SDN-WISE Anti-Attack

An SDN framework for securing IoT networks using machine learning

Master Thesis

Severin Zumbrunn

Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

January 2019

Anhang 1 / Umschlagseite

SDN-WISE Anti-Attack

An SDN framework for securing IoT networks using machine learning

Masterarbeit der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von
Severin Zumbrunn

Januar 2019

Leiter der Arbeit
Prof. Dr. T. Braun

"IoT without Security equals the Internet of Threats"

— Stephane Nappo

Abstract

In this work we propose a framework for automatic attack recognition, attacker identification and attack mitigation based on anomaly detection in combination with machine learning classification and Software Defined Networking. This framework was designed to find potential attacks based on residual analysis on network characteristics like topology and traffic statistics. The main goal is to have a framework which is not specifically designed for a certain attack, but rather to provide detection and mitigation of novel attacks in a general manner. This work provides a performance analysis and a conceptual insight to the framework. Results indicate, that the anomaly based approach for attack detection in combination with an appropriate mitigation algorithm can detect and mitigate attacks efficiently.

Prof. Dr. Torsten Braun, Communication and Distributed Systems, Institute of Computer Science, University of Bern, Supervisor

Dr. Zhongliang Zhao, Communication and Distributed Systems, Institute of Computer Science, University of Bern, Assistant

Cand. Dr. Jakob Schaerer, Communication and Distributed Systems, Institute of Computer Science, University of Bern, Assistant

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Objectives . . . 3
- 1.3 Thesis Contribution . . . 3
- 1.4 Thesis Structure . . . 4

2 Background and Literature **5**
- 2.1 Related Work on Security Issues in Wireless Networks . . . 5
- 2.2 Resource Constrained Networks . . . 7
  - 2.2.1 Internet of Things . . . 7
  - 2.2.2 Wireless Sensor Networks . . . 7
  - 2.2.3 Wireless Sensor and Actor Networks . . . 7
  - 2.2.4 Routing Protocol for Lossy Networks - RPL . . . 8
- 2.3 Software Defined Networking . . . 9
  - 2.3.1 SDN-WISE . . . 9
  - 2.3.2 ONOS . . . 9
  - 2.3.3 SDNWisebed . . . 10
- 2.4 COOJA . . . 10
- 2.5 WSN Security . . . 10
  - 2.5.1 Node-to-node communication . . . 11
  - 2.5.2 Attacks . . . 11
    - 2.5.2.1 (Distributed) Denial of Service (DDoS) . . . 12
    - 2.5.2.2 Node Replication Attack . . . 13
  - 2.5.3 RADAR: Residual Analysis for Anomaly Detection in Attributed netwoRks . . . 14
  - 2.5.4 Attack Mitigation . . . 15

3 SDN-enhanced Attack Mitigation in WSNs **17**
- 3.1 Introduction to Anomaly Detection . . . 17
  - 3.1.1 Feature Selection . . . 18
    - 3.1.1.1 DoS Attack Features . . . 18
    - 3.1.1.2 Node Replication Attack Features . . . 19
- 3.2 Attack Detection Framework . . . 20
- 3.3 Attack Mitigation . . . 21
  - 3.3.1 Attack Recovery and Mitigation Algorithm - ARMA . . . 23
  - 3.3.2 Caching . . . 24

4 Experiments **25**
- 4.1 DoS Simulation . . . 25

4.1.1 Overview . . . 25
4.1.2 Dataset . . . 26
4.1.3 Results . . . 26
4.1.4 Threshold-based DoS Detection . . . 28
4.2 Node Replication Attack Simulation . . . 30
4.2.1 Overview . . . 30
4.2.2 Results . . . 30
4.3 Multi-attack Classification . . . 31

5 SDN-WISE Framework Contribution **35**
5.1 Bugfix to OpenPath Forwarding . . . 35
5.1.1 Packet Masquerading . . . 36
5.1.2 Bugfix to current SDN-WISE implementation . . . 36
5.1.3 Bugfix to OpenPath Generator . . . 37
5.2 Topology Display of SDN-WISE Dynamic Routing Packet Processor . . . 40

6 Discussion **41**
6.1 Summary of Results . . . 41
6.2 Conclusion . . . 42
6.3 Future Work . . . 42

Chapter 1:

Introduction

Today, as the hardware integration size is halved every year, more and more devices in daily use become connected to the Internet. This phenomenon is well-known under the term Internet of Things (IoT). In the beginning of the Internet, engineers thought that the Internet will have a thousand up to ten-thousands of devices connected. Instead today there are billions of devices communicating with each other which leads to emerging problems with manageability, controllability and, especially, security. The growth in popularity of IoT let many issues and threats arise. A prominent example of such issues are Distributed Denial of Service (DDoS) attacks. As IoT devices often lack basic security measures, botnets consisting of millions of devices can easily be built up. To encounter this problem, a variety of approaches have been proposed recently that try to secure the IoT devices and the network as a whole. These mechanisms focus mostly on securing the different layers in the network stack and are implemented directly on the devices themselves. Examples are pre-shared key authentication and secure transport layer protocols like TLS. While these implementations only work for devices that possess the required non-negligible resources, it is unimaginable for most low-powered IoT devices like sensors. For that reason, other mechanisms are required that do not use great amounts of energy or computational power.

In 2005, Stanford University proposed a new paradigm called Software Defined Networking (SDN) which splits the network architecture into two virtual planes (data and control plane). While the control plane is used for establishing routes and exchanging report messages with the controller, the data plane transports the actual data. This not only improves the manageability of networks, it also introduces a completely new toolset to provide network security. With SDN, system administrators are given complete knowledge about the topology and traffic patterns of a network. Another promising field for applying SDN are Wireless Sensor and Actuator Networks (WSAN) which consist of sensors and actors that are often low-powered or have limited computational power available. Additionally, the complexity for routing is moved from the sensors to the central controller, which can save the scarce resources of the distributed nodes. Furthermore, WSNs often require point to point communication with very low delay. In this case, SDN provides full-flexibility and dynamic route management, which has been proofed to outperform conventional distributed routing algorithms (Schaerer 2018).

Recently, it has been shown that SDN is far more than just a way of routing packets in a network. Although, one of the foundations is routing, security applications can be easily developed using SDN. Mostly, because such systems have already abstracted network characteristics available, such as transmission and reception packet count per link. Using the network topology for example, it can be much simpler to detect attackers or error patterns. In combination with machine learning, this can be used to automatically recognize and mitigate such threats.

1.1 Motivation

Securing IoT networks has become a very popular and continuously growing field of research recently. There have been many studies carried out, focusing on the detection of DDoS attacks in networks by using Deep Packet Inspection (DPI) in combination with Machine Learning. In these approaches, the security algorithms are deployed to gateways like routers or firewalls that handle the traffic between the Internet and the consumer IoT devices. The basic assumption is that all packets to or from the Internet have to pass these gateways. The values of these packets are then transformed into features that can be used by appropriate classification algorithms like KDTree, SVM or Random Forest. The accuracy for detecting DDoS attacks based on those algorithms goes up to very high 99.9% (Doshi et al. 2018). Nevertheless, it is important to note that these studies focus on a single attack for consumer IoT devices and are specialized for this task only. The high accuracy can also be explained from the fact that they specialized on recognizing only DDoS attacks and even more that they are focusing on the attack rather than the attacker itself.

Common DDoS attacks, with billions of requests within a tiny fraction of time, are seldomly used against IoT devices as targets but more that the IoT devices become attackers. This comes from the fact that, IoT devices are often not sufficiently secured. Frequently, default credentials are left untouched or they do not even contain basic security considerations so that they can be hacked easily. Regardless of this fact, DDoS attacks are omnipresent and can be simulated well and so this thesis includes an experiment with DDoS for comparison with other papers. To this date, current research on IoT and especially WSN attacks focuses mainly on the detection of DDoS attacks using both centralized and decentralized approaches. However, no studies can be found on the detection of WSN attacks using SDN and machine learning. Available papers consider the detection of the attack only but not the detection of the attacking node. Therefore, the focus of this thesis is to study different security issues and attacks in WSNs using SDN and how they can be detected, prevented or mitigated using machine learning and more especially how attackers can be identified.

With IoT several characteristics are typical: limited resources, large number of nodes, heterogeneity of nodes, error prone communication, and hostile environments. All these issues increase the risk for potential security leaks and threats and need to be addressed in order to improve the overall security of IoT networks. This becomes even more clear when having a look at a particular subgroup of IoT, the Wireless Mesh Networks (WMNs).

In Siddiqui (2007) the general problem of mesh network security is described as "[..] WMN has features such as an open medium, dynamic changing topology, and the lack

of a centralized monitoring and management point, many of the intrusion detection techniques developed for a fixed wired network are not applicable in WMNs." It is therefore a logical consequence to add a central controller for management and security purposes that is able to control all network flows and is aware of ongoing traffic. Furthermore, it should have enough computational power to apply machine learning on the information it is collecting from the network to detect attacks and take countermeasures against potential attackers.

1.2 Objectives

The goal of this master thesis is to study and identify potential WSN security attacks, simulate, detect and mitigate them by using SDN-empowered security. Additionally, machine learning should be considered for classifying attacks and identifying attackers. This goal can be split into several sub goals:

- Analyze state-of-the-art approaches for attack and attacker detection.
- Find characteristic WSN attacks. Implement and simulate some of the identified attacks in SDN-WISE using COOJA.
- Extract features from the generated traffic datasets to train machine learning classifiers and apply them to recognize attacks and if possible, to detect the attacker as well
- Find ways to mitigate these attacks to prevent complete network loss

This thesis tries to answer the following research questions:

- How can WSNs be protected by applying the SDN paradigm having a centralized controller?
- Can attacks against WSNs be detected using SDN?
- How can attacks be recognized and what are useful features to detect attacks?
- How is it possible to identify an attacker during an attack?
- What possibilities can prevent an attacker from taking down or compromising a network?

1.3 Thesis Contribution

This thesis proposes a monitoring application for the SDNWisebed framework that is able to recognize attacks, identify attackers and mitigate the attacks by penalizing connections to attackers, e.g. push back. It provides a generalized approach that is flexible and not constraint to a fixed network size, or special network architecture despite other solutions proposed in the past.

Furthermore, it investigates how the SDN paradigm can be used in lossy wireless sensor networks, to enhance security and enable the mitigation of attacks in such networks. And it shows

1.4 Thesis Structure

This thesis is structured into six basic chapters. In this Chapter 1 a short overview over the topic is provided and the goal of this thesis is explained. Furthermore, in this section the structure of this thesis is described.

In Chapter 2, a literature study on related work is given including explanations for the most important terms used in this thesis. It also provides insight for different attacks for WSN and how they are commonly prevented.

Chapter 3 proposes a framework for attack mitigation in WSNs using SDN. The chapter discusses the machine learning approach, which is used to detect two WSN attacks, Denial of Service and Node Replication. It explains the selected features for the classification and proposes a framework for attack recovery and mitigation for DoS attacks.

In Chapter 4, the experiments conducted in this thesis and their results are shown. The experiments show the advantage of an SDN-based approach for attack detection in WSN. Additionally, this chapter provides measurements for multi-attack classification.

Chapter 5 highlights additional work on the SDN-WISE framework that has been done in this thesis. These side products improve the quality of future experiments with SDN-WISE and provide more advanced functionality for the framework.

In Chapter 6, the results of the experiments are discussed and evaluated. Furthermore, a summary of the work of this thesis is provided and a brief outlook for future work is given.

Chapter 2:

Background and Literature

This chapter gives an overview on the related work, the technologies and paradigms that are used and are essential for the understanding of this thesis. The first part features an introduction to give an overview on recent studies in the field of WSN and IoT Security. In the latter, different terms are introduced and explained briefly.

2.1 Related Work on Security Issues in Wireless Networks

Within fifteen years of development on WSNs, many papers have been published, focusing on the security of Wireless Sensor Networks. Although, with the current hype for the Internet of Things it is still considered a young and fresh topic.

Back in 2003, Karlof & Wagner (2003) took a survey over numerous routing protocols and highlighted their security issues. In the end of their paper, they proposed several countermeasures on the lower layers during the design stage of a protocol. But they found especially the wormhole and sinkhole attack as a significant challenge to the application of security countermeasures when a protocol has already been implemented. They concluded that security has to be an integral part of protocol design from the beginning.

Padmavathi & Shanmugapriya (2009) studied the arising problems for sensor networks in the way those networks are deployed in unattended environments, what makes them vulnerable to a variety of potential attacks. They made a survey on various network attacks that can be taken out and then listed some attacks specifically designed for WSNs.

Siddiqui (2007) studied security issues in wireless mesh networks (WMNs) and identified the lack of centralized Intrusion Detection Systems as one of the main issues in WMN security. Furthermore, they concluded that WMNs can be very reliable due to the massive amount of individual wireless transceivers, making the system error tolerant and resilient. With their work they highlight the great potential of a centralized approach to control a network based on its distributed knowledge.

Airehrour et al. (2018) proposed a framework called SecTrust-RPL which extends the Routing Protocol for Low-Power and Lossy Networks (RPL) with additional security functions. It was specifically designed to protect RPL against the prominent Rank and Sybil attacks. It is particularly interesting for this thesis, as in SDN-Wise the control packets are routed by RPL which lays the foundation for the complete SDN approach.

Wani & Revathi (2018) used SDN for an Intrusion Detection System (IDS) to recognize attacks from the outside to IoT networks. By analyzing ingress traffic to IoT networks with a back-propagation neural network they achieved accuracies of up to 99%.

In 2017, Bhunia & Gurusamy (2017) first applied SDN to IoT networks for security reasons. They propose a framework called SoftThings with attack detection, where SDN is used to control the traffic coming from and to IoT devices. It consists of a master SDN controller, several cluster SDN controller and the end devices. A machine learning module resides in the master SDN controller that is prefilled with a training set of various known TCP attacks. Their approach is based on already known and common attacks and they focus on wired instead of wireless networks. In their work, they highlight a literature study made by Sood et al. (2016) which was made explicitly to show opportunities and risks in wireless sensor networks that are driven by SDN. They state that DDoS could be a major issue in wireless sensor networks due to the scarce resources and the lack of security. They see the possibility of SDN to do attack mitigation and prevention in terms of constant analysis of the network characteristics. But they also show issues coming along when bringing SDN to WSNs, e.g. the inability of SDN to do Deep Packet Inspection (DPI) as described by Sood et al. (2016) "[..] Deep Packet Inspection (DPI) is unfortunately not supported in standard OpenFlow because currently defined match fields in order to evaluate packet are limited to the packet header only.". In the end, they conclude that there is a lack of research in the specific field of security when applying SDN to WSNs. In their work, they conducted a single experiment using Support Vector Machines for classification. However, SVMs are rather slow what can lead to problems especially in attack mitigation where detection speed is crucial.

Zhu et al. (2007), Conti et al. (2007) and Parno et al. (2005) proposed different distributed algorithms for detecting node replication attacks. All of the proposed algorithms show accuracies for detection between 60% and up to 94.5% in special cases, what clearly indicates room for improvement. These distributed approaches consume a lot of energy due to the overhead that is additionally created by the detection mechanisms. The use of a centralized approach thereof could not only save energy but also boost the performance for detecting attacks.

In the paper of Grover et al. (2011), a machine learning detector for multiple attacks in VANETs, such as *packet suppresion*, *packet replay*, *packet detention*, *position forging* and *identity spoofing aka. node replication* attacks was proposed. It is the only paper that could be found where they tried to detect node replication attacks and especially attackers with a centralized approach in combination with machine learning.

Studying this related work showed, that often countermeasures and detectors respectively are specifically designed for a single attack, but there is no general attack detection or prevention mechanism available. As WSN imposes many different attack threats it is therefore, almost impossible to provide protection against all potential attacks at

once. But previous research showed that it could be beneficial to include distributed knowledge (Siddiqui 2007) that is available within a network. Also, SDN on WSN shows much potential (Bhunia & Gurusamy 2017) but also has its drawbacks that need to be addressed. Due to the missing Deep Packet Inspection capability, there is the need for constant monitoring of the traffic flows and the evaluation of statistics that are provided by the switches (Sood et al. 2016). To conclude the related work study: SDN could be a new method for securing WSNs which is not focusing on a particular attack and how to prevent it but rather monitor the network state and traffic flows and dynamically mitigate the attack to minimize the overall harm and impact of such.

2.2 Resource Constrained Networks

2.2.1 Internet of Things

The Internet of Things (IoT) is an inflationary used term in Computer Science and describes the recent phenomenon of connecting everyday things to the Internet. IoT describes devices that can be wired or are wireless. Most of the time such devices have limited resources available like limited computational power, limited energy or limited memory and are specialized for a given task, e.g. smart fridges ordering food that is off. That is why most of these devices are not properly secured. Nevertheless, these devices have a full network stack, like TCP or UDP and are therefore frequently threatened by attacks from hackers, worms or viruses. Consequently, it is no wonder, that in some of the biggest DDoS attacks that have been carried out in history thousands of refrigerators, cameras and other typical IoT devices were involved.

2.2.2 Wireless Sensor Networks

WSNs are best described as networks built from wireless IoT devices that are using low-bandwidth connections. A variety of network forms for WSNs exist, but the most frequently used is the multi-hop sensor network where every node is sensing and/or forwarding packets. As Karlof & Wagner (2003) describe, it consists of a sink node that acts as a gateway between the low-bandwidth connection and the high-bandwidth connection. Usually, the sink node is neither battery nor computational power constraint, as it can be a full-fledged computer. A WSN can have multiple sinks.

2.2.3 Wireless Sensor and Actor Networks

In contrast to WSNs, WSNs are the super group of WSNs that additionally include actors which impose new traffic patterns. From the WSNs network perspective, every node can transmit data to every other node, not only to some distinguished sink nodes. Therefore, node-to-node (n2n) communication can be frequent. Today, this is better known under the term Machine-to-Machine communication (M2M), but in this case, the mixture of IoT devices, wired or wireless, low- or high-bandwidth is not constrained. That is why in this thesis the term n2n refers to node-to-node communication in Wireless

Sensor and Actor Networks where at least some nodes are assumed to be energy and resource constrained.

In WSNs, traffic is often made from rapid streams (e.g. video streams) or low-bandwidth communication from sensors. This mixture is very typical for WSNs as there are sensors connected which report information to actor nodes and establish control loops between them. Therefore, we often find node to node communication that is latency sensitive. Additionally, constant flows of data to controller nodes or sinks can be found due to Internet connectivity.

2.2.4 Routing Protocol for Lossy Networks - RPL

To overcome the issues in resource constrained networks, lightweight routing protocols become necessary. One famous example routing protocol is RPL.

In this protocol, every node that receives a so called DIO message, rebroadcasts it by incrementing the current degree by one. Afterwards, a routing tree is built up and nodes can start transmitting packets to a sink. At this stage, more problems show up immediately: What if a node rebroadcasts with false degree? What if a central node does not rebroadcast at all? A good analysis of these attacks and ways to prevent them were published by Airehrour et al. (2018).

After the bootstrap phase of building up the network, in order to keep the network fresh and up to date, we can then simply rebuild the RPL tree with a higher version number. Additionally, RPL provides ways to handle adding and removing nodes during runtime by DIS messages and parent selection lists. The parent selection is done by a so-called Objective Function that calculates costs for every parent node in the list. The node with the lowest costs is chosen afterwards and used for forwarding packets.

Although, this routing mechanism allows us to have node to sink communication, we are required to exchange packets with other nodes than just the sink. Therefore, an additional routing mechanism is needed. In RPL it is proposed to maintain downward routes using routing tables. This obliges all nodes to inform their parents about their children what results in having routing tables that grow with every step upwards in the tree. To not have to store all the child nodes, RPL supports a non-storing mode, where the information is just stored temporarily and then passed upwards to the sink node. In this mode, only the sink node keeps track of the children in the network and every packet has to travel through the sink and can then be routed downwards by using a node address stack that contains the next hops. Nodes receiving this packet simply pop the next hop address from the stack and forward the packet to this next address. A clear advantage of this routing protocol configuration is that it is very simple for child nodes, does not require a lot of memory but it is rather complicated for the sink node and it requires all packets to flow through the sink node what results in a bottleneck. However, with this approach we can fulfill basic routing requirements to support node to node communication.

2.3 Software Defined Networking

SDN describes the splitting of the network stack into two planes, the control and the data plane, in order to make the transmission of user data more transparent and to manage networks centrally. This opens up completely new possibilities of routing and is especially interesting for large networks, where fast and flexible routing between nodes is required. In contrast to common networks, where distributed routers make their own decisions about routing packets, with SDN a central controller calculates the best path through the network. These routing decisions are made based on network information gathered with report packets. Although, this creates a dependency and a potential security issue, however, it can improve security due to the distributed knowledge of the network nodes.

2.3.1 SDN-WISE

SDN-WISE is a framework proposed by Galluccio et al. (2015) that combines SDN and WSN. It consists of a network stack based on OpenFlow for sensor nodes, that have scarce resources available and few packets should be transmitted therefore. It features three components: nodes, sinks and visors. The latter is used to control the network and is an instance of ONOS, described in Subsection 2.3.2. The procedure for bootstrapping the network is defined as follows: First, to establish the control plane, a route is built up using the Routing Protocol for Low-Power and Lossy Networks (RPL) in non-storing mode which enables nodes to transmit REPORT packets at a regular interval via the sink to the visor. In these REPORT packets, the nodes announce their current state, their neighbours with its connection quality and additional sensor information for the current period. This helps the controller to be aware of the ongoing traffic in the network and the availability or absence of nodes. It can also be used for monitoring purposes and network control. Afterwards, when a node requires a route to another node, a route REQUEST packet is sent to the SDN controller which then eventually calculates the best route and responds to the requesting node with an OpenPath packet. This packet contains the path which should be installed. When the requesting node has been reached, it forwards the OpenPath packet on the designated path and all nodes receiving this packet subsequentially establish a route by installing appropriate entries in their FlowTables that connect the source with the destination node. This way a flow is installed and the requesting node can start transmitting packets to the target destination.

2.3.2 ONOS

With SDN-WISE the foundation for the communication between nodes, sink and visor is laid. But since the visor is monolithic, it is not very handy for development. Therefore, ONOS an SDN controller has been created. It comes with a complete toolset for managing the network topology, monitoring traffic, processing packets, etc. It is maintained by the Open Networking Foundation which is a non-profit operator led consortium that creates several OpenSource network applications.

2.3.3 SDNWisebed

The SDNWisebed as described by Schaerer (2018), enables researchers to carry out experiments with SDN on University of Bern's wireless sensor network testbed TARWIS (Hurni et al. 2012). It consists of up to 40 nodes that are distributed within two buildings of the Computer Science Institute. This testbed can be used for fast development of sensor network applications and it is able to deploy images built for Contiki fast and easy. The testbed was extended by the two earlier described components SDN-WISE and ONOS. Additionally, he proposed a protocol called Dynamic Traffic Aware Routing Protocol (DTARP) as an improvement for node-to-node communication in WSNs.

2.4 COOJA

COOJA is a simulator for wireless sensor networks based on Contiki. It can be used to tryout different network topologies with heterogenous devices. COOJA is fully SDN-WISE compatible and is therefore the first choice for fast simulations. It provides logs, a timeline, a topology view and the radio frequency channel display. In this thesis COOJA was used to generate datasets and develop and evaluate various attack detection and mitigation mechanisms for SDN-WISE.

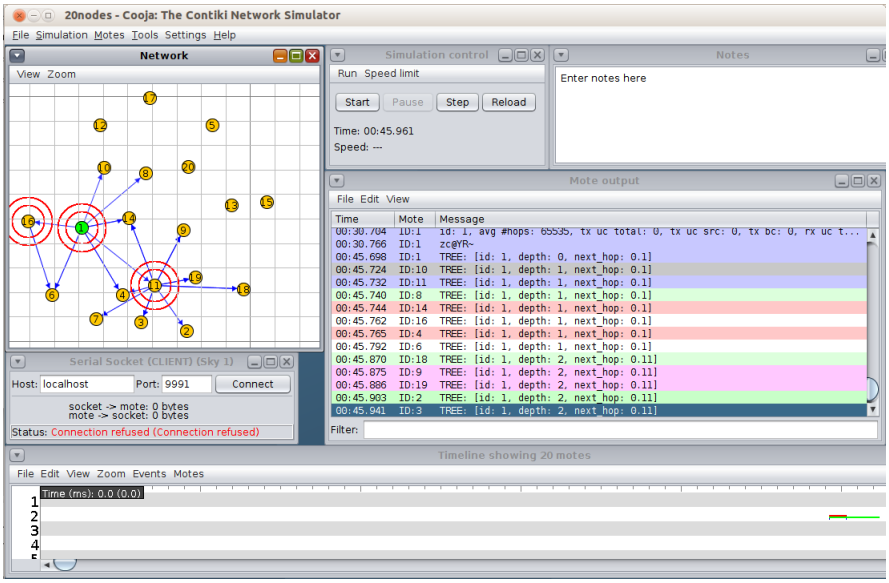


Figure 2.1: Screenshot of Cooja - The Contiki Network Simulator

2.5 WSN Security

In order to understand the security issues in a Wireless Sensor Networks, one has to understand the fundamentals of WSNs. As we have already seen, a WSN consists of a

certain number of nodes that may join or leave the network at any time. Mostly, battery powered sensor nodes with limited energy are deployed in the wild. After deployment, the nodes start to collect information about their immediate neighborhood. In this period, the first security issue arises: Whom to trust? Are all neighbors of a given node trustworthy? Only some of them? Going one step further, now we need some way of transmitting the data to some target in or even outside the network, but how do we know where to send to? This is defined by the routing protocol we use. In WSNs it is typical that a sink node initiates the creation of a route tree, e.g. by using RPL.

In RPL from the security perspective we have the problem that when the sink node fails, the complete network is lost, and nodes cannot exchange information anymore. An approach that is often seen in this case, is to have additional sink nodes. This can be achieved for example by introducing tree identifiers for every sink node and build up multiple trees for the same network. However, this creates a lot of overhead for the sake of security, which is what is not desired in WSNs.

2.5.1 Node-to-node communication

Due to the simplicity of the RPL protocol and the work of Airehrour et al. (2018) we know that the control traffic from node to sink and backwards can be secured. But how is it possible to secure direct node to node communication in WSNs efficiently? This question shows a bunch of new problems. Previously, in the case of Node-to-Sink communication we have had several tools available to secure the network traffic, e.g. with end-to-end encryption using pre-shared keys. Building a network of trust according to Djedjig et al. (2015) can help to prevent attacks. But in n2n communication using end-to-end encryption would be too heavy for simple sensor nodes. In the extreme, this would mean that every node has a key for every other node in the network. Or if the keys are not pre-shared it would require some form of RSA calculation which is clearly infeasible for sensor nodes due to the lack of sufficient computational power and the permanent energy scarcity.

Not only protecting the actual data transfer, but also the establishment and freshness of routes generates a dozen of new issues. Traffic overhead and high resource usage would lead to early battery exhaustion which is unfeasible for low-power sensor networks. To limit the effects of producing overhead for routing causes, Schaerer et al. (2018) proposed the introduction of an SDN-based routing algorithm called DTARP, the Dynamic Traffic Aware Routing Protocol for WSNs. They showed that WSNs benefit from the addition of SDN in terms of manageability, lower congestion and energy saving. In combination with their findings and the central availability of network statistics, SDN could be promising for attack detection and mitigation.

2.5.2 Attacks

As previously mentioned in Section 2.1, WSNs are threatened by various attack types that basically come from the architecture of WSNs itself. WSN attacks can be grouped into active and passive attacks. While passive attacks can be prevented by appropriate security mechanisms like encryption through pre-shared keys or calculating an encryption key adhoc, active attack require additional attention. Active attacks are generally

more difficult to detect than passive attacks, as their effects are closely related to *normal* network malfunction. Therefore, an attack detector not only has to identify network characteristic changes but also has to distinguish between real attacker or node failure.

The group of active attacks can be further split into routing attacks and application attacks, based on the network layer the attack takes place. In order to verify the proposed framework in this thesis, some attacks had to be implemented. Two very important attacks against WSNs are described more in detail in this section.

2.5.2.1 (Distributed) Denial of Service (DDoS)

Denial of Service (DoS) or Distributed Denial of Service (DDoS) attacks describe a collection of attacks where an attacker tries to take down a specific target by overusing its capability of handling requests (Douligeris & Mitrokotsa 2004). This can be any service that the target provides. In DoS, the attacker tries to request data so that the target is kept busy for as long as possible. He then repeats such requests until the target cannot response to the requests anymore and eventually fails due to stack overflow or has to be removed from the network for rebooting. In DDoS, the attack is taken out by a distributed set of attacking nodes which are orchestrated by an attacker in the way of a botnet or trojan. This set of nodes then carries out the attack on the same target at the same time during a short period of time in order to avoid recognition. Such DDoS attacks became very prominent in recent history as they were used to take down websites or servers of political opposition (Czosseck & Geers 2009) in so called cyberwars.

The problem with DoS attacks is, that there is basically always a lack of protection mechanisms as servers have to provide services and that normal requests can often not be distinguished from DoS attacks as there is nothing wrong with the requests itself, rather the huge number of packets create the problem as a whole. Especially, with IoT devices that are part of botnets the fact that they use legitimate IP addresses, it is almost impossible to discriminate between DoS and normal requests. Due to that, there have only a few countermeasures been proposed that try to limit the attack effects rather than prevent them. But very often this is only possible with great costs and investments, therefore only global players with massive infrastructure can afford it, like Googles Project Shield, Akamai or CloudFlare (Bhardwaj et al. 2018).

In the paper of Bhardwaj et al. (2018) they propose a framework called ShadowNet for DoS protection that is installed on the border router of an IoT network. It transmits small statistics packets to some ShadowNet web service about the current ongoing traffic in the network. The web service is then able to recognize DoS attacks and use the ShadowNet border routers to prevent the transmission of these packets directly before they reach the Internet.

Another approach for DoS recognition is by remembering requests per node and combining the number of requests per node with the Expected Transmission Count (ETX) metric. In the study from Esposito et al. (2008) they describe ETX as a metric, specifically designed for wireless mesh networks to find the most reliable path for transmission. But in the case of SDN, it can be used as plausibility score for DoS detection. This could be done as follows:

1. Calculate the ETX value to a given node based on previous statistical values, e.g. how many times a request was made within the recent n time frames.
2. Compare the ETX value with the number of OpenPath requests to the target node received within the last time frame.
3. If the ETX value is lower than the actual number of requests within the last time frame, the request is considered as being part of a DoS attack.

Of course, the ETX values are always influenced by the environment and can therefore be unreliable for attack detection. However, one foundation pilar of security is the availability of information and so the ETX measure combines both goals, the attack detection and the search for unavailable paths.

2.5.2.2 Node Replication Attack

A node replication attack is when an attacker seeks an existing node id in a network and then connects to the network using this already existing id. By doing that, the network nodes falsely believe to communicate with only one (the victim) node, but the attacker possibly receives all information either. The main problem considering an SDN-based network is that newly created routes lead to network disruption when establishing routes, as the reported routes do not actually exist, but only if the two nodes are considered as a single individual. An example is shown in Figure 2.2.

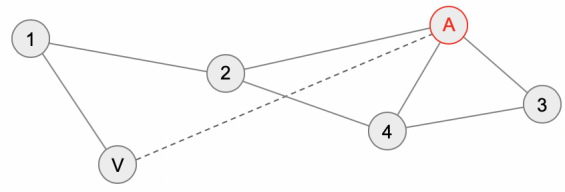


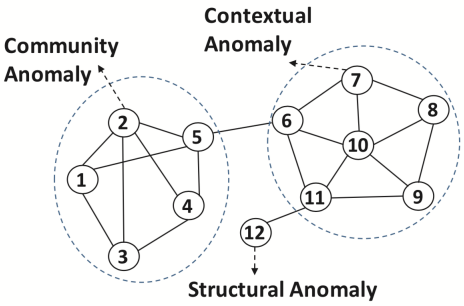
Figure 2.2: Toy example: Node Replication Attack. Attacker A and Victim V

In this example, attacker A copies the id of victim V. Node 2 may falsely believe to reach node V directly and would forward packets for V to A instead of 1. As nodes A and V do not share their network knowledge, the resulting contradiction may even lead to network disruption. Especially, in SDN networks, node replication attacks can do a lot of harm. Because the controller would falsely believe V and A to be the same and would e.g. install a route between node 1 and 3 over V, respectively A. All packets sent between the two nodes would get lost therefore. Node replication attacks are difficult to detect by distributed algorithms or require additional overhead Parno et al. (2005) what should be avoided in WSNs. Usually the attacker's goal is to disrupt the network. Therefore, he copies the victim node and moves to some other place in the network topology, so that attacker and victim are not in the same neighbourhood what maximizes routing issues and confusion because nodes do not know where to route packets to anymore. An outside observer however can help to overcome this attack by remembering the network topology and observing changes that do not seem plausible. As we will see later, SDN can be a great tool for that matter.

2.5.3 RADAR: Residual Analysis for Anomaly Detection in Attributed netwoRks

As Li et al. (2017) state "Besides, people can always develop a new type of anomaly as long as some natures of data are exploited. Therefore, it is beneficial and desirable to explore and spot anomalies in a general sense." They are possibly referring to an attacker willing to break into a network or e.g. falsify packet transmissions. They indicate with that sentence that attacks and anomalies might correlate strongly. With the fact that the previously discussed attacks, DDoS and Node Replication, are both assumed to create network anomalies in either the structure or the traffic. RADAR could help to recognize those attacks.

Li et al. (2017) propose a framework for anomaly detection that uses graph-based information to calculate anomaly scores for every node in the graph. This is done by calculating the residuals by approximation. Residual analysis bases on the relation between modeled and observed data. It describes the error that is found between these two and can therefore be used to find anomalies. As Li et al. (2017) state, "Instances with large residual errors are more likely to be anomalies, since their behaviors do not conform to the patterns of majority reference instances." RADAR uses structural data from the adjacency matrix of a graph and a matrix that holds node attributes as shown in Figure 2.3. It not only uses data directly, but it includes a community detection algorithm for attributes and topologies. This allows to have very different node types within one graph which may create highly varying traffic. But with the community detectioning part of RADAR it can handle such differencies without misclassifying such nodes as anomalies. This is specifically helpful for heterogeneous networks, like WSN and IoT.



ID	f1	f2	f3
1	2	22	48
2	11	23	48
3	3	21	47
4	4	22	46
5	3	24	49
6	12	19	51
7	11	38	55
8	10	19	56
9	13	20	49
10	14	20	49
11	11	21	53
12	12	22	52

Figure 2.3: Toy example: Topology and Attribute matrix with anomalies (Li et al. 2017)

The toy example in Figure 2.3 shows three types of anomalies that RADAR considers. *Structural Anomalies* are interpreted as the residuals of node degrees. In this example, node 12 is seen as an anomaly since it has a degree of only 1 while every other node has at least a degree of 3. *Contextual Anomalies* are defined as residuals in the node attributes. The node attributes in the toy examples are f1, f2 and f3. These have to be numerical values but their range is not constrained. In this toy example, the f2 attribute of node 7 (38) is completely different from the average f2 value of 22.6 and is therefore considered a Contextual Anomaly. The third type of anomalies, the so called *Community*

Anomalies are a combination of structural and contextual anomalies. Using appropriate algorithms, for example with the Girvan-Newman or the Minimum-cut algorithm, it is possible to find communities in a network graph. Such communities are defined as groups of nodes that share very similar attributes and are strongly interconnected. In the toy example in Figure 2.3, nodes 1, 2, 3, 4, and 5 build a community as they all share the similarity of having at least 3 edges to other members of the community. Considering the node attributes of this community, it can be seen that the f1 value of node 2 differs completely from the other nodes in that set. Therefore, node 2 is identified as an anomalous node.

2.5.4 Attack Mitigation

When discussing the detection and recognition of attacks it is also always the question of how it is possible to prevent or mitigate an attack that has successfully been detected.

A simple example for this are DoS attacks that pose a great problem to network security, most of the time, prevention is impossible, while the only way of protection against DoS is by mitigation and focusing on Quality of Service (QoS) to keep the network healthy. A typical way of doing that is by using the pushback architecture (Douligeris & Mitrokotsa 2004). This architecture is installed on several upstream-routers that analyze ongoing traffic and eventually limit the rate or drop packets for nodes identified as attackers. Rather than preventing the attack from happening, in this case we try to limit the harm to the network. A similar mitigation architecture is throttling, where the system for detecting attackers can be left to the server, which defines a throttle value such that the upstream-routers drop packets randomly according to the throttle level. By this approach, the server is able to reduce the amount of traffic before an effect can be measured. However, this can still lead to a reduced QoS for well-behaving nodes as there can be misclassification.

Preventing attacks is often not possible without extensive system changes and huge costs. Network monitoring combined with appropriate mitigation frameworks however can be a good choice. Especially, in sensor networks this is often the only viable choice due to resource scarcity and constraint possibilities.

Chapter 3:

SDN-enhanced Attack Mitigation in WSNs

This chapter explains how we used the SDN-WISE framework to detect, identify and mitigate attacks on WSNs. First, the main goal of this thesis is described in detail and afterwards the findings are discussed and evaluated eventually. The underlying architecture of this work consists of SDNWisebed, the framework proposed by Schaerer (2018) which is composed of the work from Galluccio et al. (2015) and ONOS, the SDN-controller developed by the Open Networking Foundation described in Subsection 2.3.2. The control plane is built up with RPL in non-storing mode, to enable the communication with the SDN controller. This Chapter explains what extensions were made to SDNWisebed in order to enable basic security by monitoring the network using attack detection and mitigation.

3.1 Introduction to Anomaly Detection

Anomaly detection describes the method of distinguishing the odd from the normal traffic. But in order to discriminate the two, first we have to define what is normal.

In statistical terms, normality means that a data set is said to be normal if it is well-modeled by a normal distribution, which is also called a Gaussian distribution. Furthermore, it defines that for a random variable, how likely it is that the underlying data set is normally distributed and within a normal distribution, we have that 99.7% of the data lays between 3 standard deviations of the mean.

Applying this to network security, a definition of normality of a node is required. Such a definition can be found by choosing random variables, also called features that we think model normality well in the network. After the identification of such, an algorithm is used to learn and optimize a model given an observation set. From this model, it is then possible to calculate an anomaly score for a given observation that tells how different a given sample is from others. This can be either stateless, where for every state the score is calculated independently, or it can be made dependent where the score

is calculated over a given time series. In the following subsection, the choice of features is described.

3.1.1 Feature Selection

In general, every model is only as good as its data. Therefore, the selection of appropriate features is the key to design an algorithm that recognizes anomalies well. In traditional machine learning used for intrusion detection, often raw packet data is used in combination with Deep Packet Inspection (DPI). DPI describes the way how packets are analyzed in terms of where it is sent to, from whom was the packet sent, what type of protocol was used, how frequent packets of this type are sent and especially what content the packet holds. In SDN-driven networks, there is only aggregated information available like the current network topology or the number of transmitted or received packets over a given link. This information however, is enough in order to recognize and identify attacks and attackers respectively. But the key to successful detection lies in the selection of appropriate features. In the following, the feature selection for the two attack types studied in this thesis is discussed.

3.1.1.1 DoS Attack Features

Generally, DoS attacks base on the foundation of packet transmissions. This has two basic implications: First, the size of a DoS attack linearly correlates to the number of packets transmitted. Second, a DoS attack generally focuses on a single target, rather than multiple destinations. Bhunia & Gurusamy (2017) used a bunch of features for DoS detection, such as *no. of sent requests, no. of failed authentication attempts, source of requests, bandwidth consumption, device usage at different time periods*, etc. The clear advantage of this is that they not only use the raw traffic pattern, but already build on a certain protocol. In our case, however, we try to make no assumptions of the protocol in use and therefore want to classify attacks only from network information that is typically available in networks like the total number of packets sent, the network topology and e.g. additional general node attributes. This means that we do not need to use features like *failed authentication attempts* or *protocol*, but solely use the number of packets sent.

Often, DoS attackers try to cover their malicious behaviour e.g. with carrying out their attack for a short time period only (in DDoS) or that they spoof sender addresses. For that reason it became obvious to use distributed network knowledge, that is for example, that we never use the knowledge of a single node only, but we try to use the combined knowledge of the majority instead. This is based on the assumption that the more nodes are compromised the more improbable this is. And so we may trust on our network statistics.

In SDN-WISE, every node sends a report packet to the SDN controller at a fixed interval. As it was described in Subsection 2.3.1, these report packets contain information about the current state of the node, its neighbours, the number of packets sent to neighbour X and the number of packets received by neighbour X, for every neighbour of this node. In Figure 3.1, a toy example topology having a DoS attacker A and its immediate neighbourhood is shown with the reported statistics aside each node. In order to cover-up the DoS attack, attacker A reports having sent only 1 broadcast packet since the

last report. However, his neighbours report that A has sent 20 packets during this time. Clearly, this is a contradiction and results in a logical anomaly.

As it is suggested in general, DoS attacks can be recognized from analyzing the number of packets that are sent in a period of time. Therefore, in this work, we use the sum of the packets that were reported having being sent by a given node. As we do not use the information about sent packets from a given node, this prevents misclassification by using information from potentially manipulated reports. To summarize, the resulting input features for DoS detection are simply the sum over all packets that a given node has sent, but from the perspective of the receivers rather than that of the sender itself.

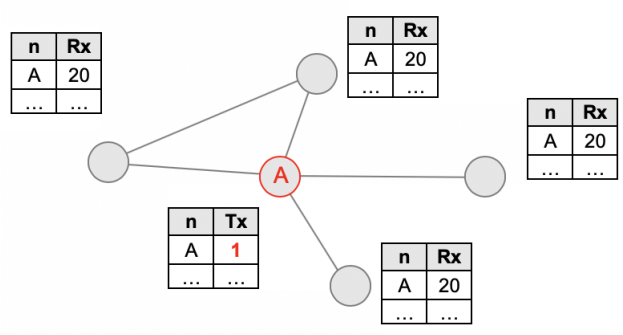


Figure 3.1: Toy example: DoS attack anomaly in attribute matrix. Each table contains the report of nodes to the SDN-controller

3.1.1.2 Node Replication Attack Features

As previously seen in subsection 2.5.2.2, node replication attacks are very hard to detect in distributed networks. This is because the potentially resulting network disruption does not appear as an attack but rather as a temporal network failure due to routing causes and due to the fact that different route reports are indistinguishable whether they are sent by victim or attacker.

In common network security based on packet anaylsis, we often face the issues described above, but having topology knowledege available in combination with additional individual node characteristics, it is possible to identify attackers or at least the attacker/victim pair.

So in order to recognize node replication attacks, it is necessary to select features that are very individual for different nodes such as the number of neighbours, the battery level or some geographical information if possible, information that cannot be reconstructed that easily. In SDN, since this information is based on report packets and nodes are identified with their MAC address, we cannot directly see changes in the reported node values and use them for our attack recognition. But we can add a change detector that scans report packet information for inconsistencies and changes. This is done by storing previously reported information and compare it to the newly received values. The outcome of this is a value that defines how much two consecutive reports differ from each other. Experiments in this thesis have shown that the best feature for node replication attack detection is by observing the set of neighbour node ids. This was calculated by

having an edge store that holds all currently known edges in the network that is then updated by the new information. During the update, the number of removed edges and the number of added edges is counted and used as the input feature for node replication attack detection. Of course, it is a very simple feature but it can be extended to account for other changes as well to map more complex issues. The focus on the neighbourhood of a node was experimentally evaluated and logically inferred from the fact that node neighbourhoods normally only change in the connecting phase of a network.

As previously mentioned, RADAR uses two input matrices: Adjacency matrix and an attribute matrix. While the former is made from the structure of the network, the attribute matrix can be defined by the user of the framework. The elegance of this framework basically is that the attribute matrix can contain very heterogeneous data while having only the single constraint that the number of rows must match the number of nodes in the adjacency matrix.

In RADAR, the features of both worlds, the structural and the contextual, are used and combined to find anomalies in all three kinds of ways. In Figure 2.3 an example for all three anomaly types is shown. Node 12 poses a structural anomaly, since it is the only node with less than 2 neighbors. Taking a look at the attribute table, node 7 is considered an anomaly because its f2 value differs a lot from that of all other nodes. Now combining structure and attributes we find that node 2 is an anomaly as well due to its neighborhood, which usually has low f1 values, while node 2 is completely different. These three examples are then listed in the anomaly score table e.g. like n12: 123.45, n2: 87.21, n7: 85.41, n6: 12.94, n8: etc. From this example, we can see that it might be difficult to tell whether a given score actually describes an anomaly or not. Comparing these scores, we find that there is a gap between anomalous nodes and normal nodes, but this gap is not well-defined and can vary a lot. For that matter, it is crucial to interpret the score values correctly. This issue is described in the next section.

3.2 Attack Detection Framework

The Attack Detection Framework proposed in this thesis consists of a pipeline built from RADAR created by Li et al. (2017), a filter for generalization, and a machine learning classifier. Figure 3.2 shows the complete pipeline with inputs and output. First, the adjacency matrix of the network graph and an attribute matrix made from columns of the features described in the previous section 3.1.1 are input into RADAR. Also the hyperparameters α, β, and γ for RADAR are set. Then, RADAR calculates anomaly scores for every node and outputs a score vector that is filtered and sorted. Afterwards, a machine learning classifier determines based on the score vector whether there was an attack, and if so, who was the attacking node. It then outputs either an attack label or an attacker label, based on the configuration.

Filter

In all classifiers that are machine learning based there exists the constraint that the size of the input vector has to be fixed. This constraint bears a huge problem for graph-based attack detectors because graphs are dynamic and vary in their structure because in a network, nodes may appear and leave at any time. However, RADAR in conjunction with a simple filter, which is proposed in this thesis, provides an easy solution for this

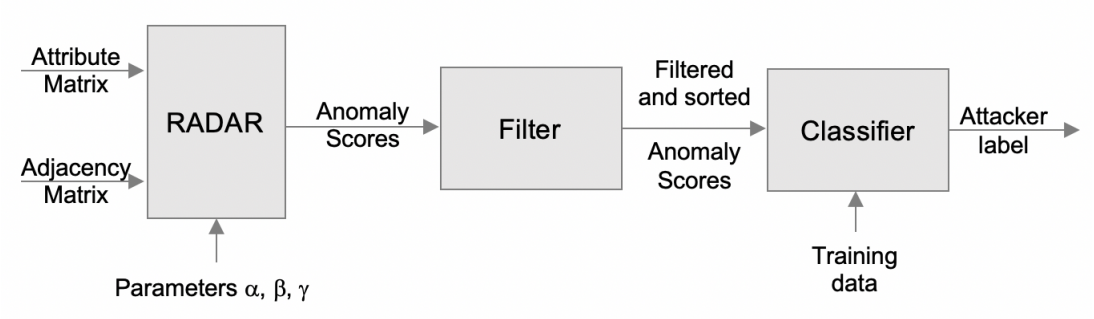


Figure 3.2: Attack(er) Detection Framework

problem. RADAR maps matrices of sizes $n \times n$ and $n \times m$ to a variable score list of size n. This list can be ordered and further used for classification by using a subset from the ordered score list of fixed size k. A more detailed explanation of this procedure is described in the following paragraph: First, the score list obtained from RADAR, which contains entries of the form (score for node 1, score for node 2, ...) can be sorted by score in descending order. Then, the k highest score values are taken (in this case, k will be a new hyperparameter). These values create the fixed feature vector for our classifier. Then from the node-score-list, a map can be made that maps the score back to the actual node identifier. Now, the classifier is trained to predict the rank from *1..k* of the attacker. For the sake of completeness, it is to note here that if the attacker is not found in the top k ranks of our training set, then the label will be set to *-1*. In our case, we try to identify a single attacker, which is usually ranked by RADAR in the top k nodes. With this setup, it is possible to have a very generalized approach for different network sizes that can train a single model for multiple network topologies and attacks. However, it can be difficult to set k appropriately. Experiments with networks of various different sizes would be required to identify the average rank of an attacker.

Because attackers will have a significantly high value for the previously mentioned features, they will also achieve the highest values from RADAR (see Figure 3.3. frequency of attacker appearance at rank). In combination with these properties and because the scores can be mapped back to the nodes, a generalized approach that deals with varying network sizes can be built by applying such a filter. Experiments have shown, that it may be beneficial to use additional features like average score or minimal score.

3.3 Attack Mitigation

As we have discussed how we can recognize attacks and attackers, this recognition is useless as long as we cannot prevent the attack from happening or at least mitigate it. Therefore, some mechanism is required to mitigate or prevent the attack after detection. Different ways have been proposed for attack mitigation. But the mitigation mechanism however depends a lot on the attack that is carried out.

A simple and effective way for attack mitigation is push back. Push back is defined as dropping packets or blocking an attacking node. It is usually achieved by broadcasting an alert to all nodes in the network with the desired behaviour e.g. *dropping packets*

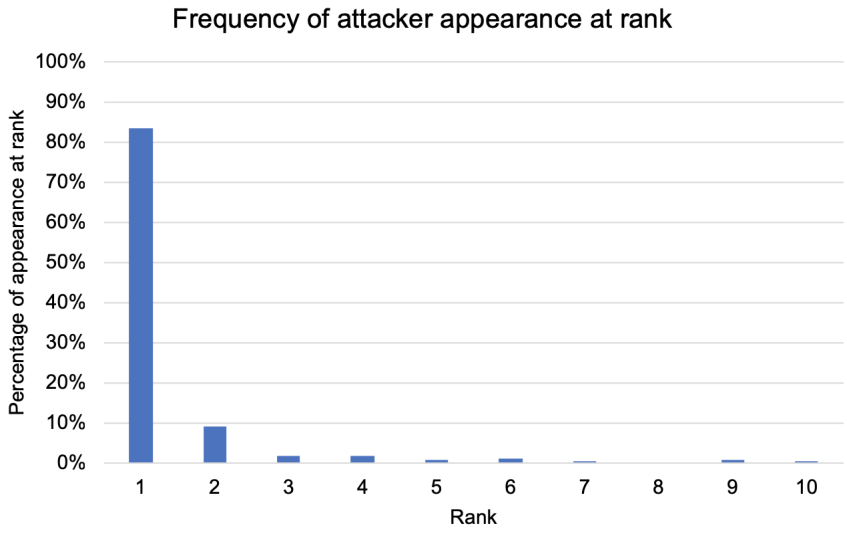


Figure 3.3: Attacker appearance at ranks in ordered score list obtained from RADAR of a network with 10 nodes

from node X. However, broadcasting is not very efficient, which is why only a subset of nodes should be informed instead. The subset consists of all nodes that are affected by the attack. For example in a DoS attack against the controller, this would be the nodes on the shortest path from the attacker to the sink node.

Assuming the topology in Figure 3.4 and that node A is carrying out a DoS attack on the controller, nodes 2 and 4 are on a shortest path to the sink node S.

Eventually, the attack is detected and the attacking node A is identified. Now, the push back strategy would be to drop packets from node A. Therefore, the sink node S would initiate a broadcast message that all traffic from node A is to block. The packet eventually reaches node 2, but might get lost and not reaching node 4. For that matter, the attacker traffic is still sent to the sink node S. Although, the packets are dropped at the sink node S, other packets may not be received anymore due to congestion.

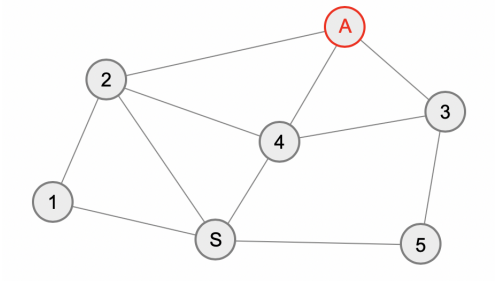


Figure 3.4: Toy example topology with attacker and sink node

3.3.1 Attack Recovery and Mitigation Algorithm - ARMA

To counteract this problem, we propose ARMA, the Attack Recovery and Mitigation Algorithm. It is based on OpenFlow and attempts to install FlowRules on certain nodes to block DoS attack traffic and afterwards tries to reset the attacker node in a further step.

The algorithm works as follows:

1. Attacker A is detected
2. Create set S with all neighbours of A
3. Find all shortest paths P with source Sink and destination A over $s \in S$
4. For every path $p \in P$, create unicast message containing FlowRule IF(P.2 = X) DROP
5. When receiving such a message:

   If $nextHop == A$ then continuously send unicast messages to X with action RESET

   else install FlowRule and forward unicast message to nextHop

Having this simple algorithm, DoS attacks can be mitigated, what can be explained with the example topology in Figure 3.3.1. In this example, there is an attacker A, several interconnecting nodes and a sink node S. First, the attacker A is identified with the proposed Attack Detection Framework. Then the neighbour set of the attacker is built $S = \{1, 2\}$. Afterwards, all shortest paths through $s \in S$ are found: $P = \{(S, 6, 3, 1, A), (S, 5, 3, 2, A)\}$. Now, for every $p \in P$ a unicast message is created containing *FlowRule IF(P.2 = A) DROP*. The two messages travel downwards the paths $(S, 6, 3, 1)$ and $(S, 5, 3, 2)$ and installs the FlowRule in every node. Then, the packets reach nodes *1* and *2*. At this point, unicast messages containing the action *RESET* are sent to attacker A. Eventually, one of the *RESET* packets reaches node A, node A is reset, and the attack was successfully mitigated. After a while, the *DROP* FlowRule is timed out and is removed from the FlowTable.

It is crucial to note that normally, DoS attackers are not trying to flood or jam the whole network, but rather want to bring down a specific node. Jamming the network would result in loosing the zombie nodes that are carrying out the attack due to not being able to communicate with the attack initiator anymore. For that reason, we assume that at some time the attacker is able to receive a RESET command. Of course, it is questionable, whether the node would actually reset itself on such a command. But it is worth a try, e.g. to reboot misconfigured or erroneous nodes. However, this step is optional, since the network traffic will already be blocked at the earliest point anyway, which is at the next hop from the attacker node.

First, little experiments with a small number of nodes have shown that this protocol can block a DoS attack effectively. However, this was not investigated further and is considered to be future work. Simulations with many tens of nodes and multiple attackers should be carried out to proof the efficiency of this algorithm.

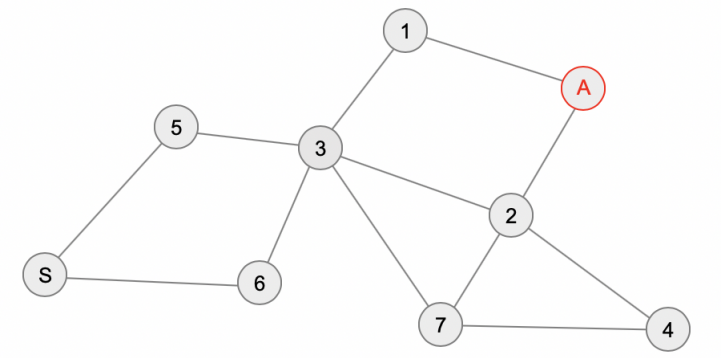


Figure 3.5: Toy example topology with attacker and sink node

3.3.2 Caching

Another simple mechanism that was implemented to prevent DoS attacks is caching. It is the first measure against server overload because it tries to minimize the computation power needed to provide a service. Caching means to store request and answer pairs in fast memory to retrieve it immediately without having to do additional calculation other than matching the request with the cache. On a cache hit, the response is retrieved and returned, while on a cache miss, the request will be processed normally.

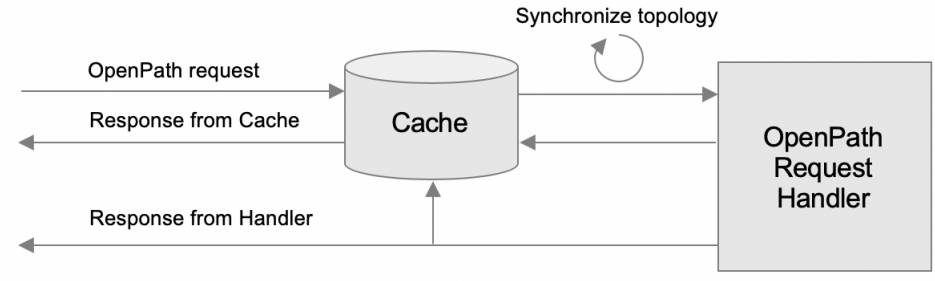


Figure 3.6: Cache for OpenPath Requests

In SDN, the SDN controller keeps track of its answers to previous OpenPath requests with a simple cache. This can be achieved using a hash table with keys containing source and destination addresses for each request and holding the response as value. With that setup, the controller can prevent calculating the same paths multiple times in a row and simply retrieve the appropriate answers from its cache. In order to be aware of new routes and route changes, the controller should check the cached routes from time to time to avoid responding with wrong path information.

Clearly, caching may not be able to prevent on-purpose DoS attacks completely, but it can help to keep the QoS stable for a longer time and therefore prolongs the interval between the first sign of attack to the time the server goes down. This is crucial for DoS detection and mitigation, as every millisecond counts.

Chapter 4:

Experiments

To test the performance of the proposed framework, several experiments with the two attack types (DoS and Node Replication) in a sample network of 10 and 11 nodes were carried out respectively. For the evaluation metric, the accuracy was used which is defined as the sum of all True Positives divided by the total number of samples. This Chapter is split into two different sections, each focusing on one of the two attacks.

4.1 DoS Simulation

4.1.1 Overview

As we have discussed previously, DoS attacks can do a lot of harm to network performance and they can be hardly detected before they have an effect. In SDN-WSNs there are different ways in which DoS attacks can be carried out. In this chapter, a specific threat will be highlighted that eventually came up during an experiment for DoS simulation where one node should have attacked another one in the network, but the experiment resulted in having lost the SDN controller temporarily, due to a stack overflow. In the bootstrap phase of the network, the SDN controller is not aware of all paths and can not provide all routes that may actually exist. Therefore, it can happen that a node requests such a path for which the SDN controller may not respond appropriately. For that matter, the requesting node may continue requesting the same path over and over. Since the processing of such an OpenPath packet requires a lot of computational power in the SDN controller for decoding, topology building, path selection, shortest path calculation, etc., this can easily lead to a DoS attack against the SDN controller. After a while the SDN controller got overloaded with requests, could not process them anymore, and crashed. Not only does this attack affect the SDN controller but also the sink node and all nodes on the RPL path to the attacking node. Therefore, it can be compared to the DNS amplification attack (Douligeris & Mitrokotsa 2004). That is, when an attacker spoofs a small packet with the originating IP address of the victim and then gets the larger responses back from the DNS server.

Additionally, it was observed that the duration of this DoS attack is actually growing with the depth of the branch in which the attacking node resides. This is mostly because of the packet stacks on every hop that are filled up and then emptied within a short period of time. It is therefore almost impossible for the SDN controller to recover from such an attack just by fulfilling its service responsibility. Furthermore, it can be problematic for the sink node and other central nodes close to the sink node, as they have to handle a huge number of packets from both sides.

4.1.2 Dataset

For the DoS attack classification, a dataset of 1800 RADAR-score samples, which are separated into two groups attack (800 samples) and normal (1000 samples) was created. The data set was sampled with COOJA and the Attack Detection Framework pipeline without Filter and Classifier. This way, the width of the data set could be reduced to only 11 columns (time stamp + 10 RADAR scores). The attack group is split into 8 equal sized groups, each of which is labeled with the node id of the attacker 2-9. The normal dataset was generated by simulating two scenarios with 30 random topologies of 10 nodes (1 sink node included). The first scenario considers one pair of nodes to be selected in the beginning, who then transmit packets in a fixed interval of 10s over a period of 5min. After that, another pair of nodes was chosen for transmission. Every 60s the flowtables are purged and, therefore, the path from sender to receiver needs to be requested from the SDN controller. All nodes generate report packets every 10s that are passed to the SDN controller. For every received report packet, the SDN controller calculates the RADAR scores and appends them to a log file. In the second scenario, every node picked another node randomly as destination to send packets to. Like in the first scenario, every 5min the destinations were changed what resulted in having randomized data. Due to the long report interval, a lot of duplicates are logged in the dataset. Therefore, the duplicates were filtered out what resulted in having only 534 unique samples in the dataset, 183 of which are attacks, and 351 are non-attacks.

As RADAR requires three hyperparameters to set the sensitivity, Li et al. (2017) propose to fix the values for beta = 0.2, gamma = 0.2 and alpha = 0.5 to let it perform best. Those values were determined empirically by Li et al. (2017). During our experiments it was found that when setting beta to such a high value, it did not detect attribute anomalies well given such a small network as in our case. Therefore, a value of 0.01 for α and β to increase the sensitivity of RADAR and setting gamma = 0.1 to give more weight to attribute anomalies than structural anomalies showed to be appropriate. However, it is difficult to tweak these hyperparameters and even small changes in those may significantly change the results. Li et al. (2017) provided a 3D plot that shows the Area under Curve (AUC) with fixed alpha = 0.5. It can be seen, that beta should not exceed 0.1 in order to have great sensitivity.

4.1.3 Results

In order to determine the best classifier for the DoS dataset, different machine learning classifiers have been tested. This was done by using Weka, a machine learning framework, which provides many different algorithms and includes training and testing functionality. As the task was to predict one or more labels given a dataset, it was logically to select

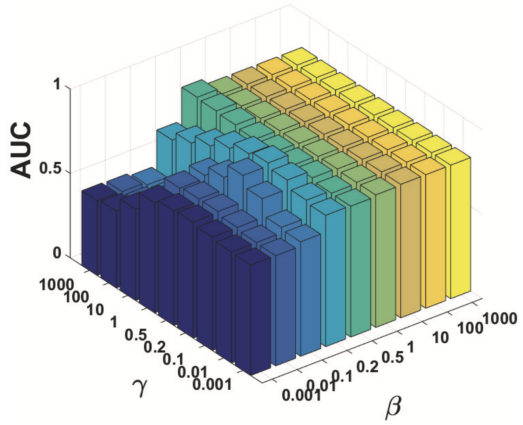


Figure 4.1: RADAR hyperparameter coherence with fixed $\alpha = 0.5$ (Li et al., 2017)

simple predictors like K-Nearest-Neighbour (K-NN), RandomTREE and KStar. Support Vector Machines (SVM) were left out, because they did not achieve a high enough score in order to be comparable with the other three algorithms.

In Figure 4.2 the accuracy results of a multi-class classification to determine the attacking node are shown. The dataset was created from samples like: Score1, ..., Score N, Node id 1, ..., Node id N, AttackLabel, Attack/Non-Attack. This structure of the dataset was necessary because RADAR does not output a list ordered by node id but by score. Therefore, the list of node ids had to be added as well to indicate what node achieved which score.

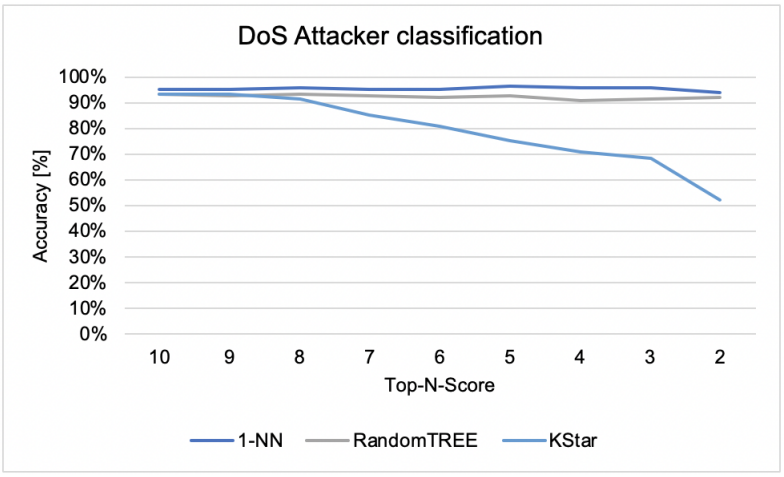


Figure 4.2: DoS attacker classification accuracy, $k = N$ scores in a network of 10 nodes. The accuracy with 1-NN and RandomTREE does not depend much on the availability of node scores

From Figure 4.2 it can be concluded, that 1-NN and RantomTREE are nearly constant in accuracy, which is quite clear from the fact, that RADAR provides a very good

anomaly detection and K-NN and RandomTREE find similar patterns over the scores. 1-NN achieved the highest score with 96.44% accuracy having the top five RADAR scores available. RandomTREE and KStar share the second rank with an accuracy of 93.43%, but there is to say that KStar required all 10 scores to get this accuracy while RandomTREE only used eight top scores. Furthermore, RandomTREE showed constant accuracy. It was found that attacker classification can be done quite performantly with only three top RADAR scores. This could be due to the fact that RADAR successfully calculates appropriate anomaly scores for the DoS attack. In combination with Figure 3.3, the proposed framework should be able to detect more than 90% of the DDoS attackers with only having scores of the top-3 nodes for any network.

Comparing the results of Doshi et al. (2018), who used data from Deep Packet Inspection (DPI) and achieved 99% accuracy, to our highest accuracy of 96.44% we achieved a similarly high accuracy having only a very small amount of attack information available. Doshi et al. (2018) used a variety of stateful and stateless features for their classification, while we used only the number of packets sent by each node in combination with the network topology for plausibility and weight correction as described in Subsection 3.1.1. It would have been interesting to compare our results with the performance of ShadowNet proposed by Bhardwaj et al. (2018) since they used a similar SDN approach. However, in their paper they only provide results for attack mitigation duration and not for the actual attack recognition. In the paper from Bhunia & Gurusamy (2017), they state to have achieved precision values for DoS classification of 98%. However, their scenario considers traffic from consumer IoT devices, like IP-Cameras only and not sensor nodes. Therefore, their experimentation results are hardly comparable to our results.

4.1.4 Threshold-based DoS Detection

In Section 3.1, we discussed how the RADAR framework can be used to detect anomalies in a network with residual analysis. Especially, DoS attacks can be recognized pretty well. This is due to the fact that DoS attacks have a big influence on network statistics. During an attack, the total number of packets in the network increases a lot, what differs much from the normal traffic in a network. In addition to the attack detection mechanism based on anomalies, a threshold-based DoS detection algorithm was tried out to check an alternative without machine learning.

For this reason a threshold-based detector was created, which can be seen as a linear classifier that distinguishes between attack and normal traffic. Of course, a non-static threshold for recognizing attacks is required, since traffic continuously varies in every normal network. Normally, this would be learned from samples of a training dataset or by training a Recurrent Neural Network (RNN) and rewarding it appropriately. But this can often be very difficult and mostly it is less efficient than simple statistical analysis. Therefore, in the threshold-based detector, a simple rolling median in addition to some scaling factor according to the expected peak packet rate and an offset representing the network size was developed (see Equation 1).

$$DTh_{\alpha,\beta,\gamma}(S,k) = \alpha * median_{k-\gamma,k}(S) + \beta \tag{1}$$

Using such a threshold that is based on time series and the total number of packets that were exchanged per interval it is possible to detect DoS attacks easily. Figure 4.4 shows

the results of this threshold-based DoS attack detector. The experiment was carried out with data from a sample topology with 10 nodes, including one attacker and one sink node. The values for the hyperparameters were set as follows: $\alpha = 1.1, \beta = 9, \gamma = 3$ In Figure 4.1.4, the total number of packets of four normal and four malicious scenarios are shown. It can be seen that in the beginning, the traffic is inconspicuous in all cases. After time stamp 14, DoS traffic clearly differs from normal traffic.

In Figure 4.4, the grey line shows the dynamic threshold, which predicts the maximum normal peak rate, and in dark blue the actual number of packets in the network. Whenever this blue line exceeds the threshold, a DoS attack is taking place.

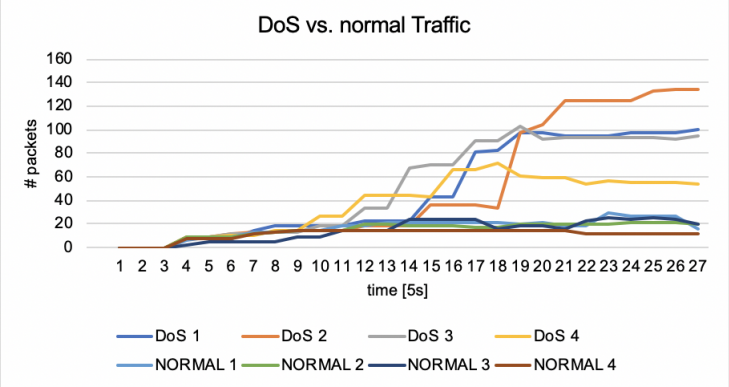


Figure 4.3: Network traffic with and without DoS attack

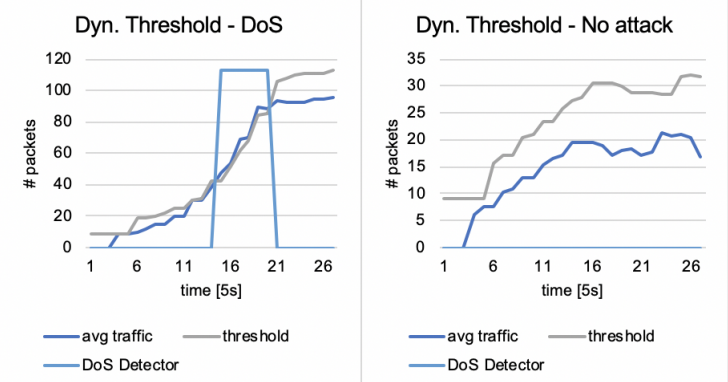


Figure 4.4: Dynamic threshold attack classification

One of the reasons why this additional experiment was conducted is that it was important to see whether the chosen DoS scenario would be realistic. With this, it could be shown that a threshold is able to detect DoS attacks from the same features we used for our Attack Detection Framework described in Subsection 3.1.1.

4.2 Node Replication Attack Simulation

4.2.1 Overview

Another experiment that was conducted during this thesis was to detect node replication attacks. Node replication attacks, as previously described, mean to have multiple instances of nodes in the network that share their id. This attack basically always changes the network topology and, therefore, the structural part of the network. Since RADAR collects information not only from node attributes, but also from the structural part of the network such as topology, RADAR provides the prerequisite to detect node replication attacks as well.

For the detection of such attacks, another dataset had to be created. This was done by adding additional nodes that clone the id of a random node. The attacker nodes have been placed such that they cover most of the network nodes to disrupt the complete network as an attacker would probably do.

When trying to detect DoS attacks, we have used attributes for RADAR that contained only the number of packets that had been sent by node X to node Y. Since node replication does not necessarily increase the number of packets sent by a certain node, at least not as its first effect, other features had to be considered. After some simulations, it became obvious that the best feature for detecting anomalies would be to look at report packets. Node replication attacks usually consist of cloning an id and putting the attacker to another place of the network, to disrupt it by false routing or gaining information from other nodes. The original node often remains untouched. Therefore, contradicting report packets would most probably show the biggest anomalies. Due to that fact, we attempted to upgrade ONOS, to also update the topology of the network by removing edges that were not reported any more in a future report packet. But as described in Section 5.2 several problems lead us to use a workaround which compared neighbourhood samples before and after a report packet of a given node. That way it became possible to observe the changes in the neighbourhood of every node.

The resulting feature vector then consisted of how many edges had to be changed after a certain report packet. As the original node and the copy of it would not report the same topology, this would lead to high values in the difference and therefore, result in a high anomaly score. Additionally, the number of current neighbours and some other values like battery, light etc. were considered.

4.2.2 Results

From the results shown in Figure 4.5. it can be seen that 1-NN (96.61%) constantly performs best, RandomTREE (94.27%) second and the worst is KStar only achieving 72.75% accuracy. Similarly to the results of DoS attacker classification, 1-NN and RandomTREE perform constantly well. In contrast to the DoS results, the accuracy drops linearly for 1-NN with decreasing amount of scores. Compared to the performance with related works Grover et al. (2011) (precision: 93%), Zhu et al. (2007) (success rate: 94.5%), Parno et al. (2005) (probability of detection: 85%) and Conti et al. (2007) (performance: 87%), we achieved better performance, assuming that the various and unspecified values are comparable.

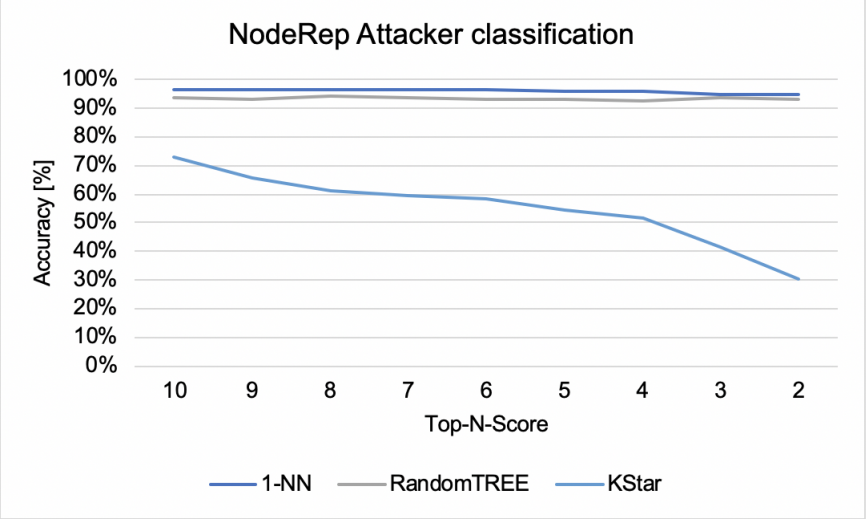


Figure 4.5: NodeRep attacker classification accuracy with $k = N$ in a network of 10 nodes. The accuracy with 1-NN and RandomTREE does not depend much on the availability of node scores

4.3 Multi-attack Classification

After having discussed the classification accuracy of attackers from datasets that only had two types *Attack* and *No-Attack*, it is interesting to see whether the classifiers can distinguish between the two attack types *Dos*, *NodeRep* and the normal type *No-Attack*. For this experiment, the complete dataset including node replication attack traffic and DoS traffic was used. This was done by merging the two datasets into one and labeling the attacks with their name as a string label. Afterwards, the dataset has been introduced into AutoWeka with 10-fold cross validation to obtain the accuracy for six different classifiers.

The results show relatively high accuracies for most of the classifiers. 1-NN performs best again, while RandomForest and Bagging achieve at least 90% accuracy. Again, it is pretty obvious why BayesNet performed less good than DecisionTable or 1-NN: BayesNet assumes normal distribution and the scores calculated by RADAR are not normally distributed at all. Classifiers that do not use distribution assumptions therefore performed better, which is why there is no distribution-based classifier found in the top 3.

Since the attack classification achieved good results, identifying the attacker in the mixed dataset was tried out in order to compare it with the previous results when only one attack type was present.

As expected, the highest accuracy when using the mixed dataset dropped by around 1.5%. This is simply due to the fact that some of the score samples are similar between DoS and NodeRep. Again, 1-NN achieved the highest accuracy of 94.80% with eight scores, followed by RandomTREE with 90.66% using nine scores. The worst performance resulted again from KStar with only 72.64%.

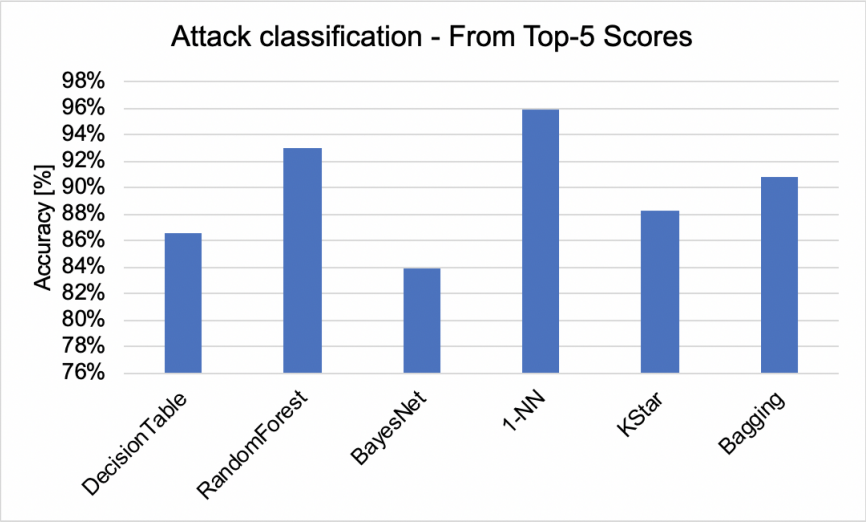


Figure 4.6: Classifying three attack types by having the top 5 scores available

Summarizing the results, RADAR provides a very good separation of attackers and non-attackers. Using a filtering and sorting algorithm that outputs the highest anomaly scores, combined with a classifier like 1-NN or RandomTREE attack and attacker classification is possible with having very high accuracies of up to 96.61%. The most appropriate classification algorithm was found to be 1-NN.

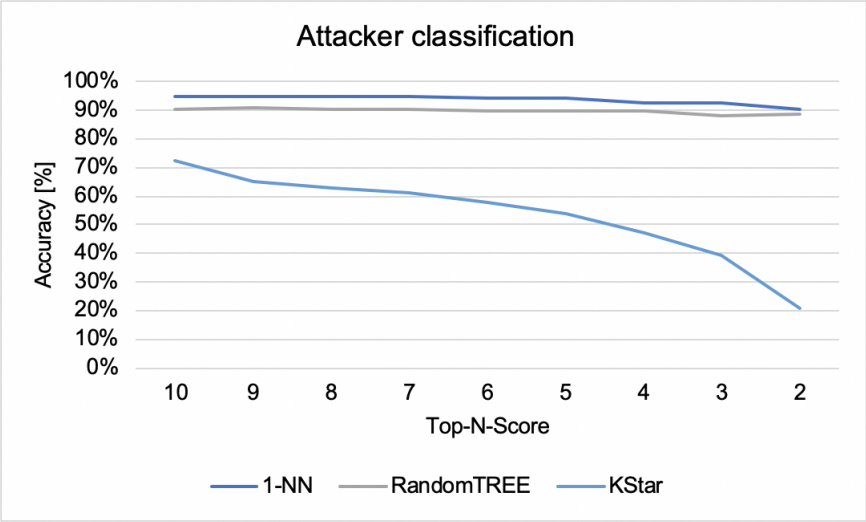


Figure 4.7: Classifying attacker from mixed dataset

Chapter 5:

SDN-WISE Framework Contribution

During the thesis, many bugs and missing features could be found. This Chapter tries to highlight the most important ones and gives an overview on how they were fixed or added respectively.

5.1 Bugfix to OpenPath Forwarding

In the current implementation of SDN-WISE for Contiki we could identify a bug with the forwarding of OpenPath response messages travelling from the controller to the requesting node. When a node on the calculated path from the OpenPath message received the packet before the requesting node, the packet was always passed along the best route defined in the OpenPath message. To clarify, the OpenPath packet holds the path information from the requesting node to the target node that is currently seen as the most favorable path between these two nodes. It is a response to the REQUEST message that was previously initiated by the requesting node. Naturally, the OpenPath response packet has to be delivered to the requesting node in the network. Therefore, the controller passes an OpenPath packet to the sink node who injects the packet in to the network according to the handle_open_path() function. Currently, every node receiving the OpenPath response packet iterated through the specified path in the packet, searched for his position and if the position exists, it would eventually forward the packet to the next node in the path. If the position did not exist in the path, it would ask the controller for a route to the requesting node. This simplified protocol implementation resulted in a bug, so that when the packet receiver is part of the path and is not the root of the path, the packet never arrived at the requesting node target. This is due to a false implementation of the algorithm that was used to deliver the packet. The misleading algorithm can be read as follows:

1. Find own position in path

   If position was not found,

ask controller for a route to the request node → exit

2. Install route for requesting (*req*) node.
3. Install route for target (*tgt*) node.
4. Select immediate subsequent node (n_{i+1}) in path
5. Forward packet to this node

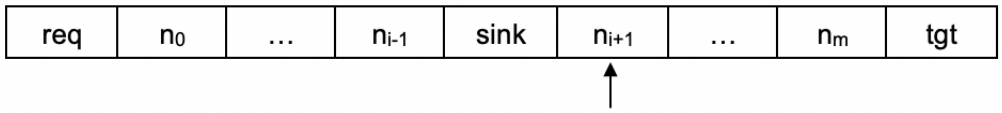


Figure 5.1: Next hop selection from OpenPath

This way, the packet could never reach the requesting node if one of the nodes in the path other than the requesting node was reached first. In order to fix this bug, several attempts have been tried. The simplest approach would have been to identify the previous hop and determine whether it was already part of the route, and if so, the packet could have been copied and passed along the path forwards and backwards, based on the position of the originator. Instead, due to the missing knowledge of the previous hop address, two other solutions were tried out.

5.1.1 Packet Masquerading

Packet masquerading describes that a packet is either encapsulated or the type is changed so that the protocol does not interpret the actual payload. This can help to handle masqueraded packets differently. In our case, this was helpful for forwarding the OpenPath packet from the sink to the requesting node. Packet masquerading was implemented as follows:

1. OpenPath packet from controller reaches the sink node
2. Change packet type to ENC_OPEN_PATH (11)
3. Forward packet to next hop according to the control layer

At the requesting node, the packet type is reverted to normal OpenPath and the packet then travels along the actual path from *req* to *tgt*.

Using this simple masquerading, it is guaranteed that the whole path is installed on every node in the corresponding path. But the drawback of this algorithm is its inefficiency. For every node receiving this message, due to the still persistent bug it could happen that there an additional OpenFlow message has to be sent to reach the requesting node what results in having to request for the destination twice.

5.1.2 Bugfix to current SDN-WISE implementation

Due to the manipulation of the SDN-WISE protocol in the previous solution, it was clear to look for a better way. Therefore, the bugfix to the actual problem was tried out.

The protocol specification for OpenPath tells that whenever a node in the OpenPath is reached, it should determine where to forward the packet based on the current position in the path, the source node and the destination node of the packet. This can be done since the OpenPath packet includes the information for the source and destination independently of the OpenPath content. So, the actual and correct implementation for the algorithm looks as follows:

1. Look for own position in the path

   If it is not found, request a path to the requesting node → exit

2. Check whether the requesting node is destination node of the packet

   If so, pass packet to previous node

   and install route for requesting and target node

   Otherwise, pass packet to next node in the path

   and install route for requesting and target node

Additionally, there was an error in the current implementation of ONOS that results in setting the wrong destination address for the OpenPath packet. Therefore, we have added a conversion algorithm to set the correct destination address in the sink node.

With this bugfix, the installation of routes uses at most two OpenPath requests and it does not violate the specification of OpenFlow. Also, as in the previous version, two packets have to be requested. One packet is used to establish a route from the sink node and all intermediate nodes to the requesting node and the actual OpenPath packet to establish the route between requesting and target node.

5.1.3 Bugfix to OpenPath Generator

In the previous part on Bugfix to OpenPath forwarding, we described ways to reduce the number of OpenPath packets for establishing routes. As these methods still had several drawbacks, because they required to request multiple paths to reach the requesting node or they based on changing fundamental parts in the OpenFlow protocol by creating an additional packet type, a real fix rather than workaround was needed. Therefore, we investigated the OpenFlow protocol and its implementation in SDN-WISE. From that, we found that parts of the protocol specification were left out or implemented wrong. E.g. the protocol specification stipulates that on every node of the path from source to destination, this route should be installed. But in the SDN-WISE implementation of OpenFlow, they installed routes for every node in the path, which not only led to a performance issues on the nodes, but most of the time also setup unnecessary routes that wasted a lot of space. Due to that fact, an update for the OpenPath installation procedure was necessary.

Whenever an OpenPath is requested, it is necessary for the controller to route the packet downwards to the requesting node. This was previously done by a method called packet matching, which means that a node requests another route to the target node from the controller. Therefore, it was necessary to request two routes for every route that was needed. This was already showed before. It became obvious, that it would make sense

HEADER	WINDOW *1*	WINDOW *2*	...	WINDOW *n*	ADDR *1*	ADDR *2*	...	ADDR *m*
10 Bytes	5 Bytes	5 Bytes		5 Bytes	2 Bytes	2 Bytes		2 Bytes

Figure 5.2: Packet definition of OpenPath packet with windows and path

NET	LEN	SRC		DST		TYP	TTL	NXH	
1	29	0	1	0	10	5	100	0	9

Window *1*					Window *2*					Path (Addr *1*, Addr *n*)									
19	0	4	0	10	19	0	2	0	2	0	9	0	2	0	9	0	1	0	9

Figure 5.3: Toy example OpenPath packet at sink node (0 1)

to make the sink node be aware of the path to the requesting node directly. With that in mind, it was clear that it would work when simply transmitting the route to the requesting node before sending the actual OpenPath packet. The procedure works as follows:

1. The best[1] path to the requesting node is calculated
2. The best[1] path from requesting node to destination node is calculated
3. The paths are merged, and the response packet holding the complete path is sent to the requesting node in a row

In the OpenPath packet, there are several window fields available that define the match filter for the packets of the flow entries. A window is consisting of 5 bytes and holds the type, the address and operators. In the current implementation of SDN-WISE the type and operator are constantly set to 19, which means to match any packet and the equal operator =. The following two bytes then define what fields of the received packet has to be matched with the two bytes following afterwards. With this, all the forwarding rules can be defined. However, in the current implementation, this installation did fail in two ways: During the path installation to the destination node, according to the specs, also the backward route should have been installed automatically. However, the installed flow rules were wrong and therefore, packet transmission could fail in the opposite direction. The second problem that appeared was that the implementation did not account for a change of path. It was rather defined that every path starts at the source node and ends at the destination. But with the proposed implementation, this was no longer true. Therefore, a workaround had to be created for that purpose. The toy example on the next pages highlight that.

In the packet header, the source address (in this case the controller), the destination address (in this case the last node in the path) and the next hop is defined. Furthermore, the packet TYP = 5 for OpenPath is set. In the payload of the packet, we first have the windows defined which tell what path should be installed and the path on which this packet should travel.

The packet starts at node 1, which is our sink node in the topology shown in Figure 5.4. The packet is then travelling along node 9, node 2 and back to node 9, this time it should install the flowrules. These are defined as follows:

[1] the best path, in this case refers to the path with the lowest cost according to the cost function in the controller

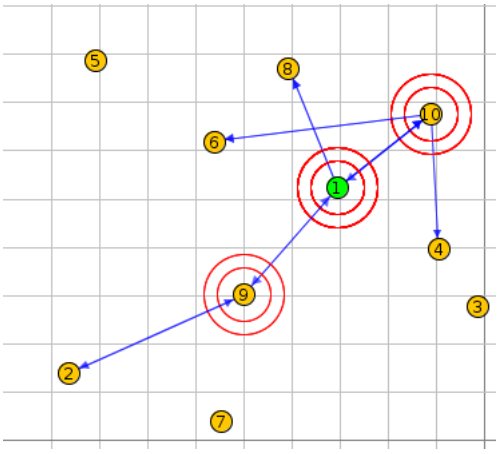


Figure 5.4: Toy example topology with route from node 2 to 10 (2, 9, 1, 10)

```
IF (P.2 = 2; P.2 = 10; P.4 = 2) {FORWARD([0 1]}
IF (P.2 = 10; P.2 = 2; P.4 = 10) {FORWARD([0 2]}
```

Figure 5.5: FlowRules installed at node 0 9

In the IF() statement, the parameters that have to match a packet to be handled according to the actions rules are defined. The first P.2 value matches the source from which the packet arrives. The second P.2 value defines the destination of the packet. With these values, it can be decided how the packet has to be handled. In the case when a packet has to travel from node 2 to node 10, the first rule will match, and the packet will therefore be forwarded to node 10. Previously, the second FlowRule was not installed correctly what could have led to accidental packet drop. Also, multiple packets had to be sent in order to set up the path, what has already been shown before with the tunneling approach.

In order to install the path only when a node resides between the requester and the destination, there had to be a workaround used. This approach consists of removing the previous node from the path, because otherwise it could happen that paths, such as the one provided in the toy example, where nodes appear twice, the current position could not be determined. Additionally, a simple check is implemented whether the current node is found in the actual path or before it. This is done by checking whether the requesting node still resides in the path. If so, the current node is not on the path that has to be installed.

To proof that this implementation improves the performance of the previous implementation of SDN-WISE, we conducted an experiment with different topologies. In Figure **??**, the average times for OpenPath route establishment for the three implementations before (old), after bugfix described in section 6.1.1 (tunneling), and after bugfix described in this section (new) are shown. The times were measured from the initiation of the request by the requesting node and the time the path was established. In all scenarios, the same topology, path request, cost function, and a message interval of 10s was used.

	old	tunnel	new
Time to establish path	11.29s	0.79s	0.92s
Total transmission time	20.29s	10.29s	10.41s
Number of packets required in total	5	3	3
Modification of OpenFlow specs	false	true	false

Table 5.1: Evaluation results of the three implementations

From Table 5.1 we can see that the tunnel protocol performed better in terms of speed than the new approach. But the time can be neglected for two reasons: First, the time difference is only 120ms on average and secondly, the tunnel protocol requires modification of the OpenFlow specs which is always something that should be avoided. Additionally, the new approach also installs the backward routes correctly, which makes up for the speed difference. To summarize, the new approach of calculating OpenPath packets improves the performance and it should be considered as a way of setting up routes using SDN on WSNs.

5.2 Topology Display of SDN-WISE Dynamic Routing Packet Processor

In the current implementation of SDN-WISE for ONOS, nodes and links are only added once and shown independently of the current state they are in. This results in having invalid topologies and thereof routing in to the wild.

For that respect, I have implemented the functionality to deactivate nodes during the startup of the network. As deactivated nodes are not considered for routing, this helps to exclude invalid or inactive nodes from routing decisions.

I have also tried to update links and remove dead nodes from the topology view. However, the implementation of SDN-WISE in ONOS has several architectural issues that makes such updates impossible without having to change the architecture completely. In future work, a total revision of the implementation should be considered therefore.

Chapter 6:

Discussion

6.1 Summary of Results

In this thesis, a framework for SDN-based Anomaly Detection for Wireless Sensor Networks has been proposed. This framework helps to detect WSN attacks in a generalized way in terms of scalability and extensibility for additional attacks by looking for anomalies using the anomaly detector RADAR proposed by Li et al. (2017) described in Section 3.1 and SDN. In combination with an additional machine learning pipeline, the framework has been shown to recognize frequent WSN attacks, such as Denial of Service or Node Replication. After recognizing attacks and identifying attackers, a simple algorithm to mitigate DoS attacks was proposed.

The literature study in Section 2.1 revealed that a lot of research has been done in the field of security for IoT, but the majority of papers focus on the detection of DDoS attacks using machine learning based on Deep Packet Inspection (Doshi et al. (2018), Meidan et al. (2018), Saad et al. (2011)) or they used SDN with additional security characteristics such as authentication features (Bhunia & Gurusamy 2017) (see Subsection 3.1.1). However, to the time this thesis was written, there has not been done any research on WSN attack detection using SDN.

The goal of this work was, therefore, to have a generalized approach for attack and attacker classification that is scaleable for different network sizes and topologies and being extensible by the means of adding additional attack features as it has been discussed in Subsection 3.1.1 of Chapter 3. The main reason for the addition of an anomaly detector was to have a system where one can change the attack to be detected based on inputting appropriate SDN network features. Furthermore, it helped to prevent false positives due to attack traffic forwarding when trying to identify the attacker. Using RADAR, these problems could be avoided.

We have not only simulated and recognized DoS attacks, but we also implemented the Node Replication attack, which is known to be difficult to detect, as the effect is often minimal or not directly visible at all. Chapter 3 showed how SDN can help to make this attack type observable and what features can be used to obtain the necessary anomaly information. It was shown that we can achieve similar or even better performance

results than compared to several related work Grover et al. (2011), Zhu et al. (2007), Parno et al. (2005) and Conti et al. (2007). But it has to be considered with caution, because first, the results have been achieved using different implementations of the node replication attack, different topologies and secondly, the terms "performance", "success rate" and "probability of detection" are not clearly defined. Therefore, the comparability is questionable. Nevertheless, an accuracy of 96.61% is relatively high, and compared to the results of DoS detection (96.44%), there is evidence that our detection algorithm for node replication attacks works well.

The experiments described in Chapter 4 showed that our framework can achieve similarly high accuracy of 96.44% compared to the results published by Doshi et al. (2018) of 99% for DoS attack detection in IoT. Additionally, we have seen that our approach may even perform better because it recognizes attacks and identifies the attacker only from anomaly scores rather than from raw features. Using such data, it is conceivable that novel attacks could be detected as well.

In the end, in Chapter 5, we described our implicit contribution in terms of bugfixes to the SDN-WISE implementation for ONOS and Contiki.

6.2 Conclusion

There is a huge variety of proposals for attack detection in IoT networks using machine learning or SDN available. But to our knowledge at this time, all of these approaches either consider DoS attacks only, use Deep Packet Inspection in order to obtain the relevant data for classification or are distributed algorithms that increase the network overhead additionally. DPI has the clear drawback of requiring intelligent gateways that forward all traffic to an analytics controller or have to classify traffic on their own. To overcome this issue, we propose to use SDN in combination with distributed knowledge, anomaly detection and machine learning. This thesis showed the architecture of our framework, an evaluation and comparison with other research studies.

In this thesis we could show that, it is possible to recognize attacks from structural data, such as the network topology, and network statistics that are checked for consistency. Furthermore, we showed that using the output of RADAR, attackers can be identified efficiently. Besides, we could show how new attacks can be recognized using this framework, by defining additional input features for RADAR.

6.3 Future Work

In this Section, a few parts that might be interesting for further research are highlighted.

This thesis focused on the potential for security mechanisms that SDN brings to WSNs. Although, security as a standalone topic is omnipresent in current research, very few studies have focused on the opportunity for SDN in WSNs. Especially, WSN security is getting more and more attention due to the new upcoming technologies like Machine to Machine Communication (M2M), 5G, etc. Thus, the future work could be to research

novel attacks and find ways to identify and mitigate them, for example by using the framework proposed in this thesis.

Generally it is true that with the increase of interconnection and interaction which is typical in WSNs, greater security risks come up. Additionally, there is a lack of studies which focus on the security issues of SDN. For example, how can a nodes establish trust against a certain SDN controller? And how could the control plane be secured in SDN-WISE?

Additionally, as the experiments of this thesis were done in a simulated environment using COOJA, it would be interesting to see how the framework performs in a real environment. Furthermore, in most Wireless Sensor and Actuator Networks we have heterogeneous nodes, rather than nodes of a single type. This could create completely different traffic patterns with data streams from high-bandwidth devices such as cameras or microphones. A lot of further research could be done therefore in this field.

Bibliography

Airehrour, D., Gutierrez, J. A. & Ray, S. K. (2018), 'SecTrust -RPL: A secure trust-aware RPL routing protocol for Internet of Things', *Future Generation Computer Systems* . **URL:** *https://linkinghub.elsevier.com/retrieve/pii/S0167739X17306581*

Bhardwaj, K., Miranda, J. C. & Gavrilovska, A. (2018), 'Towards IoT-DDoS Prevention Using Edge Computing', p. 7.

Bhunia, S. S. & Gurusamy, M. (2017), Dynamic attack detection and mitigation in IoT using SDN, IEEE, pp. 1–6.
URL: *http://ieeexplore.ieee.org/document/8215418/*

Conti, M., Di Pietro, R., Mancini, L. V. & Mei, A. (2007), A randomized, efficient, and distributed protocol for the detection of node replication attacks in wireless sensor networks, *in* 'Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing - MobiHoc '07', ACM Press, Montreal, Quebec, Canada, p. 80.
URL: *http://portal.acm.org/citation.cfm?doid=1288107.1288119*

Czosseck, C. & Geers, K., eds (2009), *The virtual battlefield: perspectives on cyber warfare*, number v. 3 *in* 'Cryptology and information security series', IOS Press, Amsterdam ; Washington, DC. OCLC: ocn489010240.

Djedjig, N., Tandjaoui, D. & Medjek, F. (2015), Trust-based RPL for the Internet of Things, *in* '2015 IEEE Symposium on Computers and Communication (ISCC)', IEEE, Larnaca, pp. 962–967.
URL: *http://ieeexplore.ieee.org/document/7405638/*

Doshi, R., Apthorpe, N. & Feamster, N. (2018), 'Machine Learning DDoS Detection for Consumer Internet of Things Devices', *arXiv:1804.04159 [cs]* . arXiv: 1804.04159.
URL: *http://arxiv.org/abs/1804.04159*

Douligeris, C. & Mitrokotsa, A. (2004), 'DDoS attacks and defense mechanisms: classification and state-of-the-art', *Computer Networks* **44**(5), 643–666.
URL: *http://linkinghub.elsevier.com/retrieve/pii/S1389128603004250*

Esposito, P. M., Campista, M. E. M., Moraes, I. M., Costa, L. H. M. K., Duarte, O. C. M. B. & Rubinstein, M. G. (2008), Implementing the Expected Transmission Time Metric for OLSR Wireless Mesh Networks, *in* '2008 1st IFIP Wireless Days', IEEE, Dubai, United Arab Emirates, pp. 1–5.
URL: *http://ieeexplore.ieee.org/document/4812900/*

Galluccio, L., Milardo, S., Morabito, G. & Palazzo, S. (2015), SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks, IEEE, pp. 513–521.
URL: *http://ieeexplore.ieee.org/document/7218418/*

Grover, J., Prajapati, N. K., Laxmi, V. & Gaur, M. S. (2011), Machine Learning Approach for Multiple Misbehavior Detection in VANET, *in* A. Abraham, J. L. Mauri, J. F. Buford, J. Suzuki & S. M. Thampi, eds, 'Advances in Computing and Communications', Vol. 192, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 644–653.
URL: *http://link.springer.com/10.1007/978-3-642-22720-2*$_6$8

Hurni, P., Anwander, M., Wagenknecht, G., Staub, T. & Braun, T. (2012), TARWIS — A testbed management architecture for wireless sensor network testbeds, *in* '2012 IEEE Network Operations and Management Symposium', IEEE, Maui, HI, pp. 611–614.
URL: *http://ieeexplore.ieee.org/document/6211968/*

Karlof, C. & Wagner, D. (2003), 'Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures', p. 15.

Li, J., Dani, H., Hu, X. & Liu, H. (2017), Radar: Residual Analysis for Anomaly Detection in Attributed Networks, *in* 'Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence', International Joint Conferences on Artificial Intelligence Organization, Melbourne, Australia, pp. 2152–2158.
URL: *https://www.ijcai.org/proceedings/2017/299*

Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Breitenbacher, D., Shabtai, A. & Elovici, Y. (2018), 'N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders', *arXiv:1805.03409 [cs]* . arXiv: 1805.03409.
URL: *http://arxiv.org/abs/1805.03409*

Padmavathi, D. G. & Shanmugapriya, M. D. (2009), 'A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks', *arXiv:0909.0576 [cs]* . arXiv: 0909.0576.
URL: *http://arxiv.org/abs/0909.0576*

Parno, B., Perrig, A. & Gligor, V. (2005), Distributed Detection of Node Replication Attacks in Sensor Networks, *in* '2005 IEEE Symposium on Security and Privacy (S&P'05)', IEEE, Oakland, CA, USA, pp. 49–63.
URL: *http://ieeexplore.ieee.org/document/1425058/*

Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Wei Lu, Felix, J. & Hakimian, P. (2011), Detecting P2p botnets through network behavior analysis and machine learning, *in* '2011 Ninth Annual International Conference on Privacy, Security and Trust', IEEE, Montreal, QC, pp. 174–180.
URL: *http://ieeexplore.ieee.org/document/5971980/*

Schaerer, J. (2018), 'SDNWisebed: A Software-Defined Wireless Sensor Network Testbed', p. 82.

Schaerer, J., Zhao, Z. & Braun, T. (2018), 'DTARP: A Dynamic Traffic Aware Routing Protocol for Wireless Sensor Networks', p. 6.

Siddiqui, M. S. (2007), 'Security Issues in Wireless Mesh Networks', p. 6.

Sood, K., Yu, S. & Xiang, Y. (2016), 'Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review', *IEEE Internet of Things Journal* **3**(4), 453–463.
URL: *http://ieeexplore.ieee.org/document/7279061/*

Wani, A. & Revathi, S. (2018), Analyzing Threats of IoT Networks Using SDN Based Intrusion Detection System (SDIoT-IDS), *in* P. Bhattacharyya, H. G. Sastry, V. Marriboyina & R. Sharma, eds, 'Smart and Innovative Trends in Next Generation Computing Technologies', Vol. 828, Springer Singapore, Singapore, pp. 536–542.
URL: *http://link.springer.com/10.1007/978-981-10-8660-1*$_4$1

Zhu, B., Addada, V. G. K., Jajodia, S. & Roy, S. (2007), 'Efcient Distributed Detection of Node Replication Attacks in Sensor Networks', p. 10.

Erklärung

gemäss Art. 30 RSL Phil.-nat. 18

Name/Vorname: Severin Zumbrunn

Matrikelnummer: 13-117-346

Studiengang: Master of Science in Computer Science

Bachelor ☐ Master ☑ Dissertation ☐

Titel der Arbeit: SDN-WISE Anti-Attack

LeiterIn der Arbeit: Prof. Dr. T. Braun

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.
Für die Zwecke der Begutachtung und der Überprüfung der Einhaltung der Selbständigkeitserklärung bzw. der Reglemente betreffend Plagiate erteile ich der Universität Bern das Recht, die dazu erforderlichen Personendaten zu bearbeiten und Nutzungshandlungen vorzunehmen, insbesondere die schriftliche Arbeit zu vervielfältigen und dauerhaft in einer Datenbank zu speichern sowie diese zur Überprüfung von Arbeiten Dritter zu verwenden oder hierzu zur Verfügung zu stellen.

Bern/22.01.2019

Ort/Datum

Unterschrift