UPPSALA UNIVERSITET

UPTEC IT 23004

Examensarbete 30 hp

Mars 2023

Intrusion Detection in the Internet of Things

- From Sniffing to a Border Router's Point of View

Victoria Bull

Civilingenjörsprogrammet i informationsteknologi

UPPSALA UNIVERSITET

Intrusion Detection in the Internet of Things

Victoria Bull

Abstract

The Internet of Things is expanding, and with the increasing numbers of connected devices, exploitation of those devices also becomes more common. Since IoT devices and IoT networks are used in many crucial areas in modern societies, ranging from everything between security and militrary applications to healthcare monitoring and production efficiency, the need to secure these devices is of great importance for researchers and businesses. This project explores how an intrusion detection system called DETONAR can be used on border router logs, instead of its original use of sniffer devices. Using DETONAR in this way allows us to detect many different attacks, without contributing to the additional cost of deploying sniffer devices and the additional risk of the sniffer devices themselves becoming the target of attacks.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Niclas Finne Ämnesgranskare: Thiemo Voigt

Examinator: Lars-Åke Nordén

Acknowledgements

I want to thank Niclas Finne and Thiemo Voigt for the time put into helping me in my technical endeavors, and the opportunity to learn from you during this project. I have learned a lot!

Contents

1 Introduction **2**
1.1 Setting . . . 2
1.2 Problem Statement . . . 3
1.3 Project Purpose and Scope . . . 3
1.4 Contributions . . . 4
1.5 Document outline . . . 4

2 Background **5**
2.1 IoT networks . . . 5
2.2 The Contiki Operating System . . . 6
2.3 Cooja and Multi-Trace . . . 6
2.4 RPL . . . 6
2.5 DETONAR . . . 9
2.6 Related Work . . . 15

3 Design **19**
3.1 Dataset . . . 19
3.2 Collection of Data with Cooja . . . 24

4 Implementation **26**
4.1 Running DETONAR on Node Logs . . . 26
4.2 Running DETONAR on Border Router Logs . . . 26
4.3 Adjustments to Nodes' Behaviour . . . 28
4.4 Automating Processes . . . 29

5 Evaluation **31**
5.1 Exploring Topology Generation . . . 33
5.2 Running DETONAR on a Border Router . . . 46
5.3 Trends for Border Router and Sniffer Approach . . . 60
5.4 Discussion of results . . . 62

6 Conclusions and Future Work **63**

Chapter 1

Introduction

1.1 Setting

Internet of Things (IoT) networks have numerous applications in modern societies, many of them including the use of sensors. A typical IoT sensor network consists of several small devices sensing their surrounding environment and communicating these measurements to one or several border routers, which act as a link between the network and the internet.

A real-world example of so called environmental sensing is forest fire detection, discussed by Zhang et al. [1], in which it is mentioned that traditional forest fire monitoring includes video, aerial, and satellite monitoring, all of which provide visual information. The IoT approach provides close to real-time measurements of for example temperature, humidity, and atmospheric pressure, making it possible to not only detect fires faster and more accurately, but also to predict the spreading of the fires based on the surrounding sensor values. With that data, fires can be fought more effectively, and their consequences reduced.

Another application of environmental sensing is that of measuring for example air- and water quality, for example to detect harmful pollution levels in the air [2]. Similar sensors used for sensing humidity or water-quality can also be part of systems for precision agriculture, where sensor data is used in management of the crop production [3]. This technique can improve quality and quantity of crops, as well as result in less waste of resources such as water and pesticides.

The interest in smart home devices is growing, and people are getting lighting control, energy management, and security-related sensors that either detect intruders or connect directly to fire departments. Even healthcare has introduced smart devices into peoples' homes, enabling remote healthcare checkups to monitor the well-being by measuring blood pressure, hormone and/or sugar level, body temperature and heart rate [4].

These are only a few examples of many in which IoT networks and devices are becoming more common. It is presented by Al-Amiedy et al. [5] that attacks on IoT devices are increasing at an alarming rate. Currently, 33% of exploited devices are IoT devices, while the same number in 2019 was only 16%.

In Ericssons Mobility Report from 2022 [6], it is estimated that there are around 13.2 billion IoT devices currently in the world, and that number is expected to increase to 34.7 billion devices by 2028. Considering these examples of applications, and the increase of IoT devices, it can be established that their functioning can directly affect the well-being of people, the environment and society. The compromise of a device can lead to disruption of the network and, following that, possibly disastrous consequences. Securing these networks are therefore of great importance, even more so as they are becoming more common in modern society.

1.2 Problem Statement

A challenge involved in working with these types of networks are the resource constraints of the devices; they typically operate with batteries as their energy supply, and have limited computational resources and memory. This means that the possibilities to implement extensive security features to keep the devices and networks secure are limited. However, considering the important applications, and the consequences of certain attacks, finding ways to secure IoT networks and the devices they consist of is an ongoing effort for researchers and businesses around the world.

Security has several aspects to it; we would like to be able to prevent attacks from taking place in the first place. But as history has revealed, it is not always easy and you are never completely safeguarded against attacks even with extensive safety measures taken. If an attack is carried out successfully, we want to be able to detect the attack, and be able to stop it as quickly as possible. Several tools and systems have been developed for the purpose of detecting attacks. One such recent system is DETONAR [7]. It was developed by the SPRITZ research group at Padua University and used for detecting and classifying attacks on IoT networks using the RPL protocol. DETONAR works by analyzing the network traffic, gathered by independent sniffer devices, and raising an alarm if any anomalies are found. The anomaly is then evaluated against a set of 14 attack signatures, and is either classified as one of those attacks or as a false alarm.

DETONAR relies on sniffers, as described by Agiollo [8], which overhear packets that are sent within their range. With the sniffer approach, there are some uncertainties around how it would work in a real world setting. The sniffers do not know which devices receive these packets, nor do they have any information about the nodes' internal states. The information the sniffers have, is the information in packets as they are in the air. DETONAR relies partly on information about the nodes' internal states, which I mention in Section 2.5, for example the receivers, transmitters and next-hops of packets. While there are many advantages not to depend on RPL traffic for security measures, and avoid running DETONAR on a resource-constrained border router, the usage of sniffers in this way is not feasible in many real life scenarios, due to that need of knowing the internal states of the nodes. The sniffer devices would also constitute for an extra cost to deploy, in both time and money, and the sniffer devices themselves may also be subjected to attacks, and therefore introduce an extra risk.

1.3 Project Purpose and Scope

In this project, I am investigating the possibility of using DETONAR on border router logs instead of using sniffer devices. The data needed by DETONAR is sent by the nodes in the network to the border router. The purpose is further to evaluate DETONAR's performance in this new context, as well as compare the performance of the border router approach to the sniffer approach in networks with different success rates.

To be able to collect border router logs, the simulation tool Cooja was used for this project. Cooja is an open source simulation tool, frequently used by the research community, which gives a lot of freedom for the users to configure the networks and define the node behaviour. NetSim, the simulation tool originally used to create input for DETONAR, is proprietary and lacks some of the flexibility of Cooja. In order to be able to use DETONAR on the Cooja logs, part of this project also consisted of adjusting DETONAR to work on Cooja logs.

1.4 Contributions

The main contribution of this project is that I show how DETONAR can be run on border router logs rather than sniffer logs, with almost no loss of attack classification, and even lower false positives compared to the sniffer approach.

Additional contributions are:

- I show how DETONAR can be used with logs from the simulation tool Cooja instead of logs from NetSim, which proved to require adjustments to some of the logic and the size of time windows used in DETONAR, due to differences in network configuration between the two tools.
- I explore through experiments how different network topologies affect the result of DETONAR, including completely random topologies, topologies with a minimum distance and topologies for which the attacker node is selected by specific criteria for some of the attacks.
- I evaluate how DETONAR performs on Cooja logs, comparing the original sniffer approach to a new approach: running DETONAR on logs only from the border router.

1.5 Document outline

The report is structured as follows; in chapter two, I introduce concepts relevant to understand the thesis content like the specific networks considered, the tools and devices used, the protocol RPL and the intrusion detection system DETONAR. Chapter two is wrapped up by a brief overview of some related work. In chapter three, I go into the dataset used and the network configuration used to acquire the dataset, as well as describe the attacks implemented. In chapter four, I describe what has actually been implemented in terms of code changes and explain the meaning of those changes. Chapter five holds the evaluation, which consists of a number of experiments with different setup. Conclusions and suggestions for future work are presented in chapter six.

Chapter 2

Background

This chapter introduces the following concepts needed to understand the thesis project; IoT networks, the simulation tools used, the RPL protocol and the intrusion detection system DETONAR. I also present some related works at the end of this chapter.

2.1 IoT networks

For this project, I am working with IoT low-power multi-hop sensor networks, which I will refer to as simply "IoT networks" in the rest of this thesis. This section will describe what characterizes these types of networks.

Sensor networks consist of devices sensing their environment in different ways. They typically consist of small, low power-devices with limited memory and limited computational resources. [9] The networks may consist of anything between 10 and hundreds, for some applications even thousands, of sensor nodes. The devices consist of a micro-controller, one or several sensors and a transceiver for wireless communication. The measurements of these devices are communicated to one or several border router devices, which acts as a link between the sensor network and the internet. The packets with sensor data are called application packets in this thesis. Border routers may also be called root nodes, sink nodes or gateway nodes, but in this thesis I use the name border router in order to distinguish them from the other devices in the network, which I call nodes.

What defines a multi-hop network is that devices are spread out, so that not all nodes directly reach the border router, but instead must have its traffic forwarded by intermediate nodes in order to reach the border router. In order words; it takes several hops for certain packets to reach the border router, so that the total range of the network is larger than the range of a single node.

Many sensor networks are unattended and in remote geographical areas [5]. Consider some examples mentioned in the introduction; sensors could be placed in the bottom of rivers with moving water or spread out in a battle-field beyond reach. Therefore, sensor networks are often also so-called Low power and Lossy Networks, LLNs. These networks are mostly powered by batteries, but they may also be harvesting energy from its surroundings, for example using solar power. LLNs are characterized by links typically supporting only low data rates, with low stability and relatively low packet delivery rates [10]. Often, the communication is not only point-to-point, but rather point-to-multipoint or mutipoint-to-point. The unstable communication might be due to environmental factors, since radio communication can be interfered with and affected by its surroundings.

To conclude: in this report, multi-hop, wireless networks with lossy communications and resource-constrained, low-power devices are considered. Further, I specifically consider only static networks, in which the nodes do not move, in this thesis project.

2.2 The Contiki Operating System

The devices in this project are using the operating system Contiki-NG [11]. The original operating system Contiki OS is no longer updated, with the last update being made in 2018, whereas Contiki-NG was forked out of Contiki OS in 2017 and has been continuously updated since then.

Contiki-NG is made for resource-constrained devices in the Internet of Things. It provides a low-power IPv6 communication stack for internet connectivity, and runs on a variety of platforms. The code footprint being as low as in the order of 100kB, and with memory usage being configurable to as low as 10kB, it is suitable for a range of applications in resource-constrained devices.

2.3 Cooja and Multi-Trace

Cooja [12] is a simulation tool for simulating networks of sensor nodes running Contiki. It was developed to provide a means of simulating IoT networks at several different levels simultaneously; that is, simulating the network level, the operating system level and the machine code instruction set level at the same time. Cooja can set up simulations of any topology and size and provides the means of in detail controlling the behaviour of single nodes. In this project, I use Cooja to simulate networks being the target of attacks. That means that one or several of the nodes in the network adjusts its behaviour in a manner corresponding to what an attacker could do.

MultiTrace, developed by Finne et al. [13] is an extension to the Cooja simulator, allowing users to generate randomized topologies in different ways, and run many simulations sequentially, to produce datasets of traces in a more automated way. It was for example used to generate diverse data for machine learning applications [14].

In this project, I use MultiTrace for the attack scenarios, to generate varied data traces of traffic from networks with different topologies to explore the performance of DETONAR.

2.4 RPL

The networks studied in this project implement a routing protocol called RPL, Routing Protocol for Low-Power and Lossy Networks. It was standardized in 2012 and is specified in the Request for Comment 6550 [10]. It is designed with consideration to low data rates and high error rates which results in low throughput, as is often the case in LLNs.

To understand RPL in the context of this project, the most important concept is the tree structure, how it is built and maintained, which includes the control packets of the protocol. In addition, RPL provides mechanisms like the use of timers to adjust the broadcasting interval of routing information, different objective functions after which a node's preferred parent is chosen, different modes of routing and more, some of which will be briefly mentioned in the following sections according to their relevance for this project.

2.4.1 DODAG

RPL relies on the concept of a tree structure called DODAG (Destination-Oriented Directed Acyclic Graph). RPL constructs and maintains a DODAG starting from the border router and growing "outwards", building the tree structure of the DODAG. Every node in the network is directly or indirectly through its parents connected to the border router, and there does not exist any loops in the DODAG.

The nodes establish themselves in the network and "invite" other nodes to the network by the use of RPL control packets; DAO packets, DIO packets and DIS packets.

2.4.2 RPL Lite

For this project, we consider the RPL implementation of Contiki-NG, which is sometimes called RPL Lite within the Contiki-community [15]. RPL Lite is a lightweight implementation of RPL, specifically tailored for reliability. While RPL originally supports two modes, storing- and non-storing mode, RPL Lite only support non-storing mode. The two modes differ in how the routing takes place, and what parts of the network holds information about the network structure. In storing mode, the nodes store the routing tables themselves, while the border router does not have any knowledge about how the network looks. In non-storing mode, which is used by RPL Lite, the nodes do not know how the network looks, they simply send their packets to their parents, while the border router keeps the routing tables and therefore holds all the knowledge about the network structure.

Additionally, RPL supports the possibility of several RPL instances within a network, while RPL Lite only supports one RPL instance per network and one DODAG per RPL instance. With several instances in RPL, an example could be that within a network of 10 nodes, one RPL instance consider the node with ID 1 the border router, and that RPL instance concerns itself with certain sensor data. Another RPL instance may exist within the same network, where for example the node with ID 10 is considered the border router, and that RPL instance may care about other sensor data. These two instances each have their own DODAG. As mentioned, RPL Lite is lightweight and does not support several RPL instances.

The following explanations of the three control packets DIO, DAO and DIS correspond to their meaning within RPL Lite.

DIO Packets

A DIO (DODAG Information Object), is broadcast from a node and is used to allow other nodes to discover a RPL instance and learn its configurations. The nodes receiving a DIO can then join the DODAG by selecting a parent from which it has received a DIO.

The network is set up by the border router transmitting a DIO, upon which any nodes within reach find out about the RPL instance and can join, and in turn transmit a DIO for nodes further away to join, and the DODAG structure is constructed.

DAO Packets

A DAO (Destination Advertisement Object) is sent upwards in the tree structure, from a child node to the border router. The DAO can be seen conceptually as a node communicating which node it has chosen as its parent. The child-parent relationship is then stored in the route state, and the node can upon receiving a DAO-ACK from the border router be seen as fully having joined the network and start transmitting DIOs. The DAO-ACK is a verification from the border router that the parent-child relationship has been established, but it is optional within RPL, so nodes may also just send a DAO and assume that it is accepted as the child of the chosen parent.

DIS Packets

A DIS (DODAG Information Solicitation) is broadcast by nodes willing to join a network, to request a DIO from neighbouring nodes, in order to be able to choose a parent and join the network by transmitting a DAO.

2.4.3 Rank and Version

The choice of a preferred parent is made by a node based on the rank of the nodes that are within its range, called the parent set. A lower rank represents a preferable parent, and often a shorter, or somehow better, way to the border router. There are different objective functions that can be used in RPL to calculate the rank of a node. In RPL Lite, the MRHOF objective function is used. This means that the border router has rank 128 (the lowest rank in the network), and every other node n calculates its rank r_n to be:

$$r_n = r_{parent} + 128 + ETX,$$

where ETX is the number of expected transmissions needed to communicate with its parent. ETX weighs in the quality of the link between itself and its parent, if there are two similarly ranked nodes to choose a parent from. Before joining a network, a node has rank 0. When joining a network, but before getting a parent, a node has rank 65535 (the highest value that can be represented by a 16 bit binary value). The rank system is illustrated in Figure 2.1

Another concept of RPL is the DODAG version. This refers to a construction of a DODAG, so that all nodes within a network can agree on a current structure. This is relevant when changes take place within the DODAG, for example when there is an error in the structure and it needs to be repaired. In RPL Lite, the version of a DODAG starts at 240. From 240, the version is increased when the DODAG is updated. The decision to update the version is normally made by the border router when discovering inconsistencies.

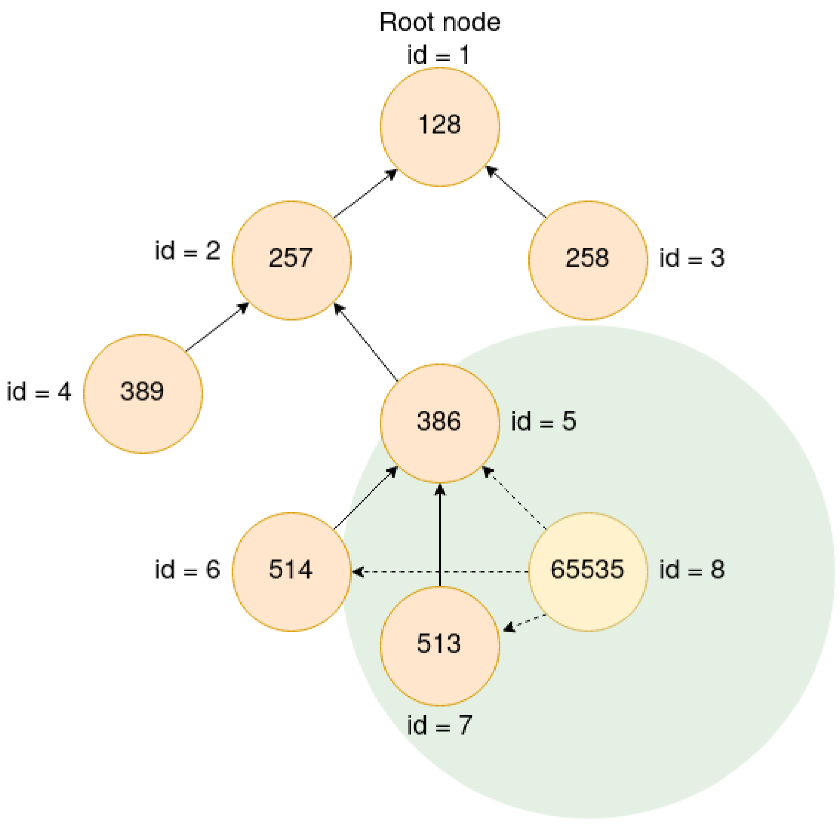


Figure 2.1: An illustration of the DODAG of a small network, to show the structure maintained by the rank system and objective function. The number on each node is its rank. The directed arrows show the preferred parents of the nodes. Node 8 wants to join the network, its transmitting range is represented by the green area. It broadcasts DIS-packet, which are picked up by any node within its reach, in this case node 5, node 6 and node 7. These nodes then each respond with a DIO to node 8. From the information in the DIOs, it will choose node 5 as its parent because it has the lowest rank out of the nodes in the parent set.

2.5 DETONAR

DETONAR [7] is an Intrusion Detection System (IDS) developed by the research group SPRITZ at the Padua university in Italy. DETONAR is developed for IoT networks implementing RPL, and uses sniffer devices to gather traffic in the network, which is then sent to a server where DETONAR runs. DETONAR can be described as a pipeline which takes network traffic as input, and gives results of whether an attack has been detected within that traffic, and if so, classifies that attack. The pipeline, as illustrated by Agiollo et al. [7] in Figure 2.2 consists of the steps; data collection, feature extraction, anomaly detection, attack classification and attacker identification.

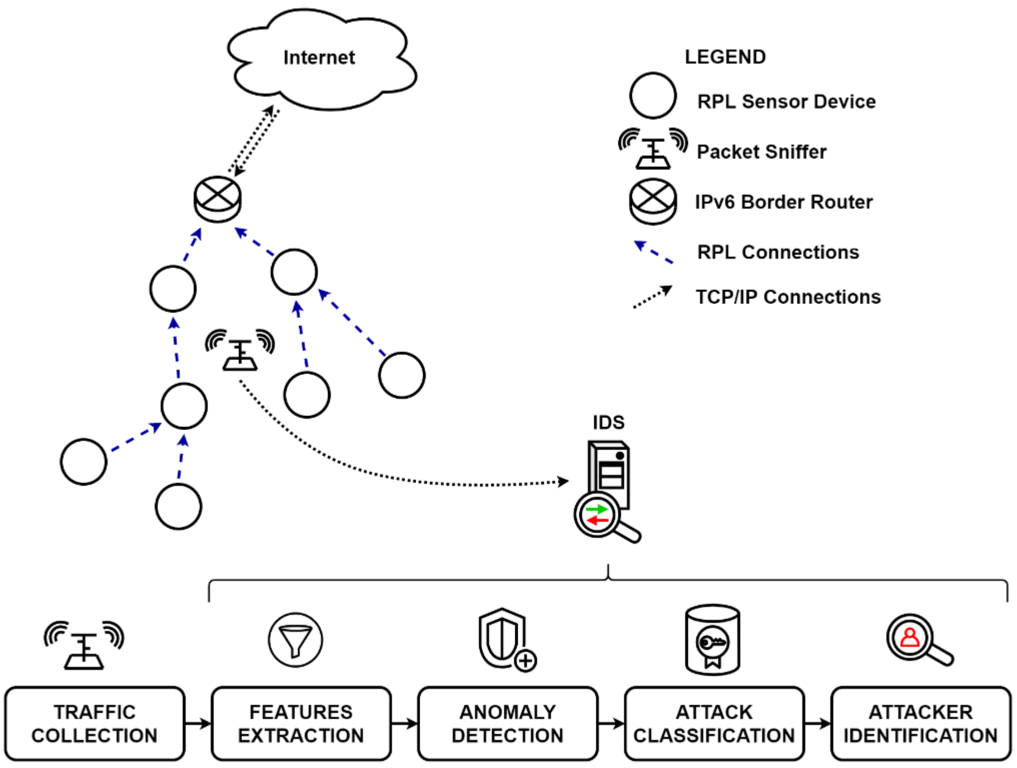


Figure 2.2: DETONAR illustrated as a pipeline.

DETONAR is developed to detect 14 different well-known attacks, which are described in Section 3.1.2. An advantage of DETONAR is the number of different detectable attacks compared to other similar IDSs, as well as the fact that it does not contribute to any RPL overhead due to its implementation of sniffers.

2.5.1 Data Collection and RADAR

The traffic data DETONAR takes as input is generated using a simulator called NetSim. The SPRITZ group used NetSim to create a dataset, called RADAR, to test their IDS. RADAR consists of packet trace files of simulations of 14 different attacks. The dataset consists of five simulations per attack, resulting in 70 simulated attacks in total. In addition to that, the dataset includes five simulations without any attacks taking place, to use for measuring false positives.

RADAR simulates the use of sniffers with perfect network coverage, so the complete traffic of the network is captured. DETONAR uses the following data points for each packet in the packet trace:

Column name	Description
Packet type	Type of control packet or an application packet.
Source id	ID of the node that initially transmitted the packet
Destination id	ID of the node that the packet is meant to reach
Transmitter id	ID of the node that transmitted the packet for this hop
Receiver id	ID of the node that received the packet for this hop
Timestamp	The time of the packet being sent
Source IP	The IP of the node that initiatlly transmitted the packet
Destination IP	The IP of the destination node
Next hop IP	The IP of the next hop in the routing of the packet
RPL Rank	The Rank of the transmitting node
RPL Version	The version of the DODAG

The simulations in RADAR last 1500 seconds each, with the attacks starting at a point in time between 500 and 700 seconds into the simulation. The margin of time prior to the attack starting in the simulation is due to the data set being primarily meant to be used as a means for performance testing the IDS. This margin is supposed to give the IDS some attack-free time for calibration.

The nodes in the RADAR simulations are configured to send application packets with an interval of 1 second.

2.5.2 Feature Extraction

DETONAR takes the input (as described in Section 2.5.1) in the form of an CSV-file and extracts information from it, while saving it to different feature files. The feature files can be divided into two different groups. The first group consists of the statistics of packets sent and received from every specific node, resulting in a feature file per sensor node as well as one feature file for the root node. For these files, each row corresponds to a time window, and the values of that row are the different packets transmitted and received during that time window. Every row includes:

- Number of DIO received
- Number of DIO transmitted
- Number of DAO received
- Number of DAO transmitted
- Number of DIS transmitted
- Number of application packets received
- Number of application packets transmitted
- Number of source IPs
- Number of destination IPs
- Number of next hop IPs

- Transmitted vs received application packets rate

The second group of feature files consists of certain columns of the complete network traffic, combined in different ways. In these files, every row corresponds to a packet being sent. The rows of these files consist of some combination of:

- Transmitter ID
- Receiver ID
- RPL Rank
- RPL version
- Packet type

DETONAR combines the use of anomaly-based intrusion detection and signature-based attack classification, so these features are used to both be able to detect an attack based on anomalies in the traffic, and to be able to classify an attack that has been detected, based on different signatures of the attacks.

2.5.3 Anomaly Detection

The anomaly detection is conducted by the use of an ARIMA (Autoregressive Integrated Moving Average) model. It is commonly used for time series analysis to predict future values, and uses only the time series itself for its prediction, without the need for prior training by any external training set. The model is fit on its own previous values. This requires the model to have an attack-free time for setup, to have legitimate attack-free history to use for it is fitting, which is reflected by the setup time in the RADAR datatraces.

The ARIMA model takes values for a time step, regresses them on their own prior values, and arrives at a future prediction. The actual values are then evaluated based on that prediction and a confidence interval. If the actual value is outside of the confidence interval, an anomaly is raised. The values used for the regression are:

- Number of DIO received
- Number of DAO transmitted
- Number of application packets received

Alarms may be raised for all types of unexpected traffic according to the model predictions, even when the traffic is not malign, since there is no way to determine that without any additional mechanism.

2.5.4 Attack Classification and Attacker Identification

To be able to decide whether an alarm was raised due to an attack or due to slight variations in legitimate traffic, and to be able to identify the attacking devices, signatures of the attacks have been defined by the team behind DETONAR. The anomaly found in the traffic is compared to these signatures, after which the anomaly is either classified as an attack or, if no signature matches, it is classified as a false alarm. The process of classification and attacker identification is described in Figure 2.3. Note that in the case of Sinkhole, Rank, Local Repair, Continuous Sinkhole and Replay attack, they are all classified as the same attack, called Rank attack by DETONAR, since they are all concerned with changed rank values.

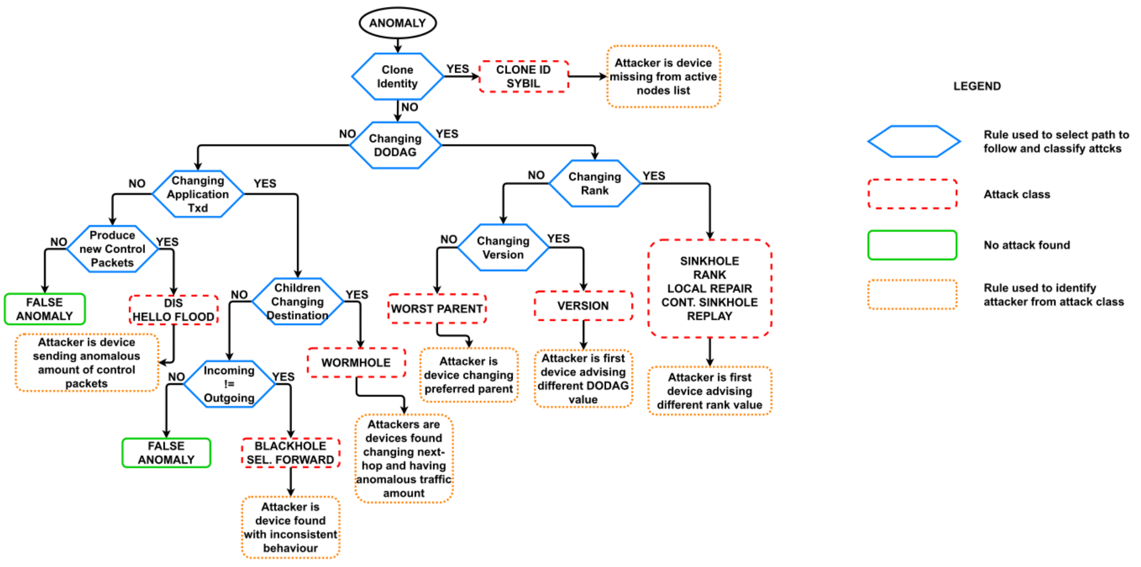

ANOMALY
Clone Identity
YES
NO
CLONE ID SYBIL
Attacker is device missing from active nodes list
Changing DODAG
Changing Application Txd
Changing Rank
Produce new Control Packets
Changing Version
FALSE ANOMALY
DIS HELLO FLOOD
Children Changing Destination
WORST PARENT
VERSION
SINKHOLE RANK LOCAL REPAIR CONT. SINKHOLE REPLAY
Attacker is device sending anomalous amount of control packets
Incoming != Outgoing
WORMHOLE
Attacker is device changing preferred parent
Attacker is first device advising different DODAG value
Attacker is first device advising different rank value
FALSE ANOMALY
BLACKHOLE SEL. FORWARD
Attackers are devices found changing next-hop and having anomalous traffic amount
Attacker is device found with inconsistent behaviour
LEGEND
Rule used to select path to follow and classify attcks
Attack class
No attack found
Rule used to identify attacker from attack class

Figure 2.3: Some caption

Once an alarm has been raised, additional information about the potential attack is extracted from analyzing the network traffic of the current time window in order to be able to classify the attack. These include:

Column name	Description
# next-hop IPs	The number of IPs the node has transmitted to during the time window
Rank changed	True if the node changed rank during the time window
Version changed	True if the version field in RPL changed during the time window
DODAG changed	True if the DODAG structure changed during the time window

2.5.5 Running DETONAR on the RADAR Dataset

DETONARs results regarding its false positives rates, its attack detection and its attacker identification as found by Agiollo when using the RADAR dataset are presented in the following section for reference, as it is interesting to relate these original results to my results, which are presented in Section 5.

False Positives

For testing false positives, Agiollo ran DETONAR on 5 simulations in RADAR that do not contain any attacks, to see if any alarms are raised and whether those alarms resulted in any attack classification. False positives were calculated as a ratio, with the total T for n devices over t time steps.

$$T = n * t$$

ARIMA false positives A depends on T and the anomalies raised a.

$$A = a/T$$

Attack classification false positives F depends on T and the number of classified attacks c.

Simulation	ARIMA false positives	Attack classification false positives
1	34.01%	0%
2	29.26%	1.58%
3	49.04%	0%
4	33.22%	0.79%
5	83.04%	0.79%
Overall	45.71%	0.63%

Table 2.2

$$F = c/T$$

In RADAR, there are 16 nodes and 150 time steps in total, of which 30 were used for training. That gives us 120 time steps remaining for predictions.

$$T = 16 * 120 = 1920$$

The results are as follows:

Simulation	ARIMA false positives	Attack classification false positives
1	2.13%	0%
2	1.83%	0.10%
3	3.06%	0%
4	2.08%	0.05%
5	5.19%	0.05%
Overall	2.86%	0.04%

Table 2.1

My comment to the above result is that another interesting thing to consider regarding the false positives, are the number of false attack classifications per time step, rather than per time step AND device. I consider this relevant, since we want to know how often an alarm is raised wrongly, as wrongly raised alarms might require some action regardless of how many devices raise the alarm that time step. The corresponding numbers for this measure would be adjusted by just using

$$A = a/t$$

and

$$F = c/t$$

That yields the following results:

Attack Detection

For each of the five simulations of every attack scenario, the attack classification and attacker identification results are the following:

Attack scenario	Correctly classified	Attacker correctly identified
Blackhole	3/5	3/5
Selective forward	5/5	5/5
Sinkhole attack	5/5	5/5
Continuous Sinkhole	3/5	3/5
HELLO Flooding	5/5	5/5
Clone ID	5/5	5/5
Sybil	5/5	5/5
Wormhole	4/5	3.5/5
Version	4/5	4/5
Rank	5/5	5/5
Replay	5/5	5/5
Worst parent	4/5	3/5
DIS	5/5	5/5
Local Repair	2/5	5/5
Average	4.29	4.39
Average percentage	86%	88%

Table 2.3

The numbers range from lowest of 40% attack classification in the Local Repair-scenario to 100% classification for 8 out of 14 scenarios. The average is 86% for classification and 88% for attacker identification. However, it is worth to keep in mind that due to the few simulations for each scenario, the significance of these results remains unknown.

A comment to the above results is that within the simulations that are considered correctly classified, there may also be false positives that are not taken into account in these results. In some cases, the specific attack conducted is detected, and nothing else, but in other cases there are other attacks detected both before and after the actual attack is detected. What to make of that may differ between different applications of an intrusion detection system.

2.6 Related Work

In this section, I will present other projects that are concerned with detecting attacks in networks using RPL. For attack definitions used in this project, see Section 3.1.2.

2.6.1 SVELTE

SVELTE [16] is an IDS that aims to detect Sinkhole, Blackhole and Selective Forward attacks. The system relies on report packets sent to the border router, which the border router use to detect attacks. Anomalies are detected by:

- Detecting inconsistencies in the reported information, like nodes reporting different ranks for the same node
- Detecting whether a node is offline or has stopped transmitting packets, for example because the packets are dropped by a blackhole attacker
- Finding inconsistencies in the DODAG, for example a child having a higher rank than its parent

This detection is based on simple thresholds, for example a threshold on the time since receiving our last packet, to raise alarm for a blackhole attack. As a measure to reduce false positives, inconsistencies are required to be consecutively reported over several time steps for the same nodes.

In terms of evaluation, SVELTE is tested on simulations of sinkhole attack to measure true positive rate, and selective forwarding attack to measure detection rate, using both lossless and lossy networks, and for three different sizes: 8 nodes, 16 nodes and 32 nodes. Following are the results:

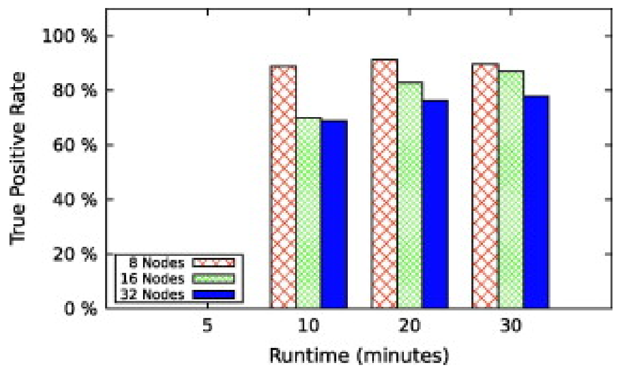
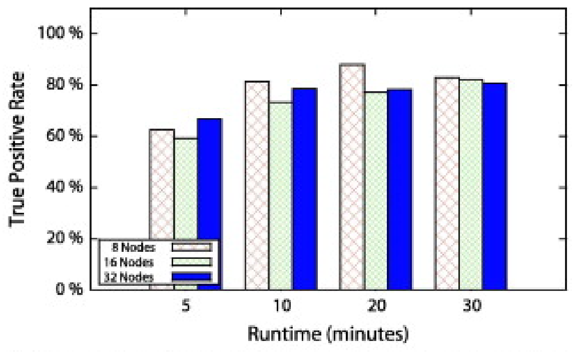
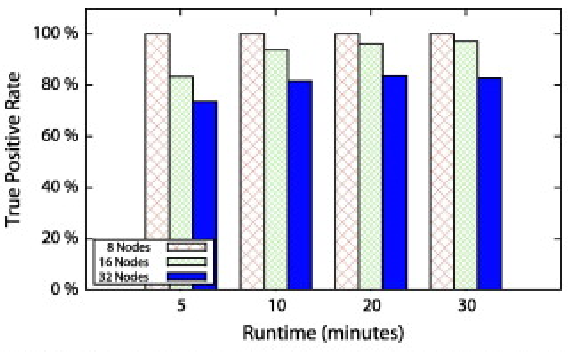


Figure 2.4: True positive rate of sinkhole attack



(a) *Lossy* network suffering from selective forwarding attack



(b) *Lossless* network suffering from selective forwarding attack

Figure 2.5: True positive rate of Selective Forward attack

For both Selective Forward in a lossless network and Sinkhole, we can see that smaller networks give better results than bigger ones. For Selective Forward in lossy networks, the results seem to depend less on the size of the networks but have overall worse true positive rates of around 70-80% compared to 70-100% of lossless networks.

2.6.2 INTI

INTI [17] is an IDS which is implemented as a modification of the RPL protocol to detect Sinkhole attacks, or rather a combination of Sinkhole and Blackhole attack according to the definitions of the attacks used in this project, found in Section 3.1.2. INTI disables the DODAG construction within RPL, and instead classifies nodes as either: free nodes, cluster member nodes, associated nodes, leader nodes and one border router. Using this structure in the network, each node has one parent which monitors its traffic and sends it to the border router to detect attacks.

INTI was tested in relation to SVELTE, for the Sinkhole/Blackhole-scenario, and was shown to perform slightly better in classification for a static network, where the nodes do not move, while it outperformed SVELTE significantly for a dynamic network, with moving nodes. The networks used for evaluation consisted of 50 devices, out of which 20-30% were border router devices.

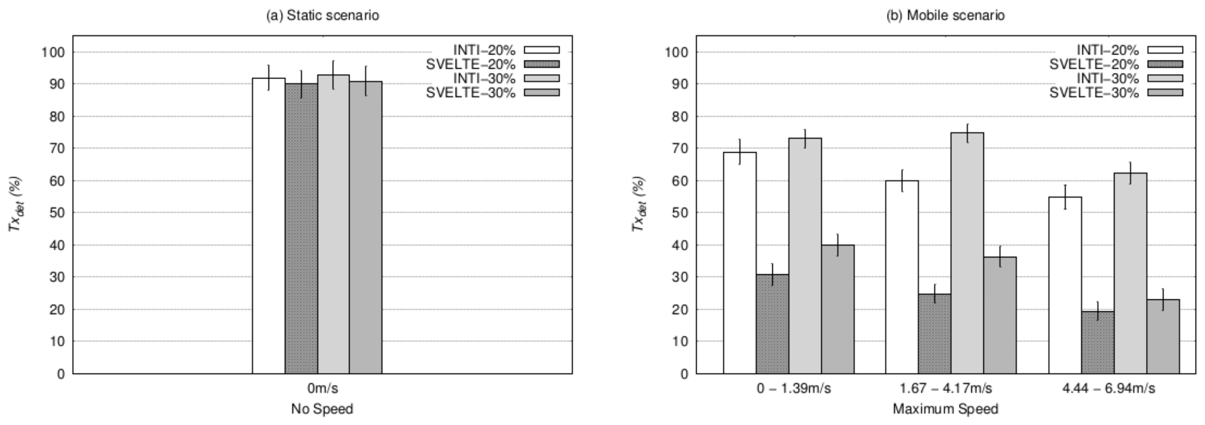


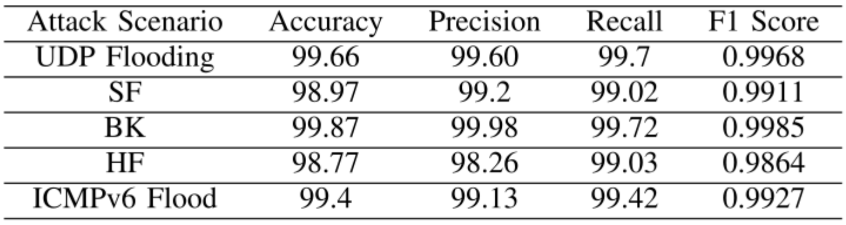
Figure 2.6: Sinkhole attack detection rate of INTI compared to SVELTE

While I do not consider mobile networks in this project, it is interesting to see the performance of such systems for comparison. INTI seems to have acquired a detection rate of around 90% for static networks, and around 55-75% for mobile networks.

2.6.3 IoT-Sentry

The IoT-Sentry [18] IDS was developed to detect the following attacks: UDP Flooding, Hello flooding, ICMPv6 flooding, Blackhole and Selective Forward. It works by a sniffer approach, just like DETONAR uses originally, to capture the network traffic. Features are extracted from the network traffic, for example the number of packets being sent for each node and the transmission rates of different packet types. The model used for detection is pre-trained on data from simulations of the different attacks before use.

Following are the recorded results:



Attack Scenario	Accuracy	Precision	Recall	F1 Score
UDP Flooding	99.66	99.60	99.7	0.9968
SF	98.97	99.2	99.02	0.9911
BK	99.87	99.98	99.72	0.9985
HF	98.77	98.26	99.03	0.9864
ICMPv6 Flood	99.4	99.13	99.42	0.9927

Figure 2.7: Detection results of different attacks of IoT-Sentry

One significant difference between this project and DETONAR is the need to train the model prior to usage. This relies on good data to train the model on, which could be a disadvantage in certain cases.

2.6.4 Related Work Conclusion

These related projects represent different approaches; in SVELTE and INTI, the approach used corresponds to the border router approach evaluated for in this project, while IoT Sentry use a sniffer approach similar to the one originally used in DETONAR. Since the above presented works are mainly concerned with a single or a few attacks, DETONAR contributes by giving us a possibility to explore a wider range of attacks.

SVELTE is evaluated using various sizes of networks, ranging from 8 to 32 nodes, while IoT-Sentry is evaluated on networks of up to 100 nodes. For INTI, only one size of networks are used, of 50 nodes. In contrast, I only run DETONAR on networks of 16 nodes in this project. That is the size used to evaluate DETONAR originally as described in Section 2.5. Testing DETONAR on different sized networks, as well as several other differences to my project and these related works may be useful to explore as future work in combination with DETONAR.

Chapter 3

Design

This chapter covers the dataset I used in this project, which consists of data traces generated by using Multi-Trace and Cooja, as well as the collection of the dataset.

3.1 Dataset

This section will outline the dataset used; starting with the network configuration, followed by the selection of attacks used for the evaluation.

3.1.1 RPL Configuration

RPL networks can vary in their configuration, which is relevant for this project since the anomalies are raised and predictions are made in regard to the control packets and application packets of the network. When doing the initial adjustments to allow for DETONAR to run on Cooja logs, the results I acquired were very bad; DETONAR raised false alarms of different attacks for almost every timestep. The results made me look at the differences between the Cooja logs and RADAR, so that I could understand what needs to be adjusted in order for DETONAR to be useful on Cooja logs.

The RADAR dataset differs quite a lot from the data traces acquired from Cooja. This is due to differences between the NetSim simulator's default configurations for RPL, which is not entirely open for anyone to take part of, and Cooja configurations for RPL. Since the NetSim RPL configurations are not openly available, I investigated the configuration through the dataset itself, as well as through experiments. Through investigation and trial and error of different configurations, I found the following differences between the two network configurations:

- NetSim does not use APP-ACK packets, sent from the border router to acknowledge that it received an application packet
- NetSim does not use DIS packets, unless explicitly configured to do so
- Different objective functions are used; NetSim has its own objective function
- NetSim uses storing mode while Cooja uses non-storing mode
- NetSim has a more aggressive use of RPL control packets than Cooja, transmitting DIO and DAO packets more frequently

Given the high amounts of false positives when initially running DETONAR on the Cooja logs, some of these points needed to be addressed. The first point in the above list was easily

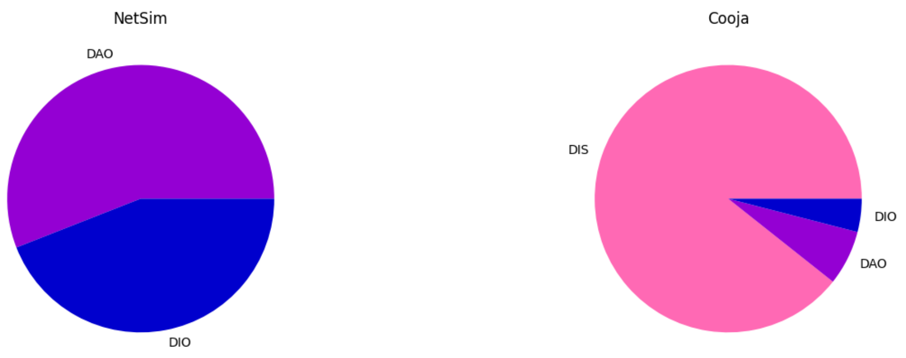


Figure 3.1: Ratio of control packets in datasets generated by NetSim and Cooja.

addressed by configuring the border router to not send APP-ACK packets. The second point is left without taking any action, since using DIS packets is preferable in low-power networks. The third and fourth points in the above list of differences in network configuration creates the need of some adjustments for running DETONAR on a dataset generated by Cooja at all. This is explained in Section 4. The fifth point needed some analysis to understand the differences.

Investigating RPL Control Packet Frequency

By investigating the RPL control packet frequencies in RADAR and in the Cooja logs, I want to understand if I need to adjust the time windows used by DETONAR and if so, how it needs to be adjusted. In RADAR, the application packets are sent once every second and the time windows are 10 seconds long, so there are 10 application packets sent within each window. When running Cooja with the same application packet intervals as RADAR, the number and ratio of control packets were substantially different, which gave extremely high false positive rates in all scenarios. To compare the rates of control packets, all control packets in the 5 simulations without attacks used for testing the RADAR dataset was used, as well as 5 simulations without attacks generated by Cooja. These numbers were found, when taking the average over the 5 simulations:

Packet type	RADAR	Cooja
DIS	0	1341.2
DAO	84972.6	99.8
DIO	66895.2	60.2
Total	151867,8	1501.2

Table 3.1: Average number of packets per simulation in RADAR and Cooja logs respectively. The average was taken over 5 simulations, with no attacks present in any simulation.

We can see that there are huge differences in both overall number of packets, and in ratio between different types of packets. As seen in Figure 3.2, RADAR consists of slightly more DAO packets than DIO packets, while the Cooja dataset is heavily dominated by DIS packets. In Figure 3.3, we can see that the ratio of DIO and DAO packets in Cooja traces are slightly different from the ratio in RADAR, such that Cooja datasets have more frequent DAO packets

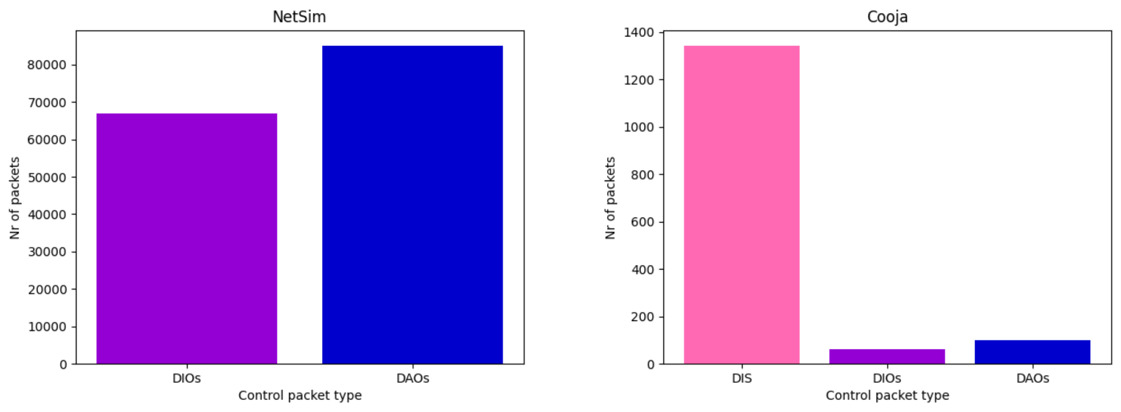


Figure 3.2: The number of control packets in RADAR (generated using NetSim) and datasets generated using Cooja. Note the different scales on the y-axis. The numbers are from 5 simulations without attacks for each simulation tool, and correspond to the average total number of control packets for a whole simulation.

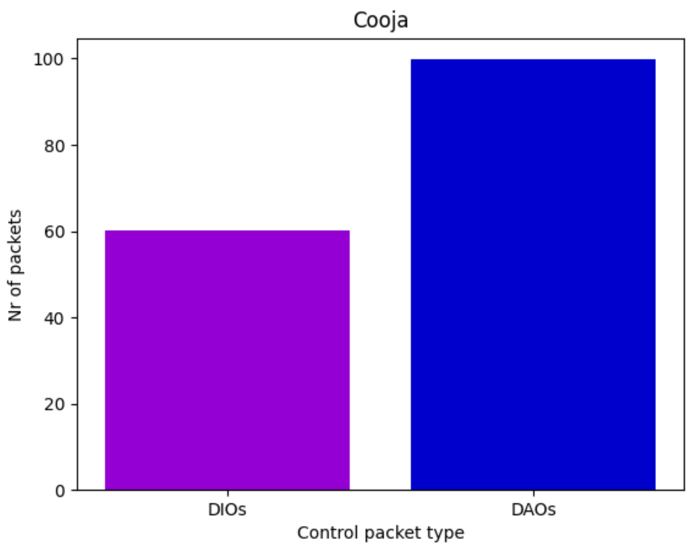


Figure 3.3: The number of DAO and DIO packets in datasets generated by Cooja. The numbers are from 5 simulations without any attacks, and correspond to the average total number of control packets for a whole simulation.

in relation to the number of DIO packets. The most notable difference is the number of packets overall.

One conclusion to draw about the NetSim RPL configurations is that it does not take resource efficiency into consideration. The fact that no DIS packets are used, for example, means that devices which are trying to join the network needs to wait for a DIO packet rather than requesting one when it is needed by transmitting a DIS packet. This may explain the high number of DIO packets transmitted constantly by all devices, which of course drains a lot of resources. In the same way, the substantially larger amount of DAO messages reflect the same lack of consideration for power consumption. Instead it seems that the constant updating of parent-child relationships in the network are prioritized within NetSim.

These differences in control packet frequency seemed to have a big effect on the predictions of the IDS, resulting in many false positives as mentioned earlier. I experimented by making the time window used by DETONAR larger, and reducing the frequency of sending application packets. Doing so, the number of false positives decreased. For the experiments to evaluate DETONAR's performance, I chose a time window of 10 minutes, which is 60 times bigger than the one used in RADAR, and I chose a rate of one application packet per minute. These changes adjust the frequency to the less aggressive RPL configurations of Cooja, while maintaining the same relationship between number of application packets per time window. Considering the resource constrained nature of IoT networks, sending application packets less frequently is often preferred in real-world scenarios.

3.1.2 Attacks on RPL-based Networks

How attacks are defined or interpreted may vary between different projects/classifications of attacks, so this section will outline what is meant by the included attacks in the context of DETONAR and this project, after briefly describing some attacks that were excluded. The definitions of the attacks are corresponding to the definitions used by Agiollo in [8], since the goal is to evaluate the performance of DETONAR in a new context. Using the same definitions allows me to easier make comparisons to the original results. All attacks are taking place in an IoT network using RPL, and in these attack scenarios, an attacker is assumed to have gained control of one node.

Attacks Excluded from this Thesis

In this project, six out of the 14 attacks handled by DETONAR were excluded. Below is a list of brief descriptions of the excluded attacks.

- Clone-ID and Sybil attacks - An attacker node takes on the identities of one or several other nodes.
- Rank attack - An attacker node advertises a high rank value, to avoid being chosen as a parent.
- Replay attack - An attacker node replays control packets it receives as if the packets originated from the attacker node itself.
- Worst parent attack - An attacker node chooses a parent with a lower rank value.
- Wormhole attack - Two attacker nodes communicate directly through a ”tunnel”, outside of the RPL network, and redirect packets through this tunnel.

Out of the 14 attacks detectable by DETONAR, certain attacks are much more destructive than others. The above attacks aim to create sub-optimal routing and/or confusion in the routing. This might waste resources of the devices in the network, and delay packets. All of the above attacks have a comparably small effect on the network, and I therefore deemed them less relevant and decided to focus on the following attacks.

Blackhole and Selective Forward Attacks

The attacker node drops all (Blackhole) or some (Selective Forward) of the application packets it receives, such that only control packets will be processed normally. This results in loosing all or some packets of the sub tree of which the attacker node is the parent. The attack mainly affects the packet delivery ratio, and its severity is highly dependent on the placement of the attacker node within the DODAG. For example, the closer to the root node, the higher the chance of more severe packet loss, since it's likely that more nodes will route their traffic through the attacker node. The consequence of this attack is either lower performance (due to high packet loss) in the sub tree of the attacker node, or complete denial of service in the sub tree of the attacker node. This attack is more effective when combined with other attacks like the sinkhole attack, since it allows the attacker node to attract more nodes to choose it as their parent.

In Cooja, I implemented Blackhole attack by having the chosen attacker check all application-packets that it processes, and drop every packet that is supposed to be forwarded. The attacker node itself keeps sending application packets normally. The selective forward attack works the same way, with the only difference being that the attacker node only drops 50% of the application packets it is supposed to forward.

Sinkhole and Continuous Sinkhole Attacks

These attacks have in common that the attacker node changes its rank value to modify the DODAG structure, though for different reasons.

In the case of a Sinkhole attack, the attacker decreases its rank value to appear as a better parent than other nodes currently chosen as preferred parents of the surrounding nodes. This will trigger the surrounding nodes to redirect their traffic to this attacker node, resulting in control over larger parts of the traffic. This attack can be especially destructive when combined with other attacks, like the Blackhole attack. I implemented this attack in Cooja by having the attacker node adopt the rank value of 256, making it appear to be a direct child of the root node.

The Continuous Sinkhole attack is characterized by the attacker node repeatedly changing its rank value. The surrounding nodes will have to keep changing their preferred parent, and compute new rank values. This attack aims to delay packets and damage the packet throughput, while draining resources of the surrounding nodes, rather than gain control of the traffic as in the sinkhole attack. In Cooja, I implemented the attack by having the attacker node change rank to a random value between 128*2 and 128*8, every time the attacker node sends a DIO.

HELLO Flooding and DIS Attacks

High amounts of control packets (DIO in the case of HELLO Flood and DIS in the case of DIS attack) are broadcast by the attacker node, which floods the network and demands processing of the neighbouring nodes. This results in higher delay and packet loss probability for the neighbouring nodes, and resource waste since the surrounding nodes need to process the flooding packets.

I implemented HELLO Flood by having the attacker node transmit DIO packets every 10 seconds, and the DIS attack by having the attacker node transmit DIS packets every 5 seconds.

Version Attack

The DIO contains information about the version of the DODAG, which should be changed only by the DODAG root. When it is changed, the nodes need to reconstruct the DODAG and therefore start transmitting control packets needed for doing so. An attacker node can with a forged DIO advertise that the version has changed, as RPL does not have any mechanism preventing it, and therefor prompts other nodes to spend their limited resources on reconstructing the DODAG. Since changing the version is done by the root node, an attacker node that changes the version will redirect traffic to itself in a manner similar to a Sinkhole attack. The result is a modified DODAG structure with a new root node, but the structure will eventually converge towards the orginal DODAG unless the attacker node keeps transmitting forged DIO packets.

I implemented the version attack by having the attacker advertise a bumped version value in one DIO packet.

Local Repair Attack

RPL has repair mechanisms (local and global), which are used if there is a routing topology failure or a node failure. In the case of a node failure (a parent node) or a link failure, a local repair mechanism tries to find a new parent or path. If there are more local failures, RPL performs the global repair which means rebuilding the whole DODAG. Inconsistencies in the network may trigger these repairs, and these can include when routing loops are detected, when a new node joins or when a node moves within a network.

The local repair attack consists of the attacker node forging packets that trigger a repair. I implemented this by having the attacker node send a forged DIO packet containing an ”infinite rank value”, the value 65535, causing the children of the attacker node to detach and look for a new parent. Reconstruction will take place once the detached nodes receives DIOs from surrounding nodes. This attack may either introduce delay or packet loss, depending on the circumstances. It also consumes resources to reconstruct the DODAG. The forged DIO packet was sent by the attacker every ten seconds after the attack started.

3.2 Collection of Data with Cooja

Part of this thesis is about making DETONAR work for Cooja, since it is an open source tool available to anyone and frequently used by the research community. The work of adjusting DETONAR to work together with Cooja and a new context was done in steps, in order to be able to both compare the performance of DETONAR between the two simulation tools, using the sniffer approach for both, and be able to compare the performance of DETONAR on Cooja logs between the two approaches. The following steps were taken:

1. Running DETONAR with the sniffer approach, but with the data collected from Cooja instead of NetSim
2. Running DETONAR, with data sent as statistics from nodes (the border router approach)

These two steps require different data traces generated by Cooja. There exists several types of logging in Cooja. One of them, the radio log, consists of all communication in the network. That is, all IP packets being sent within the network, both application packets with sensor data and control packets of different types, like the RPL control packets discussed in Section 2.4. The radio log does not correspond to any actors, like a nodes' or the border router's, perspective of the network, but gives the full traffic that has taken place within the network. In contrast, the node logs consist of anything the nodes decide to log, and may consist of very different data compared to the radio log. For example, a user of Cooja can define node behaviour, including the node logging, and include it to run on Contiki. In that way, different types of nodes within a network may also log different things.

In this project, gathering the data is done by getting the logs of the nodes, for several reasons. Firstly, the logs can be defined by us to include the data needed as mentioned previously. Secondly, they correspond to a more realistic scenario in which there is no entity with perfect knowledge of what is going on in the network, in the way that the data in the radio log might give the impression of.

3.2.1 Collecting Data from Nodes in Cooja

Since we are considering a new simulation tool for generating the data, adjustments first had to be made for Cooja to produce logs similar to the RADAR dataset. RADAR and DETONAR was analyzed to understand the differences between RADAR-logs and Cooja-logs.

The data needed for each entry in the dataset is available at the node level in Cooja. First, the nodes' behaviour had to be updated to allow for the needed information gathering, such that each node logs the correct information, as described in Section 4.3. Gathering the data in Cooja then corresponds to running Cooja on a simulation, getting the node logs for each node in that simulation and combining all the logs to a complete dataset of the traffic within the network, and parsing it to fit the requirements of DETONAR. This complete traffic is then used as input to the feature extraction of DETONAR.

3.2.2 Collecting Data from Border Router in Cooja

In the case of running DETONAR on a border router, data is sent from every node to the border router at some time interval, which requires the changes to the node behaviours described in Section 4.3. Collecting the data in this case corresponds to only taking the logs of the border router after running a Cooja simulation. The DETONAR pipeline is slightly adjusted by using the border router approach, since the feature extraction-step of DETONAR requires the full traffic of the network. The feature extraction-step is replaced by a step of statistic parsing, with the goal of arriving at a similar output as the feature extraction in order to be able to use DETONAR on the border router logs.

Chapter 4

Implementation

The implementation of this project corresponds to two things; firstly the changes and additions to the python code which DETONAR consists of. Secondly, the addition of new modules written in C, that run on the nodes in order to gather and log our needed data.

4.1 Running DETONAR on Node Logs

Running DETONAR on node logs corresponds to the ”sniffer approach”. Following is a summary of the changes made to the DETONAR code to be able to run on Cooja logs at all:

- Adding support for using another simulation tool, including some parts of DETONAR which used values specific for NetSim-simulations. This also included changes made due to the different datasets being collected in different ways; in DETONAR, every packet was logged by the receiver, while the packets were picked up mid-air in RADAR, which resulted in several logic changes.
- Adding support for being able to configure the time windows used, motivated by the difference in RPL aggressiveness.
- Change handling of rank due to differing objective functions and rank systems. This includes changes made to find the correct attacker after a rank attack has been classified.
- Change handling of reconstructing the DODAG from DAOs, due to Cooja using non-storing mode

4.2 Running DETONAR on Border Router Logs

When running DETONAR on a border router, the input data differs from the sniffer approach, due to the border router's limited access to the network traffic. This requires changes to the DETONAR code. During the anomaly detection, the number of DIOs received, the number of DAOs transmitted and the number of application packets received are used for detecting anomalies. The anomaly detection can be done in the same way when running DETONAR on border router logs instead of using the sniffer approach, since the number of different types of packets can be included in the statistics sent by nodes to the border router. However, some actions made by DETONAR after the anomaly is raised is not possible to do the same way. When an anomaly is raised, DETONAR does the following:

1. Checks whether a node has ”disappeared”, ie stopped sending and can be considered being a victim of Clone-ID or Sybil attack

2. Checks whether the rank of the nodes has changed, and if it has; which node changed rank first to determine the attacker
3. Checks whether version has changed, and if so; which node changed version first to determine the attacker
4. Reconstructs DODAG before the anomaly and after the anomaly, to see if the DODAG has changed
5. Gathers the set of ”anomalous nodes” which are defined as the nodes raising anomalies and their neighbours

The following sections will address the changes needed for DETONAR to work on a border router in regards to these points, and the design choices made when implementing the changes.

4.2.1 Checking for Clone-ID and Sybil Attacks

The trivial approach to do this would be for the border router to simply check which nodes have reported statistics within a time window and which have not. This section will describe why the trivial approach doesn't work.

For the Blackhole attack, the statistics packet of the attacker's children will never arrive at the border router. Therefore, the border router will not receive any traffic from those nodes except for the control packets. In our implementation of the Blackhole attack, control packets are still forwarded to the border router, as that was the corresponding implementation of the Blackhole attack in RADAR. Since Contiki uses RPL in non-storing mode, the DAOs are forwarded from nodes to the border router. In our case, using the RPL configuration of Contiki which takes resource efficiency into consideration, DAOs are not sent very often. This means, that most of the times, the border router will have no evidence of the attackers childrens existence once the Blackhole attack is started. Considering this, and the fact that I decided to not include the Clone-ID and Sybil attack in my evaluations, I decided to remove this functionality in DETONAR. It is worth to mention since not removing the check for Clone-ID and Sybil attack would yield results of all Blackhole and Selective Forward-attacks being classified as Clone-ID attacks.

This change in DETONAR might not be needed in all circumstances. Given a bigger time window size, it is more likely that the border router received a DAO from every node in the network during this time, and in that case this problem would not be present. However, exploring this question is out of scope for this thesis project.

4.2.2 Checking Rank and Version

The rank and version of a node can be sent to the border router together with the rest of the statistics. However, there may be several ranks for a certain node within a time window, and the mechanism of DETONAR to determine the attacking node depends on the times of rank changes.

Several approaches are possible to make the necessary data available to DETONAR. For example, one could include all ranks and versions that a node have during a time window, as well as the times of changing to that rank.

I chose a different, more trivial approach: Each node sends the rank and version it has at the time of sending the statistics packet, and the times of rank and version change. If no change has taken place, the time is zero.

4.2.3 Reconstructing DODAG

This is done in DETONAR by analyzing all the DAOs transmitted within the time window. In RADAR, the DAO is sent to the chosen parent rather than to the border router. Therefore, the structure of the DODAG can be decided by looking at the most recently transmitted DAO for every node to its parent.

However, in non-storing mode, the border router receives the DAOs from all nodes within the network, and therefor maintains its understanding of the network and its changes as time passes. Reconstructing the DODAG then just means using the DAOs sent to it. Because of the default RPL configuration, DAOs are not often sent and there is more often than not time steps that do not include any DAOs at all. It was solved by checking further back in history until the latest received DAO is encountered. An alternative could be to store the parent-child relationships in the state of DETONAR, and update it when receiving new DAOs.

4.2.4 Gather Neighborhood

Finding out the set of so called "anomalous nodes", requires us to know which nodes are close to each other within the network. The anomalous nodes are defined as the nodes raising an anomaly, as well as their neighbors. Using RADAR as input, finding the neighborhood is done by analysing DIO packets since they are all present in the dataset. Since DIO's are broadcast, they are received by all nodes within reach of the broadcasting node. That way, DETONAR knows which nodes are in proximity to each other. However, with the approach of running DETONAR on the border router, this becomes tricky since the border router does not hear all DIO packets in the network. Instead, I decided to send the list of neighbors from every node together with the statistics. Whenever a node heard a new node, it adds it to the list. This gives the consequence of sometimes sending the exact same list forever, if the network doesn't change, which of course is wasteful. The alternative is for DETONAR to keep the information of neighborhoods in the state and only send updates of new neighbors. That would require maintaining the list of neighbors at the node level which consumes resources as well. Which alternative is preferable might differ depending on circumstances.

4.2.5 Adjustments to DETONAR for Running on a Border Router

The adjustments made to DETONAR specifically for running on border router logs can be summarized as:

- Added support for parsing statistics into time windows, which replaces the feature extraction step used previously.
- Changed how to reconstruct the DODAG from DAOs, due to Cooja using non-storing mode.
- Changed handling of rank and version as described above.
- Removed Clone-ID/Sybil attacks.

4.3 Adjustments to Nodes' Behaviour

Besides the above changes in the DETONAR source code, new behaviour of the nodes in the network was defined so that it could be run on the nodes of the network, and the border router. For the sniffer approach, all nodes and the border router acted the same way; they simply logged the information needed by DETONAR for each packet they received. That information corresponds to the data points described in Section 2.5.1.

For the border router approach, I added the support for logging and sending the following statistics. This was put in a new service-module which can be included to run in Contiki. This means, if the node is running this module it is for every configurable time interval sending the data points:

- Number of DIO received
- Number of DIO transmitted
- Number of DAO transmitted
- Number of DIS transmitted
- Number of APP received
- Number of APP transmitted
- Time of rank change
- Time of version change
- Current rank at end of time window
- Current version at end of time window
- List of heard nodes

A suitable time interval to send statistics should be smaller than the time window size used in DETONAR, to make sure that the border router would receive at least one packet of statistics per time window, since there can be some fluctuation to exactly when a packet is sent. Apart from logging the statistics, all DAOs received by the border router was logged as well, to be able to reconstruct the DODAG.

Adjustments made in regards to the nodes behaviour can be summarized as implementing ways of:

- Simulating 8 attacks in Cooja, creating simulation scripts for each of the attack scenarios.
- Sending the statistics needed by DETONAR from sensor nodes to border router.
- Logging the statistics sent by the nodes, and the statistics of the border router itself, as well as the DAO packets at the border router.

4.4 Automating Processes

Apart from the code I wrote for the Contiki devices, and the changes I made in the DETONAR source code, the following additions were also a big part of making the project run smoothly:

- Automating the generation of the dataset such that a user can just place the results in the DETONAR folder and run the workflow. The following steps are done:
    1. Generates topologies using MultiTrace
    2. Run Cooja on all generated topologies

3. Parse the node logs, both for running DETONAR using the sniffer approach and for running DETONAR using the border router approach

- Automating the workflow of running DETONAR. Given a folder with the dataset in, the workflow:
  1. Extracts features/parses statistics (depending on approach) for all scenarios in the folder and saves it into features/statistics-files
  2. Runs the IDS over all features/statistics
  3. Parses the result-files and saves a summary-file of DETONARs performance over all runs

Chapter 5

Evaluation

This section will cover the different experiments made to investigate the performance of DETONAR. I start by describing the different metrics considered for the results, like attack classification, attacker identification and false positives. Then I present the experiments aiming to understand how different ways of generating topologies affect the results of DETONAR, using the sniffer approach. After that, the sniffer approach is compared to the border router approach.

In all experiments, there are 16 nodes in the network, since that was the network size on which DETONAR was originally tested. Each node has a transmitting range of 50 meters and and interference range of 100 meters. The topologies of the networks vary between the experiments. For all experiments with random topologies, the topologies are generated using MultiTrace. The nodes are randomly placed such that each node is within transmitting distance to at least one other node.

Attack Classification and Attacker Identification

I present four metrics for results, apart from false negatives, for each experiment:

- Alarm raised - The percentage of simulations in which *any* alarm was raised *after* the attack has taken place. This metric is mainly used to be able to distinguish between cases for which DETONAR completely miss the attack taking place, and cases where DETONAR recognize that an attack is taking place, but classifies it incorrectly.
- Correctly classified - The percentage of simulations in which an alarm raised *after* the attack has taken place is classified as the correct attack.
- Attacker identified - The percentage of simulations in which an alarm that was raised *after* the attack has taken place leads to the attacker node being correctly identified
- False positives - The percentage of simulations in which there are any alarms of attacks raised *before* the attack has taken place, ie false positive alarms are raised. This metric is mainly used to be able to compare between runs, since false positives are measured more extensively in separate simulations for all experiments.

When DETONAR was originally run on RADAR by Agiollo [8], the two metrics "Classified" and "Attacker identified" were presented, see Table **??**.

False Positives

I also give a table of false positives in simulations without attacks for each experiment setup. I present three measures of false positives per setup.

- False positives present in simulation - The percentage of simulations in which there is a false positive present at all, p. Any alarm raised during these simulations are false positives, so the first measure p depends on the number of simulations where at least one alarm is raised s and the total number of simulations S.

$$p = \frac{s}{S}$$

- False positives per time step - The ratio of number of false positives raised in total r and the number of total time steps T determines the false positives per time step f. T depends on the number of total simulations and the number of time steps per simulation.

$$T = S \times t$$

$$f = \frac{r}{T}$$

- False positives per time step and device - F depends on the number of raised alarms in total r, and the number of total time steps and devices.

$$F = \frac{r}{T * d}$$

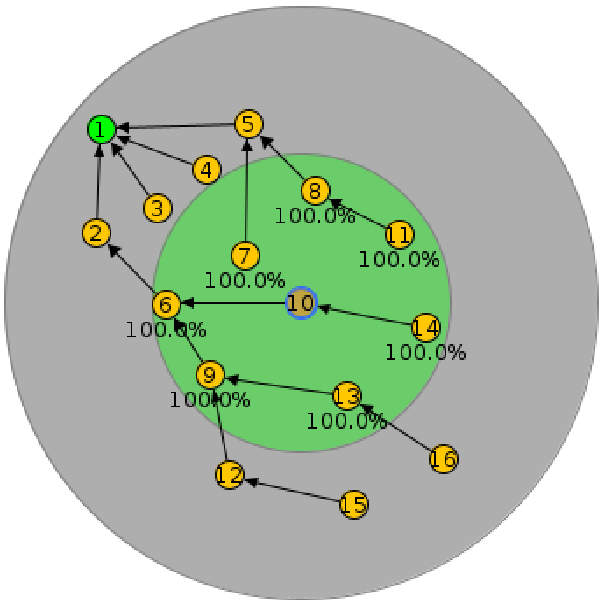


Figure 5.1: An illustration of the "initial topology", used in the first experiments. The green area is the transmitting range, and the grey area is the interference range.

5.1 Exploring Topology Generation

These first experiments of running DETONAR on Cooja logs were made using the sniffer approach, in order to explore how the results of DETONAR would be affected by using topologies that were generated in different ways; completely random topologies, topologies where a minimum distance between nodes is enforced and topologies where the attacker in certain attack scenarios is selected based on some criteria.

5.1.1 Initial Run of DETONAR on Cooja Logs

The first experiment using data traces from Cooja were run on small numbers of simulations, to get some idea of the performance. The experiments were run on the same number of simulations as originally, 5 per attack, for 10 out of the 14 original attacks.

For this experiment, all networks have a similar topology, which is a pretty even spread forming a tree, with a network diameter of 5, see Figure 5.1. In this topology, all nodes reach at least three other nodes, and some of the central nodes reach 7 other nodes, which constitutes a big part of the whole network. The attackers were chosen randomly except for the case of Blackhole and Selective Forward attacks, where I made sure that the attacker node had children for at least some time after the attack, so that it should have some effect on the traffic and therefore be detectable.

Experiment Setup

- 5 simulations per scenario
- Simulation time: 27000 seconds
- Attacks start at 21000-23000 seconds
- Using an "initial topology", shown in Figure 5.1

Attack Classification and Attacker Identification

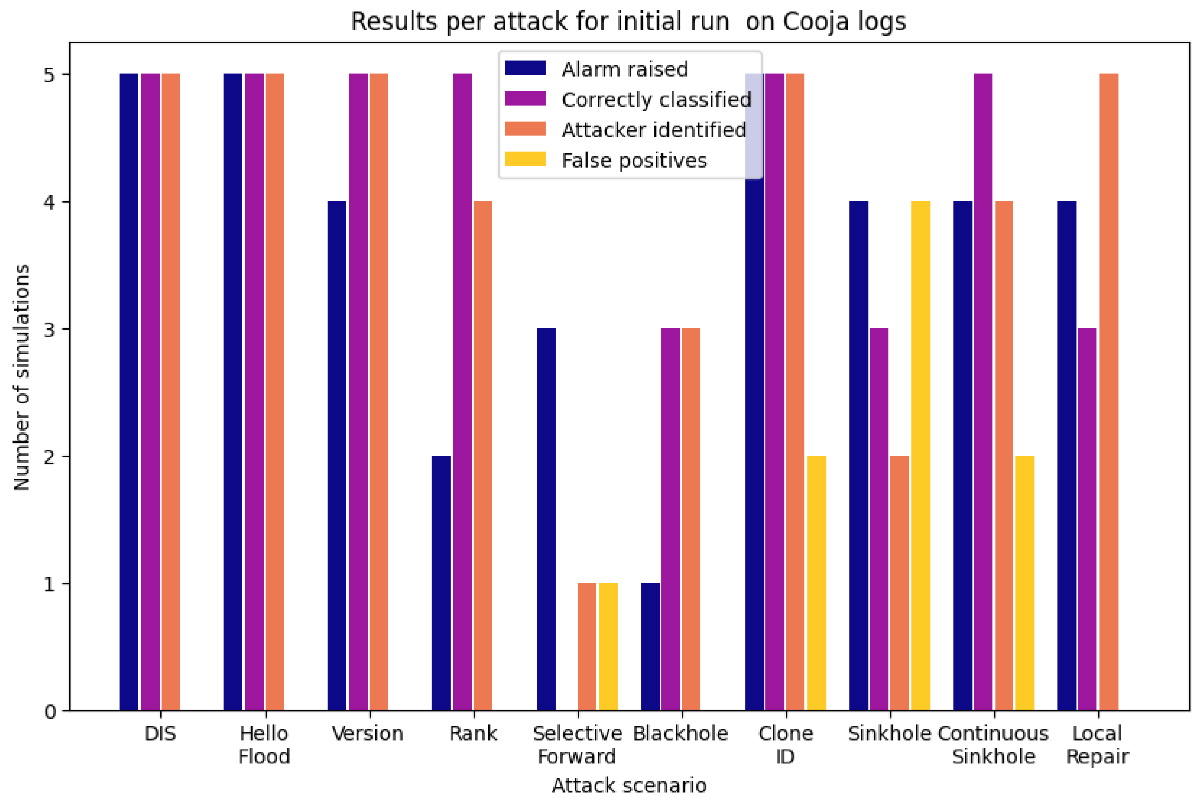


Figure 5.2

False Positives

The total number of false positives over the 5 simulations are 0.

Measure	Percentage
False positives present in simulation	0%
False positives per time step	0%
False positives per time step and device	0%

Table 5.1

Analysis

Corresponding results for running DETONAR on RADAR were on average 86% correctly classified attack and 88% correct attacker identification, while my average results are 78% correctly classified and 78% correct attacker identification. DETONAR on RADAR gave false positives in 60% of the simulations, 0.63% false positives per time step and 0.04% false positives per time step and device, while my false positives are 0% over these 5 simulations. Apart from the false positives, our initial results are not as good as the result on RADAR, but there are many parts of the experiment setup which can be investigated further to see how the results of DETONAR are affected. From these results, we can see that DETONAR at least works on the Cooja logs, but we need a bigger set of simulations to be able to draw any conclusion from the result.

These initial runs included two attacks which are no longer included in the upcoming experiments, Clone-ID and Rank attack. They were later excluded due to time limitations.

5.1.2 Networks with Random Topologies

For this experiment, I am running DETONAR on random topologies. This is interesting to gain insight into whether the results depend on the topologies, and also since the ”initial topology” is not representative for all networks. Also, considering only five runs does not give us sufficient insight into what may affect the results of DETONAR. Therefore, we need a way of generating many simulations while allowing them to have variation in topology. This is done with Multi-Trace [13], since it would be infeasible to create any reasonable amount of topologies by hand. For these experiments, all attackers were chosen randomly from the full set of sensor nodes.

Experiment Setup

- 100 simulations per scenario
- Simulation time: 27000 seconds
- Attacks start at 21000-23000 seconds
- Random topologies
- Random choice of attacker node

Attack Classification and Attacker Identification

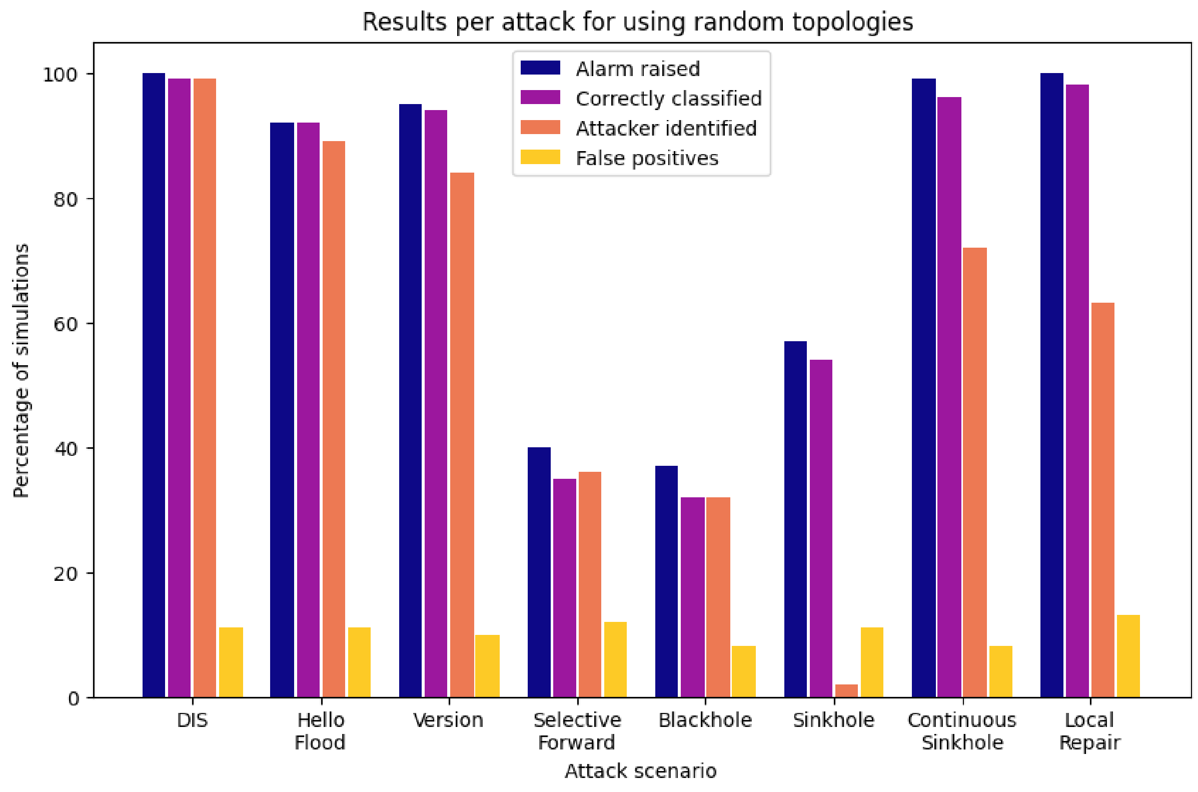


Figure 5.4

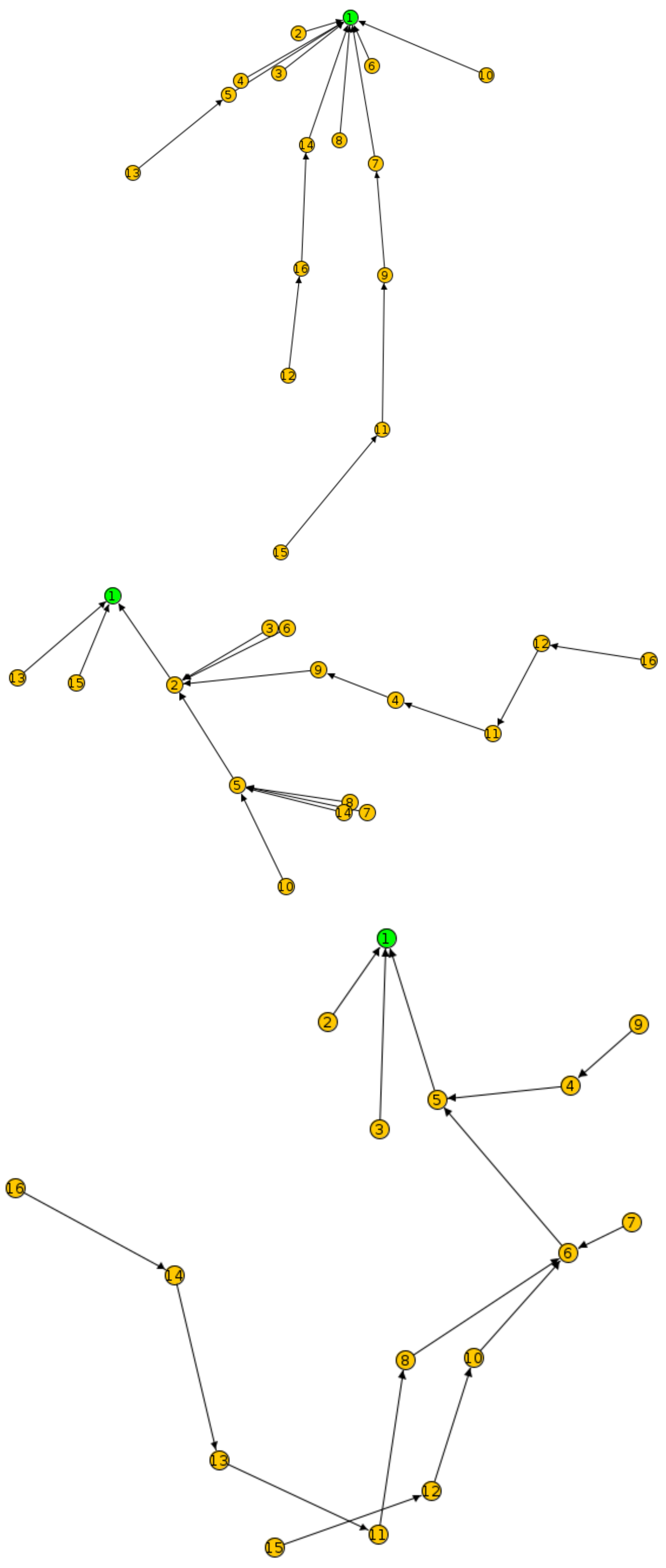


Figure 5.3: An illustration of 3 of the random topologies used in the second experiment.

False positives

The total number of false positives over the 100 simulations are 21. The training period is 30 time steps of 600 seconds, which corresponds to 18000 seconds. This gives us 9000 seconds of possibly detecting attacks, corresponding to 15 time steps.

Measure	Percentage
False positives present in simulation	19%
False positives per time step	1.27%
False positives per time step and device	0.08%

Table 5.2

Comparison to Initial Run

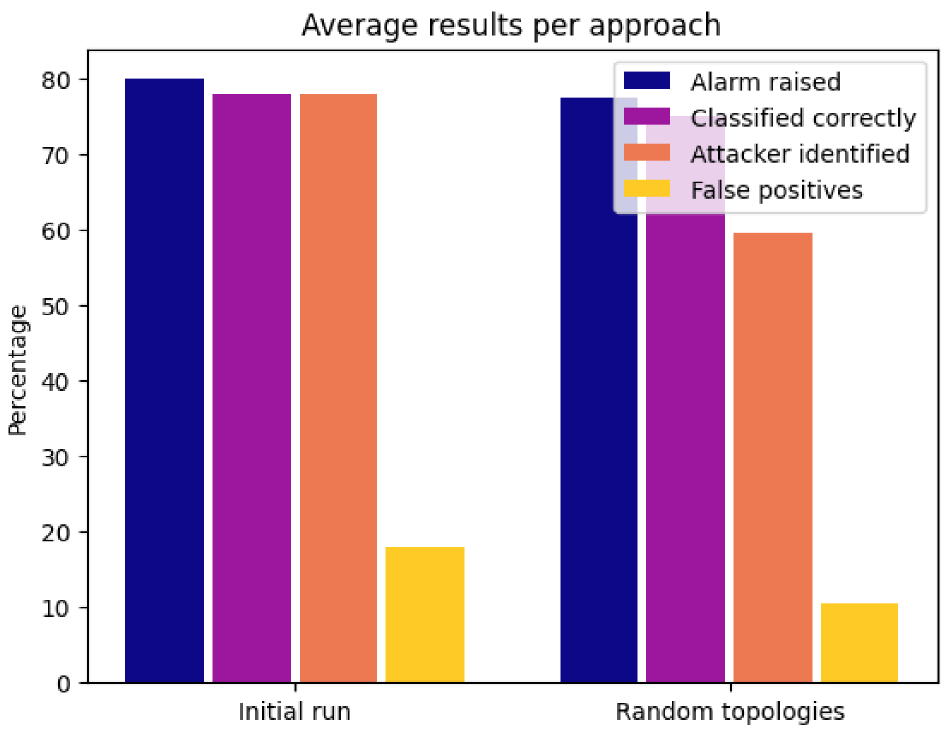


Figure 5.5

Analysis

All measures are lower on average for the randomly generated topologies than what was indicated by the run on the "initial topology". We can also see that there is a big difference between the different scenarios; the attack classification and attacker identification of DIS, HELLO Flooding and Version attacks are very high, while the same values for Blackhole, Selective Forward and Sinkhole attacks are very low, with around 30% for the former ones, and for Sinkhole 54% and as 2% for classification and attacker identification respectively. There are several reasons which may contribute to this, which I will go into in the following paragraphs.

Certain attacks effects are highly dependant on the placement of the attacker within the network. Regarding the scenarios Blackhole and Selective Forward, it is quite obvious that if the attacker is chosen to be a leaf node, the attack will have no effect, since it won't have any children from which it can drop application packets.

Dense and sparse networks may produce quite different results. Looking at some examples of topologies out of the randomly generated ones, we see great variation in the spread of the networks, see Figure 5.3. If we look at the first of those randomly generated topologies, we can see that 9 out of 15 nodes are directly connected to the border router, and therefore have rank 256. For the Sinkhole attack, the attacker being any of those 9 nodes would result in an undetectable attack, since the fake rank being advertised is 256. Another case of undetectable Sinkhole attacks are the cases where the attacker is in a neighborhood where there is already a better, or equally good, parent present, such that the fake rank would not attract any new children.

Similarly, for all attacks where the rank value is altered; Sinkhole, Local Repair and Continuous Sinkhole, it is likely that the results will be worse in cases where there are very few neighbors who can actually choose the attacker node as a new parent or switch parent from the attacker node to another node. However, the above results for Continuous Sinkhole and Local Repair are relatively good, at least when it comes to attack classification, even with the random topologies.

One important difference between RADAR and the Cooja logs which affects the result, is the time of the logging. In RADAR, the packets are sniffed and thus the time recorded in the dataset is maintained by the sniffer devices, which are supposed to have synchronized time between themselves. This is not mentioned explicitly, but is an assumption made by me since the timestamps in the logs are used in a way which indicates synchronized time. One strong indicator of this is the timestamps of rank changes and version changes, which are used for identifying the attacker. Whichever node changed its rank first, is identified as the attacker node in certain attacks. This would not be accurate unless the time is synchronized. However, in Cooja every node has its own time and we use the node logs with un-synchronized times as input. This will definitely affect the attacker identification in all scenarios concerned with rank or version: Sinkhole, Local Repair, Continuous Sinkhole and Version attack, but it is possible that the different times affect other scenarios as well. Since the system relies on analyzing events in different time windows, different devices' time window may not correspond to the same global time and therefore not mirror the events in the network accurately. This is probably also contributing to worse performance than running DETONAR on RADAR.

To summarize the conclusions of the first experiment, the results are differing quite a lot from the original results of running DETONAR on RADAR. This is likely due to several different reasons; the random topologies generated, as well as the placement of the attacker node and the fact that the time of devices within the network is not synchronized. Different scenarios are likely more or less affected by these factors; for example the Blackhole and Selective Forward scenarios are likely affected the most by the placement of attacker nodes.

5.1.3 Networks with Minimum Distance

Looking at the random topologies used in the previous experiments, there is a huge difference in spread and shape of the networks. Clusters of nodes are formed in certain topologies. In a real world scenario, these are likely less interesting topologies given that the sensor nodes are supposed to sense their surroundings, and the clusters give no additional insight since the same area is covered by all nodes in the cluster. It is also likely that clustering affects the results, as discussed above. For these reasons, I tried running DETONAR while enforcing a minimum distance of 40 meters between nodes to guarantee some spread in the networks. The transmission range is 50 meters, so the minimum distance ensures that all nodes have at least one node at 40-50 meters distance, and no nodes closer than that. Three examples of these topologies are shown in Figure 5.8.

Experiment Setup

- 100 simulations per scenario
- Simulation time: 27000s
- Attacks start at 21000-23000 seconds
- Random topologies
- Random choice of attacker node
- Minimum distance between nodes is 40 meter

Attack Classification and Attacker Identification

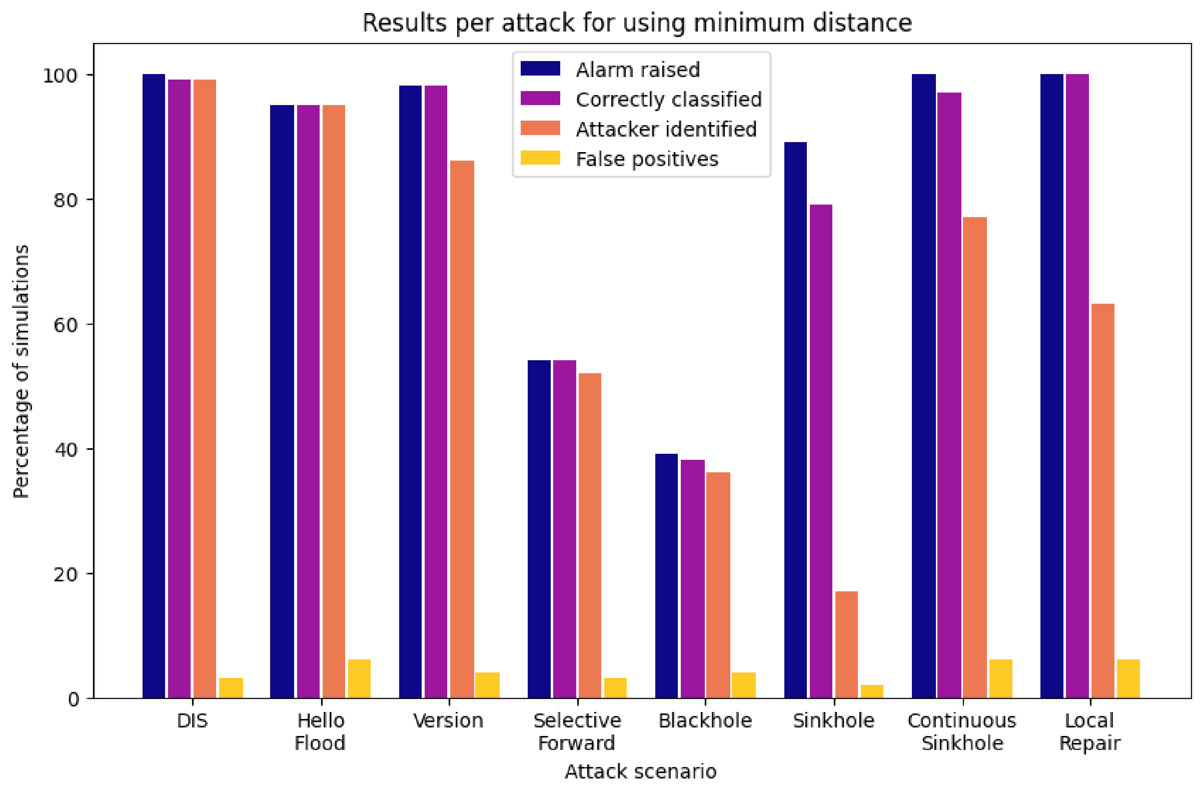


Figure 5.6

False Positives

The total number of false positives over the 100 simulations are 16. The training period is 30 time steps of 600 seconds, which corresponds to 18000 seconds. This gives us 9000 seconds of possibly detecting attacks, corresponding to 15 time steps.

Measure	Percentage
False positives present in simulation	12%
False positives per time step	1.07%
False positives per time step and device	0.07%

Table 5.3

Comparison to Random Topologies

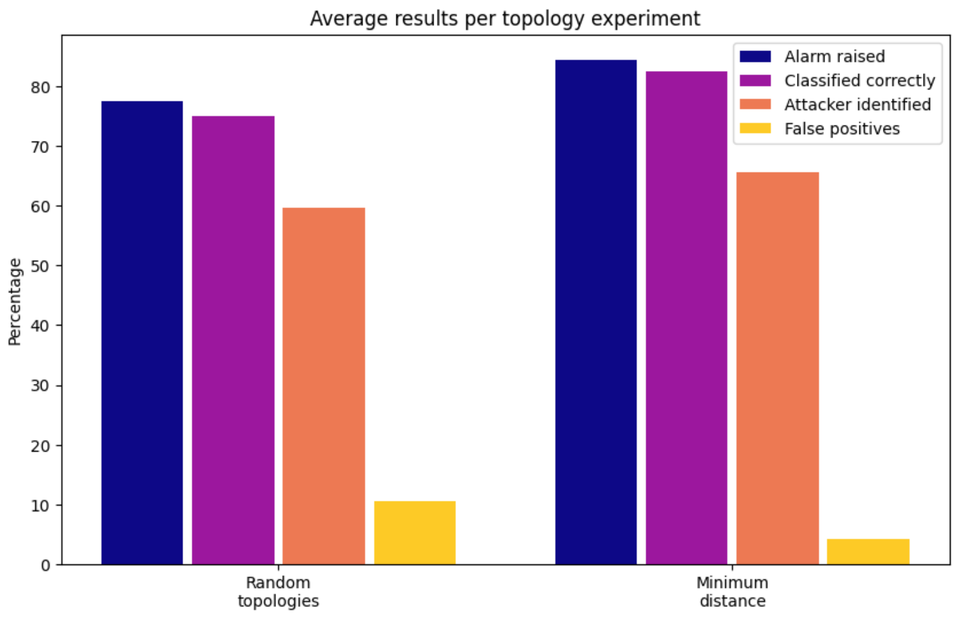


Figure 5.7

Analysis

From the above results, it seems that enforcing a minimum distance quite close to the transmitting range of the nodes gives improved results in all measures taken into account. For individual scenarios, we still have bad performance for detecting Blackhole and Selective Forward, while we see some improvement for Sinkhole.

It seems that the improvement in raising any alarm at all, improved from an average of 77.5% to an average of 85.8% might be the most significant difference, since the attack classification and attacker identification is completely relying on the fact that an alarm is raised. The reduction of false positives is also quite significant, going from an average of 10.5% over all scenarios to an average of 3.9%.

5.1.4 Selected Attackers

Since several of the attacks heavily rely on the placement of the attacker node in order to be effective, I wanted to see whether it would affect the results to add some mechanism to the choice of attacker instead of choosing the attacker node randomly. I first investigated the results by looking at a few Blackhole-simulations that were wrongly classified, and chose another node as an attacker. From the 4 simulations I looked at, two of them were correctly classified when choosing a different attacker. This indicates that the bad results of detecting Blackhole

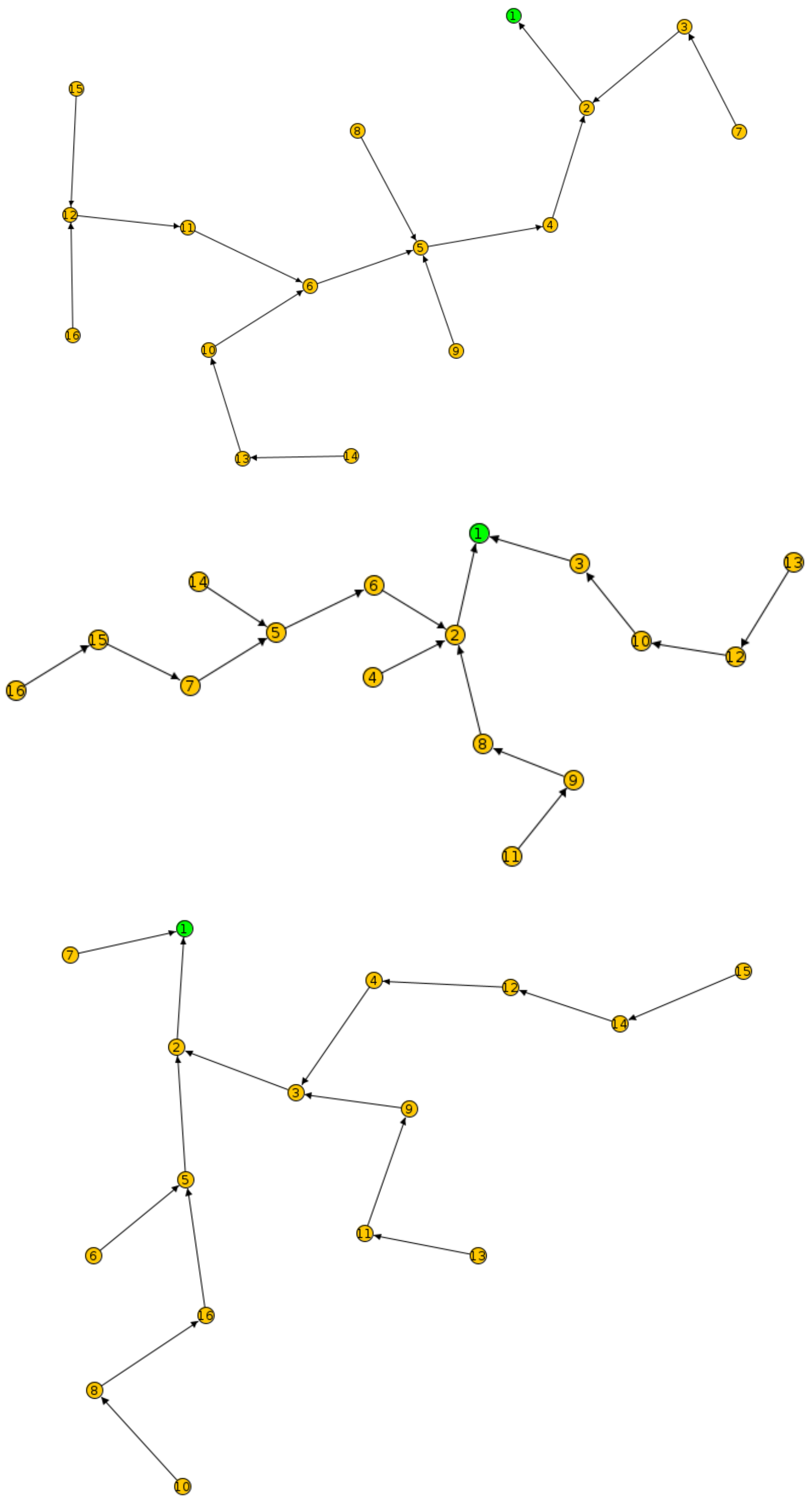


Figure 5.8: An illustration of 3 of the random topologies with a minimum distance of 40 meters used in the third experiment.

and Selective Forward attacks might be improved by using another way of picking the attacker. To be able to run experiments without choosing attacker by hand, I added some criteria for the attacker node for the following attacks:

1. Blackhole and Selective Forward: There are more than 1 node within the attacker node's transmitting range and at least one of those nodes are further away from the sink than the attacker node.
2. Sinkhole, Continuous Sinkhole, Local Repair: There are more than 2 nodes within the attacker node's transmitting range and at least one of those are further away from the border router than the attacker node. Also, the border router must not be within range of the attacker.

These criteria were added to the Cooja simulation scripts. I also needed to reduce the time for running these experiments due to time limitations, so the simulation time was reduced from 27000s to 24000s. The attack was scheduled to start around 21000s into the simulation. This reflects in the lower false positive rates, since there is less time for the system to raise alarms before the attack takes place. I did not run a legitimate scenario for separate false positive results for this experiment, since I am interested in the differences in attack classification and attacker identification for the affected attack scenarios.

Experiment Setup

- 100 simulations per scenario
- Simulation time: 24000s
- Attacks start around 21000 seconds
- Random topologies
- Minimum distance between nodes is 40 meters
- Attackers for certain attacks are chosen based on criteria presented above

Attack Classification and Attacker Identification

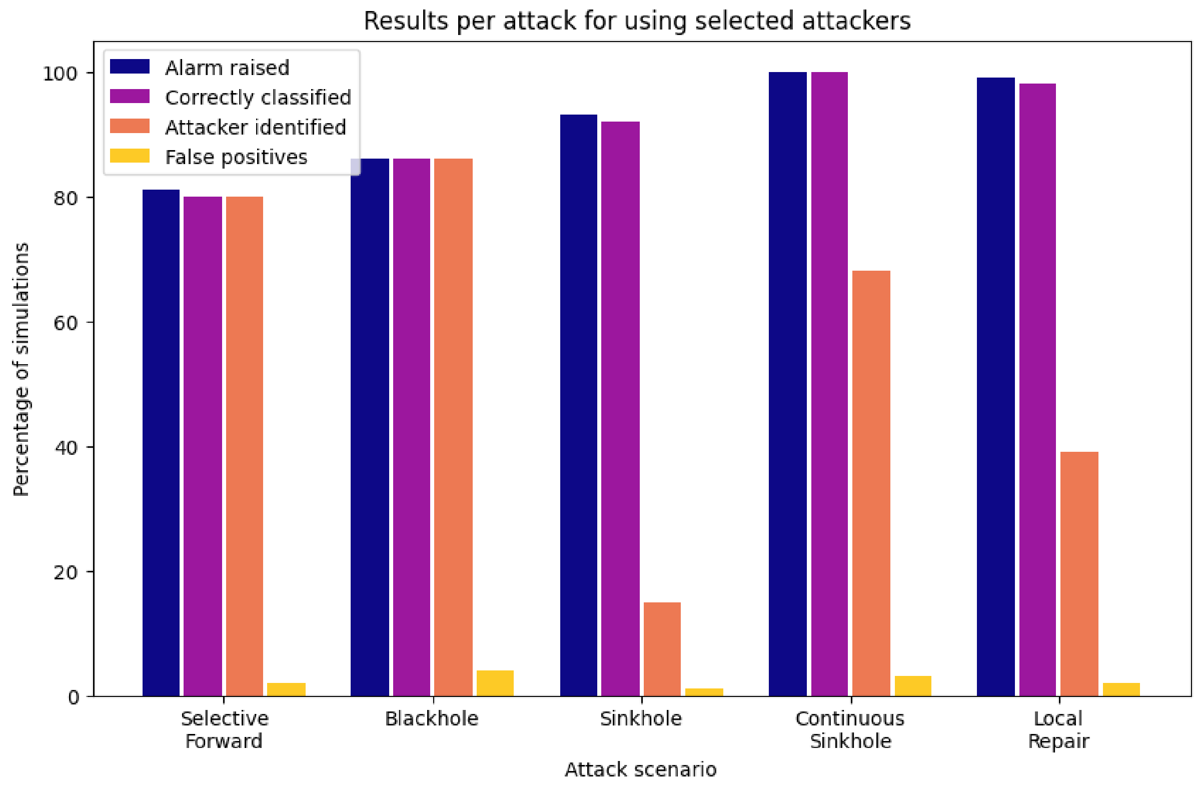


Figure 5.9

Results before using Selected Attackers

These below Figure are the same results as in Figure 5.6, but presented for only the relevant scenarios for better readability and comparison.

- 100 simulations per scenario
- Simulation time: 24000s
- Attacks start around 21000 seconds
- Random topologies
- Minimum distance between nodes is 40 meters
- Random choice of attacker node

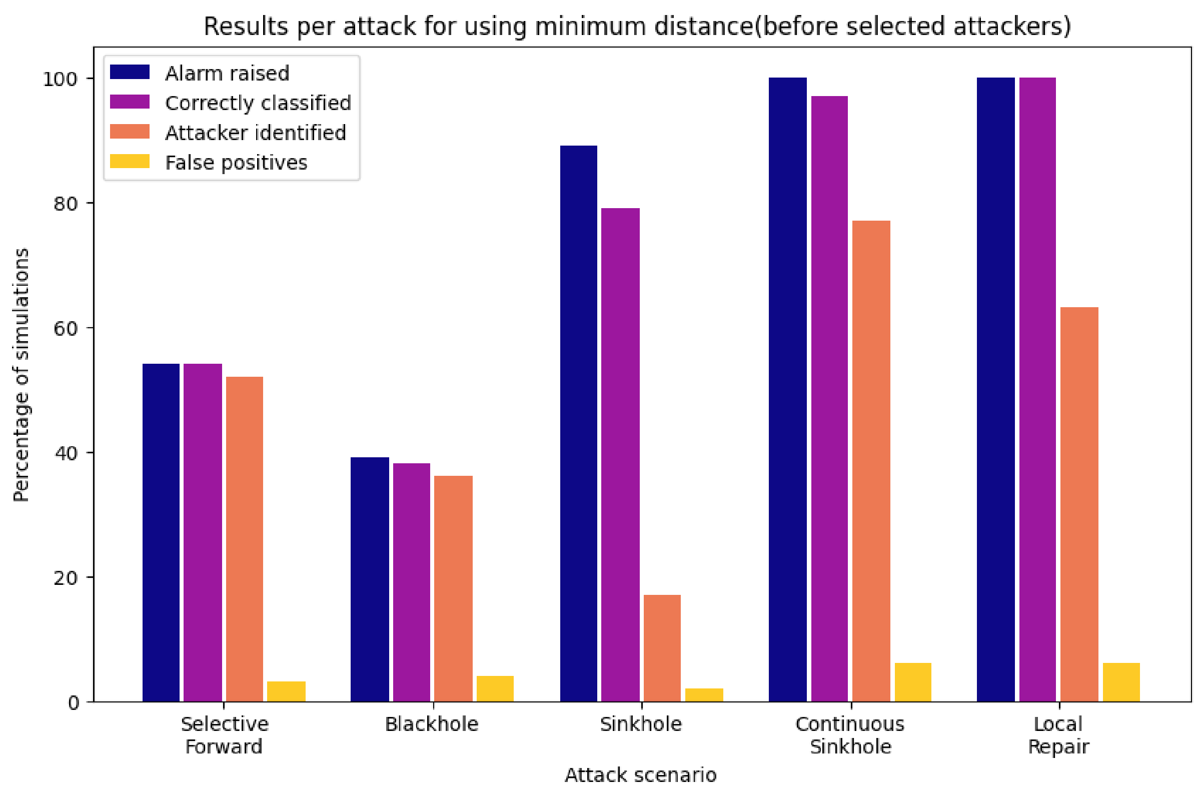


Figure 5.10

Comparison to Minimum Distance

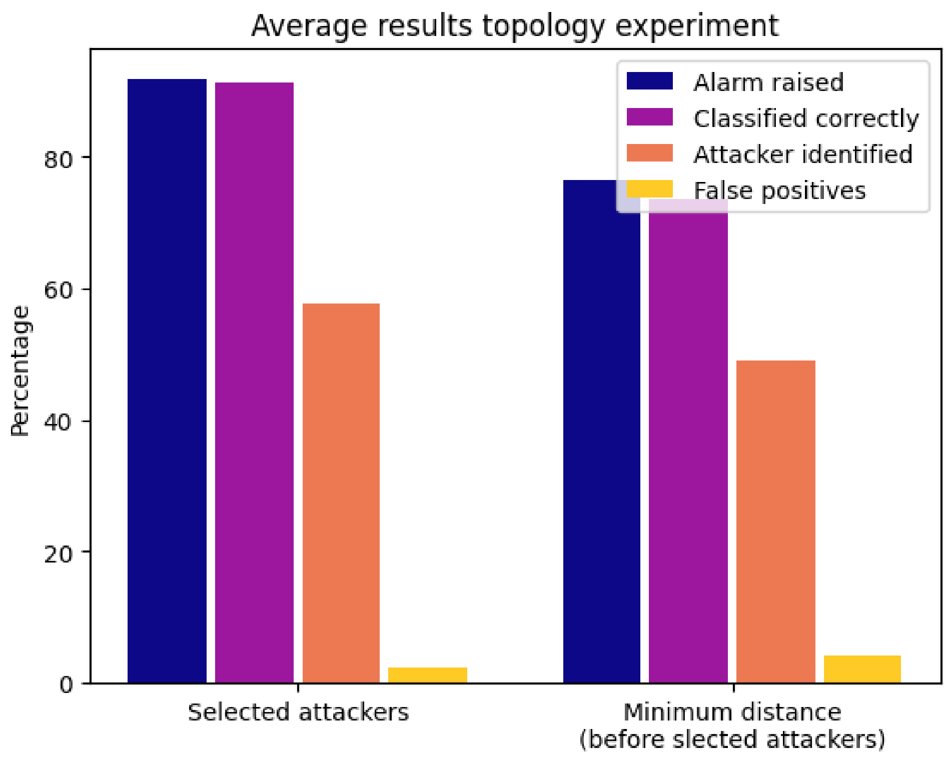


Figure 5.11: This result is corresponding to the average of the five attack scenarios: Blackhole, Selective Forward, Sinkhole, Continuous Sinkhole and Local Repair

Analysis

It is worth to mention, that even if these criteria likely would give a better choice of attacker, in terms of the effect they have on the network and therefore the relevance for detecting it, there is nothing that guarantees that these criteria will work the way I want them to during attacker selection. Since I am using 100 simulations for each scenario, it is infeasible to go through them all and look at whether the criteria actually gave the intended results.

We can see some changes in the results; mainly for Selective forward and Blackhole, where the attack classification is increased from 54% and 29% to 81% and 86% respectively. The attacker identification follows the same pattern. There also seems to be an improvement in attack classification for the Sinkhole-scenario. However, the attacker identification is lower for all attacks where a change in rank takes place. It's difficult to know for certain that a change in results is because of the introduced criteria. I am assuming that these changes in attacker identification is random and not necessarily connected to the introduced criteria, since there is no pattern in the result changes.

We can draw the conclusion that the results improved overall from using this approach, mainly for the Blackhole and Selective Forward attacks.

5.1.5 Conclusion of Topology Exploration

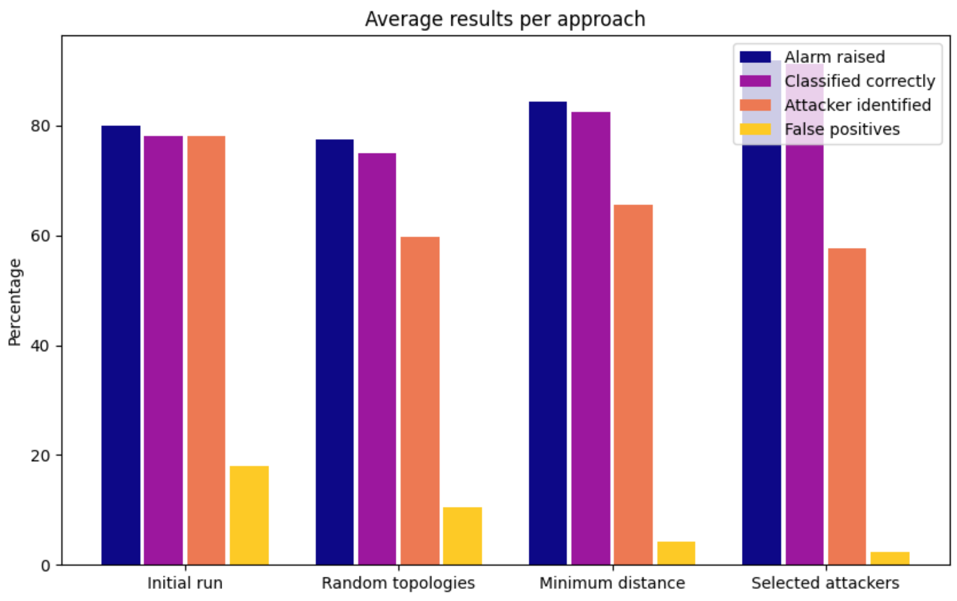


Figure 5.12

A conclusion to draw from these experiments is that adding the minimum distance in order to get less clustering in the networks, and selecting the attackers in certain attack scenarios improves the number of alarms raised and the number of attacks classified. The false positives also seem to decrease. However, the attacker identification seems to not be affected by these changes; although there are small differences between the different experiments, there is no consistent trend.

5.2 Running DETONAR on a Border Router

As described in Section 4.2, adjustments were made to DETONAR to be able to run it on data from a border router instead of the full network traffic. For the following experiments, I wanted to test the performance of the border router-approach in relation to the sniffer-approach, without the result depending on differences in the random topologies being generated. Therefore, for all the following experiments I produced logs that could be used in both the sniffer approach and the border router approach for the same dataset. DETONAR was then used once for each approach.

In real world applications, the radio medium is likely not perfect, which means there will be some loss of packets. As mentioned by Lohier et al. [19], network communication depends on many factors, like distance, which MAC-layer is used and the environment of the network. To accommodate for the fact that it is difficult to know what level of loss is common in real life scenarios, I evaluated how DETONAR would work in networks with several different success rates. By success rate, I refer to the rates of which packets are successfully transmitted from a node and received by a node, both of which are configurable in Cooja. The total loss of packets due to the success rate will depend on both. Let's say we have 90% transmission success rate and 90% reception success rate in a network. Out of the 90% which are successfully transmitted, 90% will be received successfully. In total, that is $90*90 = 81\%$ success rate overall.

It is worth mentioning that even if packets are lost due to the imperfect radio medium, unicast packets, like application packets and DAO packets, are re-transmitted by the nodes up to seven times due to default MAC-layer configurations of Contiki-NG [20]. However, packets which are broadcast, like DIS and DIO, are not re-transmitted. The transmission of those packets are determined by the RPL trickle timers. The loss of packets introduced in these experiments will therefore mainly effect DIO and DIS packets.

5.2.1 Networks with 100% Success Rate

In this initial comparison, I wanted to see the difference between the border router approach and the sniffer approach while using the insight of which factors affect the result of DETONAR from previous experiments. The network success rate for this experiment is 100%, which means there is no loss of packets due to the radio medium.

Experiment Setup

- 100 simulations per scenario
- Simulation time: 24000s
- Attacks start around 21000 seconds
- Random topologies
- Minimum distance between nodes is 40 meters
- Attackers for certain attacks are chosen based on criteria presented above in Section 5.1.4
- 100% network success rate

Attack Classification and Attacker Identification - Sniffer Approach

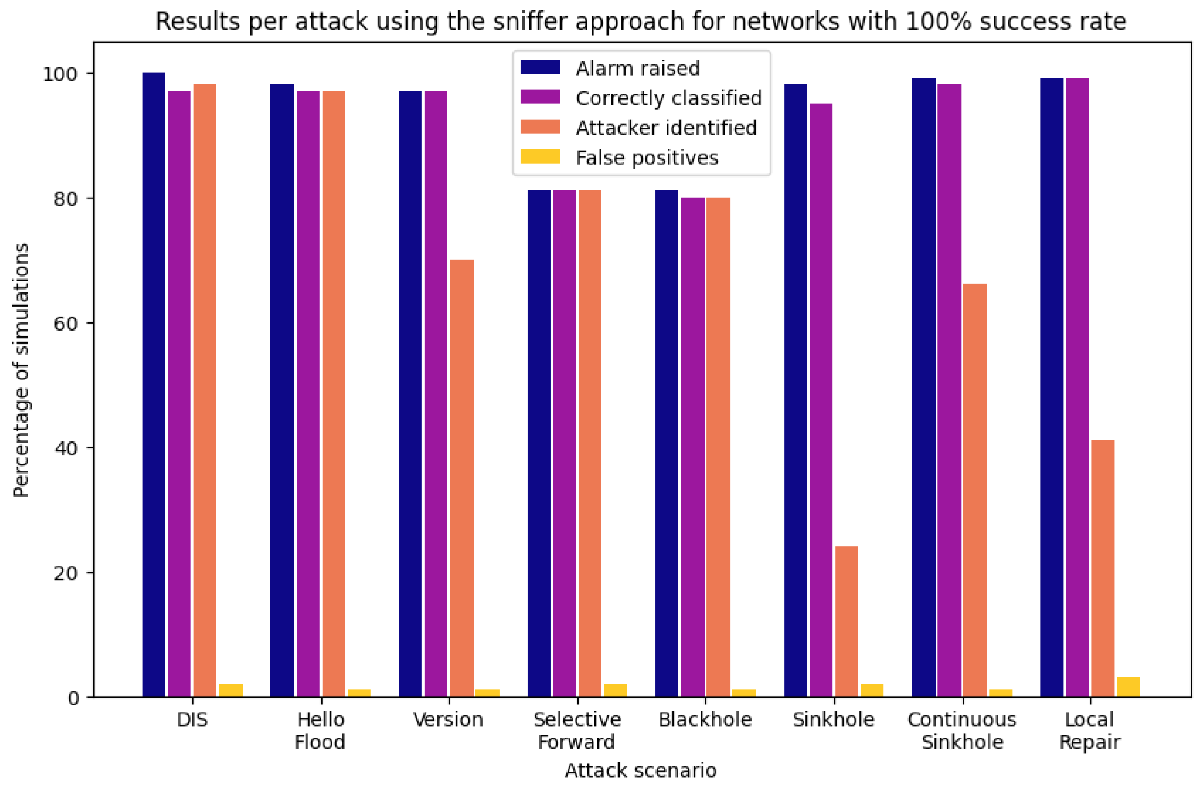


Figure 5.13

False Positives - Sniffer Approach

The total number of false positives over the 100 simulations are 8. The training period is 30 time steps of 600 seconds, which corresponds to 18000 seconds. This gives us 6000 seconds of possibly detecting attacks, corresponding to 10 time steps.

Measure	Percentage
False positives present in simulation	6%
False positives per time step	0.8%
False positives per time step and device	0.05%

Attack Classification and Attacker Identification - Border Router Approach

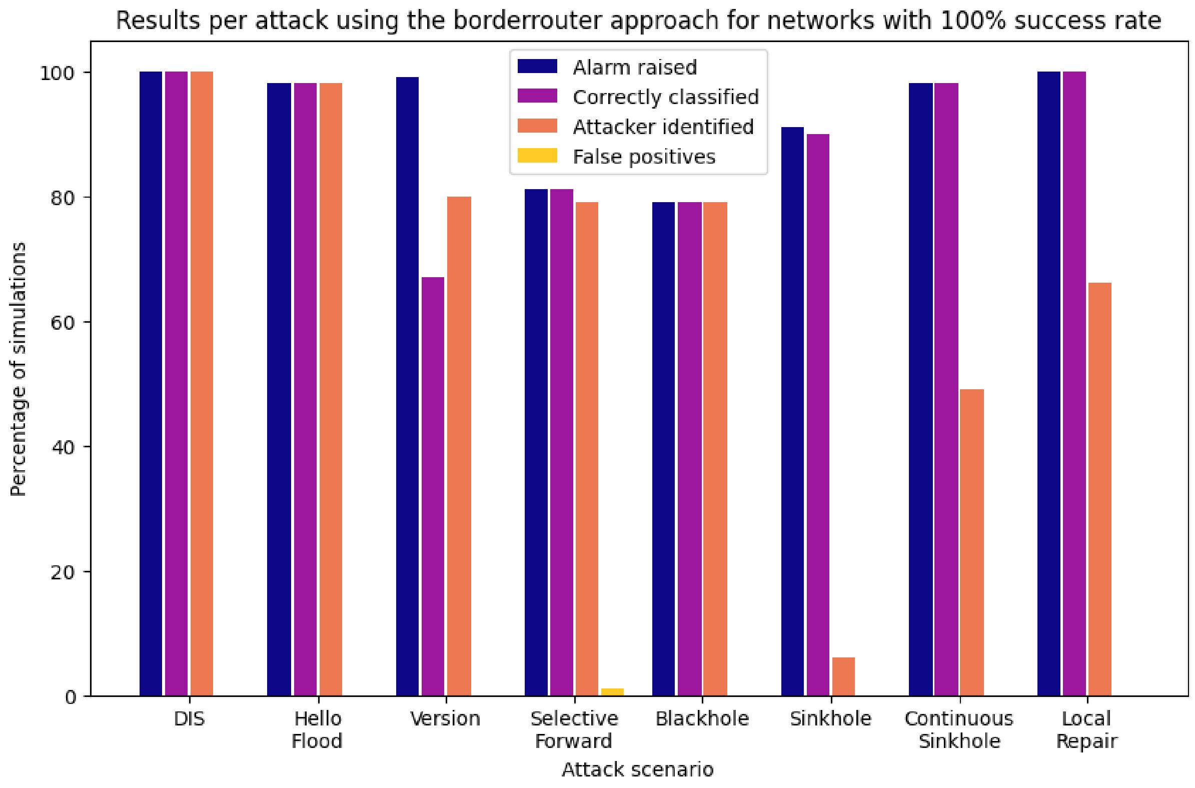


Figure 5.14

False Positives - Border Router Approach

The total number of false positives over the 100 simulations are 0.

Measure	Percentage
False positives present in simulation	0%
False positives per time step	0%
False positives per time step and device	0%

Table 5.4

Comparison between approaches

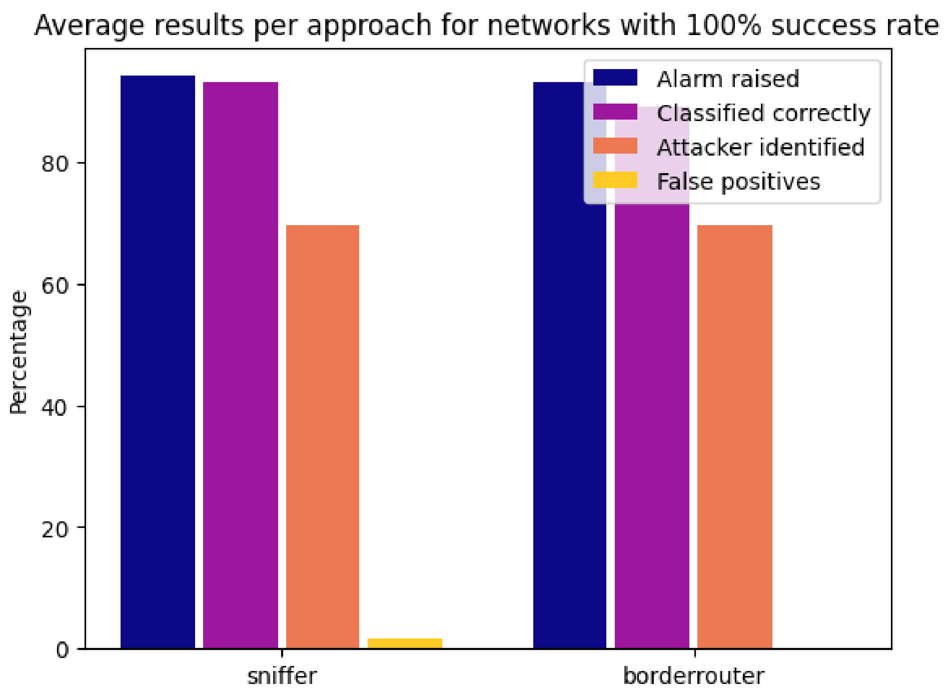


Figure 5.15

Analysis

I will compare the acquired results to the ones of DETONAR on RADAR. For this comparison, it is worth mentioning that the 8 attacks evaluated in this thesis had an attack classification of 80% and attacker identification of 87.5% for DETONAR on RADAR. If we instead consider all attacks, even the ones not implemented in this thesis, the attack classification for DETONAR on RADAR is 86% and the attacker identification is 88%. Using the sniffer approach on Cooja logs, the attack classification is on average 93% and the attacker identification 69.63%. The border router approach on Cooja logs gives comparable results of 89.13% for attack classification and 69.63% for attacker identification.

Comparing the two approaches to each other, we see that Blackhole and Selective Forward attacks have similar results in both classification and attacker identification for both approaches, around 80%. Considering the fact that some of the attacks not classified correctly are likely undetectable due to the attacker being a leaf node, I consider that a pretty good result.

The attacks standing out the most are all attacks concerned with the change of rank and version. As described earlier, these attacks depend on synchronized time to be correctly classified and to have the attacker correctly identified. Since the percentage of raising alarm is so high, we can at least draw the conclusion that these attacks are not simply left unnoticed by DETONAR. Looking at some wrongly classified attacks of for example Version, having the highest number of wrongly classified attacks for the border router approach, there are mainly false alarms of DIS and HELLO Flood attacks. This makes sense, since the change in Version makes the network want to rebuild itself, and the nodes will start transmitting control packets in order to do so. If any of the nodes who notice a change in Version from a neighbor was started earlier than the neighbor, that attack might be classified as something else than a Version attack, especially if the node with "earlier time" it is not considered an anomalous node by DETONAR.

The attacker identification suffers similarly for Sinkhole, Continuous Sinkhole and Local Repair.

The above results show that the two approaches have small differences in results. The sniffer approach seems to perform slightly better in terms of attack classification, while the border router approach seems to have a lower number of false positives.

5.2.2 False Positives

This experiment was deidcated entirely to investigate the false positives of the different approaches. To be able to compare against DETONAR's original false positives when run on RADAR, I wanted to measure false positives in the same way. In RADAR, the simulations are 1500 seconds, and each time window is 10 seconds. That gives us 150 time steps for running DETONAR on the entire simulation. 30 time steps are used as training, which gives us 120 time steps for the actual evaluation. In my case, the time window is 600 seconds long, so I ran DETONAR on simulations of 90000 seconds, since that gives the same number of timesteps for intrusion detection, 120. I use the same measures of false positives as described in the beginning of this chapter.

Experiment Setup

- 100 simulations
- Simulation time: 90000s
- Random topologies
- Minimum distance between nodes is 40 meters
- No attacks in simulation

False Positives - Sniffer Approach

The total number of false positives over the 100 simulations are 73.

Measure	Percentage
False positives present in simulation	46%
False positives per time step	0.61%
False positives per time step and device	0.04%

Table 5.5

False Positives - Border Router Approach

The total number of false positives over the 100 simulations are 0.

Measure	Percentage
False positives present in simulation	0%
False positives per time step	0%
False positives per time step and device	0%

Table 5.6

False Positives - DETONAR on RADAR

Measure	Percentage
False positives present in simulation	60%
False positives per time step	0.63%
False positives per time step and device	0.04%

Table 5.7: These are the numbers from the original running of DETONAR on RADAR by Agiollo [8], presented in Section 2.5.5.

Analysis

Comparing the false positive rates to each other, it is possible to say that running DETONAR on Cooja logs with the sniffer approach gives results that are quite similar to the results on RADAR.

It seems that running DETONAR using the border router approach outperforms the sniffer approach by far, giving us no false positives at all. It was indicated already in the previous experiment that the false positives for the border router approach would be close to zero, but this experiment adds to the significance since the simulation time is much longer. It allows us to have more confidence in the reduction of false positives using the border router approach. When I look at some of the results of the simulation, I can see that there are still anomalies raised from the analysis of the traffic. However, none of those anomalies result in an attack classification. It is difficult to know why that is the case, but since there are a few changes made to DETONAR in order to run it on the border router logs, one of those changes or the combination of them may have affected the false positives. Exploring the reason for the low false positives could be an interesting thing to look at in the future.

5.2.3 Networks with 90% Success Rate

This experiment was run with the same setup as previous, except that both the transmitting success rate and the receiving success rate was 90%. In total, that is $90 * 90 = 81\%$ success rate overall.

Experiment Setup

- 100 simulations per scenario
- Simulation time: 24000s
- Attacks start around 21000s
- Random topologies
- Minimum distance between nodes is 40 meters
- Attackers for certain attacks are chosen based on criteria presented above in Section 5.1.4
- 90 % transmitting success ratio
- 90 % receiving success ratio

Attack Classification and Attacker Identification - Sniffer Approach

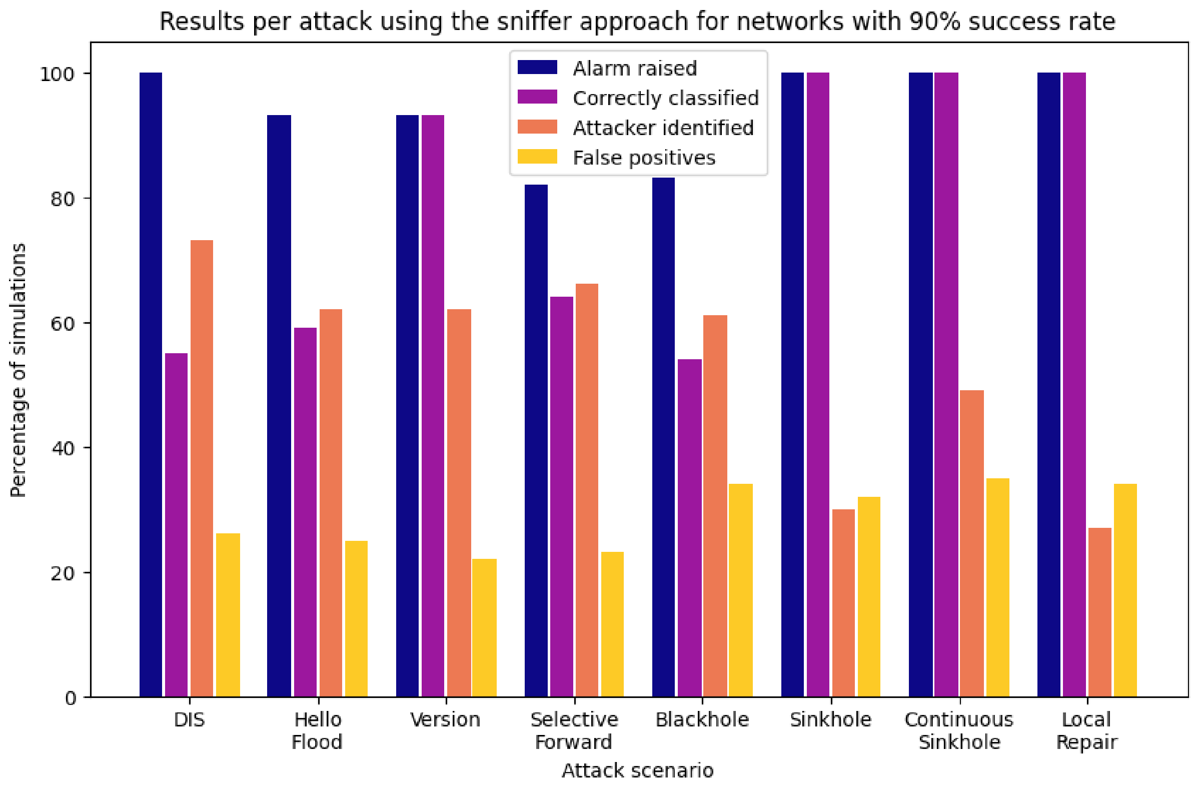


Figure 5.16

False Positives - Sniffer Approach

The total number of false positives over the 100 simulations are 119. The training period is 30 time steps of 600 seconds, which corresponds to 18000 seconds. This gives us 6000 seconds of possibly detecting attacks, corresponding to 10 time steps.

Measure	Percentage
False positives present in simulation	60%
False positives per time step	6%
False positives per time step and device	0.38%

Table 5.8

Attack Classification and Attacker Identification - Border Router Approach

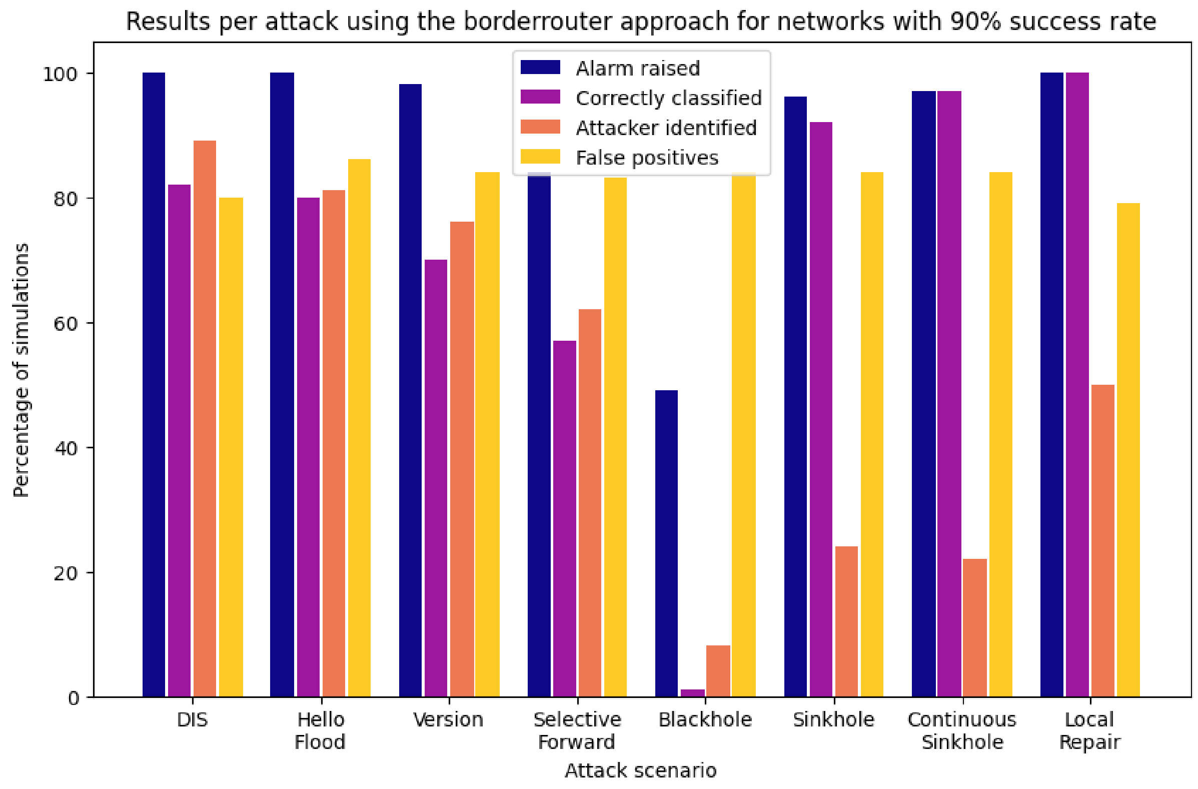


Figure 5.17

False Positives - Border Router Approach

The total number of false positives over the 100 simulations are 208. The training period is 30 time steps of 600 seconds, which corresponds to 18000 seconds. This gives us 6000 seconds of possibly detecting attacks, corresponding to 10 time steps.

Measure	Percentage
False positives present in simulation	82%
False positives per time step	20.8%
False positives per time step and device	1.3%

Table 5.9

Comparison between approaches

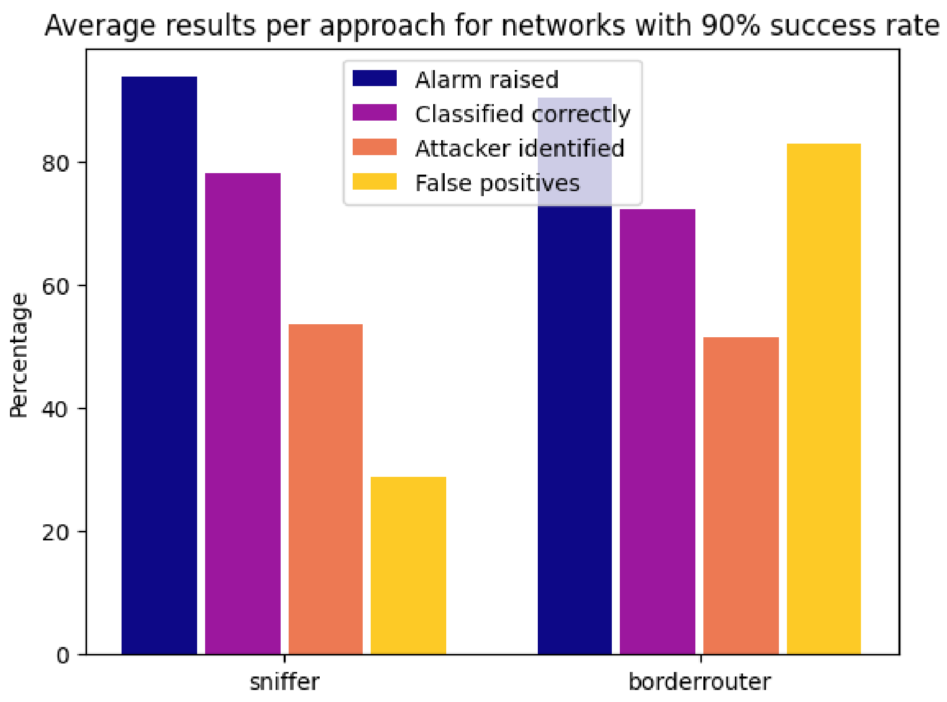


Figure 5.18

Analysis

The results of running DETONAR with only 90% success rate are quite disastrous. The false positives skyrocketed, making it pretty unnecessary to even consider the raised alarms, attack classification and attacker identification of the scenarios tested, since many of those alarms are likely false ones. Looking at the attacks found in the scenarios, it seems that a majority of all false negatives are classified as Rank-attacks, which explains the pretty good performance of Sinkhole, Continuous Sinkhole and Local Repair attack classification.

The fact that DIS and HELLO Flood are still performing well makes sense, since the rates in which the control packets are transmitted in those attacks are still much higher even if only 90% are successfully transmitted and received. For the Version attack, the attacker sends out one control packet with a bumped version once, and if that only malicious packet is dropped, the attack is effectively stopped. Having 90% success rate for both transmission and reception of packets, would give us a 81% success rate of a single packet being both successfully transmitted and successfully received. Therefore, it is understandable that the Version attack classification suffers in the manner shown in the results.

These results could probably be compensated for by adjusting the model used for anomaly detection. There are several configurable parameters which may not be suitable for a lossy network, judging from these results. Another conclusion to draw from these results are that the border router approach seems more sensitive to loss of packets than the sniffer approach, considering the extremely high false positive rates.

5.2.4 Networks with 99% Success Rate

This experiment was conducted using a 99% success rate, for both transmitting packets and receiving packets. In total, that is $99 * 99 = 98.01\%$ success rate overall.

Experiment Setup

- 100 simulations per scenario
- Simulation time: 24000s
- Attacks start around 21000s
- Random topologies
- Minimum distance between nodes is 40 meters
- Attackers for certain attacks are chosen based on criteria presented in Section 5.1.4
- 99 % transmitting success ratio
- 99 % receiving success ratio

Attack Classification and Attacker Identification - Sniffer Approach

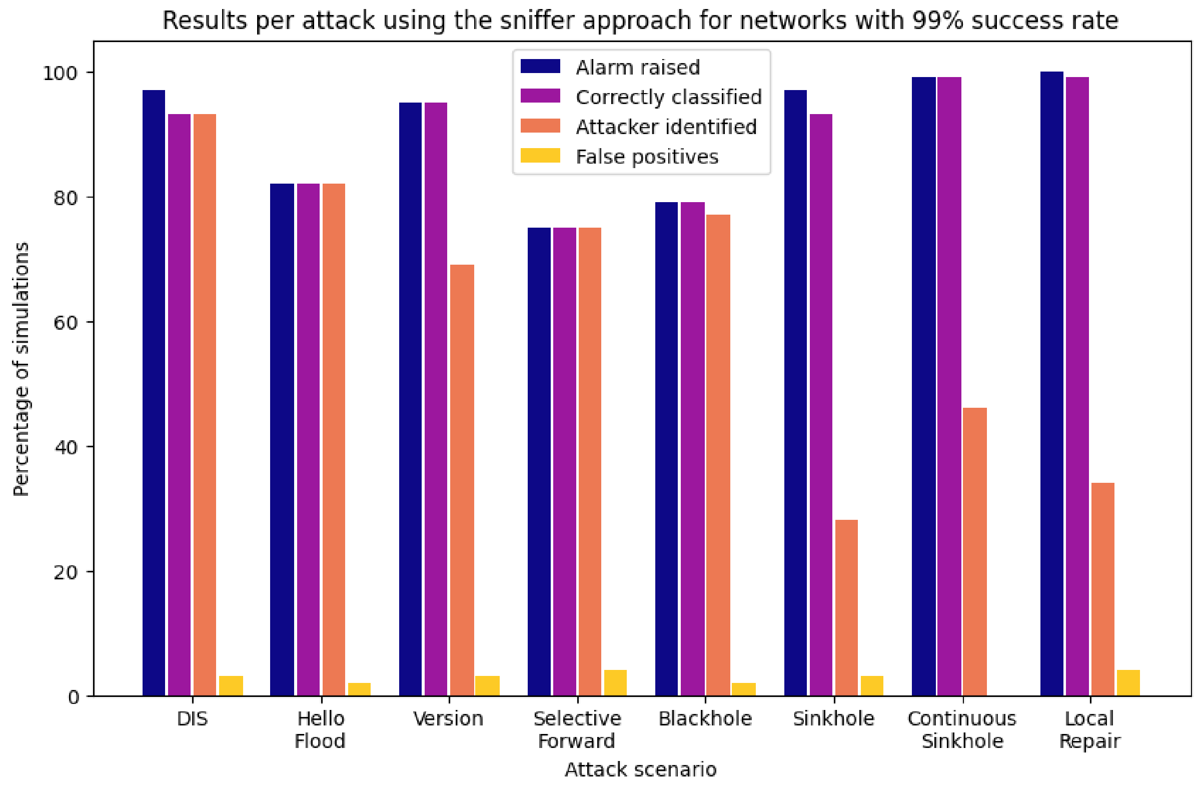


Figure 5.19

False Positives - Sniffer Approach

The total number of false positives over the 100 simulations are 5. The training period is 30 time steps of 600 seconds, which corresponds to 18000 seconds. This gives us 6000 seconds of possibly detecting attacks, corresponding to 10 time steps.

Measure	Percentage
False positives present in simulation	5%
False positives per time step	0.5%
False positives per time step and device	0.03%

Table 5.10

Attack Classification and Attacker Identification - Border Router Approach

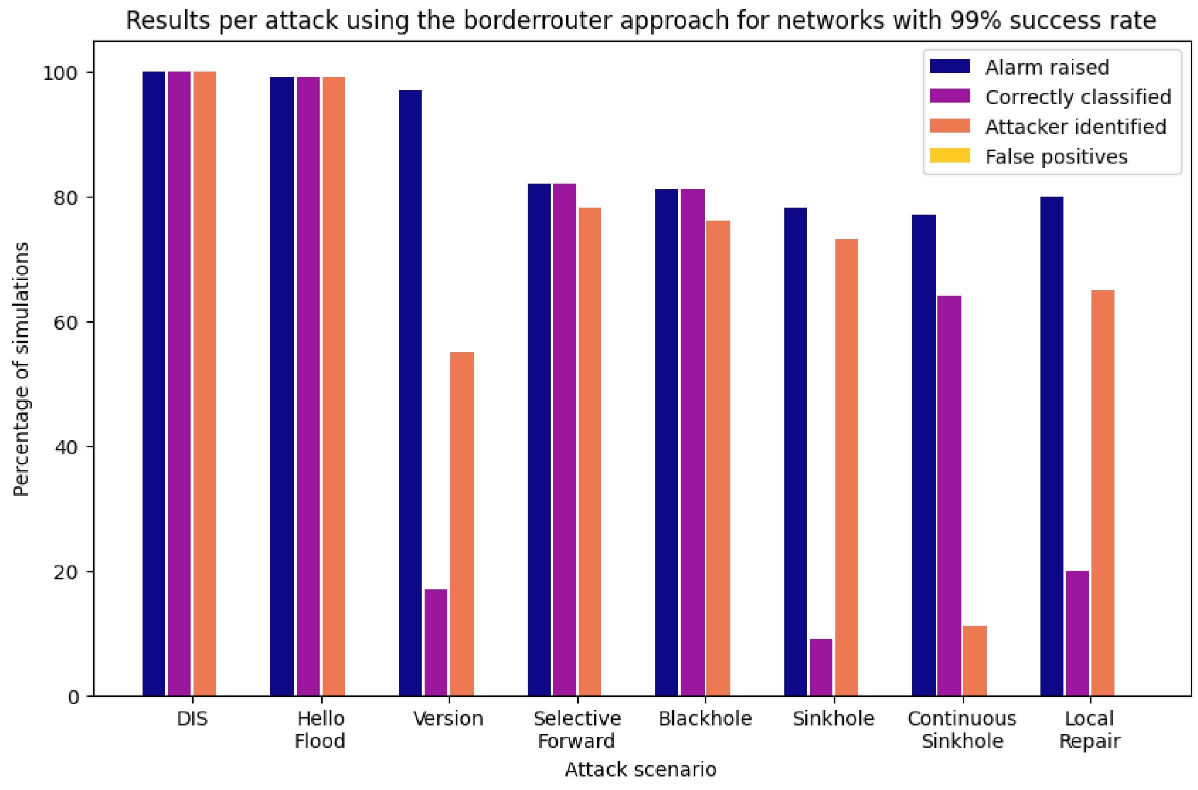


Figure 5.20

False Positives - Border Router Approach

The total number of false positives over the 100 simulations are 0. The training period is 30 time steps of 600 seconds, which corresponds to 18000 seconds. This gives us 6000 seconds of possibly detecting attacks, corresponding to 10 time steps.

Measure	Percentage
False positives present in simulation	0%
False positives per time step	0%
False positives per time step and device	0%

Table 5.11

Comparison between approaches

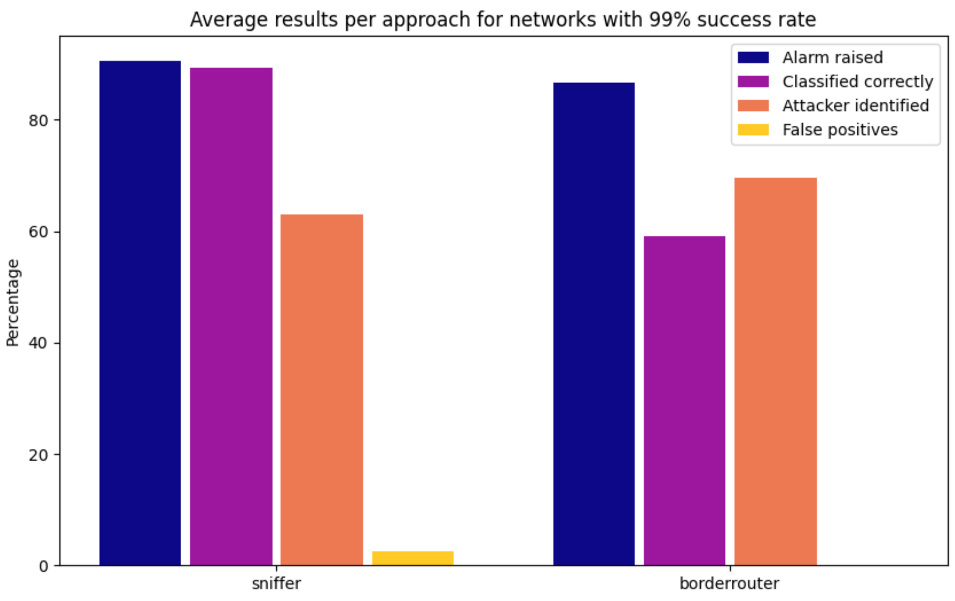


Figure 5.21

Analysis

These results also show that even much less loss in the network seems to affect the result of the border router approach substantially more. For the sniffer approach, the number of alarm raised are slightly reduced which results in somewhat lower attack classification and attacker identification. For the border router approach, the correct classification dropped drastically from the number of alarms raised, from 86.75% alarms raised to only 59% attacks classified on average. The attack scenarios with the lowest classification rate are Version, Sinkhole and Local Repair. Among those simulations being wrongly classified for these attack scenarios, HELLO Flood is especially over-represented. It is not all that surprising that new rank values or version values used in these attack scenarios prompt nodes to send DIO packets in order to potentially adjust the DODAG corresponding to the new ranks or versions. The attacker identification does not seem to be affected that much, comparing to the network with 100% success rate, where the average attacker identification was around 70%.

5.2.5 Networks with 99.5% Success Rate

I decided to make an additional experiment with less loss to get an indication about which level of success rate the attack classification starts suffering in the border router-approach.

Experiment Setup

- 100 simulations per scenario
- Simulation time: 24000s
- Attacks start around 21000s
- Random topologies
- Minimum distance between nodes is 40 meters
- Attackers for certain attacks are chosen based on criteria presented in Section 5.1.4

- 99.5 % transmitting success ratio
- 99.5 % receiving success ratio

Attack Classification and Attacker Identification - Sniffer Approach

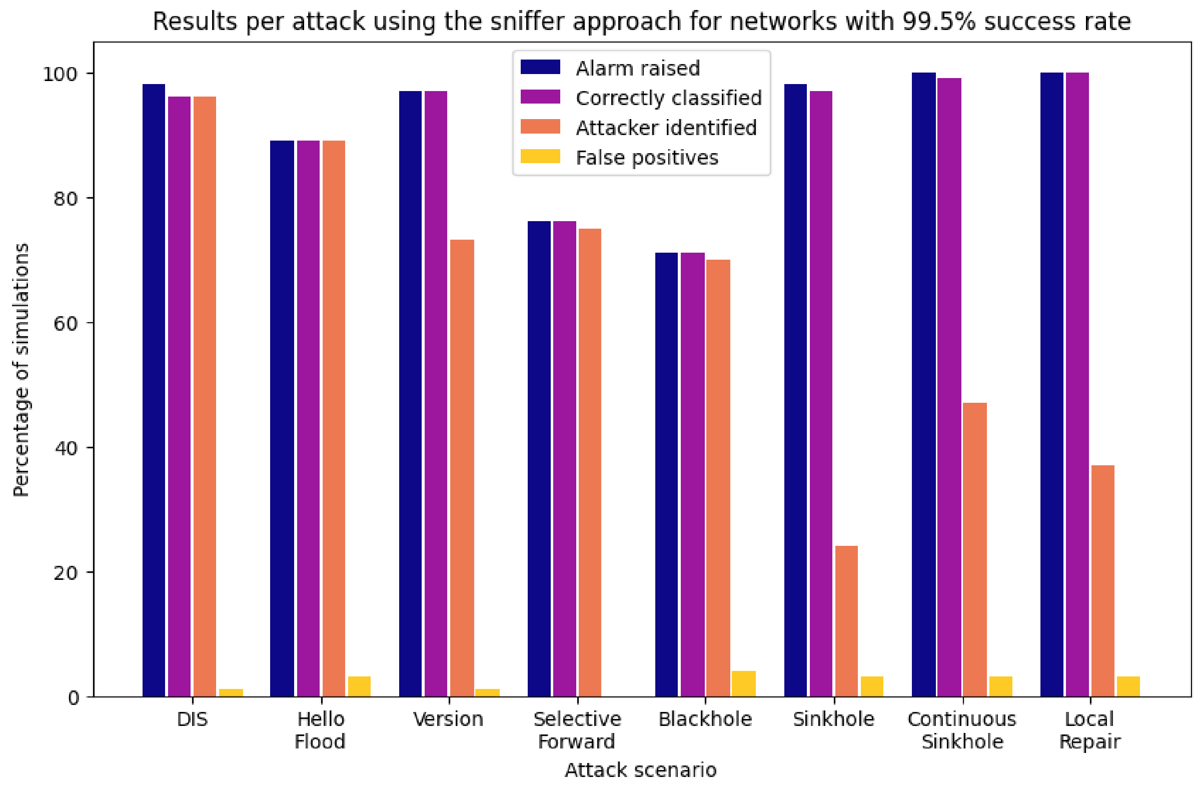


Figure 5.22

False Positives - Sniffer Approach

The total number of false positives over the 100 simulations are 6. The training period is 30 time steps of 600 seconds, which corresponds to 18000 seconds. This gives us 6000 seconds of possibly detecting attacks, corresponding to 10 time steps.

Measure	Percentage
False positives present in simulation	5%
False positives per time step	0.5%
False positives per time step and device	0.03%

Table 5.12

Attack Classification and Attacker Identification - Border Router Approach

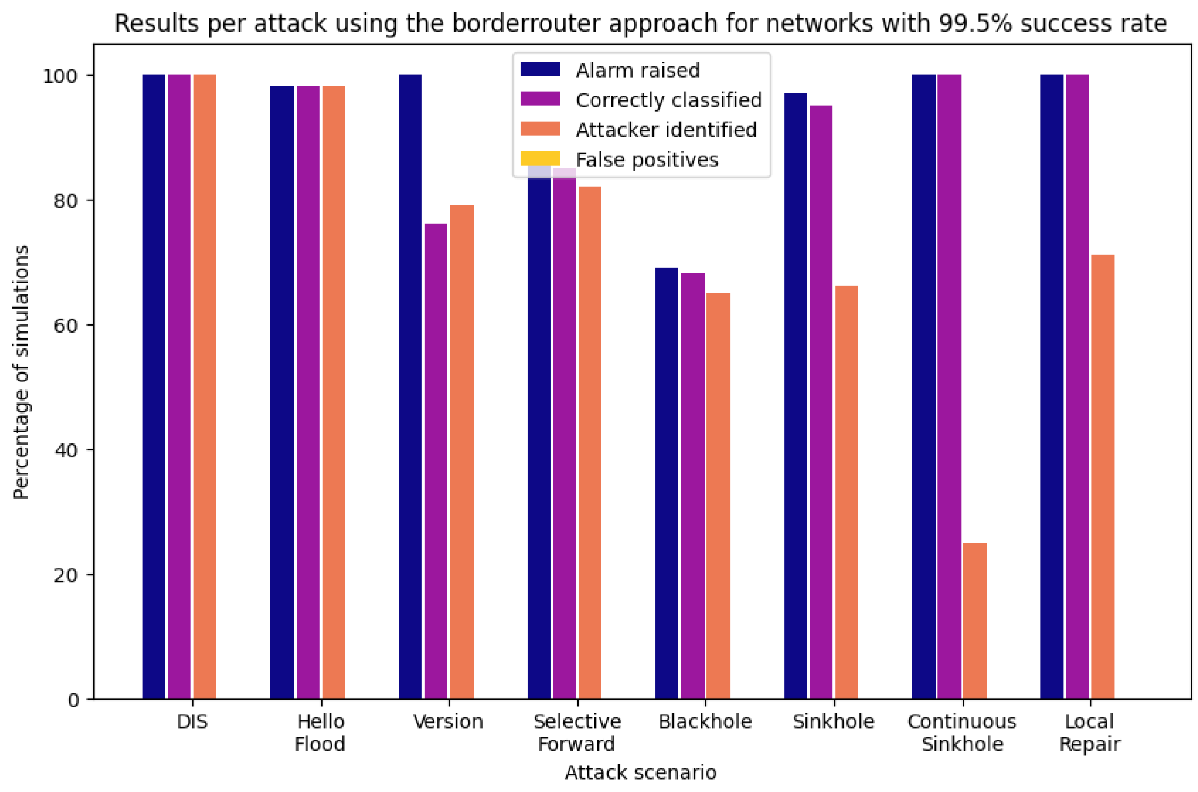


Figure 5.23

False Positives - Border Router Approach

The total number of false positives over the 100 legitimate simulations are 0.

Measure	Percentage
False positives present in simulation	0%
False positives per time step	0%
False positives per time step and device	0%

Table 5.13

Comparison between approaches

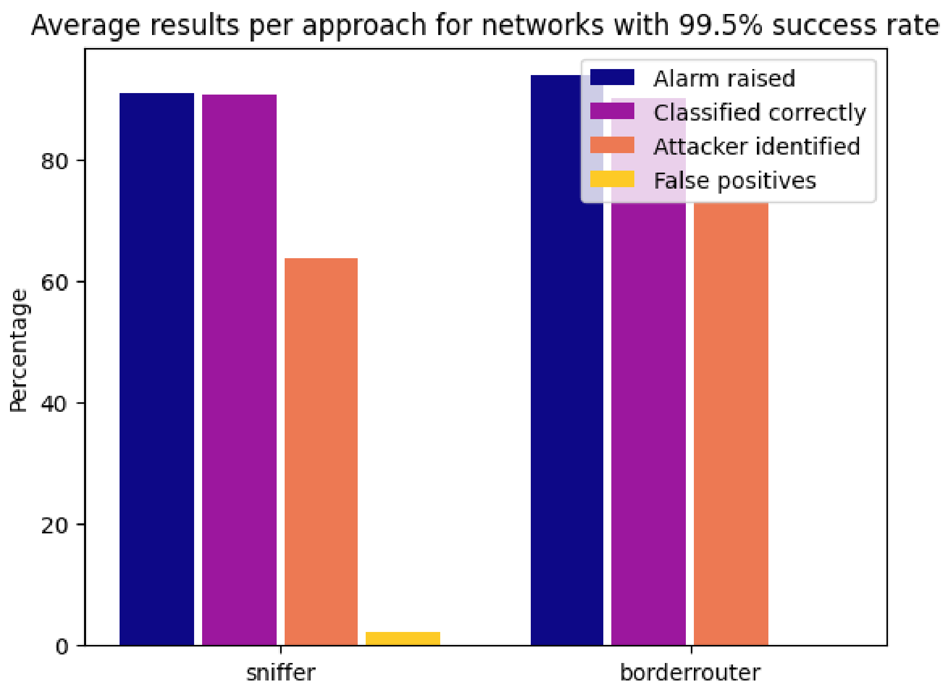


Figure 5.24

Analysis

The results of running DETONAR on a network with 99.5% success ratio gives very similar results to running on a lossless network, with the exception that the border router approach seems to outperform the sniffer approach for the attacker identification. The miss-classified attacks of the Version attack simulations, which has the biggest difference between discovering an attack and correctly classifying it for the border router approach, are predominantly DIS attacks. This is not surprising, since the version change makes the nodes request new DIOs from their neighbors in order to rebuild the DODAG correctly.

From this we can draw the conclusion that DETONAR can produce comparable results in a slightly lossy network, but that it's possibly some breaking point somewhere between 99.5% and 99% success ratios, where the performance of mainly the attack classification starts suffering for the border router approach.

5.3 Trends for Border Router and Sniffer Approach

The results acquired during the experiments on networks with different success rates, and using different approaches can be visualized as:

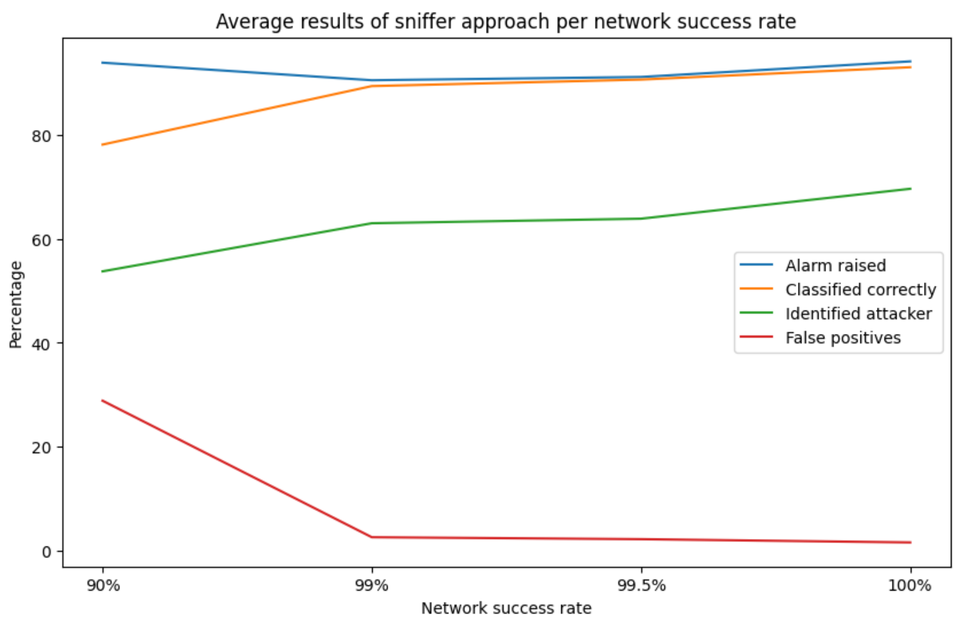


Figure 5.25: Results for sniffer approach

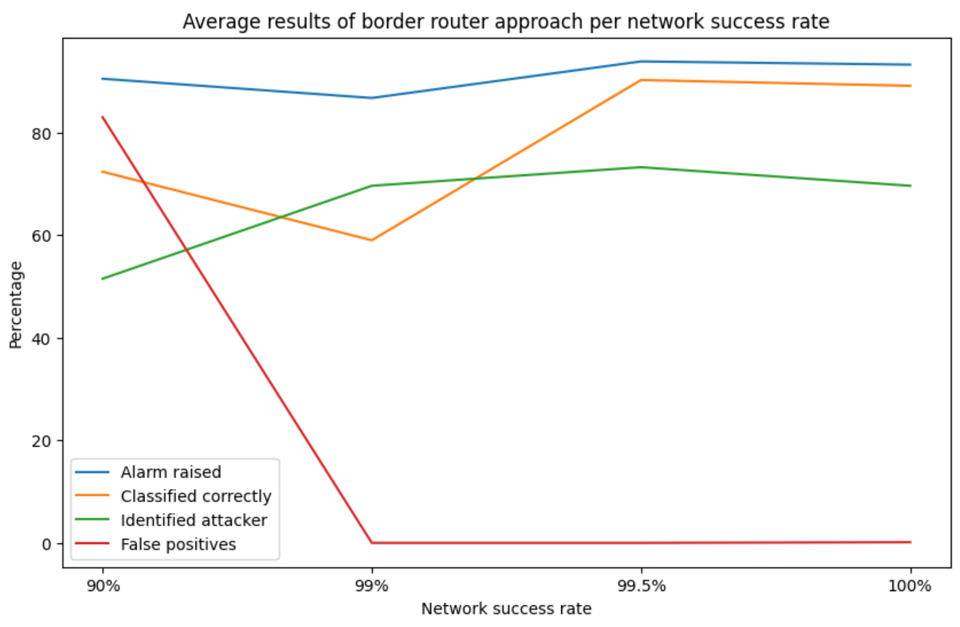


Figure 5.26: Results for border router approach

From these charts, we can confirm what we have previously seen in the charts for single experiments. Notice that the sniffer approach acquires comparable results for 99% success rate and higher, while the border router is more sensitive to the loss of packets, and acquires comparable results for 99.5% success rate and higher. Mainly the attack classification suffers from the loss in the border router approach.

5.4 Discussion of results

I will here briefly compare my results to the results of SVELTE [16], for which the results are presented as the true positive rate, TPR, given by

$$TPR = \frac{TP}{P}$$

The results of SVELTE for networks of 16 nodes, range between 70-90% TPR for the Sinkhole attack, depending on runtime of the experiments with higher true positives for longer runtime. In the case of Selective forward attack, SVELTE achieved from 80% to almost 100% success rate, also depending on runtime. Following are the results from the corresponding attacks from my experiment of using networks with 100% success rate and the border router approach.

Attack scenario	Correctly classified (TP)	Wrong attack classified (FP)	$\frac{TP}{P}$	TPR
Selective forward	81%	15%	$\frac{81}{96}$	84.37%
Sinkhole	91%	33%	$\frac{91}{124}$	73.99%

Table 5.14

My above presented results are within the same range as the SVELTE results, although on the lower end of the ranges. Compared to the SVELTE results, my work explores many different randomly generated topologies and how different constraints of generating topologies may affect the results, as well as 8 different attacks.

Higher classification rates would be desirable for commercial use, and there are several aspects which could be explored further and offer room for improvement. I believe the work of exploring the performance of DETONAR conducted in this project could be a valuable start in order to achieve commercially viable results in the future.

Chapter 6

Conclusions and Future Work

Running DETONAR on border router logs instead of using a sniffer approach in a lossless network gives comparable results; for the 8 attacks implemented, the average attack classification is 89.1% for the border router approach and 93% for the sniffer approach. The two approaches seem to have similar attack identification percentage, which is 69.6% on average for both approaches. This lower percentage, relative to running DETONAR on RADAR, is likely partly due to unsynchronized time between the Contiki devices. The problem with unsynchronized time between nodes has been tackled many times, for example by Yadav et al. [21], and Vera-Pérez et al. [22]. Using time synchronization would probably improve the results of mainly the attacker identification, but possibly other metrics as well, especially for the attack scenarios concerned with changes in rank and version.

Besides the performance of DETONAR on Cooja logs in a lossless network, the performance of DETONAR was found to be affected by loss in the network, especially for the border router approach. Experiments were conducted with different lossyness, simulating networks with 90% success ratio, 99% success ratio and 99.5% success ratio. For the case of 90% success ratio, neither approach gave reasonable results since the false positive rates were really high; 29.25% and 83% for sniffer and border router approach respectively. For using DETONAR in networks with that type of loss, the anomaly detection would need to be tweaked, possibly by adjusting confidence intervals or other parameters of the model used. In the case of 99% success ratio, the sniffer approach gave results comparable to having no loss at all, while the attack classification suffered a lot in the border router approach, with only 59% of attacks correctly classified on average over all scenarios. However, the attacker identification average of the border router approach was still comparable to that of lossless networks. For the networks with 99.5% success ratio, all measures were shown to be comparable to running DETONAR on a lossless network for both approaches, and the border router approach even outperformed the sniffer approach for attacker identification.

One major upside to the border router approach compared to the sniffer approach shown by the experiments are the very low false positives, extremely close to 0%, both for lossless networks and networks with 99% success ratio or more.

This project can be seen as a first step of exploring the use of DETONAR on a border router. The next step could be to analyze and make the changes needed to take attacker behaviour more into account. For example, it is possible that statistics might not be sent if an attacker is malicious, or that for example a fake rank is sent in the statistics instead of the actual malicious rank of a sinkhole attacker. In that case, an IDS would have to be able to pick up on that malicious behaviour in another way, probably through the neighbors of the attacker, for example by comparing values reported in between neighboring nodes.

Lastly, I have only considered one size of networks in this project, and I have not considered the overhead introduced to the network due to sending statistics to the border router instead of using a sniffer approach. These two factors would also be an interesting thing to investigate to further evaluate the viability of this approach.

Bibliography

[1] J. Zhang, W. Li, Z. Yin, S. Liu, and X. Guo, "Forest fire detection system based on wireless sensor network," in *2009 4th IEEE Conference on Industrial Electronics and Applications*, 2009, pp. 520–523.

[2] P. Arroyo, J. Lozano, J. I. Suárez, J. L. Agustín, and P. Carmona del Barco, "Wireless sensor network for air quality monitoring and control," *Chemical Engineering Transactions*, vol. 54, pp. 217–222, 01 2016.

[3] H. M. Jawad, R. Nordin, S. K. Gharghan, A. M. Jawad, and M. Ismail, "Energy-efficient wireless sensor networks for precision agriculture: A review," *Sensors*, vol. 17, no. 8, 2017. [Online]. Available: https://www.mdpi.com/1424-8220/17/8/1781

[4] O. Çetinkaya and O. Akan, *Use of Wireless Sensor Networks in Smart Homes*, 04 2016, pp. 233–258.

[5] T. A. Al-Amiedy, M. Anbar, B. Belaton, A. H. H. Kabla, I. H. Hasbullah, and Z. R. Alashhab, "A systematic literature review on machine and deep learning approaches for detecting attacks in rpl-based 6lowpan of internet of things," *Sensors*, vol. 22, no. 9, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/9/3400

[6] M. A. M. P. Jonsson, S.A Yehya, "Ericsson mobility report," 2022, accessed: 2023-01-09. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/november-2022

[7] A. Agiollo, M. Conti, P. Kaliyar, T. Lin, and L. Pajola, "Detonar: Detection of routing attacks in rpl-based iot," *IEEE Transactions on Network and Service Management*, 2021.

[8] A. Agiollo, "Dna: Detection of networking attacks in rpl based iot networks via traffic analysis," p. 117, Jan 2020.

[9] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128601003024

[10] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6550

[11] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The Contiki-NG open source operating system for next generation IoT devices," *SoftwareX*, vol. 18, p. 101089, 2022.

[12] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, 2006, pp. 641–648.

[13] N. Finne, J. Eriksson, T. Voigt, G. Suciu, M.-A. Sachian, J. Ko, and H. Keipour, "Multi-trace: Multi-level data trace generation with the cooja simulator," 07 2021, pp. 390–395.

[14] H. Keipour, S. Hazra, N. Finne, and T. Voigt, "Generalizing supervised learning for intrusion detection in iot mesh networks," in *Ubiquitous Security*, G. Wang, K.-K. R. Choo, R. K. L. Ko, Y. Xu, and B. Crispo, Eds. Singapore: Springer Singapore, 2022, pp. 214–228.

[15] "Contiki-NG rpl lite documentation," https://docs.contiki-ng.org/en/develop/_api/group__rpl-lite.html#details, accessed: 2022-12-22.

[16] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2661–2674, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870513001005

[17] 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)*, 2015, pp. 606–611.

[18] Kamaldeep, M. Malik, M. Dutta, and J. Granjal, "Iot-sentry: A cross-layer-based intrusion detection system in standardized internet of things," *IEEE Sensors Journal*, vol. 21, no. 24, pp. 28 066–28 076, 2021.

[19] S. Lohier, A. Rachedi, E. Livolant, and I. Salhi, "Wireless sensor network simulators relevance compared to a real ieee 802.15.4 testbed," in *2011 7th International Wireless Communications and Mobile Computing Conference*, 2011, pp. 1347–1352.

[20] "The contiki-ng configuration system," https://docs.contiki-ng.org/en/develop/doc/getting-started/The-Contiki-NG-configuration-system.html, accessed: 2023-01-16.

[21] A. Yadav, A. Waghmare, and A. S. Sairam, "Exploiting node heterogeneity for time synchronization in low power sensor networks," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, 2014, pp. 828–832.

[22] J. Vera-Pérez, D. Todolí-Ferrandis, S. Santonja-Climent, J. Silvestre-Blanes, and V. Sempere-Payá, "A joining procedure and synchronization for tsch-rpl wireless sensor networks," *Sensors*, vol. 18, no. 10, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/10/3556