Observability in Machine Learning based Intrusion Detection Systems for RPL-based IoT

Malin Strand
Abstract

As IoT devices become more and more present in our daily lives, security in IoT networks has become a major concern. A promising approach for detecting attacks is the use of machine learning based Intrusion Detection Systems (IDSs). The attack studied in this thesis is the blackhole attack, an attack causing parts of the network to disconnect. The IDS identifies malicious activities in the network by analyzing the network traffic. However, in the case of a blackhole attack, not all activity can necessarily be seen by the IDS. We study the impact of limited observability for an IDS and simulate attacks on RPL-based IoT networks using the Cooja simulator together with the Multi-Trace extension. The data obtained from the simulations are processed to match the observability of the sink node, and then used to train and test a Deep Neural Network machine learning model for an IDS. The DNN is trained on a conventional blackhole attack and then tested on variations of that attack in order to evaluate the efficiency of the IDS ability to detect previously unseen but similar attack types. We use a data preparation method that allows the IDS to detect attacks online, as opposed to retrospectively. Additionally we modify the network simulations to be more realistic and assess how the IDS is affected by the more realistic network simulations. Our study found that the IDS was not significantly affected by the more realistic simulations, however observability proved to be a critical factor for detecting attacks. We show that an IDS with access to full observability of network activities achieved greater performance, with a detection rate of approximately 92%, compared to an IDS with limited observability, which achieved a detection rate ranging from 45% to 78%. Our findings highlight the importance of considering and developing new techniques for enhanced observability, in order to further improve and develop IDSs.
Contents

1 Introduction 2

2 Background 4
   2.1 6LoWPAN 4
   2.2 RPL 5
   2.3 Attacks on RPL 6
      2.3.1 Blackhole attack 7
   2.4 Application layer 7
      2.4.1 MQTT Protocol 8
   2.5 Machine Learning 9
      2.5.1 Deep Neural Network 10

3 Related work 10

4 Method 11
   4.1 Generating the datasets 13
   4.2 IDS design 15
      4.2.1 Observability 15
      4.2.2 Data Preparation 16
      4.2.3 Modelling 17
   4.3 Implementing a more realistic network environment 17
   4.4 Client implementation 18

5 Evaluation Metrics 20
   5.1 Recall 20
   5.2 Accuracy and Precision 21
   5.3 ROC 21

6 Results 22

7 Discussion 30

8 Conclusions and Future work 32
1 Introduction

Internet of Things (IoT) devices have become increasingly present in our daily lives, as they now maintain an important role in a variety of critical areas such as healthcare, traffic, agriculture and even aerospace [1]. Therefore, security in IoT networks has become a major concern. In 2016, the Mirai botnet, comprised of IoT devices among others, launched massive distributed denial-of-service (DDoS) attacks on several targets [2]. In 2021, the number of IoT attacks exceeded 1 billion [3].

Defence against network attacks often include three components [4]: prevention, detection and mitigation. The first component aims to prevent attacks from happening in the first place. If an attacker nevertheless succeed to cross the first line of defense, the second component aims to detect that the network is under attack. Once the attack is detected, the third component aims to mitigate and recover from the attack.

The purpose of second line of defense is to detect suspicious behaviour in the network, which is done via Intrusion Detection Systems (IDSs). This thesis will focus on attack detection, the second line of defense.

However, designing efficient IDSs for IoT networks is challenging due to resource constraints in IoT devices. One method for intrusion detection that has been proven effective is the use of machine learning [5] where attacks are detected through the identification of network anomalies and previous experiences. To be effective, the machine learning-based algorithms require a lot of training data. In addition, a more diverse dataset allows for more knowledge and insight to be captured by the ML model. The efficiency of such algorithms might therefore to a large extent rely on the availability of data within the IoT network. Valuable data for an IDS model is derived from past attacks and the specific configuration of the system. Hence, this data can only be obtained after the event of an attack rather than before the attack. Therefore the IDS may fail to detect new attack types due to a lack of sufficient training samples.

As IoT nodes are resource constrained devices, one of the common options [6] is to place the IDS on the sink node because it includes more resources. In that case, the ML-based IDS has to be trained on only the data that is observable to the sink node. However, not all activity in the network is necessarily observable, for instance when an attack disconnects parts of the network. In that case, messages from the disconnected nodes will never reach the sink node and that data is considered unobservable. However, it is important to distinguish between messages that are expected but do not arrive, and messages that are random and unpredictable. While unexpected messages that do not arrive are unobservable, the absence of expected messages can still be considered observable. Detecting attacks might therefore rely on expected data that is not seen, rather than solely relying on observed data.

In this thesis, we simulate attacks on RPL-based low-power multi-hop wireless networks using the Cooja simulator [7] together with the Multi-Trace [1] extension to generate datasets. We focus on the case when not all activity in a network can be observed, specifically when attacks disconnect the network.
previous work, the data generation was performed with the Cooja simulator and all messages that was sent to the sink node were available and used as training data for the machine learning algorithm. The algorithm was therefore trained under the assumption that all data was observable for the sink node, when in reality some data may be unobservable due to attacks disconnecting the network. This can be problematic when the amount of data that the IDS observes in real-world usage is significantly less compared to the training data. Additionally there is a weakness in previous work regarding data preparation methods as the machine learning algorithm receive data after running the network for 5 hours and label it as attack or non-attack. This is a retrospective method that only detects attacks after 5 hours, and not online method. Furthermore, the nodes are set to send application layer messages to the sink node at a periodic rate which is not quite realistic.

We deviate from previous work in that we address these issues and only consider observable data when generating the datasets. In that way, the machine learning algorithm can be trained on only observable data and the training data is more representative of the real-world usage. In addition we use an online approach for data preparation which allows the classifier to detect attacks based on most recent data. We also modify the network traffic to be more realistic and consider two ways of sending messages. We generate two types of datasets containing data from simulations where messages are sent periodically, as in previous work, and as a Poisson process respectively. This allowed us to assess how the more realistic network would impact the IDS performance.

Contributions
In this thesis we have made the following contributions:

- First, we study the impact of limited observability and train an IDS on observable data. This is explained with more detail in Section 4.2.1.

- Second we use a data preparation method that allows the IDS detect attacks online based on sliding window. The data preparation is explained in Section 4.2.2.

- Third, we have considered two ways of sending application layer messages. The first involves sending periodic messages as previously in [8], while the second way is that we considered the network traffic to be more realistic if it is not assumed to be periodic, but rather follows a Poisson process. In Section 4.3 we explain how it has been implemented.

After presenting background on IoT networks, attacks and machine learning in Section 2 the previous work upon which this thesis is based is presented in Section 3. Section 4 describes the method that was used to generate the datasets and prepare it for machine learning, as well as the method used for implementing more realistic network simulations. To generate attack variations, we replicate the attack types implemented by [8] explained in Section 4.1. Section 5 presents the evaluation metrics used to evaluate the performance of the IDS, and the
results are presented in Section 6. Section 7 is a discussion of the results and Section 8 describes the conclusions that could be drawn as well as a proposal for future work.

2 Background

IoT networks consist of devices with limited resources, such as low processing power, memory and battery life. Hence, it is important to keep the power consumption low when it comes to communication between nodes. This makes it so that standard protocols designed for regular networks are not applicable to IoT networks, and there is a need for a protocol specialized on low power communication. 6LoWPAN was implemented to address the needs of low power wireless networks and is the most commonly used protocol standard for IoT networks. Section 2.1 provides a detailed description of this standard.

In IoT networks that consist of a large number of IoT devices, there is typically a sink node that acts as the root of the network. The sink node is more powerful than regular nodes in that it consists of more resources and therefore serves as a central point for collection and aggregation of data. Similar to wireless sensor networks (WSN), the nodes collect some sort of data from the physical environment that are transmitted to the sink node. In turn the data can be processed and analyzed depending on the purpose. In a wireless network, all nodes are not necessarily in range of the sink node and nodes might need to forward packets on behalf of one another. RPL is a routing protocol implemented for resource constrained wireless networks and has support for tree-like topologies with a sink node as the root. Section 2.2 provides a description of RPL.

Subsequent sections explore attacks on RPL, with a subsection dedicated to the blackhole attack, which is the main focus of this thesis. Additionally there will be a discussion on application layers for IoT. The final section introduces machine learning and the DNN classifier.

2.1 6LoWPAN

IPv6 over Low-Power Wireless Personal Area Networks, or 6LoWPAN, is a standard protocol that enables the use of IPv6 in wireless networks consisting of resource constrained nodes, such as limited power or memory. In order to achieve this, the IPv6 header is compressed by adding an adaptation layer (called LoWPAN adaptation layer, see Figure 1) and using fragmentation to match the Maximum Transition Unit (MTU) of IPv6 (1280 bytes), reassembled with the adaptation layer [9], [8].

While there are a variety of other protocols for low power wireless networks, 6LoWPAN is the most widely used protocol standard for IoT networks [10]. As for the transport layer, TCP is not commonly used with 6LoWPAN but is rather utilized with UDP or ICMP protocols because of performance and complexity issues with TCP [11]. 6LoWPAN nodes are connected to the Internet via a IPv6
Border Router (6BR) that translates IPv6 packets originating from the Internet to 6LoWPAN standards and vice versa. In addition, 6LoWPAN supports multi-hop communication, meaning that nodes can forward packets on behalf of other nodes, which is essential in a wireless network since not all nodes are necessarily in range of the 6BR node.

2.2 RPL

Routing Protocol for Low power and lossy networks, or RPL, is a standard routing protocol for IoT networks [12] and is most widely used for routing in 6LoWPAN networks. The characteristic of RPL routing is its tree-like topology called a Destination Oriented Directed Acyclic Graph (DODAG), containing one or more nodes called root or sink node. An example of an RPL-based 6LOWPAN network topology can be seen in Figure 2.

The sink node can act as a Border Router (BR6) which enables communication between the 6LoWPAN network and the Internet [13]. Each node has a unique ID (an IPv6 address), as well as a parent and a list of neighbouring nodes. In addition, each node maintains a rank value that indicates its position in the topology with respect to the sink node. The rank value strictly decreases towards the sink node, and is determined by an Objective Function (OF) that considers factors such as topological distance to the sink, link metrics or other properties [14].

In RPL, control messages are used in order to exchange routing information between nodes. There are a few different types of control messages [13]: DAO (Destination Advertisement Object), DAO-ACK (DAO Acknowledgement), DIO (DODAG Information Object) and DIS (DODAG Information Solicitation). DAO messages are sent by children to their respective parents and they contain information required for downward traffic (towards the leaves). The DAO-ACK message is used for confirmation that a DAO message has been received. DIO messages are exchanged during the set up in order to build the
The green node represent the sink node acting as a Border Router connecting to the Internet. The numbers on the nodes represent their ranks.

DODAG, and DIS messages are used for requesting graph related information from neighbours. A parent could for example use a DIS message to request a DAO message from its children.

2.3 Attacks on RPL

Attacks on RPL can be divided into three main categories: resource, traffic and network topology based attacks [15]. The classification of attacks on RPL together with some examples of attacks in each category can be seen in figure 4.

Resource based attacks [16] are attacks that aim to consume network resources such as node memory, battery, processing etc. These attack types causes the network to become unavailable. As an example, the flooding attack exhaust the system’s resources by sending a high volume of traffic, consuming its resources by having it process a bunch of none sense traffic.

Traffic attacks [16] generally aim to eavesdrop on network activities or to gain a better understanding about the network as a first step for other attacks. Sniffing and traffic analysis attacks focuses on intercepting data and analysing network traffic with the intent of eavesdropping. It does not affect the performance of the network itself, but rather is an attack on integrity and confidentiality.

Network topology attacks aim to disrupt network performance or isolate nodes by manipulating the network topology [16]. The network topology attacks fall further into two categories: sub-optimization and isolation attacks.
The sub-optimization attacks aim to decrease network performance by introducing new non-optimal packet routes. For instance, the Routing table poisoning attack uses forged DAO control messages to modify routing information in order to build sub-optimal paths.

Isolation attacks on the other hand, serve to isolate nodes or parts of the network and essentially disconnecting them from the rest of the network. The Blackhole attack is an example of an isolation attack and will be discussed in greater detail below.

2.3.1 Blackhole attack

The Blackhole attack is classified as a network topology attack, and more specifically an isolation attack, causing nodes or parts of the network to disconnect. In a Blackhole attack, one malicious node simply drops all packets that it is supposed to forward. If placed strategically in the network topology, the attacking node may cause several nodes to become isolated and disabling their communication with the sink node. Figure 5 displays an RPL network topology under normal circumstances compared to when its under attack.

The malicious node accomplishes this by broadcasting a low rank value to its neighbors [17], making it seem like routing through the malicious node is the most optimal path towards the sink node. This attracts the neighbors to choose the malicious node as their routing parent, creating a sub-optimal path topology. Next, the malicious node simply drops all packets originating from other nodes, silently isolating all of its children from the sink node.

2.4 Application layer

IoT networks consist of devices that serve as either sensors, such as temperature sensors, or they are actuators that receives commands and act on them, such as light bulbs with different color options. Application layer messages are utilized to communicate this data between devices.

As discussed in section 2.2, the IoT nodes use control messages in order to maintain or modify the network topology in response to any disruption or
Node 2, the red node, is the attacker and as can be seen is placed in a strategical way such that it isolates almost the entire network except for node 1.

change in the network. In this thesis the IoT nodes send their rank and other information to the sink node, akin to how a temperature sensor transmits data to a sink node. To simulate a practical IoT network, it is reasonable to have this data at the application layer.

2.4.1 MQTT Protocol

There are a variety of application layer protocols that are suitable for IoT. Some examples are XMPP, AMQP and MQTT, the latter being the most popular one [18]. The reason for MQTT being highly suitable for IoT its lightweight architecture and its intent to work on small devices with low power. These protocols are all usable according to different situations, for example the XMPP protocol is suitable when there is a need for a secure and reliable many-to-many (M2M) connection [19] but requires a higher bandwidth compared to MQTT. Hence, the choice of using the MQTT protocol was simply because it being the one most widely used one and there was no need for a more complex protocol.

The MQTT protocol utilizes the publish/subscribe [20] messaging principle where devices can subscribe to specific topics that are related to them. There is an additional entity in the network called a broker that receives all messages published from clients (any device running an MQTT library and is connected to the broker [20]) and forwards them to the correct destinations. When a publisher has a message to send, it is sent to the broker where the message is distributed to all clients subscribed on the corresponding topic. Figure 2 shows an example of an MQTT architecture with a temperature sensor acting as publisher.

![Figure 4: An RPL network under normal circumstances compared to under attack.](image-url)
2.5 Machine Learning

Machine learning can be categorized into two main approaches: supervised learning and unsupervised learning [22]. The key difference between the two approaches is whether the training data is labeled or not. In supervised learning, the dataset is labeled with corresponding outputs, and the goal is to learn a general rule or correlation between inputs and outputs. On the other hand, unsupervised learning does not require labeled data and instead analyzes data in a different way to discover patterns and structure. Typically, supervised learning algorithms are utilized for classification and regression problems, whereas unsupervised learning algorithms are suitable for clustering, association, and dimensionality reduction tasks.

A classifier in supervised learning is an algorithm that takes data as input and maps it to one of several category labels. In the context of Intrusion Detection Systems, the categories (labels) that correspond to the datasets are generally binary: attack or non-attack. The classifier is trained on this labeled dataset where the goal is to learn a general rule or correlation between the input data and the corresponding label in order to make accurate predictions on new data. Some examples of classifiers used to train Intrusion Detection Systems include Support Vector Machines (SVM), Random Forest Classifier (RFC) and Deep Neural Networks (DNN) [23], [8]. Unsupervised learning algorithms are generally used for tasks that do not involve categorizing data but rather to discover patterns and relationships in the input data. Clustering algorithms such as K-means and K-dimensional trees are examples of unsupervised learning algorithms that can be used for IDSs to identify similar patterns in the input data without the need for explicit category labels.

Overall, supervised and unsupervised learning algorithms play different roles in machine learning, and the choice of approach and algorithms depends on the specific problem and available data.
2.5.1 Deep Neural Network

A Deep Neural Network (DNN) [24] is a type of supervised learning algorithm that can be used as a classifier. It is an artificial neural network modeled after the structure and function of biological neural networks [25]. DNNs are composed of multiple layers, including an input layer, one or more hidden layers that perform complex computations, and an output layer. Figure 6 illustrates an example of a DNN architecture.

The DNN layers are composed of nodes, inspired by neurons in biological brains. Each node is connected to all nodes in the adjacent layer and the connections are assigned a numerical value called weight. A heavier weight will result in a greater impact on the next layer [26]. For each layer, the input is multiplied by the weight, resulting in a value between 0 and 1. This value will be the input for the next layer. In the final layer, the neuron with the highest value determines the output.

During the training process, the DNN is presented with training data that is labeled with the corresponding output. The DNN makes a prediction and this predicted output is compared to the actual output using a loss function, which determines the magnitude of the possible error. Depending on the value of the loss function, the DNN makes adjustments to the weights. The goal of the training process is to minimize the value of the loss function by adjusting the weights of the interconnected neurons and gradually improve the predictions.

3 Related work

The authors of [1] highlight the lack of sufficient data traces needed to train ML-based IDSs in wireless low-power, multihop networks. The reason for the lack of data traces, according to the authors, is that there is no single entity that can overhear all the traffic in the network. To tackle this challenge, the researchers propose Multi-Trace - an extension to the Cooja simulator that produces multilevel datatraces suitable for training machine learning algorithms.
The authors of [23] present an ML-based IDS which is trained using datasets generated by Cooja and Multi-Trace. Their findings show that diverse training data leads to better generalization and higher accuracy of the models, compared to those trained with less diverse data. In their research, the authors tested the SVM, DNN, and RFC classifiers where the latter achieved the highest accuracy and precision.

Pettersson et al. in [8] use Multi-Trace to generate datasets and implements variations of the Blackhole attack and evaluates how these attack variations impacts an IDS model developed by [23] trained on conventional Blackhole attacks. In their work, each node is set to send an application layer message to the sink node periodically, including the control messages that that node has sent/received. These periodic messages are sent at a rate of precisely one message per minute. The classifier that is used is the Random Forest Classifier.

It is important to point out two weaknesses of the aforementioned papers that make them challenging to be used in real world applications:

- As data generation in the papers described above is performed with the Cooja simulator, all the application layer messages that have been sent to the sink node are available in a log file. Therefore, the machine learning algorithms used in the papers described above are trained under the assumption that all logs are observable from the sink node. In reality, some logs may be unobservable due to the attack disconnecting parts of the network.

- The applied data preparation methods in the papers described above assign a label (attack/non-attack) to the network as a whole. More precisely, the classifiers receive all logs generated after running the network for 5 hours and determines whether the network was under attack or not. This retrospective approach is not an online method and it assumes that all logs are observable.

4 Method

As described by authors in [7], the distributed nature of wireless sensor networks and their longer compile and run-time make software development for IoT networks difficult. By using a network simulator however, the development is simplified and we can study the system behaviour and develop algorithms in a controlled environment. For this reason, the Cooja network simulator was used to simulate IoT networks.

Cooja is a flexible Java-based tool that is part of the Contiki operating system designed for simulating wireless sensor networks [7]. The reason for Cooja being flexible is that it enables cross-level simulation and that all levels of the system can be altered, such as node hardware (low-level) or radio transmissions (high-level). Together with Cooja, Multi-Trace [1] was used to generate the datasets needed for training and evaluating an ML-based IDS. Multi-Trace is an extension of Cooja that captures and logs data at different levels.
The data that is generated from the simulations, captured by Multi-Trace, are used as training data for an ML-based IDS. In this thesis we use a supervised DNN algorithm for the classification task. It is essential that the data is diverse and covers a broad range of scenarios to provide the ML algorithm with a greater understanding and knowledge. Figure 7 provides an overview of the end-to-end approach, from simulation generation to model training.

![Figure 7: Illustration of the method](image)

Initially we have 23 attack variations, introduced in section 4.1, that are composed of base simulation files with nodes ranging from 5 to 20, that is 16 x 23 base simulation files. For each of the base files composed of 5, 10, 15 or 20 nodes, we generate 50 new simulation files by randomizing the position of the nodes. Next we run all the simulations and prepare the datasets for the machine learning model, described in section 4.2.

Our dataset differs from previous studies in that we modified the network traffic to be more realistic and added more features regarding control message statistics. We used the same attack variations and method for creating the simulation scenarios as authors in [8]. However, we considered two ways of sending messages; periodically and as a Poisson process, and generated datasets for both of these scenarios.

Transmitting messages periodically makes it easier for an IDS to detect patterns and identify regular network behavior. Expected messages that are not received at a periodic rate could be identified as anomalies. However, a predictable IDS can also be exploited by attackers, making it easier to avoid detection if the predictability can be taken to their advantage. Introducing more randomness to the messages by having them model a Poisson process can make the network traffic less predictable for the IDS and harder to identify patterns in normal traffic behavior. This in turn can make it harder to identify malicious behavior, but a less predictable IDS can not be exploited in the same way by attackers.

Therefore, to balance predictability and randomness, we considered both periodic and Poisson process message transmission. This allowed us to evaluate the impact of a more realistic network environment on the IDS performance,
while also taking into account the need to establish some unpredictability to make it more difficult for attackers to exploit network vulnerabilities.

Our implementation and modifications of the nodes are described more in depth in section 5. The following sections will describe the dataset generation.

4.1 Generating the datasets

Base scenarios
In order to evaluate the impact of attack variations on an IDS trained on conventional attacks, we use the attack variations implemented by authors in [8]. The reason for this approach is that attackers ideally want to avoid detection and therefore might alter the attack accordingly. Testing the IDS on attack variations that it has not specifically been trained on allow us to evaluate its effectiveness in detecting previously unseen, but similar, attacks.

There are 23 different attack scenarios which are all given an ID from 1 to 23. Each scenario contains a variation of the Blackhole attack, see Table 1. In addition there is one scenario without an attack, called the normal case. In total, there are 24 scenarios. For each scenario there are several base simulation files, containing nodes ranging from 5 to 20. This means that every scenario (ID) contains 16 base simulation files, which in turn means that there are 384 (16x24) base simulation files.

<table>
<thead>
<tr>
<th>ID</th>
<th>Attack variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic attack</td>
</tr>
<tr>
<td>2-6</td>
<td>On/off (1 attacker)</td>
</tr>
<tr>
<td>7-12</td>
<td>Decreasing rank (1 attacker)</td>
</tr>
<tr>
<td>13-17</td>
<td>Sequential On/off (2 attackers)</td>
</tr>
<tr>
<td>18-23</td>
<td>Sequential decreasing rank (2 attackers)</td>
</tr>
<tr>
<td>normal</td>
<td>none</td>
</tr>
</tbody>
</table>

Table 1: Attack variations in base simulation files.

The Basic attack is considered the conventional, "normal", Blackhole attack, where a randomly selected node starts an attack 15 minutes after the start of the simulation. In the On/off variation, the attacking node pauses and resumes the attack in intervals during the simulation, as opposed of being constantly active. In the Decreasing rank variation, the attacking node initially advertises a higher rank and gradually decrease it, rather than starting with a low rank. Furthermore, for both of these scenarios, there are additional variations where a second attacker is introduced, and the attackers alternate being the attacker.

Randomize node positions from base
We generate 50 new simulations using a Python script that randomizes the node positions for each of the base simulation files given to it. Only simulation files with 5, 10, 15, or 20 nodes are considered, which results in 50 new simulation files for each case where the nodes have different positions.
In summary, there are 24 distinct attack (or normal) scenarios, and for each scenario, we use the base files with 5, 10, 15 or 20 nodes and randomize their positions 50 times. This process results in a total of $24 \times 4 \times 50 = 4800$ simulation files.

**Running simulations in Cooja**

A Python script is used to automatically run all simulations. It uses Cooja’s no-GUI ability in order to accelerate the process. After a successful run, a folder containing the data traces collected during the simulation is generated. Figure 8 displays the log files obtained after a successful run.

![Figure 8: Logs after a successful simulation run](image)

The most informative file from which we extract information and data is the mote-output.log. It contains output and events from all motes during the entire simulation, where an event is for example sending, receiving or forwarding a packet. A sample of a mote-output.log file can be seen in Figure 9.

![Figure 9: Mote-output.log](image)

The image above shows a sample of mote outputs during a simulation. During this particular time in the simulation, mote 2 sends a UDP packet to mote 1 (the sink node) on row 731: ”Sending periodic message 4”. Next it prints some data, then the path taken by the UDP packet can be seen on row 733-734. Mote 1 receives the UDP packet: ”Received message” on row 735, and sends a response back to mote 2: ”Sending response” on row 736. The response travels back towards mote 2 on row 737-738 and is received by mote 2 on row 739: ”Received response”.

**Parsing RPL statistics**

After running all simulations and obtaining the log files, a Python script is used to extract crucial data from the logs into a .csv file. Throughout a simulation, each node continuously reports RPL statistics (as seen in Figure 9, row 732).
These are parsed from the mote-output logs and stored in a .csv file containing the features presented in Table 2.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time stamp of event in the simulation</td>
</tr>
<tr>
<td>Mote</td>
<td>ID of node</td>
</tr>
<tr>
<td>Seq</td>
<td>Sequence number of message</td>
</tr>
<tr>
<td>Rank</td>
<td>Rank value of node</td>
</tr>
<tr>
<td>Version</td>
<td>DODAG version</td>
</tr>
<tr>
<td>DIS-R</td>
<td>No. of DIS messages received (unicast + multicast)</td>
</tr>
<tr>
<td>DIS-S</td>
<td>No. of DIS messages sent (unicast + multicast)</td>
</tr>
<tr>
<td>DIO-R</td>
<td>No. of DIO messages received (unicast + multicast)</td>
</tr>
<tr>
<td>DIO-S</td>
<td>No. of DIO messages sent (unicast + multicast)</td>
</tr>
<tr>
<td>DAO-R</td>
<td>No. of DAO messages received</td>
</tr>
<tr>
<td>DAO-S</td>
<td>No. of DAO messages sent</td>
</tr>
<tr>
<td>Rpl-tot-s</td>
<td>Total no. of RPL messages sent</td>
</tr>
<tr>
<td>Rpl-tot-r</td>
<td>Total no. of RPL messages received</td>
</tr>
<tr>
<td>Tot-s</td>
<td>Total no. of messages sent</td>
</tr>
<tr>
<td>Tot-r</td>
<td>Total no. of messages received</td>
</tr>
</tbody>
</table>

Table 2: Features parsed from nodes’ RPL statistics

DIS and DIO messages can be either unicast, which refers to data being transferred between a single sender and receiver, or multicast, which refers to data being transferred between multiple senders and receivers. The total number of DIS or DIO messages sent or received is the sum of both unicasted and multicasted messages.

4.2 IDS design

4.2.1 Observability

Not all features in Table 2 can be assumed to be observable at all times. During an attack, some nodes may be disconnected from the network, and their reported data may never reach the sink node. To train the IDS using only observable data, we need to generate a dataset that includes only the data that is observable to the sink node. This involves iterating through the logs in the mote-output file and determining whether the data reached the sink or was dropped by an attacking node. If the data was dropped, we replace it with 0s in the table to indicate that it was unobserved. An example of a data table containing observable data for a specific attack scenario is provided in Table 3.
| Node | Time | Rank | DIS-S | DIS-R | DIO-S | DIO-R | DAO-R | ........
|------|------|------|-------|-------|-------|-------|-------|--------
| 2    | 1000000 | 256  | 100   | 50    | 10    | 5     | 12    | ........
| 2    | 2000000 | 256  | 120   | 60    | 20    | 10    | 22    | ........
| 2    | 3000000 | 256  | 140   | 70    | 30    | 15    | 32    | ........
| 2    |      |      |       |       |       |       |       | ........
| 4    | 1000000 | 200  | 112   | 15    | 11    | 100   | 7     | ........
| 4    | 2000000 | 0    | 0     | 0     | 0     | 0     | 0     | ........
| 4    | 3000000 | 0    | 0     | 0     | 0     | 0     | 0     | ........
| 4    | 4000000 | 0    | 0     | 0     | 0     | 0     | 0     | ........
| 4    |      |      |       |       |       |       |       | ........
| 5    | 1000000 | 210  | 140   | 31    | 14    | 190   | 3     | ........
| 5    | 2000000 | 0    | 0     | 0     | 0     | 0     | 0     | ........
| 5    | 3000000 | 0    | 0     | 0     | 0     | 0     | 0     | ........
| 5    | 4000000 | 0    | 0     | 0     | 0     | 0     | 0     | ........
| 5    |      |      |       |       |       |       |       | ........

Table 3: Observable data for an attack scenario resulting in a disconnection of nodes 4 and 5.

The "..." at the top of the table indicates the remaining features in Table 2. That is; DAO-S, Rpl-tot-s, Rpl-tot-r, Tot-s and Tot-r. The values are simply examples and are not collected from real data.

4.2.2 Data Preparation

Data pre-processing is a crucial step that involves processing data to prepare it for machine learning. The goal of this step is to transform the data into a format that is compatible with the machine learning algorithm being used.

To begin with, we generate tables that combine the data gathered from simulations involving 5, 10, 15, and 20 nodes into individual tables, as shown in Figure 10. As we have conducted 50 simulations containing randomized node positions for each case, each table will hold the data from 50 simulations. Moreover, considering the different attack variations described in Section 4.1, we obtain 4 tables for each attack variation, resulting in a total of 96 tables.

Each row of the table corresponds to a specific simulation time and contains data reported from all the nodes. If a node is unable to communicate with the sink node due to an attack, its reported data is recorded as 0. The last column of the table contains the label that indicates whether the network was under attack at that particular time, with 0 denoting no attack and 1 denoting an ongoing attack. Since the number of features or columns is always 11 times the number of nodes, the table size increases proportionally as the number of nodes grows larger.
Once the tables have been generated, the feature values are averaged over 5-minute time intervals and stored as vectors. To create each vector, the feature averages are calculated for a time chunk of 5 minutes, starting from the first 5-minute interval and moving forward by 1 minute at a time. For example, the feature averages for time intervals 1-5 are calculated first, then for intervals 2-6, and so on. These resulting vectors are used as the input data for the DNN model.

### 4.2.3 Modelling

Once the data pre-processing is complete, the tables containing feature vectors can be used as input for the supervised deep neural network (DNN) classifier. The size of the input layer in the DNN model will depend on the size of the feature vectors. For example, if the vector contains data from 5 nodes, the input layer will have 55 nodes (11 features multiplied by 5 nodes), whereas for 10 nodes, the input layer will be 110 nodes, and so on. The DNN algorithm is trained using the data generated in previous steps.

### 4.3 Implementing a more realistic network environment

In Figure 6, an example of running a simulation in Cooja using the graphical mode can be seen.

In this scenario, there are five regular nodes referred to as clients, and a single sink node known as the server. On the right-hand side is the mote output containing sections for Time, Mote and Message, displaying messages transmitted by the nodes (referred to as motes in Cooja) at specific time stamps. In this example, the clients are implemented to send a simple UDP message to the server in a periodic manner, which is then echoed back as a response from
the server. In addition, the nodes report data regarding their control message statistics.

This is a simple implementation and does not accurately reflect real-world scenarios. To make it more realistic, we aimed to add an application layer using the MQTT protocol (see section 2.2). When discussed further however, it was concluded that implementing the MQTT protocol would not be the best option. This is because the MQTT protocol comes with some overhead and would not enhance observability. Instead, we decided to take a different approach and modify the implementation of the client nodes to make the network traffic more realistic.

### 4.4 Client implementation

Prior to running the simulation, a software for the clients as well as the server must be specified. The software for the Cooja motes are implemented in the C programming language and the algorithm for the clients can be summarized as follows:

```c
/* Some initial UDP connection setups */

/* Define send interval (60 seconds) means 1 packet / min */
int send_interval = 60 * clock_second

/* Define timer using etimer library */
static struct etimer periodic_timer;

/* Set timer to random number < send_interval */
etimer_set(&periodic_timer, send_interval % random_rand())

while(1)
{
    /* Wait until timer is finished */
    process_wait_event_until(etimer_expired(&periodic_timer))
    /* Log statistics and send UDP message */
    log_rpl_statistics()
    send_udp_message()
    /* Reset timer */
}```
The listing above showcases the nodes sending a UDP message at a periodic rate. A more realistic scenario would be to have the UDP messages behave as Poisson events, and the series of UDP messages model a Poisson process.

In a Poisson process, the average time between events is known, but the exact timing of events are random. The rate parameter, \( \lambda \) (lambda), represents the average time between Poisson events. Using the same send interval as in Listing 1, we can calculate \( \lambda \) to be 1/60, that is 1 packet every 60 seconds on average. In a Poisson process, the time between events follows an exponential distribution. Therefore, by modifying the timer in Listing 1 to be set to an exponentially distributed random variable, the UDP messages will model a Poisson process.

The steps for generating exponentially distributed random numbers goes as follows. Firstly, we generate uniformly distributed random numbers \( u \) using the function `random_rand()` and dividing the result by 65536, resulting in a random number between 0 and 1. The exponentially distributed random variable \( X \) can then be calculated using the following formula:

\[
X = -\frac{\log(1 - u)}{\lambda}
\]

By setting the timer to \( X \) after each loop, the timer will have an exponential distribution and the UDP messages will model a Poisson process. The listing below shows the implementation of the clients with an exponential distributed timer:

```c
... etimer_reset(send_interval)
Listing 1: UDP-client periodic (original)

* Some initial UDP connection setups *
/* Define lambda (events on average per unit of time) */
float lambda = 1/60
/* Define exponential timer using etimer library */
static struct etimer exponential_timer;
/* Start with a static timer */
etimer_set(&exponential_timer, 180 * clock_second)
while(1)
    /* Wait until timer is finished */
    process_wait_event_until(etimer_expired(&exponential_timer))
    /* Log statistics and send UDP message */
    log_rpl_statistics()
    send_udp_message()
    /* Reset timer to exponentially distributed random variable */
    1) Generate random number 'u' between 0 and 1 */
    float u = random_rand() / 65536 /*
    2) Calculate 'next_time' with formula */
    float next_time = -log(1 - u) / lambda
    etimer_set(next_time)
Listing 2: UDP-client exponential (modified)
```
With this implementation, we repeat the process of generating datasets by using the same base simulation files but changing their client software to the UDP-client exponential. Next we will compare the results of the IDS when using the UDP-client periodic and the UDP-client exponential in order to see how the IDS performance is affected by a more realistic behaviour network environment.

5 Evaluation Metrics

In order to evaluate the performance of the ML-model, we use evaluation metrics that are derived from a Confusion matrix (see Figure 12). The matrix shows the number of accurate and inaccurate predictions made by the model, by comparing predicted and actual labels. Based on these values, we calculate evaluation metrics such as recall, accuracy, precision and area under the Receiver Operating Characteristic (ROC) curve to evaluate the performance of the model.

![Confusion Matrix](image.png)

Figure 12: Confusion Matrix

5.1 Recall

The first metric used is recall, which is defined as the ratio between true positives (i.e., accurately predicted attacks) and total positives (total no. of attack instances in the dataset), as shown in Equation 1.

\[
Recall = \frac{TruePositives}{TruePositives + FalseNegatives}
\]

\[
= \frac{TruePositives}{TotalPositives}
\]  

Recall is a useful metric in situations where the cost of a false negative (i.e., predicted no attack but was, in fact, attack) is higher than the cost of a false positive (i.e., predicted attack but was no attack), as is often the case in IDSs. In an IDS, missing an attack (false negative) can have severe consequences, while incorrectly identifying a safe network as under attack (false positive) may be less costly.
5.2 Accuracy and Precision

Accuracy and precision are other metrics that are used to measure the performance of the ML-model. Accuracy is a measurement of the proportion of correct predictions out of the total predictions, see Equation 2.

\[
\text{Accuracy} = \frac{\text{TruePositives} + \text{TrueNegatives}}{\text{CorrectPredictions}} = \frac{\text{TruePositives} + \text{TrueNegatives}}{\text{TruePos.} + \text{TrueNeg.} + \text{FalsePos.} + \text{FalseNeg.}}
\]

(2)

Accuracy is a metric that provides an overall indication of the correctness of the model’s predictions. It considers the performance of the model in detecting both attacks and safe networks.

**Precision** measures the proportion of true positives (accurately predicted attacks) out of the total predicted positives (predicted attacks), see Equation 3.

\[
\text{Precision} = \frac{\text{TruePositives}}{\text{PredictedPositives}} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalsePositives}}
\]

(3)

Precision is an important metric in cases where the cost of a false positive is high, as opposed to recall which considers false negatives. In the context of IDSs, high precision implies that the model accurately predicts the absence of attacks well. However, precision should not be the main focus when evaluating IDSs since the cost of false negatives tends to be much higher than the cost of false positives.

5.3 ROC

A Receiver Operating Characteristic (ROC) curve is a graphical plot that maps a model’s true positive rate (TPR) against its false positive rate (FPR). It is a tool for evaluating the performance of machine learning algorithms for binary classification tasks.

The TPR, or recall, is a measure of positive instances (attacks) in the dataset that are accurately classified as positive by the model. The FPR measures the proportion of negative instances (no attack) that are incorrectly classified as positive. A perfect classifier would have a 100 % TPR and 0 % FPR, and the ROC curve would pass through the upper left corner, whereas a classifier whose predictions are completely random would have a TPR and FPR of 50 % and the ROC curve would follow a diagonal line.

AUC stands for Area Under the Curve and measures the area under the ROC curve. An AUC score close to 1 indicates perfect performance, while an AUC of 0.5 indicates that the ROC curve is diagonal and the classifier is no better than random guessing.
6 Results

We have studied the impact of limited observability on an ML-based IDS using the DNN algorithm for the classification task. Additionally, we have implemented more realistic network simulations composed of Poisson process message transmission, as opposed to periodic message transmission in order to assess if the more realistic networks made any impact on the IDS regarding its performance.

The IDS relies on analyzing the network traffic in order to detect malicious behavior. With limited observability of network activities, it could lead to a significant loss of information about the network that impacts the IDS ability to identify attacks correctly. Therefore, it is important to study the impact of limited observability on an IDS in the context of isolation attacks and develop techniques to enhance observability on IDSs for RPL-based IoT networks. Additionally, it is important to study the impact of the IDS ability to detect attacks in more realistic networks since it introduces more randomness to the message transmissions and could make the network traffic less predictable for the IDS. By studying the impact of the more realistic networks, we can assess how the IDS handle the more complex and random nature of real-world networks and identify strengths and weaknesses of the IDS in real-world scenarios, which could help in optimizing and improving the IDS.

The model is trained on normal scenarios and basic attacks and tested on the attack variations On/Off and Decreasing rank. The On/Off attack has multiple variations in which the attacker pauses and resumes the attack at different intervals ranging from 10 to 30 minutes. Similarly, the Decreasing rank attack has multiple variations in which the attacker advertises a high rank value and gradually decreases it at different rates ranging from 1/min to 32/min. The model is tested on these attack variations for networks consisting of 5, 10, 15 and 20 nodes respectively.

Figure 13 shows the ROC curves and AUC for the model trained and tested on simulations where node messages were sent periodically. The attack variations involving two attackers for On/Off and Decreasing rank are not displayed, but the results are similar to those of the corresponding number of nodes with one attacker. We can see that all attack variations for On/Off and Decreasing rank have a similar ROC curve with poor detection for small networks, and better detection as the network grows larger. We can also see that the ROC curves are more diagonal and AUC closer to 0.5 for the On/Off variations compared to the Decreasing rank variations.

Figure 14 displays the precision, recall, and accuracy for the attack variations involving periodic transmissions. We can see that the precision is high, close to 1.0, for all scenarios and the performance score is generally higher for the Decreasing rank variation compared to the On/Off variation. Additionally, recall and accuracy appears to strongly correlate since the shapes of the graphs are closely parallel or almost identical.

The results for the model being trained and tested on attacks where messages from nodes modeled a Poisson process, the more realistic networks, can be...
seen in figures 15 and 16. The ROC curves and AUC scores follow the same patterns as in the periodic case, where the AUC scores are lower for the On/Off variations compared to the Decreasing rank variations. The AUC score also seem to increase proportionally as the number of nodes increases, as in the periodic case. The Poisson simulations follow the same patterns as the periodic simulations for precision, recall and accuracy.

Table 4 presents the average accuracy, recall and AUC scores for the attack variations. However, it is important to note that the results for accuracy and recall were obtained from figures 14 and 16 and therefore might not be exact but rather estimations based on the figures. As the precision is close to 1.0 for all cases, it is not included in the table. We can see that all scores are higher for the decreasing rank variation, while the periodic simulations seem to have a slight increase in average accuracy and recall. The average AUC however is higher for the Poisson simulations.

<table>
<thead>
<tr>
<th></th>
<th>On/Off attack</th>
<th>Periodic</th>
<th>Poisson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.488</td>
<td>0.440</td>
<td>0.421</td>
</tr>
<tr>
<td>Recall</td>
<td>0.474</td>
<td>0.633</td>
<td>0.641</td>
</tr>
<tr>
<td>AUC</td>
<td>0.633</td>
<td>0.440</td>
<td>0.641</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Decreasing rank attack</th>
<th>Periodic</th>
<th>Poisson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.796</td>
<td>0.779</td>
<td>0.779</td>
</tr>
<tr>
<td>Recall</td>
<td>0.777</td>
<td>0.836</td>
<td>0.863</td>
</tr>
<tr>
<td>AUC</td>
<td>0.836</td>
<td>0.779</td>
<td>0.863</td>
</tr>
</tbody>
</table>

Table 4: Average performance

Overall, the results show that with limited observability, the IDS trained on the basic attack could identify the Decreasing rank variation and basic attack well, while it had more trouble detecting the On/off variation. Limited observability seemed to impact the IDS greatly for the On/Off variation, and less for the Decreasing rank variation. When trained and tested on the more realistic networks, there was generally little to no impact on the performance compared to the periodic simulations. Additionally, the IDS performed better at detecting attacks in larger networks, most likely because there are more messages that circulates in those networks and the IDS either received more data which could be analyzed or could more easily identify the absence of messages.
Figure 13: Periodic ROC and AUC

(a) 5 nodes On/Off variation  
(b) 5 nodes Decreasing variation  

(c) 10 nodes On/Off variation  
(d) 10 nodes Decreasing variation  

(e) 15 nodes On/Off variation  
(f) 15 nodes Decreasing variation
Figure 13: Periodic ROC and AUC

(g) 20 nodes On/Off variation
(h) 20 nodes Decreasing variation

Figure 14: Periodic precision, recall and accuracy

(a) Performance 5 nodes On/Off
(b) Performance 5 nodes Decreasing
(c) Performance 10 nodes On/Off
(d) Performance 10 nodes Decreasing
Figure 14: Periodic precision, recall and accuracy

(e) Performance 15 nodes On/Off  
(f) Performance 15 nodes Decreasing  
(g) Performance 20 nodes On/Off  
(h) Performance 20 nodes Decreasing
Figure 15: Poisson ROC and AUC

(a) 5 nodes On/Off variation
(b) 5 nodes Decreasing variation
(c) 10 nodes On/Off variation
(d) 10 nodes Decreasing variation
(e) 15 nodes On/Off variation
(f) 15 nodes Decreasing variation
Figure 15: Poisson ROC and AUC

(g) 20 nodes On/Off variation
(h) 20 nodes Decreasing variation

Figure 16: Poisson precision, recall and accuracy

(a) Performance 5 nodes On/Off
(b) Performance 5 nodes Decreasing
(c) Performance 10 nodes On/Off
(d) Performance 10 nodes Decreasing
Figure 16: Poisson precision, recall and accuracy

(e) Performance 15 nodes On/Off
(f) Performance 15 nodes Decreasing

(g) Performance 20 nodes On/Off
(h) Performance 20 nodes Decreasing
7 Discussion

The model demonstrated a high precision close to 1.0, meaning that it could identify safe networks well. However, this result should be interpreted with caution since the model was not tested on pure no-attack scenarios, but rather only on attack scenarios. During the simulations, the networks were only in a safe state for the first 15 minutes before the attack started. As a result, the model was only tested on no-attack scenarios of a short amount of time, leading to a high precision score due to a lack of test data for no-attack scenarios.

The high performance achieved when tested on the basic attack is because the model was originally trained on this attack. Therefore, it is natural that when tested on the same attack as the training session, the model would perform better at recognizing it compared to the attack variations.

Generally the model achieved better performance when tested on the decreasing rank attack in comparison to the on/off attack. Additionally, the model achieved higher average accuracy and recall scores for the simulations composed of periodic transmissions, whereas simulations where messages modeled a Poisson process resulted in a higher average AUC score. However, the difference between the results for the periodic and Poisson simulations was significantly small for most cases. Therefore, the model did not show a clear indication of which transmission rate it could predict attacks best for since there is no clear patterns of such a preference in the result data. For some cases, the performance was better for the Poisson simulations, while the opposite was true in other cases. Hence, it is difficult to say which transmission rate it preferred. However, one conclusion to be made is that there was no clear difference in performance between the two cases, indicating that the different transmission rates did not significantly affect the model’s performance.

In the case of periodic messages, the IDS expect messages at a static rate and therefore could easily identify expected messages that are not observed, which in turn could be an indication of an attack. When introducing some randomness to the transmission of messages as in the Poisson process simulations, the expectation of messages are no longer static as in the periodic case, hence making it more difficult to identify the absence of messages. This could account for the slight increase in accuracy and recall for the periodic simulations. Nevertheless, the model demonstrated good performance in both the realistic and periodic scenarios, indicating its ability to classify attacks despite the introduction of randomness to messages.

The IDS in previous work with access to full observability of network traffic demonstrated a great performance, achieving an average recall score of 0.92 for both the decreasing rank and the on/off attack variations. In other words, it detected around 92% of all attacks. In contrast, the IDS in this thesis with limited observability, only detected around 45% of the on/off variations and 78% of the decreasing rank variations. These results highlight the critical role of observability in the effectiveness of the IDS, as limited observability significantly affected its ability to detect attacks. However, it is important to note that the differences in data preparation methods and classifiers used in this thesis and
previous work may contribute to the differences in results.
8 Conclusions and Future work

In this thesis, we have simulated attacks on RPL-based IoT networks and generated datasets which we used to train an ML-based IDS. Specifically we have studied the case when attacks disconnect the network, the blackhole attack, and considered the fact that limited observability of network activities for an IDS may impact its ability to detect attacks. We trained a DNN classifier on observable data and studied the impact of limited observability. Additionally we considered a data preparation method that allowed the IDS to detect an attack online based on the most recent data. Furthermore we implemented a more realistic way of transmitting messages where we considered the network traffic to be more realistic if the message transmission of nodes modeled a Poisson process. Hence, we considered two ways of sending messages, the first way as periodical transmission as in previous work and the second way as a Poisson process. This allowed us to establish a point of reference in which we could compare the results for an IDS given datasets from simulations where messages were sent periodically and as a Poisson process respectively.

The model was trained on a conventional blackhole attack, which is referred to as the basic attack, along with data from networks with no attack. The model was then tested on variations of the blackhole attack: on/off and decreasing rank attacks. Our results showed that the performance of the IDS was not significantly affected by the more realistic network, but observability was found to be a critical factor for detecting attacks.

The IDS from previous work used a retrospective approach for detecting attacks and had access to full observability of network traffic, resulting in the ability to classify attacks almost perfectly. In contrast, our IDS with limited observability using an online approach only detected around 50% of the On/Off attacks and 80% of the Decreasing rank attacks. This showcases that observability of network traffic is a critical part for detecting malicious behavior by an IDS. The retrospective approach used in previous work could also be a factor in the differences in the IDS performance, since the attacks are detected based on historical data, while our online approach involves real-time detection based on most recent data. Analyzing historical data could help the IDS to identify patterns and detect attacks, although it has the disadvantage that it does not detect attacks in real-time.

Overall, the study found that the model performed similarly for the periodic and Poisson process transmissions, and introducing a more realistic network did not impact the IDS performance to a great deal. Observability proved to be a critical factor for detecting attacks, while using an online approach as opposed to a retrospective approach could also be a factor.

The simulations conducted in this thesis were limited to a fixed setting, with no mobility, arrival or departure of nodes. In wireless networks however, devices can move around, go out of range, run out of battery, or join the network. A proposal for future work is to introduce some mobility, disconnection or arrival of nodes in order make the simulations more realistic.
Another proposal for future work is to explore other type of attacks that target the network topology in the same way as blackhole attacks, such as sinkhole or wormhole attack, in addition to implementing more variations of the blackhole attack. This would make the IDS even more generalized and provide it with more insight and knowledge of potential threats.
References


