. Figure 8.22 shows that even though the border router always is on in experiment FEON, event-trust still sends a lot of *TRU-Messages* before a node receives it.

Based on the results discussed above, and considering the assumption that the border router is not a node with restricted resources, the border router should be in listening mode all the time. This gives more energy consumption at the border router, but it also improves the results significantly.

Should event-trust update or time-trust update be used as the trust update for the IDS? While the event-trust update uses the least resources, concerning the file size and memory usage, the event-trust update uses several more packets to send the observations to the border router. As discussed above, time-trust update is

more "correct" because every packet is received at the border router. Time-trust update also captures every attacker during the experiment, while event-trust update captured 90% of the attackers (when the border router always are listening). However, it also introduces more false positives. These false positives are connected to the way the system detects the sinkhole attack and not whether or not the system uses time-trust or event-trust. Therefore, to decide which trust update to use, it is important to evaluate what is most important to the RPL-protocol. Should the IDS use a few resources, or is it more important that the network is not overloaded with *TRU-Messages*.

9.5 Concerning the length before an attack is introduced.

When an attack is detected, it is important to an IDS because if the attack spends hours to days in the system, it can be crucial for the system. Figure 8.11 displays how long it took to detect attacks in FTON. The figure shows that the first attack is detected at 1 minute while the last node (node 48) is detected after 84 minutes. The reason for this gap is node 48 had received several more positive observations before the negative observation started arriving (after the attack was initiated). This raised another question: What if the attacker does not initiate the attack before the attacker node has received a lot of positive observation. This would affect the detection time.

A solution to the problem above is to only take into consideration the last x of a node's observations. This would give an upper limit to how much the attacker can gain on staying in the network receiving positive feedback. How many observations should be stored for each node is discussed in future work.

9.6 Different attacks

Even though this intrusion detection system only captures the sinkhole attack, it can easily be extended to detect other attacks. The border router receives *TRU-messages* without the knowledge of what type of attack caused the packets to be sent. Hence, to introduce detection for a new attack, the detection of the attack needs to send the *TRU-Messages*, based on its observation, and the IDS will take care of the rest.

Implementing detection for new attacks will not have the same increase in resource usage as seen before. The IDS is already implemented, and the thing left to implement is to detect the attack and sending the *TRU-messages*. One of the reasons for having a centralized trust propagation is that the border router is assumed to have more resources than the other nodes. The detection of the attacks should not use a lot of resources because it will be used by nodes that have limited resources.

As mentioned in section 9.3, the IDS introduces many false positives. This is due to the way the intrusion detection system detects the sinkhole attack. The reason for why it introduced many false positives is discussed in section 9.3. Note that to fix these problems is only a small part of the IDS. One of the problems discussed in the mentioned section, is the implementation problem regarding the function `uip_ds_route_head()`. If this is the problem causing the false positives, a new way of detecting children in Contiki could fix the problem with detecting sinkhole, hence, fix the IDS.

Another way to detect sinkhole attack is to intercept every data packet coming from a node's neighbor, testing if the packet is coming from a child. If the node is a child, the rank in the data packet should be evaluated. This could not be performed due to the same problems as with selective forward. Because the packets are unicasted, and the data packets were not captured at the physical layer, it could not be implemented.

Chapter 10
Conclusion

10.1 Conclusion

This master thesis has introduced TIDS which uses a trust-based technique to detect attacks on the RPL protocol. Ideally, the IDS detects the attacks intended and does not use a lot of resources, while not introducing false positives or false negatives either. The implemented IDS has successfully detected the sinkhole attack while introducing false positives.

TIDS uses between 5-10% of the available storage and between 5-10% of the memory. The energy consumption of the IDS is large at the border router because the border router needs to receive every *TRU-Message*, while the energy consumption in the other nodes does not increase as much. The event-trust update used fewer resources than the time-trust update.

Regarding the detection rate of the implemented system, this thesis has concluded that the border router should always be in listening mode to get the best detection rate. The time-trust update detected every attacker after 90 minutes while event-trust update detected 90 % of the attackers after 4 hours and 30 minutes. The event-trust update introduced 2 false positives, while the time-trust update introduced 15. These false positives are introduced because of the way the sinkhole attacks are detected in this IDS. Hence, if the way the IDS detects sinkhole attack is improved, the IDS will also improve.

Even though TIDS introduced false positives, this thesis has shown good results. TIDS does not introduce significant amount of extra energy consumption in the normal nodes. The storage and memory usage is significant, however, it can be reduced. The results have also demonstrated that it detects attacks. This thesis has shown that the system can react to positive and negative observations. If suitable methods for detecting attacks can be implemented and extended to TIDS, then TIDS could be used with the RPL-protocol.

10.2 Future Work

This thesis has shown promising results. However, there is still work to in order to make the IDS ready for implementation in the RPL protocol. This section discusses potential improvements of the IDS presented in this thesis.

10.2.1 Interruption Rather Than Passive Waiting

In the main program of the border router, the border router is actively waiting for new observations from node in its DODAG. This means that every 0.1 second, a timer times out, and the border router tests whether or not the node has received new data. If the intrusion detection system instead interrupts the main program when new observation data is received, the energy consumption will be reduced. Therefore, the system notifies the main program when a trust update is received. After receiving the notification, the border router updates the received trust. Because the border router now waits for an observation, rather than checks if a new observation has arrived each 0.1 second, the energy consumption at the border router is reduced.

10.2.2 Implement different attacks

For this intrusion detection system to be complete, the most known attacks should be detected. This thesis explains how the selective forward attack can be detected, however the attack is not implemented. Another attack that should be detected is the version number attack, which can be detected by testing whether or not the version claimed by a neighbor is correct. When receiving a version number from a neighbor which is higher than the current nodes version number, the current node expects to receive an message from the border router telling the node to update its version number. If this message has not arrived before a timer expires, it sends a negative observation to the border router, because the node claims that it has a higher version number than the actual number. If the current node receives the version number update before the timer expires, it sends a positive observation.

Implementing other attacks should not be resource demanding, considering the intrusion detection system does not need to be implemented again. The detection of the attack, combined with the sending of the *TRU-Message* is the two things needed in order to detect another attack. Before implementing an attack, it is important to consider that sending observations about several attacks might affect the detection rate of the intrusion detection system. If a node performs a sinkhole attack, while the node is receiving positive observations because of the selective forward detection, it might not be detected as a malicious node. This is due to the node is acting according to the RPL protocol, and receiving positive observations from the selective forward test. So even though the sinkhole test causes a parent to send negative

observation to the border router, the border router might not assume the attacker is malicious if this is not considered.

10.2.3 Safety of the TRU-messages

In this thesis, the safety of the *TRU-messages* has not been discussed. The data is sent unencrypted and unprotected from attacks. This should be considered as the attacker might exploit the messages. If the attacker can modify *TRU-messages*, the IDS might not work as intended.

10.2.4 A better way to detect children

As mentioned in the discussion, `uip_ds_route_head()` might return routes to nodes that are not children of the node. This causes problems in the implementation of the sinkhole attack. Hence, a better way of testing if a node is a child is needed.

10.2.5 Testing different detection rates

In this thesis, the trust-observations are sent at once, or between four and six minutes. This is to compare whether or not time-trust update is detecting more nodes than event-trust update. An interesting new experiment could be to find the ideal time for when the trust-observation should be sent. It is interesting as one wants to send as few packets as possible, but still receive the best possible results.

References

[Adv10] S.L Advancare. Z1 Datasheet. http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf, 2010. [Online; accessed 20-May-2017].

[BE15] B A Bagula and Zenville Erasmus. Iot Emulation With Cooja. http://www.cs.uwc.ac.za/~abagula/cos730/docs/ICTP-Cooja-Presentation-version2.pdf, 2015. [Online; accessed 20-May-2017].

[Con16] Contiki. Contiki: The Open Source OS for the Internet of Things. http://www.contiki-os.org/index.html, 2016. [Online; accessed 08-October-2016].

[CPNS15] C. Cervantes, D. Poplade, M. Nogueira, and A. Santos. Detection of sinkhole attacks for supporting secure routing on 6lowpan for internet of things. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 606–611, May 2015.

[For12] Internet Engineering Task Force. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. https://tools.ietf.org/pdf/rfc6550.pdf, 2012. [Online; accessed 02-June-2017].

[GCT17] Jia Guo, Ing-Ray Chen, and Jeffrey J.P. Tsai. A survey of trust computation models for service management in internet of things systems. *Computer Communications*, 97:1 – 14, 2017.

[GD13] Ing Pietro Gonizzi and Simon Duquennoy. Hands on Contiki OS and Cooja Simulator: Exercises (Part II). https://team.inria.fr/fun/files/2014/04/slides_partI.pdf, 2013. [Online; accessed 24-May-2017].

[GMS15] J. Granjal, E. Monteiro, and J. Sá Silva. Security for the internet of things: A survey of existing protocols and open research issues. *IEEE Communications Surveys Tutorials*, 17(3):1294–1312, thirdquarter 2015.

[Gro17] Autonomous Networks Research Group. MAC protocols in ContikiOS. http://anrg.usc.edu/contiki/index.php/MAC_protocols_in_ContikiOS, 2017. [Online; accessed 07-June-2017].

[HNBH11] Philipp Hurni, Benjamin Nyffenegger, Torsten Braun, and Anton Hergenroeder. On the accuracy of software-based energy estimation techniques. In *Proceedings*

of the 8th European Conference on Wireless Sensor Networks, volume 6567 LNCS of *EWSN'11*, pages 49–64, Berlin, Heidelberg, 2011. Springer-Verlag.

[Ini12] IEEE Internet Initiative. Towards a Definition of the Internet of Things (IoT). http://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf, 2012. [Online; accessed 25-May-2017].

[Jøs01] Audun Jøsang. A logic for uncertain probabilities. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 9(3):279–311, June 2001.

[KC07] Barbara Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature reviews in Software Engineering Version 2.3. *Engineering*, 45(4ve):1051, 2007.

[KH17] Z. A. Khan and P. Herrmann. A trust based distributed intrusion detection mechanism for internet of things. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 1169–1176, March 2017.

[Kna98] Knapskog, S J and Jøsang, A. A metric for trusted systems. *Proceedings of the 21st National Security Conference*, pages 16–29, 1998.

[Mak] MakeLinux. Memory Management in Linux. http://www.makelinux.net/ldd3/chp-15-sect-1. [Online; accessed 02-June-2017].

[MTS14] T. Matsunaga, K. Toyoda, and I. Sasase. Low false alarm rate rpl network monitoring system by considering timing inconstancy between the rank measurements. In *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*, pages 427–431, Aug 2014.

[MY11] Sureyya Mutlu and Guray Yilmaz. A Distributed Cooperative Trust Based Intrusion Detection Framework for MANETs. *ICNS 2011 : The Seventh International Conference on Networking and Services*, pages 292–298, 2011.

[PTPB10] C. R. Perez-Toro, R. K. Panta, and S. Bagchi. RDAS: Reputation-Based Resilient Data Aggregation in Sensor Network. In *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 1–9, June 2010.

[Res17] Symantec Security Response. Ransom.Wannacry. https://www.symantec.com/security_response/writeup.jsp?docid=2017-051310-3522-99, 2017. [Online; accessed 29-May-2017].

[RS13] Kévin Roussel and Ye-Qiong Song. A critical analysis of Contiki's network stack for integrating new MAC protocols. Research Report RR-8776, INRIA Nancy, December 2013.

[RWV13] Shahid Raza, Linus Wallgren, and Thiemo Voigt. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad Hoc Networks*, 11(8):2661 – 2674, 2013.

[SU16] M. Surendar and A. Umamakeswari. Indres: An intrusion detection and response system for internet of things with 6lowpan. In *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 1903–1908, March 2016.

[Udi12] Jeremy Udit. Contiki OS. http://jeremyudit.blogspot.no/2012/08/contiki-os.html, 2012. [Online; accessed 25-May-2017].