

# Applying Machine Learning for Identifying Attacks at Run-Time

Nurit Devir



# Applying Machine Learning for Identifying Attacks at Run-Time

Research Thesis

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Computer Science

**Nurit Devir**

Submitted to the Senate  
of the Technion — Israel Institute of Technology  
Adar aleph 5779      Haifa      February 2019



This research was carried out under the supervision of Prof. Orna Grumberg and Prof. Shaul Markovitch, in the Faculty of Computer Science.

## Acknowledgements

First of all, I would like to thank my main advisor, Prof. Orna Grumberg. Thank you for the weekly meetings, the willingness to help even when the field is relatively far from your main research subjects. Thanks for the encouragement, guidance and attention to even the smallest details. Thank you for being available for me at any time, sometimes even in less conventional hours. I won an amazing supervisor, both professionally and personally, and I would like to thank you for making this time so pleasant for me.

I would also like to thank my associate advisor, Prof. Shaul Markovitch, for devoting time and thought in busy and stressful times. Thank you for the additional points of view you gave me, about the new ideas and sharp thinking. Thank you for your great part in this work.

I would also like to thank Dr. Gabi Nakibly. Thank you for the direction and the exposure to a field which I was not familiar with. Thank you for your willingness to contact various organizations in order to contribute to the quality of the research. Thank you for your cooperation and for your help and advices during your research.

This research was partially supported by the Technion Hiroshi Fujiwara cyber security research center in the Technion and I would like to thank them for it.

I would like to thank my friend Harel Cain. Thank you for your availability, for your desire to invest thought and time for promoting my research. Thanks for the tips you gave me along the way. Knowing that I had someone who would gladly help me with any problem, small or big, gave me a sense of calm. Thank you for the good feeling and the great help, even in less convenient times.

I would like to thank my friends who shared this journey with me. Thank you for being there even in more stressful times. Thank you for your encouragement, support, and help when needed. This way would have been much harder without you.

Finally, I would like to thank my family. My parents, Menachem and Shoshi, my sister Michal, my brother and his wife - Ori and Reut, and my three nephews - Ariel, Yoav and Shachar. Thank you for inserting joy and happiness to this period. Thank you for your help, for your endless support and love. Thank you for always being there for me. Thanks to my parents for teaching me to aim high, to invest and not to despair, even when you still do not see the light at the end of the tunnel. Thank you for pushing me and help at any time. Without you it would not have happened. Special thanks to my sister Michal for the great professional help. I am grateful to you for sharing your great knowledge and creativity with me, in a pleasant and loving way. It is said that you do not choose your family, but if I had the ability to choose - I would choose you.

The generous financial help of the Technion is gratefully acknowledged.



# Contents

## List of Figures

<b>Abstract</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
<b>2 Preliminaries</b>	<b>7</b>
2.1 OSPF Basics . . . . .	7
2.2 Cisco implementation of OSPF . . . . .	9
2.3 Formal Modeling . . . . .	9
<b>3 Threat Model</b>	<b>11</b>
<b>4 Applying Machine Learning for Identifying Attacks at Run-Time</b>	<b>13</b>
4.1 Introduction . . . . .	13
4.2 Problem definition . . . . .	13
4.3 Solution outline . . . . .	14
4.4 Example generation by simulation . . . . .	14
4.5 Example tagging . . . . .	15
4.5.1 Estimating the attack severity . . . . .	16
4.5.2 Ordered categorical tags . . . . .	17
4.6 The Features . . . . .	17
4.7 The Learning Algorithm . . . . .	18
4.7.1 Choosing a Split . . . . .	19
4.7.2 Determining the Final Tag . . . . .	21
4.7.3 Feature Importance . . . . .	22
<b>5 Empirical Evaluation</b>	<b>25</b>
5.1 Experimental Methodology . . . . .	25
5.2 The Network Topology Generator . . . . .	26
5.3 System Performance . . . . .	27
5.4 Feature Importance . . . . .	29
5.5 Transfer Learning Setup . . . . .	30

5.5.1	Introduction . . . . .	30
5.5.2	The Target Belongs to the Same Family as the Source . . . . .	30
5.5.3	The Target Belongs to a Different Family from the Source . . . . .	31
5.6	Dynamic Networks . . . . .	32
5.6.1	Introduction . . . . .	32
5.6.2	Topology Changes Between the Training and Prediction Phases .	33
5.6.3	Topology Changes During Training and Prediction Phases . . . . .	33
5.7	Detection Times . . . . .	35
5.8	Evaluation with Real Data . . . . .	36
5.8.1	Introduction . . . . .	36
5.8.2	Real OSPF Data . . . . .	36
5.8.3	Results . . . . .	37
<b>6</b>	<b>The Monitor Placement Algorithm</b>	<b>39</b>
6.1	Introduction . . . . .	39
6.2	Locating One Monitor . . . . .	39
6.3	Locating More Than One Monitor . . . . .	41
6.4	Optimal Set of Monitors . . . . .	44
<b>7</b>	<b>Related Work</b>	<b>49</b>
<b>8</b>	<b>Discussion</b>	<b>51</b>
8.1	Applying the Framework to Other Protocols . . . . .	51
8.2	Working with Balanced Dataset . . . . .	51
8.3	Accuracy Measures . . . . .	52
8.3.1	Classification Measures . . . . .	52
8.3.2	Experiments with the New Terminology . . . . .	53
<b>9</b>	<b>Conclusions and Future Work</b>	<b>55</b>
9.1	Conclusions . . . . .	55
9.2	Future Work . . . . .	55
	<b>Hebrew Abstract</b>	<b>i</b>



# List of Figures

4.1	Illustration of choosing a feature for the split. The left side demonstrates the using of feature $f_1$ for the split and the left side demonstrates the using of feature $f_2$ ('+' represents the tag 0 while '-' represents the tag 1).	20
5.1	A specific topology created by the algorithm . . . . .	28
5.2	Several rounds of cross validation algorithm (from Wikipedia) . . . . .	29
5.3	Weighted accuracy as a function of the number of examples used in the training process . . . . .	30
5.4	Weighted accuracy as a function of the n and d. . . . .	31
5.5	Weighted accuracy as a function of the distance between the target and the source families . . . . .	32
5.6	Weighted accuracy as a function of delta. Topologies with 6, 12 and 16 routers, from left to right respectively. . . . .	33
5.7	The organization's network topology . . . . .	37
6.1	Weighted accuracy as a function of the number of monitors. Topologies with 12 and 16 routers, from left to right, respectively. . . . .	45



# Abstract

With the increase in malicious activity over the Internet, it has become extremely important to build tools for automatic detection of such activity. There have been attempts to use machine learning to detect network attacks, but the difficulty in obtaining positive (attack) examples, led to using one-class methods for anomaly detection.

In this work we present a novel framework for using multi-class learning to induce a real-time attack detector. We designed a network simulator that is used to produce network activity. The simulator includes an attacker that stochastically violates the normal activity, yielding positive as well as negative examples. We have also designed a set of features that withstand changes in the network topology. Given the set of tagged feature vectors, we can then apply a learning algorithm to produce a multi-class attack detector. In addition, our framework allows the user to define a cost matrix for specifying the cost for each type of detection error.

Our framework was tested in a wide variety of network topologies and succeeded to detect attacks with a high accuracy. We have also shown that our system is capable of handling a transfer learning setup, where the detector is learned on one network topology but is used on another topology from the same family. Another setup we tested is dynamic networks in which changes take place in the topologies. Finally, we also referred to choosing the router(s) which should be chosen to record the traffic and transfer this information to the detector, in order to achieve high performances.

We anticipate the presented framework will enable any organization to defend itself with an attack detector that is automatically adapted to its particular setting.



# Chapter 1

## Introduction

Over the years, the use of the Internet has become increasingly widespread and its applications became extremely wide with enormous importance to billions of people around the world. In parallel, malicious attacks have become a critical threat to the Internet activity all over the world. Thus, it has become extremely important to build tools for automatic detection of such activities.

Network intrusion detectors are designed to spot malicious activity by identifying protocol violations. It is desirable to detect such activity as early as possible to prevent the attacker from damaging the infrastructure. There are many challenges making the intrusion detection task harder than other areas [SP10]. One main challenge is related to the background in which such a system works. In this field of attack detection, there is an ongoing race between attackers and defenders. Each one tries to improve its own abilities. Thus, the attackers try to hide their actions and make them similar to a normal behavior of the system. Such an attack sometimes involves a small number of actions, and these actions can be identical to legal actions in term of their structure [SGN13].

Most initial attempts to build attack detectors relied on manually defining rules or patterns that characterize attack activities [HS14, GM14]. Such an approach suffers from several weaknesses. First, it is very difficult to design such a set of rules that covers a wide variety of attacks. Second, the set of rules will usually be able to detect only known attacks. To overcome these difficulties, new example-based machine-learning based methods were developed. Since positive examples (of attack activity) are very difficult to obtain, strong binary or multi-class algorithms could not be used, and these solutions had to resort to weaker one-class learning approach to create a model of trustworthy activity, and then compare new behaviors against this model [TDJ<sup>+</sup>14]. Such an *anomaly detection* approach has significant difficulties in intrusion detection context as network traffic is often very diverse, where the bandwidth or the duration of connections can exhibit immense variability. These changes are normal and occur in networks on an ongoing basis, which makes it difficult to characterize normal behavior.

To account for the difficulties of characterizing normal behavior, several works

have made an attempt of using multi-class learning algorithms for the detection task [SM03, CFF16, KJS17]. These works assumed the existence of a relevant tagged set of examples, both positive and negative. This assumption, however, may not hold in practice. First, the strategies of the attackers are continuously developed and evolved. Thus, a model trained on previous attacks would have a problem identifying newer types of attacks. Second, an attack may exploit characteristics of the particular network topology. Therefore, a model trained on other topologies may not perform well. Third, obtaining a real multi-class training set for intrusion detection is extremely difficult [SP10]. It is especially difficult to obtain positive examples of attacks, mainly due the potential sensitivity of the data.

In this work, we present a new framework for example-based learning of an intrusion detector. Our approach overcomes the lack of datasets by including a simulation-based example generator. The example generation algorithm simulates both attacks and normal activity in a given network topology. The network activity records are then passed to an automatic tagging module and then to the feature extractor that derives a special set of features, yielding a multi-class training set. Our framework allows the user to specify a cost matrix (for identifying the cost for each error type). We therefore developed a cost-sensitive version of Random Forest algorithm, which is a well known learning algorithm that uses a committee of decision trees. We used our learning algorithm on the generated training set to induce an intrusion detector tailored to the given network topology.

We have implemented this framework for the widely used routing protocol OSPF. This protocol allows routers to calculate their routing tables within a cluster of networks - an autonomous system (AS). An attacker that leverages OSPF to attack an AS may have a catastrophic effect on it. A single malicious router within an AS can poison the routing tables of all other routers of that AS by sending false routing messages, thereby subverting the entire routing process. Finding attacks on the routing protocol is a demanding task as the exact nature of the attack may be unknown. We performed an extensive series of experiments to test our framework characteristics. Our experimentation environment includes a network topology generator that enables us to test our algorithm in a variety of environments. Our empirical evaluation showed promising results, with a very low error rate. We have shown that the learned models can withstand situations of transfer learning and dynamic networks, where the learned model is applied to a different topology than the one trained on. In addition, we propose a way to choose the location where the detector should be placed in order to get a high accuracy rate.

Our new framework has a significant practical potential. An organization can easily define its network topology and cost matrix and start the training process. The system will produce a runtime intrusion detector that is adapted to the specific topology and cost matrix. Furthermore, as we later show, our algorithm can recommend the particular router where the detector should be located at.

Our work is an involved application of learning techniques for enhancing security in the Internet. The main contribution of our work is the development of an adaptive learning-based runtime intrusion detector. Our framework is unique in that it adapts itself to the user’s specific setting (topology, severity of attacks, cost of different misclassification errors, and more). This framework will enable an organization to defend itself against malicious activities, according to its needs. Specifically, our contributions include:

- Generating examples that are relevant to a specific desired topology.
- Enabling the user to define severity of attacks.
- Learning a model that withstands transfer learning and dynamic networks setups.
- Using a cost sensitive version of a known learning algorithm. Our modifications of the algorithm take into account a cost matrix given as an input.
- Recommending a particular router where the intrusion detector should be located at, for achieving a high accuracy rate.

This work is structured as follows: Chapter 2 presents the OSPF protocol, its standards and the implementations. Chapter 3 describes the threat model we used. Chapter 4 formally defines the problem and presents our solution components. Chapter 5 describes the experiments we conducted, including setups of transfer learning and dynamic networks. Chapter 6 discusses the question of which router(s) needs to be chosen to locate the detector at, and proposes an answer to that question. Chapter 7 reviews previous works done in the field. Finally, Chapters 8 and 9 discuss our framework, summarize the results achieved and present a possibility to extend the work in the future.





## Chapter 2

# Preliminaries

### 2.1 OSPF Basics

When talking about computer networking, a routing table is a database held by every router in the network. This table tells the router to which of its neighbors it should forward a packet in order for it to reach its destination through the optimal route.

Open Shortest Path First (OSPF) is one of the most widely used interior gateway routing protocols on the Internet. Its aim is to allow routers within a single autonomous system (AS) to construct their routing tables, while dynamically adapting to changes in the autonomous system's topology. OSPF is currently used within many autonomous systems on the Internet. It was developed and standardized by the IETF's OSPF working group. On a given network, each one of the routers sends its neighbors a message called "Link State Advertisement" (LSA). The main field in this message contains the links to the neighbors known to the router who generated the LSA, and their costs. We refer to this field as *Links*. This information actually describes the router's local view of the network. When a router receives such a message from another router, it updates its view about the topology accordingly and floods the message to all of its other neighbors. At the end of this process, each one of the routers has a database which contains the up-to-date information about the network and about the costs of sending a message from one router to its neighbors. According to this information, using Dijkstra's algorithm [DIJ59], a next hop is derived for each destination, which forms the router's routing table. Now, if router  $x$  wants to send a message to router  $y$ , router  $x$  looks at its routing table and sends the information to the neighbor so that the overall route to router  $y$  will be of a minimal cost. From the description above, it is obvious that an attacker who compromised a router residing within the AS can easily manipulate the routing information distributed by the protocol. It may block the flooding of LSAs to some of its neighbors, change the contents of LSAs before sending them to other routers or simply advertise false LSAs on behalf of the compromised router or other routers in the AS. Using such manipulations, an attacker may poison the routing tables of all other routers in the AS, thereby controlling the routing process in

the AS. Such control may allow the attacker to re-route selected traffic streams through the router it controls in order to eavesdrop or alter the traffic. It may also substantially degrade the performance of the AS by selecting longer routes or disconnecting portions of the AS all together. Indeed several works have shown different attacks that abuse OSPF.

The OSPF protocol dynamically adapts to changes in the network: every few seconds, each router sends a ‘hello’ message to its neighbors in order to inform them it is alive. This type of message is sent in a fixed time interval (“hello interval”). If a router does not receive such a message from its neighbor within a certain predefined period of time, it infers that the connection between them went down and they are no longer neighbors. In the opposite case, when a router receives a ‘hello’ message from a new router that has not sent such a message before, it infers that a connection between them was inserted and they are new neighbors. According to this knowledge, the router updates its information about the network topology and sends a new LSA with its updated local view. In addition, every 30 minutes the router sends a new LSA instance, even if there is no change since the last one. This action is called “refresh the LSA”.

**Definition 2.1.1 (LSA fields).** The following are some of the relevant fields included in an LSA:

- **Sequence number** - a serial number of the LSA. This number can not exceed the value of “MaxSeqNum” parameter. This number increases by one on every new instance of the LSA, which is either sent periodically or sent when the router detects a change in the network. An LSA with a lower sequence number will be replaced by an LSA of a higher sequence number that was originated by the same router. The replacement will take place in the database of the router that received the LSAs.
- **Age** - the time in seconds since the LSA was originated. This field is increased by one every passing second as long as the LSA is stored in the router’s database, and also when the LSA is moved to another router. This field is never incremented past “MaxAge” parameter. When an LSA reaches the value of “MaxAge” in its Age field, it is finally flushed from the router database.
- **Advertising router** - this field identifies the router which generates this LSA.
- **Links** - as described, this field contains the information about the links to the router’s neighbors that the advertising router knows about, and their costs.

We also add some more fields that contain more information about the LSA:

- **From** - the router that sent this LSA.
- **To** - the router that received this LSA.

- **Time** - the absolute time in which the LSA was sent or received in the network.

An *OSPF run*  $\pi$  is a sequence of LSAs, sent or received by the routers of the network, according to the protocol rules. For the problem we face in this work, we need to define the following:

**Definition 2.1.2 (run from the point of view of a router).** Given an OSPF run  $\pi$  and a router  $r$ , a run from the point of view of  $r$ , denoted  $\pi_r$ , is a sequence of LSAs, obtained by restricting  $\pi$  to those LSAs that were sent or received by the router  $r$  ( $r$  appears in the To or From fields of the LSAs).

OSPF has a mechanism called **fight-back** which helps it to deal with attacks: When a router receives an LSA which claims to be generated by itself, and after some checks finds out that this LSA was not generated by itself, it sends a new LSA. This new LSA contains the correct information about its links. It also includes a sequence number which is higher by one than the sequence number of the malicious LSA. The new LSA is also flooded in the system and because its sequence number is higher, it replaces the malicious LSA in the databases of all other routers. This will cause the attack to fail of being permanent in the system. Note that because of the flooding action, if an attacker sends an LSA on behalf of a router, this router will often receive this LSA and “understand” that something is wrong, because it did not originate such an LSA. This router can then activate the fight-back mechanism.

In spite of the fight-back mechanism, previous works [SGN13] show that OSPF is vulnerable to attacks. As mentioned, the goal of our work is to detect and alert of attacks at runtime, when they actually occur.

## 2.2 Cisco implementation of OSPF

In this research, we are going to track the traffic messages and will try to find properties that indicate that the system is under attack. In addition, we want our work to be used in the real world - real OSPF running. For that, we checked the commands Cisco provides the users in order to see the OSPF running mode in a specific time. Our learning will be based on these commands, so users will be able to use our model in a real OSPF setting.

## 2.3 Formal Modeling

Some previous works analyzed OSPF protocol. Most of them have not modeled the protocol in its entirety but rather abstract away some details (see, for instance, [MKSUKW12] and [SGN13]). In our work, we decided not to model the routing table calculation. This is because we would like to detect an attack, even if it does not affect the routing table. Note that the routing table calculation is based on Dijkstra’s

algorithm and therefore significantly increases the running time of the protocol. Thus, implementing it will add unnecessary overhead to the process of generating runs which will be examples to the learning algorithm.

Our modeling is suitable for static topologies, in which the links between routers are unchanged. It is also expanded to dynamic networks, in which changes occur in the structure of the topology.

The modeled functionality includes the LSA message structure, the LSA flooding procedure, the fight-back mechanism, the hello message structure, the refresh LSA procedure and the flushing operation.

We implemented this protocol using Python. The protocol was simulated on a variety of topologies, each one of them contains legitimate OSPF routers and can also contain a single malicious router. For the protocol simulation, we used an existing code <sup>1</sup> and extended it to our needs.

---

<sup>1</sup><https://github.com/audm/ospf-sim>

## Chapter 3

# Threat Model

We adopt the common threat model found in the literature [WVW97, WCJ<sup>+</sup>, JM06, NKGB12]. This model assumes the attacker has the ability to send LSAs to at least one valid router in the AS and that router processes them as valid LSAs. This assumption can be trivially achieved by an insider, namely an attacker who gained control over just a *single* router in the AS. We assume nothing about the location of the compromised router. The attacker can gain control of a router, for example, by remotely exploiting an implementation vulnerability on the router. Several such vulnerabilities have been published in the past (e.g., CVE-2018-0167, CVE-2018-0175, CVE-2015-0235).

OSPF employs per-link LSA authentication where each link is associated with a secret shared by all the routers directly attached to that link. A router authenticates a received LSA using the secret associated with the link on which it was received. Note that no end-to-end authentication is employed in OSPF. The threat model assumes that the attacker knows only the secrets associated with the links that are directly attached to the router it controls. It does not know the secrets associated with other links in the AS. Consequently, the attacker may only be able to send false LSAs to its immediate neighbors (which they in turn flood to their neighbors). Specifically, the attacker cannot send an LSA directly to a remote router.

We assume that attacker may be able to do illegitimate actions. The set of these actions is described in detail in Section 4.4. In this work we monitor the routing protocol traffic, which allows the detection system to identify attacks as soon as they unfold, even before they have an affect on the routers' routing tables. Identifying an attack based on changes to the routing tables may be too late - the damage has already been done.

We assume the defender has one or more monitoring points on the routers in the AS. Detection logic may rely on information gleaned from those routers only. Specifically, this includes the OSPF traffic that passes through the monitored routers. We assume the detection system does *not* have a global view of the routing state of the AS.



## Chapter 4

# Applying Machine Learning for Identifying Attacks at Run-Time

### 4.1 Introduction

In the following sections, we formally define a learning problem and discuss the different components of the learning algorithm. In general, a learner is given a set of examples, each of which is tagged with a value describing the subset it belongs to. The goal is to build a classifier that can identify the tag of a new, previously unseen, example. Each example is first transformed into a set of features and then analyzed by the learning algorithm.

In our setting, the examples are runs of the OSPF protocol from the point of view of a specific router, called *monitor*. Our goal is to identify whether the run contains an attack, and tell how severe the attack is.

We now describe a learning problem more formally [Fri17]: Let  $O$  be a set of objects. Let  $L = \{l_0, \dots, l_{k-1}\}$  be a set of  $k$  possible labels, also called “tags”. Let  $f : O \rightarrow L$  be a function which gets  $o \in O$  and returns its label  $l \in L$ . Let  $D \subseteq \{(o, f(o)) | o \in O\}$  be a set of tagged examples. Let  $F = \{f_1, \dots, f_n\}$  be a set of functions  $f_i : O \rightarrow I_i$  when  $I_i$  is some domain representing possible values of the feature  $f_i$  of an object. Then, the training set is represented by feature vectors:  $S = \{(\langle f_1(o_i), \dots, f_n(o_i) \rangle, y_i) | (o_i, y_i) \in D\}$ , where  $y_i$  is the tag of  $o_i$  and  $\langle f_1(o_i), \dots, f_n(o_i) \rangle$  is the feature vector associated with  $o_i$ . A learning algorithm takes  $S$  as an input and finds a function  $g : O \rightarrow L$ , called a classifier, which approximates the function  $f$ .

### 4.2 Problem definition

Let  $N = (V, E)$  be a graph representing a network topology, where  $V$  represents the set of routers and  $E$  the set of connections. Let  $M \in V$  be a router, denoted as the *monitor*. Let  $\pi_M$  be a run from the point of view of the monitor  $M$ . Let  $T = \langle t_0, \dots, t_{k-1} \rangle$  be a set of attack categories ordered by their severity (where  $t_0$  represents “no attack

in the system”). The motivation for using ordered categories is to provide the user with well-defined severity levels rather than continuous values, which are difficult to interpret. Such categorical tags can tell an organization how severe the attacker actions are, according to its own definition of severity, and can help it to arrive at a more well-informed decision on how to treat a detected attack. We assume the existence of an oracle  $o$  that receives  $\pi_M$  and returns its true category in  $T$ . Our goal is to build  $o'$ , an approximation of  $o$ .

Furthermore, we assume that different types of errors may carry different costs, and these costs are specified by a given  $k \times k$  *cost matrix*  $C$ , whose  $C_{i,j}$  entry defines the cost of assigning the tag  $t_i$  to an instance whose actual (real) tag is  $t_j$ . Note that typically, entries on the main diagonal of this matrix are all 0 (no error). Our goal is to minimize the error cost according to  $C$ .

### 4.3 Solution outline

We propose to solve the above problem using a machine learning approach. There are several obstacles in applying learning for this problem, mainly the lack of examples. In the following subsections we describe the components of our solution:

1. A simulation-based example generator.
2. An analysis-based tagging algorithm.
3. An OSPF-based feature extractor.
4. A cost sensitive learning algorithm.

### 4.4 Example generation by simulation

One significant challenge in training a model to be used as an attack detector is the lack of public datasets to train the model with. In this section we present an alternative approach, where the learning system produces its own training examples using simulations.

The example generator works by simulating normal OSPF activity and periodically simulating an attack. When simulating the OSPF runs, some of those runs contain an attacker where others are normal. On each run, one of the routers is chosen to act as the monitor, which means that the messages received and sent by this router are recorded. Our simulations also contains an attacker defined as follows.

**Definition 4.4.1 (Attacker).** An attacker is an entity with predefined set of capabilities. The attacker takes control of one of the routers in the network and can do the following actions:



- When getting an LSA, the attacker can change the *Links* field. In addition, the attacker can decide which of its neighbors to forward the LSA to, if any. The motivation of such an action is to falsify the knowledge the other routers have about the topology without doing something that can clearly reveal the attack (such an action requires only the changing of the *Links* field of the LSA).
- The attacker can also generate a new LSA by choosing arbitrary values for the fields and deciding which of its neighbors to send this message to.

In this model, an attack is initiated by any action made by the attacking router, which is not consistent with a normal protocol behavior, even if that action did not have an effect on the routing process. The definition above contains a large collection of capabilities from which the attacker can randomly generate different combinations of actions resulting in different attacks. We do not restrict the attacker’s sequence of actions at all. This means that the attacker can choose an action in every step, thus creating a sequence of messages that constitutes an unknown attack. By doing that, we enable the attacker to generate new attacks and our framework would be able to identify them.

Given a network topology  $N$  and a monitor router  $M$ , we generate  $m$  runs. At each run, we stochastically decide if the run contains an attacker  $A$ , and if so, which router is the attacker. We simulate the OSPF protocol in a distributed manner. In addition to the protocol simulation, the attacker can inject and execute actions that violate the protocol, according to its capabilities, described above.

During a run  $\pi$ , we save records of  $\pi_M$  and  $\pi_A$  (the run from the points of view of the monitor and the attacker, respectively). Once the simulation of  $\pi$  is finished (after a predefined number of steps  $\rho$ ), we partition  $\pi_M$  and  $\pi_A$  to prefixes at fixed time intervals. This way, at a specific point in time, we have the prefixes of  $\pi_M$  and  $\pi_A$ , up to that time. For a router  $r$ , we denote by  $PR(\pi_r, t)$  the prefix of  $\pi_r$  at time  $t$ , i.e., the sequence of messages in  $\pi_r$  in the time interval  $[0, t]$ .

Both, the attacker prefixes and the monitor prefixes are later used by our tagging algorithm. However, only the monitor prefixes will be available during detection, and therefore, only these prefixes will be used for extracting features during training. Note that using these prefixes simulates real-time mode in which we are given a run up to a certain point in time. The example generation algorithm is listed in Algorithm 4.1.

## 4.5 Example tagging

The tagging algorithm receives a pair of prefixes, the monitor prefix and the attacker prefix, from the example generator, and produces a tagged example. The tagging process consists of two steps: first, computing a continuous value reflecting the severity of the attack, and second, transforming the value into one of the categories in  $T$ .

---

**Algorithm 4.1** GENERATEEXAMPLES( $N = (V, E)$ ,  $M$ )

---

```
1:  $S \leftarrow \emptyset$ 
2: for  $i = 1, \dots, m$  do
3:    $attack\_flag \leftarrow$  flip a coin
4:   if  $attack\_flag$  then
5:      $A \leftarrow random\_selection(V)$   $\triangleright$  choose router A as attacker
6:   simulate the OSPF protocol on  $N$  for  $\rho$  time units.
7:   for  $j = 1, \dots, \rho$  do
8:      $S \leftarrow S \cup \{ \langle PR(\pi_M, j), PR(\pi_A, j) \rangle \}$ 
```

---

#### 4.5.1 Estimating the attack severity

The goal of the first step is to assign the generated example a value reflecting the severity of the attack. For doing so, we use the *IllegalActionCost* field, added to every LSA with an initial value of 0. During the simulation, if the attacker performs an action that deviates from the protocol, *IllegalActionCost* is assigned an integer positive value, reflecting the severity of this deviation. For example, if according to the protocol, a message should be sent to a set of neighbors, and the attacker blocks it from getting to some of them, the severity of the attack is proportional to the number of neighbors blocked. Note that the value of the *IllegalActionCost* field can only be determined during the simulation, and not after it, since only then we know what was expected by the protocol and how the attacker actually acted.

To determine the continuous value of a monitor prefix  $PR(\pi_M, t)$ , we use the corresponding attacker prefix  $PR(\pi_A, t)$ . The value should reflect the severity of *all* abnormal actions performed by the attacker along the prefix. To evaluate this value, we take into account two factors:

- We sum up the values of the *IllegalActionCost* fields in all LSAs that appear in the attacker prefix. This sum is denoted by *TotalCost*. A larger value of *TotalCost* means more severe deviations from the protocol.
- Let *FirstAttack* represents the *Time* field of the first LSA in the attacker prefix, where *IllegalActionCost*  $\neq 0$ . This is the first time the attacker did something illegal. Let *PrefixLength* denotes the *Time* field of the last LSA in the prefix. The value  $PrefixLength - FirstAttack$  reflects the absolute time during which the attack had been in the system, had the potential of spreading throughout the network and influencing its behavior. The larger this value is, the more severe the attack is.

We now compute the continuous value of a prefix of  $\pi_M$  as:

$$TotalCost \cdot (PrefixLength - FirstAttack)$$

### 4.5.2 Ordered categorical tags

Once the example (monitor prefix) is assigned a value reflecting its severity, we transform this value to a tag  $t \in T = \{t_0, \dots, t_{k-1}\}$  according to the problem definition. One simple way to do this is a uniform discretization of the range of continuous values to  $k - 1$  intervals. These  $k - 1$  intervals represent the  $k - 1$  tags which symbolize an attack (belong to the group  $T \setminus \{t_0\}$ ). Prefixes without an attack are automatically tagged as  $t_0$ .

## 4.6 The Features

The examples used for training are tagged monitor prefixes. We designed a set of features that aggregate various values over the prefix. To make the features suitable for both fixed and dynamic networks, we decided to make the features relative rather than absolute. For example, if the model is trained on one topology but is applied on another one (transfer learning), the number of neighbors of the monitor can be different between these two networks. Therefore, instead of a feature describing the number of neighbors an LSA was sent to, we use the difference between these values in two consecutive LSAs. In addition, for the same reason, we do not define a feature for each router in the system, but rather define aggregator features that summarize the desired values for all the relevant routers. We have features of several types:

- **F1:** Features related to Sequence Numbers of LSAs.
- **F2:** Features related to the Links field in the LSAs - the number of links, the connections they describe, and their costs.
- **F3:** Features related to the times in which LSAs were sent or received by the monitor.
- **F4:** Features related to the neighbors to which the monitor sent the LSAs.
- **F5:** Features related to the ‘hello’ messages, received or sent by the monitor. This type of features is used when dynamic networks are considered, where changes take place and connections go up and down. In a fixed network, these ‘hello’ messages are sent periodically to keep the routers in the network knowing their fixed set of neighbors. However, in a dynamic network, these messages give information about the changes occurred, and are taken into consideration while trying to distinct between normal changes or attacks.

As it can be seen, some of the features are specific to the OSPF protocol and relate to specific messages (LSA, ‘hello’) and specific fields of them (such as the *Links* field). However, other features are more general and relate to the traffic passing through the network (such as the number of neighbors a message is sent to).

Note that in some cases some of the features cannot be extracted in a certain prefix, usually when considering a short prefix. In such a case it is possible that there is not enough information about the run. We are not dealing with such examples and remove them from the example set.

## 4.7 The Learning Algorithm

In general, there are two families of learning problems:

1. Classification - predicting a discrete number of values. For example, predicting if a person is in a high risk of a specific disease or not.
2. Regression - predicting a continuous value. For example, predicting the price of a house.

While choosing the learning algorithm to use, two issues are needed to be taken into account:

- The tags are discrete values, like in a classification problem.
- There is an order between the possible tags, like in a regression problem.

Considering these properties of the tags, the user can define the penalty for each type of misclassification error. For example, he can define a higher penalty for prediction of  $t_0$  to an example whose real tag is  $t_3$ , than a prediction of  $t_0$  to an example whose real tag is  $t_1$ . Such a decision takes into account the order between the different tags and gives a more severe penalty for error between more distant tags. In other words, some misclassification errors can be considered worse than others, as specified in the cost matrix. Thus, we have modified the Random Forest algorithm, so that it can take into consideration the cost matrix. By using this matrix, the user can define different costs to different misclassification errors. In addition, the matrix is not required to be symmetric, which means that the user can prefer, for instance, false positive over false negative.

Our learning process uses the framework of *Random Forest* algorithm, which is a well known learning algorithm that uses a committee of decision trees. Each decision tree presents a sequence of tests conducted on a given example. At the end of each test sequence, a decision is made and the tag of the new example is determined. Each test is represented as an internal node of the tree and has several possible outcomes, which constitute the children of that node. An internal node in the tree can be viewed as implementing a split of the examples arriving at that node, based on the test in the node. A leaf in the tree represents the tag will be given to an example that will arrive to this leaf at the end of the test sequence.

A committee is a set of decision trees that commonly predict the tag of a given example. Each one of the trees gives a prediction to the example and the final prediction of the classifier is defined based on the predictions of the trees.

Random Forest is a learning algorithm that uses a committee. It builds a forest of several decision trees, each of which is constructed with a subset of examples. Further, in each splitting test node, a subset of features is considered, from which one feature is chosen to be used for this split. When we are interested in predicting the tag of a new instance, each one of the trees in the forest conducts its test sequence on the given instance and outputs its prediction for the object. All the tree predictions are taken into consideration when calculating the final tag of the instance. For example, in a classification task, the final tag is determined to be the tag which was predicted by the majority of the trees. This decision minimizes the number of trees that do not agree with this final tag. In a regression problem, when we are interested in predicting a continuous value, the final tag is determined to be the average of the tree decisions. In order to take the cost matrix into account, we made two changes in the classic version of Random Forest. One change is done when choosing the feature to split a node by and the second change is done when calculating the final tag according to the predictions of the trees in the forest.

#### 4.7.1 Choosing a Split

Assume a set of tagged examples arrived at node  $v$  of a decision tree. The tag of  $v$  will be determined so that it minimizes the average misclassification error in the node, *according to the cost matrix*. For calculating this tag we check all the possible tags. For each possible tag  $t$ , we check the penalty we would pay if we determine the tag of all the examples arrived the node to be  $t$ . Note that this computation of the node tag is done in the training phase, when we have the real tags of the examples, thus we can calculate the penalty we would pay for the different possibilities of the node tag. At the end of this process, we divide the results by the number of examples arriving at the node and define the tag of the node to be the one that achieves the lowest cost. More formally, recall that the set of possible tags is denoted by  $T$  and the cost matrix by  $C$ . Let  $E_v$  be the set of examples reached node  $v$ . For each example  $e \in E_v$ , let  $r(e) \in T$  be the real tag of  $e$ . Then the tag of node  $v$  is calculated by the following formula:

$$Tag(v) = \operatorname{argmin}_{t \in T} \frac{\sum_{e \in E_v} C_{t,r(e)}}{|E_v|} \quad (4.1)$$

When having the tag of the node, we can compute its average error by placing it in the expression that represents the average misclassification error in the node. Therefore, the average misclassification error for node  $v$ ,  $Err(v)$ , is given by the following formula:

$$Err(v) = \frac{\sum_{e \in E_v} C_{Tag(v),r(e)}}{|E_v|} \quad (4.2)$$

According to Occam's razor, simpler solutions are more likely to be correct than complex ones. In terms of learning, the meaning of this principle is that a simpler and smaller

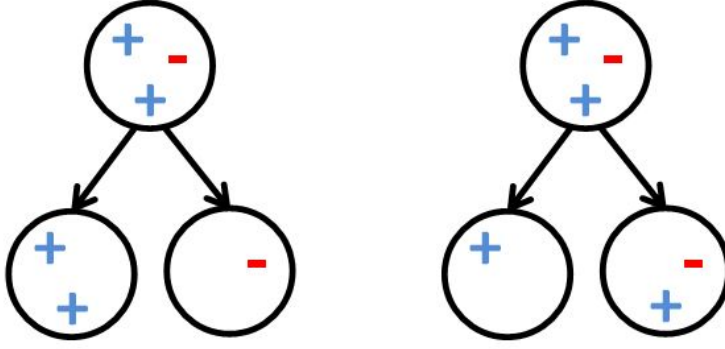


Figure 4.1: Illustration of choosing a feature for the split. The left side demonstrates the using of feature  $f_1$  for the split and the right side demonstrates the using of feature  $f_2$  ('+' represents the tag 0 while '-' represents the tag 1).

tree is preferred to a more complex one. Therefore, when building the trees of the learning algorithm, we are interested in the smaller trees. However, constructing such an optimal tree is a NP-complete problem ([HR76]). Therefore heuristics are used. One heuristic which is used for this purpose is called 'Choose-Attribute'. This heuristic is a greedy algorithm which chooses to split a node by the feature which results in the best local separation. For example, consider a classification task, which is not cost sensitive, in which we have a node with 3 examples, two of them have the real tag 0 and the other one has the real tag 1. Suppose we have two possible features to use in order to split the node. One feature,  $f_1$ , separates the examples in this node to two successors - one with the 2 examples tagged as 0 and the second with the last example (tagged as 1). The other feature,  $f_2$ , also separates the examples in the node to two successors, but in a different way: one successor has 2 examples, one that is tagged as 0 and one that is tagged as 1, and the third example (tagged also as 0) is passed from the node to the second successor. In this case, the first feature executes a better local separation so we will choose it to split the node (see Figure 4.1 for illustration).

In our cost sensitive version, we are also interested in small trees. We use a heuristic which takes into account the cost matrix. Recall that for a given feature  $f$  that splits the node, the examples of the node are divided between the different successors of the node according to the values they have for this feature. In our algorithm, the feature chosen to split a node  $v$  is the one that minimizes the weighted average on  $Err(v_i)$  for all successors  $v_i$  of  $v$ , generated by the feature. We denote by  $succ_f(v)$  the set of successors achieved when splitting the node  $v$  by the feature  $f$ . Note that all of the examples in the node are divided between the node's successors. Therefore, it holds that  $\sum_{v_i \in succ_f(v)} |E_{v_i}| = |E_v|$ . We denote the weighted average of the successor errors as 'WASE', and given a node  $v$  and a feature  $f$  to split the node by,  $WASE(v, f)$  is

defined as follow:

$$WASE(v, f) = \sum_{v_i \in succ_f(v)} \frac{|E_{v_i}|}{|E_v|} \cdot Err(v_i) \quad (4.3)$$

The feature that minimizes the weighted average of the successor errors is finally chosen to split the node. This feature is denoted by  $Feature(v)$  and formally defined by the following formula:

$$Feature(v) = \underset{f \in Features}{\operatorname{argmin}} \{WASE(v, f)\} \quad (4.4)$$

#### 4.7.2 Determining the Final Tag

As said, using a platform of committee as in Random Forest, each one of the trees in the committee gives a prediction for a new given example. The final prediction of the classifier is defined according to the predictions of the trees. In our cost sensitive case, we define the final tag to be the one that minimizes the average weighted error of the trees. As before, this calculation is performed by using the cost matrix and checks all the possibilities for the final tag. For each tag  $t$ , we check the average weighted error we would get if we determine the tag of the new example to be  $t$ , considering the different tree predictions. Let the set of trees in the committee be denoted by  $F$ . Recall that the set of possible tags is denoted by  $T$  and the cost matrix by  $C$ . For a given example  $e$  and a specific tree  $tree \in F$ , let  $tree(e)$  be the prediction of  $tree$  on  $e$ . Then, for determining the final tag of  $e$ , we use the following formula:

$$final\_tag(e) = \underset{t \in T}{\operatorname{argmin}} \frac{1}{|F|} \cdot \sum_{tree \in F} C_{tree(e), t} \quad (4.5)$$

In the case where more than one tag achieved the minimal error, we choose the one that achieved more ‘votes’ from the trees.

For clarification, let take a look at a specific example. Assume the set of tags given as an input is  $T = \langle t_0, t_1, t_2 \rangle$  and the given cost matrix  $C$  is the cost matrix presented in Table 4.1. Suppose we are interested in predicting the tag of a new example  $e$ , whose real tag (unknown to us) is  $t_0$ .

In one case, assume we have 3 trees in the forest that give the following predictions:  $t_0$ ,  $t_1$  and  $t_2$ . We denote by  $Err(t_i)$  the average misclassification error we received when determining the example’s tag to be  $t_i$ . Then:

$$Err(t_0) = \frac{1}{|F|} \cdot (C_{t_0, t_0} + C_{t_1, t_0} + C_{t_2, t_0}) = \frac{1}{3} \cdot (0 + 2 + 4) = 2$$

$$Err(t_1) = \frac{1}{|F|} \cdot (C_{t_0, t_1} + C_{t_1, t_1} + C_{t_2, t_1}) = \frac{1}{3} \cdot (2 + 0 + 2) = 1.33$$

$$Err(t_2) = \frac{1}{|F|} \cdot (C_{t_0, t_2} + C_{t_1, t_2} + C_{t_2, t_2}) = \frac{1}{3} \cdot (4 + 2 + 0) = 2$$

Therefore, the chosen tag for  $e$  is  $t_1$ , which is a mistake. Note that also in a regular classification process (not a cost sensitive version), in such a case it is very likely to get a wrong prediction according to the situation in which each one of the three trees gives a different prediction and only one of them predicted the right tag.

Table 4.1: Example for the cost matrix. For each matrix element, the row represents the predicted tag while the column represents the true tag.

	$t_0$	$t_1$	$t_2$
$t_0$	0	2	4
$t_1$	2	0	2
$t_2$	4	2	0

Now suppose that the learning algorithm works better and builds an additional tree that predicts the tag  $t_0$  for  $e$ . Then:

$$Err(t_0) = \frac{1}{|F|} \cdot (C_{t_0,t_0} + C_{t_1,t_0} + C_{t_2,t_0} + C_{t_0,t_0}) = \frac{1}{4} \cdot (0 + 2 + 4 + 0) = 1.5$$

$$Err(t_1) = \frac{1}{|F|} \cdot (C_{t_0,t_1} + C_{t_1,t_1} + C_{t_2,t_1} + C_{t_0,t_1}) = \frac{1}{4} \cdot (2 + 0 + 2 + 2) = 1.5$$

$$Err(t_2) = \frac{1}{|F|} \cdot (C_{t_0,t_2} + C_{t_1,t_2} + C_{t_2,t_2} + C_{t_0,t_2}) = \frac{1}{4} \cdot (4 + 2 + 0 + 4) = 2.5$$

Predicting  $t_0$  and predicting  $t_1$  yield the same minimal error. However, the prediction  $t_0$  got 2 votes by the trees, where  $t_1$  got only 1. Therefore, the chosen tag for  $e$  in this case is  $t_0$ , which is the correct tag.

Obviously, if we add an additional tree whose prediction for  $e$  is  $t_0$ ,  $Err(t_0)$  will remain unchanged while  $Err(t_1)$  and  $Err(t_2)$  will increase. Therefore  $Err(t_0)$  will be the minimal value and the final tag will be again  $t_0$ , which is the correct tag.

### 4.7.3 Feature Importance

One of the most interesting issues when solving a machine learning problem is realizing which ones of the features were helpful to the learning process. For finding these features we implemented an algorithm which takes our new model classifier as an input and outputs for each feature a value that represents the percentage of the feature's contribution to the learning process. The algorithm first initializes a zeros-array, where each cell of this array is associated with one of the features and contains the feature's score. The algorithm traverses the trees of our model and in each internal node  $v$  (not a leaf) that is splitted by a specific feature  $f$ , we calculate the two factors below:

- **Error reduction:** The decrease in the average error achieved by using  $f$  at this node. A larger value of this factor means a greater contribution to the learning process. As mentioned in Subsection 4.7.1,  $Err(v)$  represents the average error of node  $v$ , and the weighted average of the errors of the successors of  $v$  is given by  $WASE(v, f)$ . Therefore, for a node  $v$  splitted by a feature  $f$ , the error reduction factor is calculated by the following formula:

$$ErrorReduction_{v,f} = Err(v) - WASE(v, f)$$

- **Number of examples:** The number of examples in node  $v$ . This is actually the number of examples that are divided to smaller groups, using feature  $f$ . The larger this factor is, meaning the feature helps to more examples, and



the contribution of this feature to the learning process is larger. Formally,  $NumberOfExamples_v = |E_v|$ .

We now compute the *Local score* of feature  $f$  at a specific internal node  $v$  as  $LocalScore_{f,v} = ErrorReduction_{v,f} \cdot NumberOfExamples_v$ , and add this value to the array cell associated with feature  $f$ . Algorithm 4.2 describes the algorithm that calculates the features' score array.

---

**Algorithm 4.2** FEATUREIMPORTANCE(CLASSIFIER)

---

```

1:  $features\_scores \leftarrow zeros(number\_of\_features)$  ▷ zeros-array
2: for tree  $t$  in classifier do
3:   for node  $v$  in  $t$  do
4:     if  $v$  is not a leaf then
5:        $f \leftarrow Feature(v)$ 
6:        $LocalScore_{f,v} \leftarrow ErrorReduction_{v,f} \cdot NumberOfExamples_v$ 
7:        $features\_scores[f] = features\_score[f] + LocalScore_{f,v}$ 

```

---

After having the features' score array, we use it to calculate the percentage of each feature's contribution by the following expression:

$$Contribution(f) = \frac{feature\_score[f]}{\sum_{g \in Features} feature\_score[g]} \cdot 100 \quad (4.6)$$



## Chapter 5

# Empirical Evaluation

We have performed an extensive empirical study to test our new framework.

### 5.1 Experimental Methodology

The learning process requires from the user the following input: network topology, monitor router, the set of possible tags and the cost matrix. For the experiments described in this section we have used these settings:

1. *Network topology*: To allow experimentation in a diverse set of topologies, we have designed a topology generator that is capable of generating topologies according to specified parameters. This topology generator is described in the following subsection.
2. *Monitor location*: We have randomly selected a node (a router) in the produced topologies to act as a monitor.
3. *Ordered set of tags*: Our framework is capable of working with any ordered set. For the experiments described here we have used 4 categories: *No Attack*, *Weak Attack*, *Medium Attack* and *Severe Attack*.
4. *Cost matrix*: The cost matrix we used is specified in Figure 5.1. For each matrix element, the row represents the predicted tag while the column represents the true tag.

Since we need to measure performance with respect to the cost matrix, we have defined a new measure called *weighted accuracy*.

**Definition 5.1.1 (Average error cost).** Given an example  $e_i$ , let  $t_i$  denote the true tag of  $e_i$  and  $p_i$  denote the predicted tag of  $e_i$ . Let  $\max(C)$  denote the maximum value specified in the cost matrix  $C$  (this is the maximal error cost we would pay for a misclassification error). Given a cost matrix  $C$  and a test set of examples

$E = \{e_1, \dots, e_n\}$ , we define the average (normalized) error cost as  $AER_{C,E} = \frac{\sum_{i=1}^n \frac{C_{p_i, t_i}}{\max(C)}}{n}$ .

Table 5.1: The cost matrix used for the experiments.

	No Attack	Weak Attack	Medium Attack	Severe Attack
No Attack	0	$\frac{1}{3}$	$\frac{2}{3}$	1
Weak Attack	$\frac{1}{3}$	0	$\frac{1}{3}$	$\frac{2}{3}$
Medium Attack	$\frac{2}{3}$	$\frac{1}{3}$	0	$\frac{1}{3}$
Severe Attack	1	$\frac{2}{3}$	$\frac{1}{3}$	0

**Definition 5.1.2 (Weighted accuracy).** Given a cost matrix  $C$  and a test set of examples  $E = \{e_1, \dots, e_n\}$ , we define the weighed accuracy as  $1 - AER_{C,E}$ .

## 5.2 The Network Topology Generator

A network topology is the set of routers and the connections between them. Thus, a topology can be represented by a simple (no self-loop, no parallel edges) undirected graph. Topologies working with the OSPF protocol are usually connected graphs. We characterize a topology by its number of nodes  $n$  and its average degree  $d$ , and define  $\mathcal{F}_{n,d}$  to be the family of all connected topologies with  $n$  routers and an average degree of  $d$ . The motivation of characterizing the topologies with these two elements is our hypothesis that these two factors will have a large impact on the performance of our system on the topology. On the one hand, we assumed that when the network is larger, in terms of the number of routers, the attacker has more options on which router to take control of. Therefore, when the network is larger, the variety of possible attacks is likely to be greater, which causes it to be more difficult to achieve good performance while trying to identify attacks in that topology. On the other hand, we hypothesized that the higher the level of connectivity in the network, the faster the information spread. Therefore, identification of an attacker in such a network is expected to be easier and faster than in a network where the average degree is relatively low.

Assume for now that  $n \cdot d$  is an even number. We have designed an algorithm that, given  $n$  and  $d$ , generates a random set of topologies in  $\mathcal{F}_{n,d}$ .

Getting the number of routers  $n$  and the average degree  $d$ , we first check the validity of the given parameters (for example, it must hold that  $d \leq n - 1$ ) and if the parameters are valid, we generate a random topology from  $\mathcal{F}_{n,d}$ . The algorithm that creates the topology has two phases: the first one is for creating a connected graph with  $n$  vertices and the second one is for making the average degree to be  $d$  (see algorithm 5.1).

According to the algorithm, we continue to add edges to  $E$  as long as  $|E| < \frac{nd}{2}$ . Therefore, at the end of the process  $\frac{nd}{2} \leq |E| < \frac{nd}{2} + 1$ . Since we assume  $nd$  is an even number,  $|E| = \frac{nd}{2}$ . The average degree of the resulting graph is  $d' = \frac{2 \cdot |E|}{n}$  (each edge

---

**Algorithm 5.1** CREATETOPOLOGY( $n, d$ )

---

```
1:  $V \leftarrow \emptyset, E \leftarrow \emptyset$ 
2:  $routers = \{r_1, \dots, r_n\}$ 
3: choose a first router  $first\_r$  from  $routers$ 
4:  $V \leftarrow V \cup \{first\_r\}$ 
5:  $routers \leftarrow routers \setminus \{first\_r\}$ 
6: while  $routers$  is not empty do
7:   choose randomly router  $r$  from  $routers$ 
8:    $routers \leftarrow routers \setminus \{r\}$ 
9:   choose randomly one edge  $e$  from  $\{(r, r') | r' \in V\}$ 
10:   $E \leftarrow E \cup \{e\}, V \leftarrow V \cup \{r\}$ 
11: while  $|E| < \frac{nd}{2}$  do ▷ The second phase starts here
12:   choose randomly an edge  $e$  such that  $e \notin E$ 
13:    $E \leftarrow E \cup \{e\}$ 
14: return  $G = (V, E)$ 
```

---

increases the degree of its two end nodes by one). Therefore the average degree of the generated graph is  $d' = d$  as required.

Note that for arbitrary (not even)  $nd$ , our algorithm generates a member in  $\mathcal{F}_{n,d'}$  where  $|d' - d| \leq \frac{2}{n}$ , which becomes very small for large values of  $n$  (by multiplying the above formula by  $\frac{2}{n}$  we get that  $d \leq \frac{2 \cdot |E|}{n} < d + \frac{2}{n}$ ).

Figure 5.1 shows an example of a topology with 8 routers and an average degree of 5 that was created by the above algorithm. Each node is associated with a unique number identifying it.

After generating a network topology, we randomly choose a value reflecting the bandwidth of the connection between each pair of neighbor routers.

### 5.3 System Performance

In order to evaluate our performance on different networks, we conducted experiments in which we check the accuracy of our tool while using different network topologies. We generated a number of topologies, with different number of routers and average degrees. For each topology, we ran the simulator and tagger to produce a set of examples. We balanced the set across categories which means that for every tag  $t \in T$ , we have an equal number of examples that are tagged as  $t$ . Then we applied the feature extractor in order to get the tagged feature vectors. Finally, we have conducted a cross-validation experiment. This is a widely used technique for estimating how accurately our model will perform in practice (see ‘Cross-validation (statistics)’ in Wikipedia). In this technique, we use some of the given data to train the system with and the other data to test our system on. One round of cross-validation involves partitioning a sample of data into complementary  $k$  subsets, performing the analysis on the union of  $k - 1$  subsets (called the training set), and validating the analysis on the other subset (called the

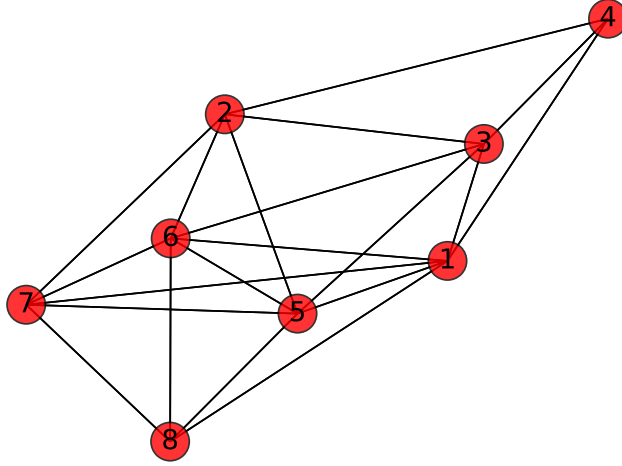


Figure 5.1: A specific topology created by the algorithm

validation set or testing set). Usually multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds to give an estimate of the model's predictive performance (see Figure 5.2 for illustration). In our experiments, we used 10-fold stratified cross validation which means we conducted 10 rounds where each data subset has the same percentage of examples for each tag. During the cross validation process we recorded the average weighted accuracy that was achieved.

Figure 5.3 shows the learning curve, which describes the weighted accuracy as a function of the number of examples used in the training phase. The figure represents the learning curve for one of the tested topologies. We can see a typical learning behavior where performance quickly improves up to about 2000 examples, and then stabilizes. This result means that a learning process is actually taking place and succeeding in generalizing the accumulated experience from the training phase on new unseen examples.

In addition, we conducted more experiments to check how different parameters influence the performance of the system. One of these parameters is the network size. The left-hand side of Figure 5.4 shows the obtained results. The X axis represents the number of routers in the tested topologies ( $n$ ), and the Y axis the average weighted accuracy.

Another factor we take into account is the connectivity of the graph. The right-hand side of Figure 5.4 presents the result of experiments done with topologies with 12

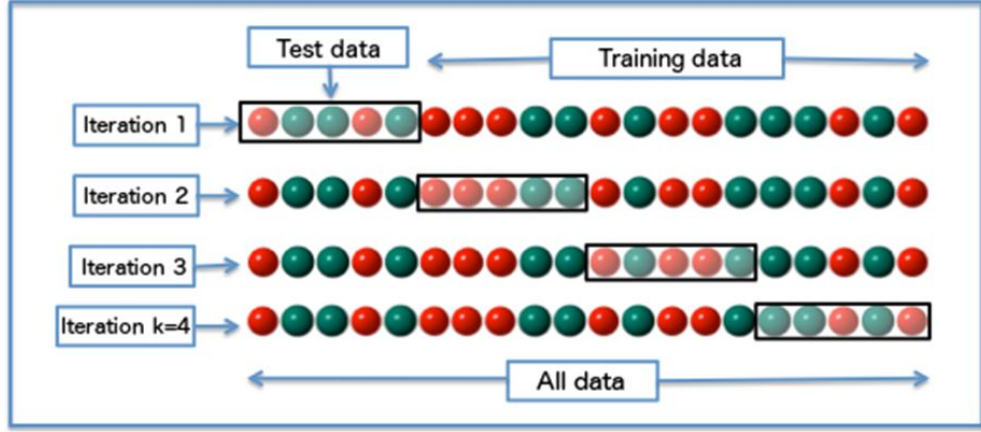


Figure 5.2: Several rounds of cross validation algorithm (from Wikipedia)

routers. In the experiments, we create a number of topologies, each one with 12 routers but different average degree, and repeat the process of conducting the experiments, as described above. The X axis in the right image of Figure 5.4 represents the average degree of the tested topologies ( $d$ ), and the Y axis the average weighted accuracy.

From the left-hand side of Figure 5.4 we can see that, as expected, the weighted accuracy decreases when the network becomes larger and contains more routers. However, the decline stabilizes, showing that our framework can handle even large network topologies. From the right-hand side of the same figure we can see an upward trend in the graph curve as the average degree of the tested topology gets higher. We expected to see this trend as we assumed that a topology with a higher average degree should yield a higher performance, resulting from the fact that the traffic flows in the topology more quickly. More reinforces to this phenomenon are displayed in Subsection 5.6.2 and in the experiments part of Section 6.2.

## 5.4 Feature Importance

We conducted some experiments in order to see which of the features help in recognizing to which class a given prefix belongs. We generated random topologies and used our simulator in order to generate examples for these topologies. Then we used our tagger and the feature extractor and built a classifier according to these feature vectors. When having the classifier, we applied algorithm 4.2 and checked the values the different features received.

We notice that in different topologies there are different features which have more impact, and therefore get higher scores. In general, it seems that the features which are very helpful to the learning process are features related to the *Links* field in the LSAs - the number of links, the connections they describe, and their costs.

Usually, attacker wants to falsify the knowledge the other routers have on the structure of the topology, in order to route packets through it or prevent them from

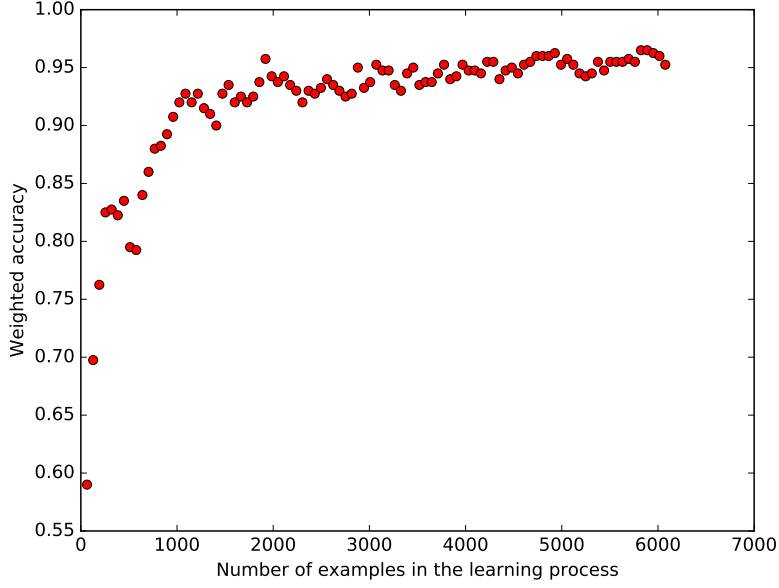


Figure 5.3: Weighted accuracy as a function of the number of examples used in the training process

reaching their destinations. Thus, the LSA’s field which is usually used in the attack is the one that related to the structure of the network - the *Links* field. Therefore it is not surprising that the features related to this field are very helpful in detecting attacks.

## 5.5 Transfer Learning Setup

### 5.5.1 Introduction

We envision the following use case for our new method: A customer supplies our algorithm with its specific topology, and applies the learning mechanism to generate an intrusion detector. In realistic setups, however, we would like sometimes to be able to learn a detector for a family of topologies. Recall that we define  $\mathcal{F}_{n,d}$  to be the family of all connected topologies with  $n$  routers and an average degree of  $d$ . If a customer has a topology of that family, he can use the learned detector, without any further training. This is a classical transfer learning setup where the predictor is learned on one problem and is applied on another.

As mentioned in Section 4.6, we presented a set of features that use relative terms to help adaptation to dynamic setups such as transfer learning. We experimented with our algorithm to test its flexibility in such setups.

### 5.5.2 The Target Belongs to the Same Family as the Source

In our first experiment we learned on a source topology of a given family, and tested the performance on a target topology of the same family. Table 5.2 lists the obtained results. The first two columns specify the family parameters. The third column specifies



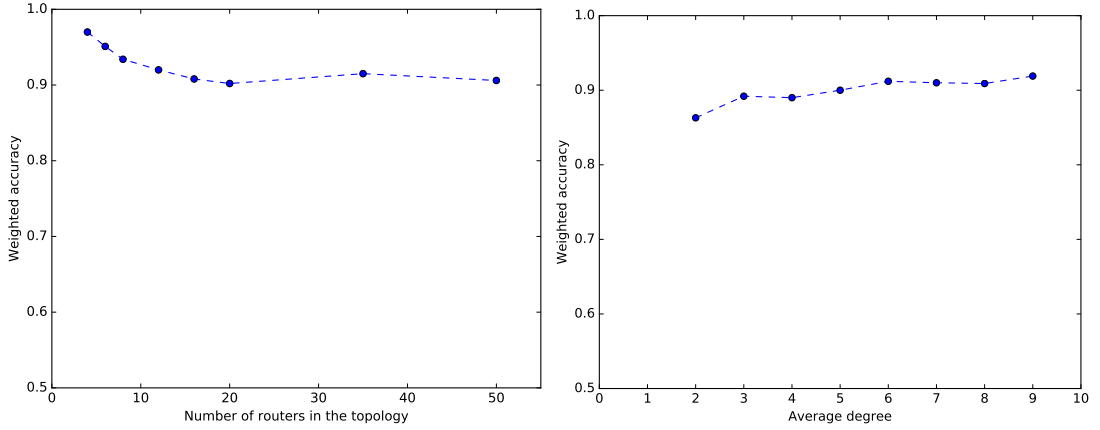


Figure 5.4: Weighted accuracy as a function of the  $n$  and  $d$ .

the weighted accuracy for the normal setup where source = target, and the fourth one displays the weighted accuracy for the transfer setup where the target is a different topology from the same family as the source.

Table 5.2: TRANSFER LEARNING RESULTS

Family		source = target	source $\neq$ target
$n$	$d$	weighted accuracy	weighted accuracy
4	2	0.970	0.937
8	5	0.934	0.887
12	5	0.920	0.843
16	7	0.908	0.870

Obviously, testing on the same topology yields better results than when testing on a different one, but since they are of the same family, the difference is modest.

### 5.5.3 The Target Belongs to a Different Family from the Source

The second experiment tested the effect of the dissimilarity between the family of the source topology and the family of the target topology on performance. We define the distance between  $\mathcal{F}_{n_1, d_1}$  and  $\mathcal{F}_{n_2, d_2}$  to be  $|n_1 \cdot d_1 - n_2 \cdot d_2|$ . Figure 5.5 shows the results for several target topologies. The X axis stands for the distance between the family of the source topology and that of the target topology. The Y axis specifies the weighted accuracy of the model learned on the source topology when tested on the target test set.

We can see a very typical transfer learning behavior with a decline in performance when the target is getting further from the source.

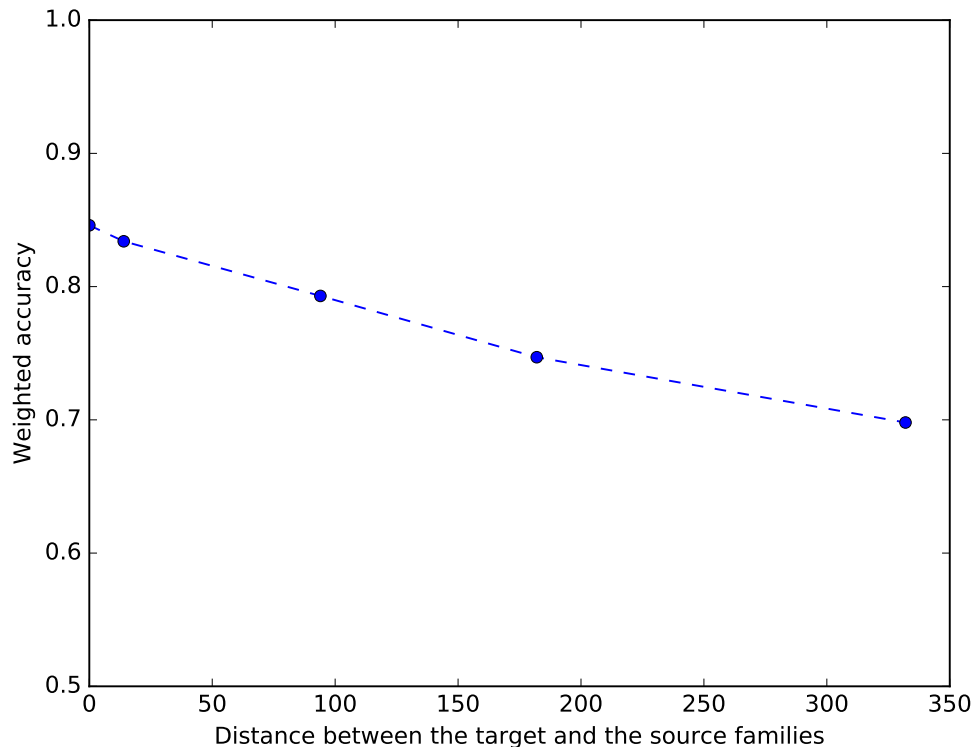


Figure 5.5: Weighted accuracy as a function of the distance between the target and the source families

## 5.6 Dynamic Networks

### 5.6.1 Introduction

Network topologies often go through continuous changes, such as addition or removal of connections between routers. This may present a problem to a detector that was trained on somewhat different topology. One obvious way to overcome the problem is to perform retraining with the modified topology. As retraining is computationally expensive, we prefer to deal with the problem differently by building a model that can tolerate small topology changes.

In this section we consider two types of dynamic networks: the first scenario assumes that the topology has dynamically changed after the training phase, and the learned and predicted topologies are fixed, but different from one another. The second scenario assumes that the changes take place both during the training and during the prediction phases.

Similarly to transfer learning, in these cases the target topology is not identical to the source one.

### 5.6.2 Topology Changes Between the Training and Prediction Phases

In this subsection we talk about the first scenario where the topology is dynamically changed after the training phase.

We conducted several experiments in order to check how much our model can tolerate such changes. In each experiment, we have generated a random topology and applied our learning algorithm to generate a detector. We then produced a sequence of topologies created by randomly choosing edge(s) to be removed or inserted from/to the topology. A test set was produced for each member in the sequence and the learned model was tested on these sets. Given two topologies with the same number of routers, we define as *delta* the number of edges that exist in one of them but not in the other.

Note that in the described experiments, we made sure that we did not cancel a change that already took place (not insert and remove the same edge). Figure 5.6 presents the results of some of the experiments.

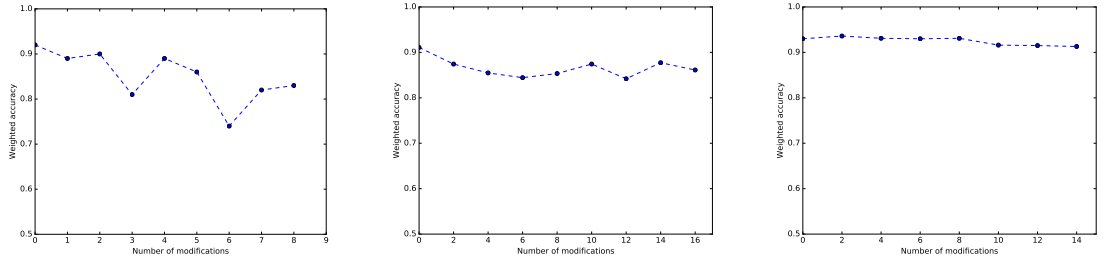


Figure 5.6: Weighted accuracy as a function of  $\delta$ . Topologies with 6, 12 and 16 routers, from left to right respectively.

We can see that the weighted accuracy is still high even for topologies that are different than the one we used for training. It is particularly true for larger networks in which the ratio between the number of changes and the size of the topology is smaller, and therefore such a change is less significant.

We also note that at certain points in the graph the accuracy got better even though the number of changes increased. These points are the result of adding edges to the graph. We conjecture that increasing the graph connectivity may overcome the disadvantage of having different learned and predicted topologies. This phenomenon also reinforces the assumption examined in Section 5.3, that increasing the connectivity of the graph may be beneficial to the performance of the system.

### 5.6.3 Topology Changes During Training and Prediction Phases

In this section we discuss the second scenario, in which the topology is dynamically changed during the training and prediction phases.

## The Simulator

For conducting experiments to check how much our model can tolerate such changes, we have changed our OSPF simulation a bit. At the beginning of the simulation, a random integer number is chosen. This number represents the number of connections that will be changed (removed or inserted) during the simulation. Let us denote this number as  $n_{changes}$ . Then we randomly choose  $n_{changes}$  pairs in which the first element is an identifier of a connection (edge) and the second one is a time in which this connection will be changed. Then we start running the simulator, while executing the insertion or removal of the connections at the time specified for each one of them. This way, our simulator actually represents an OSPF simulation while the network topology is dynamically changed during the run. From now on, when we refer to the OSPF simulator with its dynamic version, we call it the *dynamic-simulator*.

## Additional Features

When considering a network which is dynamically changed during the run, we should pay attention not only to the LSA messages which describes the different connections and is spread over the network, but also to the local 'hello' messages. This type of messages is sent every few seconds between neighbor routers. If a router does not receive such a message from its neighbor within a specific time interval, it infers that the connection between them went down and they are no longer neighbors. In the opposite case, when a router receives a 'hello' message from a new router that has not sent such a message before, it infers that a connection between them was inserted and they are new neighbors. This way, the 'hello' messages help the OSPF protocol to dynamically adapt to changes in the network. Therefore, when working with dynamic topologies, we include these messages. As with LSA messages, we include only hello messages that were sent or received by the monitor. We used these messages and extract features from them, as described in section 4.6.

## Experiments

We conducted several experiments in order to check how much our model can deal with this type of dynamic networks. In each experiment, we have generated a random topology and used our *dynamic-simulator* for generating the examples. Then we applied the tagger and balanced the set across categories. After that we applied the features extractor including the features for the 'hello' messages. Finally, we have conducted a stratified cross-validation experiment and recorded the average weighted accuracy.

Tables 5.3 and 5.4 present the results achieved when executing the experiments on two network topologies - with 12 and 16 routers. In each table, the rows represent the type of the network on which the training process was done where we considered three types on networks: a stable network (without changes at all), a network in which a few changes occur during the run and a network in which many changes occur during

Table 5.3: DIFFERENT DYNAMIC TYPES - A TOPOLOGY WITH 12 ROUTERS

	Stable	Few changes	Many changes
Stable	0.930	0.905	0.880
Few changes	0.900	0.935	0.912
Many changes	0.857	0.883	0.940

Table 5.4: DIFFERENT DYNAMIC TYPES - A TOPOLOGY WITH 16 ROUTERS

	Stable	Few changes	Many changes
Stable	0.923	0.91	0.897
Few changes	0.906	0.938	0.911
Many changes	0.847	0.867	0.949

the run. The columns in the tables represent the type of the tested network: stable, few changes and many changes. Each table cell (i,j) represents the weighted accuracy achieved when training the detector on the topology specified in row  $i$  and testing the performance on the topology specified in column  $j$ .

We can see that in both tables, the highest performance achieved on the diagonal, when the tested network is taken from the same setup of the topology on which the training process has been applied. For example, consider the second column, which represents the situation in which the tested network is a dynamic network in which a few changes occurred during the run. We can see that the highest value in this column achieved in the second row, which means we received the best performance when the detector was trained on a network of the same type. We can see this phenomenon also in the other columns.

This situation is consistent with the real world situations in which the learned network is usually the same as the network we want to predict (For example, if a network of a specific organization is dynamic, it is unlikely that the organization will be able to stop the changes for the training phase).

## 5.7 Detection Times

We conducted more experiments in order to check the detector performance in terms of how long an attack exists in the network before it is revealed. In these experiments we only distinguish between non-attacked (tag *No Attack*) and attacked (tags *Weak/Medium/Severe Attack*) runs, ignoring the level of the attacks. Tables 5.5 and 5.6 show the accuracy rate of our framework as a function of the normalized detection time. We define the *normalized detection time* as the actual detection time divided by the time it takes for a packet to traverse the network's diameter, which is the longest path between any two nodes in the network. We present results for mid-size networks having a diameter of 2 and 6 links. The different lines of the tables display the normalized

Table 5.5: NETWORK WITH A DIAMETER OF 2

Normalized detection time	Accuracy rate
0 – 2.3	0.925
2.3 – 7.14	0.982
above 7.14	0.995

Table 5.6: NETWORK WITH A DIAMETER OF 6

Normalized detection time	Accuracy rate
0 – 3.2	0.965
3.2 – 4.18	0.99
above 4.18	1

detection times and the accuracy we achieved in each one of them. From the tables we can deduce that the longer an attack exists in the system, the higher our accuracy rate is, which means that we are more likely to detect the attack. However, for both networks, the accuracy rate is very high even for short detection times. This implies that our algorithm is able to detect attacks in a very short time after the attack starts. For example, for a network having a diameter of 6 links, an accuracy rate of 0.965 is obtained, when the normalized detection time is under 3.2. Assuming the time it takes for a packet to traverse the longest path in the AS is 50 ms, then the attack will be detected with high probability within 160 ms ( $3.2 \cdot 50$ ).

## 5.8 Evaluation with Real Data

### 5.8.1 Introduction

As said in chapter 1, one of the most significant challenges when working in the field of attack detection is the lack of appropriate public datasets for assessing the systems [SP10]. The primary reason to this situation arises from the fact that the inspection of network traffic can reveal highly sensitive information. Any breach of such information can be harmful. Therefore researchers frequently encounter insurmountable organizational and legal barriers when they attempt to provide such datasets.

### 5.8.2 Real OSPF Data

Although there is no available data to try our system on, we succeeded to achieve real runs of the OSPF protocol from an ISP that prefers to remain anonymous. The organization provided us with data of real OSPF runs, where all the runs are normal. The network topology of that organization includes 17 routers and is presented in Figure 5.7.

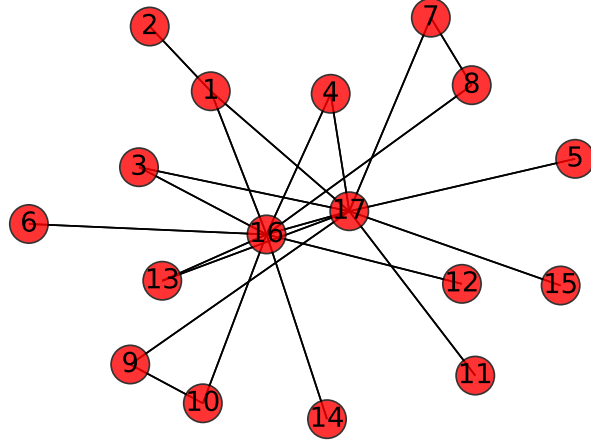


Figure 5.7: The organization’s network topology

One main difference between our simulation and the data provided to us is that in our simulation the monitor is located on one **router** while the data we received reflects the traffic on one **connection** - the one between routers 16 and 17. To deal with this difference, we performed a second processing on our examples to make them reflecting the traffic on that connection only.

In addition, when considering a monitor located on a connection and not on a router, some of our features became irrelevant and were omitted from the learning process. For example, having only the record of the traffic on that connection, we cannot know how many neighbors the router at the end of this connection sent an LSA to. Therefore, we did not use all of our features and thus expected a lower accuracy rate. After making the adjustment to this real data, we could conducted experiments to see how well our system performs on it.

### 5.8.3 Results

We received from the provider a layout of its topology, and encoded this layout into our system. We then trained a detector on this topology and tried it on the normal runs supplied by the ISP. This is not an appropriate testing as the real test data contained only normal runs, but it could still be indicative, especially if our system would have performed poorly on the real data. We found, however, that our detectors achieved high weighted accuracy results ( $\approx 95\%$ ). This result reinforces our hypothesis that the model we built succeeds in identifying normal runs of the protocol.





## Chapter 6

# The Monitor Placement Algorithm

### 6.1 Introduction

Our framework assumes that the user specifies which router(s) serves as the monitor(s), but it is not clear how to choose the router(s) for this purpose. In this chapter we propose a way to choose the location of the monitor(s) wisely in order to get a high accuracy rate.

### 6.2 Locating One Monitor

In this section we assume the organization has resources to invest in only one monitor. The organization needs to choose the monitor location wisely in order to achieve a good identification of attacks. There are several possibilities for choosing this location. One possibility is to choose it randomly. A more intelligent decision is to choose the router with the highest degree in the topology. The motivation of such a decision is the idea that a router with more neighbors gets more information of the traffic in the network and therefore has a better position to detect an attack.

Another option is to use the measure of betweenness centrality, which is a measure of centrality in a graph based on shortest paths. The *betweenness centrality* of a node  $v$  is given by the expression  $g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$  where  $\sigma_{st}$  is the total number of shortest paths from node  $s$  to node  $t$  and  $\sigma_{st}(v)$  is the number of those paths that pass through  $v$ .

Another intelligent option for choosing the location for the monitor is to choose the router that has the highest potential to transmit a large amount of traffic through it. We call this router the ‘max-flow router’. In order to evaluate which router has the potential to transfer a large amount of traffic, we used a technique called ‘random walk’. We built an algorithm which executes a large number of random walks,  $n_{walks}$ , on the given topology. In each walk, a random router is chosen to be the source of the walk

and at every step the algorithm randomly chooses to finish the walk or to continue and randomly move to one of the neighbors of the current router. For each router we maintain a counter which increases by one every time a walk arrives to this router. Finally, we choose to place the monitor on the router with the largest counter value. Algorithm 6.1 displays this algorithm.

Our flexible framework allows us to design another interesting strategy for choosing a router for the monitor. We train the detector on each one of the possible routers and estimate its performance. We then select the router that achieves the highest accuracy rate to be the monitor.

---

**Algorithm 6.1** FINDMAXFLOWROUTER( $N = (V, E)$ )

---

```

1:  $counters \leftarrow zeros(|V|)$  ▷ zeros-array
2: for  $i = 1, \dots, n_{walks}$  do
3:    $r \leftarrow$  choose the first router randomly
4:    $counters[r]++$ 
5:    $to\_continue =$  choose if to continue the walk, with prob  $P_c$ 
6:   while  $to\_continue$  do
7:     choose an arbitrary neighbor  $r'$  of  $r$ 
8:      $r \leftarrow r'$ 
9:      $counters[r]++$ 
10:     $to\_continue =$  choose if to continue the walk, with prob  $P_c$ 

```

---

## Experiments

We conducted several experiments in order to compare all the above options for choosing the location of the monitor. In each experiment, we built a network topology and checked the accuracy achieved when locating the monitor according to the different strategies. Table 6.1 presents these experiments. The first two columns describe the checked topology parameters - the number of routers and the average degree. The other columns describe the accuracy achieved when the monitor has been located according to the above different techniques.

Table 6.1: SELECTING ONE MONITOR

topology		random router	router with the highest degree	highest betweenness centrality	max-flow router	our algorithm
number of routers	average degree					
6	3	0.93	0.944	0.93	0.93	0.96
12	2	0.809	0.866	0.873	0.852	0.873
12	6	0.892	0.934	0.934	0.934	0.934
16	7	0.91	0.92	0.934	0.94	0.94
30	8	0.888	0.92	0.92	0.92	0.921

The table demonstrates the advantage of our approach for selecting the monitor

location, achieving the best accuracy rate in all experiments. No other methods achieved high performances as ours, in all the different experiments.

We can see again that the connectivity of the topology helps in receiving a higher performance (the results in the third row in the table is much higher than the results in the second row), which also reinforces the assumption examined in Section 5.3.

### 6.3 Locating More Than One Monitor

In this section we assume the organization has resources to invest in more than one monitor. When working with several monitors, each one of them records the network traffic from its point of view. As in the previous case, the organization needs to choose the monitor locations wisely in order to achieve a good identification of attacks.

When using more than one monitor, we consider two options for building the final detector that will predict the tag of a new unseen example. The first option is to build one classifier for all the monitors: each one of the monitors will transmit its point of view of the run to this classifier. In this way, the classifier is the final detector and it has all the monitors points of view of the same run. This means that the model has more knowledge and is likely to achieve better detection results. In this case we use an extended feature vector, which is the concatenation of the feature vectors of the different monitors to each other (and even can include more features that describe relations between the features of different monitors). We thus get a new feature vector that will be used as an example (both for the training phase and the prediction phase).

Another option is to build a classifier for each one of the monitors separately by generating examples to the training phase from its point of view. These examples are generated by the simulator and are tagged by our tagger. Then we generated the feature vectors and executed the training process for each monitor individually. This process is done for every monitor and resulting in independent classifiers, each one of them is fitted to its monitor. When executing a new run of the protocol, each monitor has its point of view of the run which gets into its classifier using the regular feature extractor. This way, each one of the monitor classifiers outputs a tag for the new run, relying on its knowledge about the run (its monitor point of view). Then all these local decisions of the classifiers are considered for determining the final decision (tag) for the run. In this case, the detector is the one that takes all the classifier predictions into account and determines the final tag accordingly. There are several methods for determining the final tag using the decisions of the classifiers. Here are some of them:

1. Define the final tag to be the most severe attack among the predictions of the classifiers. This decision makes sense by predicting the worst case.
2. Define the final tag to be the severity that the majority of the classifiers reported on.

3. Define the final tag to be the average severity of the attack among the severity reported by the classifiers. This decision makes sense by taking into account **all** the classifier predictions and not just one or several of them as in the previous options.
4. Define the final tag to be the tag that minimizes the average weighted error of the classifiers. We calculate this tag using the cost matrix, in a similar way to Subsection 4.7.2. We go over all the possible tags and for each possible tag  $t \in T$  we calculate the average weighted error we would get if we determine the tag of the new example to be  $t$ , considering the different classifier predictions. Finally we choose the tag that achieves the lowest cost. As in Subsection 4.7.2, in the case where more than one tag achieved the minimal error cost, we choose the one that achieved more ‘votes’ from the classifiers. Formally, denote the set of the classifiers as  $S$ . In this case, the size of  $S$ ,  $|S|$ , is the number of monitors, as we build a classifier for each monitor. For a classifier  $s \in S$ , we denote by  $s(e)$  the classifier prediction of a given example  $e$ . Using the cost matrix  $C$  and the set of possible tags  $T$ , the final tag of  $e$  is determined by the following formula:

$$final\_tag(e) = \underset{t \in T}{\operatorname{argmin}} \cdot \frac{1}{|S|} \sum_{s \in S} C_{s(e),t} \quad (6.1)$$

We conducted some experiments to check these two options of building the detector. In each experiment, we generated a random topology and used our simulator while putting a monitor on some of the routers. Then we used the tagger and the feature extractor in order to get feature vectors - one feature vector to each one of the monitors. After having the tagged examples, we checked the two options described above (building one classifier for all the monitors and building a classifier for each one of the monitors and then determining the final tag using the classifier predictions). In the second approach, we tested the different options of determining the final tag of an unseen example, as described above. We refer to these options as (1), (2), (3), (4), as they appeared in the above explanation. Table 6.2 presents the results achieved. In the different experiments done in this part, in each topology, we checked different sets of monitors. For each set, we used the feature vectors that are relevant to the chosen routers and conducted a stratified cross-validation experiment while recording the average weighted accuracy achieved.

From the above experiments, we concluded that the preferred option is to build a single classifier for all the monitors, that will act as the detector. This result confirms our hypothesis that when the classifier has more information about the traffic in the network (more routers’ points of view), its prediction on an unseen example will be more accurate. The case that emphasizes this idea is the following: Let assume that we have a network topology with  $n$  routers and we also have enough resources for locating  $n$  monitors - one on each router. In the scenario where we build one classifier for all the

monitors, this classifier actually receives all the routers' points of view, which includes all the messages that were sent or received by a router in the network. In this case specifically, no matter on which router the attacker takes control, the classifier also has the information from the attacker point of view. This situation can help the classifier in recognizing the presence of an attacker in the system. In the second option, when building a classifier for each monitor separately, each one of the classifiers predicts the tag for a new unseen example. The final tag is determined by the detector using these local predictions. With the same logic, no matter on which router the attacker takes control, there is a classifier that has the information from the attacker point of view. However, the detector takes this classifier's prediction into account with more  $n - 1$  predictions of the other classifiers, which reduces the effect of this classifier when determining the final tag. In this case, it is harder to get a general pattern of attacks and therefore the accuracy decreases.

### Extending the Feature Extractor

When building one classifier for more than one monitor, we use a feature vector which is the concatenation of the feature vectors of the different monitors. However, we can also take into account additional features that can be extracted from the example contains all the monitors' points of view of the prefix. These additional features are the results of applying mathematical operations on the features from each monitor. For example, consider the case of two monitors. Let  $f_i^{M_1}$  be the feature  $i$  of monitor  $M_1$  (e.g. the maximum over the difference between the number of links in two adjacent LSAs generated by the same router and received by  $M_1$ ). Let  $f_i^{M_2}$  be the same feature calculated for  $M_2$ . Then we can define a new feature  $f_i^{M_1, M_2}$  to be the result of some mathematical operation between  $f_i^{M_1}$  and  $f_i^{M_2}$ . Such a feature takes into account all the monitors' points of view and maybe can help to the learning process.

We conducted several experiments for testing the extended feature extractor in the case of using one classifier for more than one monitor. In each experiment, we generated a random topology and chose different combinations of more than one monitor. Then we used our simulator and tagger to get tagged examples. After that, we generated two feature vectors for each example (that includes the record of the traffic from the points of view of all the selected monitors): One with the regular feature extractor that concatenates the feature vectors of the different monitors to each other, and the other one with the extended feature extractor which adds additional features (describe relations between features of different monitors) to this concatenation. We used these two sets of feature vectors in order to execute two learning processes. We conducted stratified cross-validation experiments while recording the average weighted accuracy achieved in the different settings. In all the experiments we saw an increase of about 0.5% in the performance when using the extended feature extractor. This can indicate that these new shared features help a little to the learning process.

## 6.4 Optimal Set of Monitors

Using different combinations of routers to act as the monitors yields different accuracy results. We want to be able to choose the best set of monitors, one that does not contain a large number of monitors in order to save resources, and on the other hand, succeeds in achieving high performances in identifying the attacks. When having a small network topology, and enough resources, one can check all the possible combinations of routers for this purpose. However, in a large topology, this check is very expensive, having an exponential number of such combinations. Therefore, we decided to set the number of monitors,  $n_{monitors}$ , according to the customer resources. We implemented an algorithm for choosing the  $n_{monitors}$  monitors. The algorithm is based on the following idea: When we are interested in one monitor, the best choice is to choose the one that achieved the highest accuracy when working as a single monitor. Therefore we start with choosing this router. Then, if  $n_{monitors} > 1$ , we add more routers to the monitor set. For determining which router to add, we search for the  $k\%$  of the other routers, which achieved the highest accuracy results when working as a single monitor. We call this set of monitors  $BEST_k$ . From the set  $BEST_k$ , we choose to insert the router which is the farthest from the monitors that were already inserted to the monitor set. This idea relies on the thought that we want the monitors not to be grouped in one area of the network but rather to be spread out on large parts of it. Note that a distance of a router  $r$  from the monitor set is calculated as the length of the shortest path between  $r$  and a monitor  $m$  in the set. In addition, when we have more than one router, from the  $BEST_k$  set, with the same maximal distance from the monitor set, we select the one that achieved the highest accuracy when working as a single monitor and add it to the monitor set. The algorithm for choosing the monitor set is presented in algorithm 6.2.

---

**Algorithm 6.2** CHOOSEMONITORS( $n_{monitors}, k, N = (V, E)$ )

---

```

1:  $M \leftarrow \emptyset$ 
2:  $R \leftarrow V$ 
3: for  $i = 1, \dots, n_{monitors}$  do
4:   if  $M = \emptyset$  then
5:      $M \leftarrow \{\text{the router } r \in R \text{ which achieved the highest accuracy}\}$ 
6:      $R \leftarrow R \setminus \{r\}$ 
7:     continue
8:    $BEST_k \leftarrow k\% \text{ routers from } R \text{ with the highest accuracy results}$ 
9:    $r' \leftarrow \text{the router } r \in BEST_k \text{ that is farthest from } M$ 
10:   $M \leftarrow M \cup \{r'\}, R \leftarrow R \setminus \{r'\}$ 

```

---

## Experiments

We can divide the experiments we conducted in this section to two parts: The first one is done in order to test the performance of our algorithm for choosing the set of monitors. In the different experiments done in this part, we generated random topologies and

used our simulator while putting a monitor on each one of the routers in the topologies. Then we used the tagger and the feature extractor in order to get feature vectors - one feature vector for each monitor. In each topology, we checked different sets of monitors. One of these sets is the one selected by our algorithm using  $k = 40$ . For each set, we used the feature vectors that are relevant to the chosen routers and conducted stratified cross-validation experiments while recording the average weighted accuracy achieved. As we concluded that building one classifier for all the monitors is the preferred option, we conducted these experiments using this approach. Note that the case where we are interested only in one monitor is actually already displayed in section 6.2, as our algorithm choose the monitor with the highest accuracy in this case. For this reason, here we display the results of using more than one monitor. Table 6.3 presents the results achieved in this part.

From the experiments in this part we can see that our algorithm succeeds in selecting a set of monitors that yields high performance.

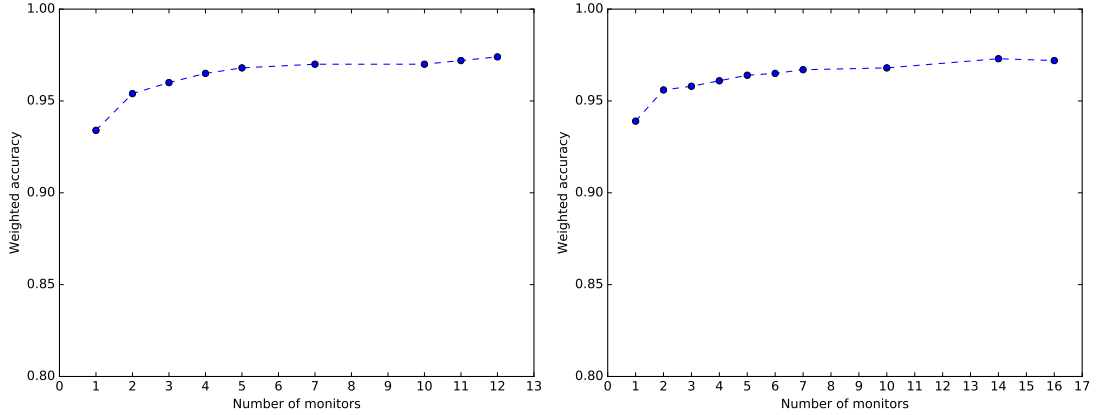


Figure 6.1: Weighted accuracy as a function of the number of monitors. Topologies with 12 and 16 routers, from left to right, respectively.

The second part includes experiments that check the number of monitors. As in many other fields in the domain of machine learning, such as increasing the size of the committee or increasing the size of the training set, investing more resources will improve the system performance until a certain threshold value is reached. At this point, it is not useful to invest more resources because the performance would not be improved significantly. We assume that in our setting we will see a similar phenomenon: Using a larger number of monitors will improve the system performance up to some point where it will not be useful to add more monitors. We conducted several experiments to check it. In the experiments, we generated random topologies and used our simulator while putting a monitor on each one of the routers in the topologies. Then we used the tagger and the feature extractor in order to get feature vectors. After that, we used our algorithm (with  $k = 40$ ) for choosing the monitors when examining different number of monitors. For each set selected by our algorithm, we used the feature vectors that are

relevant to the chosen routers for building the trained detector. We then conducted stratified cross-validation experiments while recording the average weighted accuracy achieved. Again, as we concluded that building one classifier for all the monitors is the preferred option, we conducted these experiments using this approach. Figure 6.1 presents the results of some of the experiments.

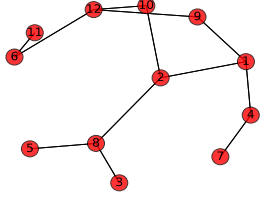
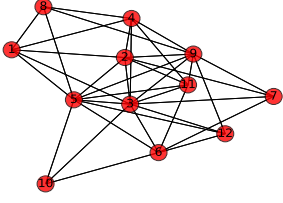
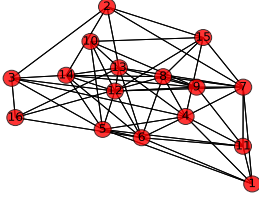
We can see the typical behavior we expected where performance improves while adding more monitors, helping to observe more information about the traffic in the network. Then, at some point, as expected, the improvement in performance slows down.



Table 6.2: BUILDING THE FINAL DETECTOR

Number of monitors	Number of routers	Method of building the classifier(s)	Method of determining the final prediction	Weighted accuracy
2	16	Single classifier	The classifier prediction	0.95
		Per monitor classifier	(1)	0.925
			(2)	0.903
			(3)	0.926
			(4)	0.903
	50	Single classifier	The classifier prediction	0.92
		Per monitor classifier	(1)	0.88
			(2)	0.89
			(3)	0.88
			(4)	0.89
3	12	Single classifier	The classifier prediction	0.96
		Per monitor classifier	(1)	0.934
			(2)	0.933
			(3)	0.94
			(4)	0.936
	16	Single classifier	The classifier prediction	0.958
		Per monitor classifier	(1)	0.943
			(2)	0.937
			(3)	0.943
			(4)	0.939
12	12	Single classifier	The classifier prediction	0.974
		Per monitor classifier	(1)	0.887
			(2)	0.92
			(3)	0.91
			(4)	0.925
16	16	Single classifier	The classifier prediction	0.972
		Per monitor classifier	(1)	0.924
			(2)	0.93
			(3)	0.93
			(4)	0.94

Table 6.3: THE LOCATIONS OF MORE THAN ONE MONITOR

The topology	Number of monitors	Location of the monitors	Weighted accuracy
	2	1, 2	0.892
		5, 11	0.848
		4, 10	0.884
		2, 4	0.893
		6, 8	0.876
		Our algorithm - 2, 12	0.893
	3	1, 2, 10	0.898
		5, 7, 9	0.896
		Our algorithm - 1, 2, 12	0.902
	2	6, 9	0.934
		11, 12	0.924
		2, 7	0.936
		3, 8	0.953
		4, 10	0.937
		Our algorithm - 3, 5	0.954
	3	1, 6, 12	0.952
		3, 8, 10	0.96
		Our algorithm - 2, 3, 5	0.96
	2	1, 2	0.937
		7, 8	0.953
		1, 10	0.932
		4, 9	0.952
		3, 7	0.953
		Our algorithm - 4, 12	0.956
	3	1, 2, 16	0.95
		2, 6, 13	0.954
		Our algorithm - 4, 7, 12	0.958

## Chapter 7

# Related Work

In [SP10] the authors emphasize the difficulties faced by researchers who use machine learning for network intrusion detection. In our work we face similar difficulties and provide successful solutions for dealing with these challenges.

The issue of attack detection has been studied over the years. There are two main approaches for the tagged examples, used as a training set for the learning process. One approach builds a model solely based on normal training examples and evaluates each testing case to see how well it fits the model. Some works, such as [SVG10, Bao09, MEAN13] use this approach for network intrusion detection. For example, in [TDJ<sup>+</sup>14] the authors present an approach for detecting sensor spoofing attacks on a cyber-physical system. They use only examples which were collected during normal runs of the system. By those examples, they learn a safety envelope which contains all the states that the system can reach when there is no sensor attack. Then they use this envelope to check if the system's state falls outside of it, and raise an alarm if so.

Another approach attempts to learn the distinction between normal and abnormal behaviors using both normal and attack examples in the training set. Works taking this approach include [CFF16, SM03, KJS17]. For example, in [CFF16] the authors used the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) dataset in which a hybrid of real modern normal activities and attack behaviors were generated. They used those examples for the training set of their learning algorithm (which includes a combination of two machine learning methods) in order to classify normal and abnormal behaviors. The accuracy achieved was 98.76%. In [KJS17] the authors created a detection algorithm and used the NSL-KDD dataset, which is a much improved version of the original KDDCUP'99 dataset, to evaluate it. This dataset contains 41 features and is tagged as either normal or an attack, with exactly one specific attack type. The authors used a combination of classifiers for their algorithm and classified incoming network traffic as normal or as an attack. The accuracy achieved was 89.24%.

However, while these works use existing examples as their training data, we generate the examples using sophisticated simulation. Generating the examples by our simulator

overcomes some issues that exist when using existing data (see Chapter 1), one of them is the ability to fit the training data to a specific network topology when using such a simulator. This ability can improve the accuracy rate because the training data is more specific to the testing setup.

Previous works analyzed OSPF security vulnerabilities ([SGN13, NSM<sup>+</sup>14]), searching for message falsification attacks ([WVW97]) and attacks with a persistent effect ([NSM<sup>+</sup>14]). These works are based on the assumption that the attacker sends legal OSPF messages in terms of their structure, but the router’s behaviour deviates from the protocol rules. Our work is also based on this assumption.

The works above search for vulnerabilities of the OSPF protocol, but cannot identify an attack at runtime. In contrast, in [TDJ<sup>+</sup>14] the authors build an attack detector monitor that runs continuously, monitors the state of the system, and raises an alarm whenever it believes there is an attack during the runtime. This work is different from ours in that it develops an attack detector to a different problem. More importantly, it uses anomaly detection, which means only non-attacked examples were used for training.

In our work we build a multiclass learning-based attack detector using both normal and attack examples for the training set. We use this detector in order to identify attacks on the OSPF protocol at runtime. Our work is different from other works also because it builds a detector that is suitable to the user’s setting, such as its network topology or the different costs it gives to the possible misclassification errors (as specified in the cost matrix).

## Chapter 8

# Discussion

### 8.1 Applying the Framework to Other Protocols

In order to apply our framework to other protocols than OSPF, two changes have to be done: The first one is the simulation itself (including the attacker simulation), which needs to fit the new protocol standards. The second one is the feature extraction, which also depends on the specific protocol. Note that there are features which are not related specifically to the OSPF protocol and can be used also for other routing protocols. However, there are also features which are very specific to the fields of the messages used in the OSPF protocol, which have to be changed when working with other protocols.

Despite those changes, there are several components that can remain unchanged even when working on another protocol: Using a simulator in order to achieve examples with all the possible tags, using relative and not absolute features for dealing with changes occur in the network, using our cost sensitive version of the Random Forest algorithm, and more.

In this work we developed a general framework for building an intrusion detector adapted to the user's setting. We fitted our framework to the OSPF protocol but it can be adjusted also to other protocols.

### 8.2 Working with Balanced Dataset

In all of our experiments, we used a balanced set of examples according to their tags. This case is not the typical one in reality, where the number of normal runs is much higher than the number of runs which contain an attacker. However, a balanced dataset enables a good assessment of our model. This assessment could not be achieved by using, for example, 90% of normal instances and 10% of attacked ones. In this case, even a basic classifier which always says "normal" is right in 90% of the cases. Therefore, by balancing the dataset, we can get a reliable assessment of our work.

## 8.3 Accuracy Measures

In order to evaluate our performances with respect to the cost matrix, we defined a new measure, weighted accuracy, which takes into account not only the number of successes but also the penalty on each one of the misclassification error types. However, we can also extend the discussion and talk about more regular classification measures. For doing that, we need to limit our tags, in order to talk about false negative and false positive. First, let us define some terms [HKP12]:

- **True positive:** These are the positive examples that were correctly labeled by the classifier. Let TP be the number of true positives in a given test set.
- **True negative:** These are the negative examples that were correctly labeled by the classifier. Let TN be the number of true negatives in a given test set.
- **False positive:** These are the negative examples that were incorrectly labeled as positive by the classifier. Let FP be the number of false positives in a given test set.
- **False negative:** These are the positive examples that were incorrectly labeled as negative by the classifier. Let FN be the number of false negatives in a given test set.

Table 8.1 summaries the terminology we used for using the terms above, with respect to the set of tags specified in Section 5.1. We refer to the *No Attack* tag as "negative" and to all the other tags (*Weak/Medium/Severe Attack*) as "positive".

Table 8.1: TERMINOLOGY

Name	Predicted tag	Real tag
True Positive	Weak/Medium/Severe Attack	Weak/Medium/Severe Attack
True Negative	No Attack	No Attack
False Positive	Weak/Medium/Severe Attack	No Attack
False Negative	No Attack	Weak/Medium/Severe Attack

### 8.3.1 Classification Measures

We are interested in the following classification measures:

- Sensitivity - measures the proportion of actual positives that are correctly identified as such. A high sensitivity test is reliable when its result is negative, since it rarely misdiagnoses those attacked runs. Mathematically,  $sensitivity = \frac{TP}{TP+FN}$ .
- Specificity - measures the proportion of actual negatives that are correctly identified as such. A high specificity test is reliable when its result is positive, since it rarely misdiagnoses those normal runs. Mathematically,  $specificity = \frac{TN}{TN+FP}$ .

Table 8.2: CLASSIFICATION MEASURES

Number of routers in the topology	Number of monitors	Sensitivity	Specificity
12	1	0.96	0.903
	2	0.985	0.975
	3	0.988	0.972
16	1	0.96	0.87
	2	0.98	0.91
	3	0.99	0.92

### 8.3.2 Experiments with the New Terminology

We conducted some experiments to check our performances in terms of the classification measures described in subsection 8.3.1. In each experiment, we generate random topologies with different number of monitors chosen by our algorithm (see section 6.4). Then we used our simulator, tagger and feature extractor to get tagged examples. We conducted a stratified cross-validation experiment while recording the classification measures using the new terminology. Table 8.2 describes the results of some of these experiments.

From the experimental results we can see that our algorithm achieves high performances in the terms of sensitivity and specificity. It can be seen that generally the algorithm is somewhat more reliable in providing a negative prediction (a higher performance in sensitivity), saying there is no attack in the system. Because in the vast majority of cases the implications of saying there is no attack when there is, are more severe than in the opposite case, this is an encouraging result, which implies that our algorithm is valuable. Here we can also see that having more monitors improves the performance in terms of sensitivity and specificity, similarly to the improvement in performance we mentioned in Section 6.4.





## Chapter 9

# Conclusions and Future Work

### 9.1 Conclusions

In this work we produced a new framework for example-based learning of an intrusion detector. We overcame the challenge of lacking data (especially data including attacks) to learn from. This is done by building a simulator, which generates the examples.

An important strength of our approach is its being adjustable to the requirements of a specific user. The user defines the possible tags, the network topology and the error cost for each misclassification error. These factors help in providing the user an intrusion detector that is fitted to his own needs.

Another added value of our framework is its ability to recommend the user of where to locate the monitor(s) in order to be able to receive good performances.

In addition, we tested our framework with a setup of transfer learning and received good accuracy rate. Such a setup can be used in the real world for shortening the time of the learning process. For example, if an organization has a topology of the same family as the topology of another organization, the first one can purchase the detector of the second one and avoid the training process which takes time and resources. Another setup we tested was dynamic networks, where we also received good results. This strengthen our model by adapting to dynamic networks as well as to stable ones.

Our new framework has a significant practical potential when the user can define his preferences and the system will produce a runtime intrusion detector that is adapted to this particular setting. We believe that the new learning framework, presented in this work, can be practical and useful for small as well as large organizations, and significantly enhances their network security.

### 9.2 Future Work

**Deep learning.** One area of future work is trying to use deep learning algorithms. Using our algorithm, we presented the features as values which are the result of a mathematical computation (mean, max, etc.) on several message fields or other data.

In the future, researchers can represent the traffic as a serial of messages and try to use Recurrent neural network (RNN) which works well with time-series.

**Possible commands for the user.** As mentioned in Section 2.2, our work is done based on the commands Cisco provides to the user for examining the mode of an OSPF run. This is reflected in the form of the traffic records and the features we have used. However, another direction of possible future work is to check if extending the commands the user can use in order to know the system states at different times can enable to achieve a higher accuracy. For example, a command that enables checking "what the router does with the arrived LSA - insert it to its database or ignore it" can help to extract a new feature and may improve the detector performances.

**The networks.** In this work we applied our model to networks with point-to-point connections between the routers. However, OSPF protocol is used also in more complicated networks, such as a network which has a border router that is used to establish a connection between backbone networks and different OSPF areas. It can be interesting to extend the simulation to such networks and test the detector performances.

**Other protocols.** As said in Section 8.1, one can use our framework for other protocols than OSPF. It is interesting to see the model performance for different protocols.

# Bibliography

- [Bao09] Cui-Mei Bao. Intrusion detection based on one-class SVM and SNMP MIB data. In *Proceedings of the Fifth International Conference on Information Assurance and Security, IAS 2009, Xi'An, China, 18-20 August 2009*, pages 346–349, 2009.
- [CFF16] Md Nasimuzzaman Chowdhury, Ken Ferens, and Mike Ferens. Network intrusion detection using machine learning. In *Proceedings of the International Conference on Security and Management (SAM)*, 2016.
- [DIJ59] E.W. DIJKSTRA. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [Fri17] Lior Friedman. Recursive feature generation for knowledge-based induction. Master’s thesis, Department of Computer Science, Technion – Israel Institute of Technology, 2017.
- [GM14] Wei Gao and Thomas H. Morris. On cyber attacks and signature based intrusion detection for MODBUS based industrial control systems. *JDFSL*, 9(1):37–56, 2014.
- [HKP12] Jiawei Han, Micheline Kamber, and Jian Pei. Data mining concepts and techniques, third edition, 2012.
- [HR76] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Inf. Process. Lett.*, 5(1):15–17, 1976.
- [HS14] Neminath Hubballi and Vinoth Suryanarayanan. False alarm minimization techniques in signature-based intrusion detection systems: A survey. *Computer Communications*, 49:1–17, 2014.
- [JM06] E. Jones and O. Le Moigne. OSPF Security Vulnerabilities Analysis. Internet-Draft draft-ietf-rpsec-ospf-vuln-02, IETF, June 2006.
- [KJS17] Jasmin Kevric, Samed Jukic, and Abdulhamit Subasi. An effective combining classifier approach using tree algorithms for network

- intrusion detection. *Neural Computing and Applications*, 28(S-1):1051–1058, 2017.
- [MEAN13] Eitan Menahem, Yuval Elovici, Nir Amar, and Gabi Nakibly. ACTIDS: an active strategy for detecting and localizing network attacks. In *AISec’13, Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*, pages 55–66, 2013.
- [MKSUKW12] Saif Malik, S K. Srinivasan, S U. Khan, and L Wang. A methodology for ospf routing protocol verification. 12 2012.
- [NKGB12] Gabi Nakibly, Alex Kirshon, Dima Gonikman, and Dan Boneh. Persistent OSPF attacks. In *Proceedings of NDSS*, 2012.
- [NSM<sup>+</sup>14] Gabi Nakibly, Adi Sosnovich, Eitan Menahem, Ariel Waizel, and Yuval Elovici. OSPF vulnerability to persistent poisoning attacks: a systematic analysis. In *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014, New Orleans, LA, USA, December 8-12, 2014*, pages 336–345, 2014.
- [SGN13] Adi Sosnovich, Orna Grumberg, and Gabi Nakibly. Finding security vulnerabilities in a network protocol using parameterized systems. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 724–739. Springer, 2013.
- [SM03] Andrew H. Sung and Srinivas Mukkamala. Identifying important features for intrusion detection using support vector machines and neural networks. In *2003 Symposium on Applications and the Internet (SAINT 2003), 27-31 January 2003 - Orlando, FL, USA, Proceedings*, pages 209–217, 2003.
- [SP10] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berkeley/Oakland, California, USA*, pages 305–316. IEEE Computer Society, 2010.
- [SVG10] Osman Salem, Sandrine Vaton, and Annie Gravey. A scalable, efficient and informative approach for anomaly-based intrusion detection systems: theory and practice. *Int. Journal of Network Management*, 20(5):271–293, 2010.

- [TDJ<sup>+</sup>14] Ashish Tiwari, Bruno Dutertre, Dejan Jovanovic, Thomas de Candia, Patrick Lincoln, John M. Rushby, Dorsa Sadigh, and Sanjit A. Seshia. Safety envelope for security. In *3rd International Conference on High Confidence Networked Systems (part of CPS Week), HiCoNS '14, Berlin, Germany, April 15-17, 2014*, pages 85–94, 2014.
- [WCJ<sup>+</sup>] S. F. Wu, H. C. Chang, F. Jou, F. Wang, F. Gong, C. Sargor, D. Qu, and R. Cleaveland. Jinao: Design and implementation of a scalable intrusion detection system for the OSPF routing protocol. *ACM Transactions on Computer Systems*, 2:251–273.
- [WWW97] Feiyi Wang, Brian Vetter, and Shyhtsun Felix Wu. Secure Routing Protocols: Theory and Practice. Technical report, North Carolina State University, May 1997. Found on Google only available copy is through Google view.

להשיג מידע של תעבורה אמיתית, הצלחנו להשיג מידע כזה ולהשתמש בו לצורך ביצוע ניסויים. בנוסף, בעבודה זו הצענו דרך לבחור את המיקום בו כדאי להציב את הגלאי במטרה לקבל ביצועים טובים.

המערכת שבנינו הינה בעלת פוטנציאל מעשי משמעותי. ארגון הרוצה לשפר את האבטחה שבו יכול בקלות להגדיר את טופולוגית הרשת שלו ואת מטריצת המחיר ולהתחיל בתהליך האימון. המערכת תייצר עבורו גלאי המסוגל לזהות התקפות בזמן אמת, באופן המתאים למאפיינים שהגדיר. בנוסף, האלגוריתם שלנו יוכל להמליץ לארגון על איזה מבין הנתבים כדאי למקם את הגלאי שנבנה.

העבודה שלנו מהווה יישום של טכניקות למידה לצורך שיפור האבטחה באינטרנט. התרומה העיקרית של עבודתנו הינה פיתוח של גלאי התקפות לצורך זיהוי התקפות בזמן אמת, תוך התאמה לדרישות המשתמש. הייחודיות שבמערכת שלנו הינה התאמה לדרישות הספציפיות שמגדיר המשתמש (כגון מבנה הרשת, התיוגים האפשריים וכן המחיר עבור השגיאות השונות). מערכת זו תוכל לאפשר לארגון להגן על עצמו מפני פעילות זדונית העשויה להתרחש ברשת שלו. באופן ספציפי יותר, תרומות המחקר כוללות:

- יצירת דוגמאות המתאימות לטופולוגיה ספציפית, לצורך תהליך האימון.
- אפשרות למשתמש להגדיר חומרות התקפה.
- פיתוח מודל שעמיד בפני מצבים בהם הרשת אותה רוצים לחזות אינה זהה לרשת עליה אומנה המערכת.
- שימוש בגרסת cost sensitive של אלגוריתם למידה מוכר. השינויים שביצענו באלגוריתם מתחשבים במטריצת המחיר שהגדיר המשתמש.
- נתינת המלצה למשתמש לגבי מיקום הגלאי על גבי אחד הנתבים ברשת, במטרה להשיג ביצועים טובים.

בעבודה זו אנו מציגים מערכת חדשה מבוססת למידה חישובית, שמטרתה לבנות מודל שיגלה התקפות בזמן אמת. הגישה שלנו מתגברת על המחסור בדוגמאות לצורך שלב האימון על ידי כך שהיא כוללת בתוכה סימולאטור המייצר את הדוגמאות. האלגוריתם בו נעשה שימוש לצורך יצירת הדוגמאות מחקה הן התקפות והן התנהגות נורמאלית עבור טופולוגית רשת נתונה. סימולציית ההתקפות נעשית באמצעות מודל כללי של תוקף. במודל זה, התקפה נוצרת על ידי פעולה שבוצעה על ידי הנתב עליו השתלט התוקף, אשר אינה עולה קנה אחד עם התנהגותו הנורמאלית של הפרוטוקול. התקפות שונות עשויות להכיל סדרה של פעולות זדוניות שנבחרו באקראי על ידי התוקף. אימון המערכת על דוגמאות הנוצרות באופן זה מאפשרות לו להיות מסוגל לזהות התקפות חדשות בעלות מאפיינים שאינם ידועים.

הפעילות המתרחשת ברשת מתועדת ומועברת לשלב התיוג האוטומטי לצורך קביעה האם מדובר בריצה נורמאלית או בהתקפה (ואם כן, באיזו חומרת תקיפה). לאחר מכן הדוגמה המתויגת מועברת לאלגוריתם המחליץ ממנה תכונות שעשויות להיות רלוונטיות לצורך קביעת התיוג עבור דוגמאות חדשות. לאחר תהליך זה, המתבצע עבור כל הדוגמאות שיוצר הסימולאטור, אנו מקבלים אוסף דוגמאות, בעל כל סוגי התיוג הקיימים, המהווה את קבוצת האימון שלנו. המערכת שלנו מאפשרת למשתמש להגדיר מטריצת מחיר בה מוגדר המחיר של כל שגיאת חיזוי. דבר זה הביא אותנו ליישם גרסה שהיא cost-sensitive עבור אלגוריתם למידה ידוע. בסופו של התהליך קיבלנו גלאי התקפות שאומן על אוסף הדוגמאות המתויגות שיצרנו ועל כן הוא מותאם באופן ספציפי לדרישות המשתמש ולטופולוגית הרשת שהוא סיפק.

אנו מימשנו את המערכת עבור פרוטוקול התקשורת הנפוץ OSPF. פרוטוקול זה מאפשר לנתבים ברשת לקבל תמונה מלאה של מבנה הרשת ובהתאם לכך לחשב את טבלאות הניתוב שלהם במטרה לדעת לאיזה שכן להעביר הודעה מסוימת על מנת שתגיע ליעד שלה במסלול האופטימאלי. ישות שמבצעת התקפה על פרוטוקול זה יכולה להשפיע עליו בצורה הרסנית. נתב זדוני אחד הקיים במערכת יכול לזהם את טבלאות הניתוב של כל הנתבים האחרים ברשת על ידי שליחת הודעות המכילות מידע שקרי. באופן זה לפעולות התוקף תהיה השפעה על אופן הניתוב כולו. המערכת שאנו בנינו מתעדת את התעבורה העוברת ברשת, ולא את טבלאות הניתוב של הנתבים השונים. ניטור פעילות המערכת באופן זה מאפשרת לגלאי לזהות התקפות ברגע שהן מתרחשות, אפילו לפני שיש להן השפעה על טבלאות הניתוב. זיהוי התקפה בהתבסס על שינויים שנעשו בטבלאות הניתוב עלול להיות מאוחר מידי, שכן הנזק כבר נגרם. מציאת התקפות בפרוטוקולי תקשורת הינה משימה בעלת קושי רב, שכן אופי ההתקפה וצורתה עשויים להיות לא ידועים.

אנו ערכנו מגוון רחב של ניסויים במטרה לבחון את ביצועי המערכת שלנו. סביבת הניסויים שלנו כללה אלגוריתם ליצירת טופולוגיות, במטרה לבחון את ביצועי המערכת שלנו עבור סביבות שונות. הניסויים שביצענו הראו תוצאות מבטיחות, עם שגיאה נמוכה מאוד. כחלק מהניסויים, הראנו שהמערכת שבנינו יכולה לעמוד בפני מצבים של transfer learning ורשתות דינאמיות, בהם החיזוי מבוצע על רשת שונה מזו שהמערכת אומנה עליה. כמו כן, על אף הקושי

# תקציר

במהלך השנים השימוש באינטרנט הלך וגדל והשימושים שלו התרחבו והפכו חשובים לאנשים וארגונים רבים ברחבי העולם. במקביל לכך, מתקפות זדוניות הפכו לאיום ממשי על פעילות התקינה של האינטרנט בכל העולם ולכן עלתה חשיבותה של בניית כלים אוטומטיים שיזהו מתקפות כאלה.

גלאי חדירה לרשת מתוכננים למצוא פעילות זדונית על ידי זיהוי הפרות של פרוטוקולים. אנו מעוניינים לזהות פעולות אלה כמה שיותר מוקדם על מנת למנוע מהתוקף לפגוע באופן חמור בתשתיות. עם זאת, ישנם אתגרים רבים שהופכים את פעולת זיהוי החדירה לרשת לקשה יותר מתחומים אחרים. אחד האתגרים המשמעותיים קשור לרקע בו מערכת כזאת פועלת: בתחום של זיהוי התקפות ישנו מרוץ מתמשך בין התוקפים למגינים על הרשת כאשר כל אחד מן הצדדים מנסה לשפר ולשכלל את יכולתיו. כחלק מזה, התוקפים מנסים להסתיר את פעולותיהם ולגרום להן להיות דומות לפעולות הרגילות המתרחשות במערכת. מתקפות כאלה מכילות לעיתים מספר קטן של פעולות הזרות במבנה שלהן לפעולות חוקיות המתבצעות בפרוטוקול, ומכאן הקושי לזהותן.

רוב הניסיונות ההתחלתיים לבנות גלאי התקפות הסתמכו על הגדרה ידנית של חוקים או דפוסים שמאפיינים מתקפות. עם זאת, גישות אלו סובלות מכמה חולשות. ראשית, קשה מאוד להגדיר אוסף חוקים שיכסה מגוון רחב של התקפות. שנית, אוסף החוקים הזה יהיה מסוגל לזהות רק התקפות ידועות בעלות מאפיינים ותבניות מוכרות.

כדי להתגבר על קשיים אלה, פותחו שיטות חדשות מבוססות למידה חישובית. מאחר ודוגמאות חיוביות (של התקפות) הינן קשות מאוד להשגה, לא ניתן היה להשתמש באלגוריתמים המערבים למידה מדוגמאות בעלות כל סוגי התיוג הקיימים (כאלה עם התקפות וכאלה ללא התקפה). לכן היה צורך להצטמצם לשיטות של one class, בהן נעשה ניסיון להשתמש בדוגמאות שליליות בלבד לצורך בניית מודל שישקף כיצד נראית פעילות מהימנה של הרשת, ולאחר מכן להשוות התנהגות חדשה מול מודל זה. עם זאת, גם לגישה זו ישנם מספר קשיים משמעותיים. המרכזי שבהם הוא האופי הדינאמי של רשתות, בהן רוחב הפס או חיבורים בין נתבים עלולים להשתנות. שינויים שכאלה נחשבים נורמאליים ומתרחשים ברשתות על בסיס קבוע, דבר המקשה על האפיון של התנהגות נורמאלית.

על מנת להתמודד עם קושי זה, נעשו ניסיונות להשתמש בדוגמאות עם כל סוגי התיוג האפשריים (כאלה עם התקפות וכאלה ללא התקפה) בשלב הלמידה. עבודות אלו הניחו שישנם מאגרים קיימים של דוגמאות מסומנות כאלה. הנחה זו, עם זאת, אינה מתיישבת עם המציאות. ראשית, מאגרים כאלה הינם קבועים ומתיישנים, בעוד אסטרטגיות התוקפים הולכות ומתפתחות. לכן, מודל שאומן על סמך מאגרי מידע אלה ייתקל בבעיה בעת ניסיון זיהוי של התקפות חדשות יותר. בנוסף, התוקפים יכולים לכלול בהתקפותיהם מאפיינים שונים של הרשת הספציפית אותה הם תוקפים. מכיוון שכך, מודל שאומן על רשת שונה לא יצליח לבצע הכללה נכונה על הרשת הנחזית, דבר שיביא לביצועים נמוכים. לבסוף, השגת מאגר מידע שכזה הינו משימה קשה מאוד, שכן חברות אינן מעוניינות לפרסם את התעבורה העוברת ברשת שלהן, שעשויה להכיל מידע רגיש.





המחקר בוצע בהנחייתם של פרופסור ארנה גרימברג ופרופסור שאול מרקוביץ', בפקולטה למדעי המחשב.

## תודות

ראשית ברצוני להודות למנחה האחראית שלי, פרופ' ארנה גרימברג. תודה על הפגישות השבועיות, הנכונות להירתם ולעזור גם כאשר התחום רחוק יחסית מנושאי המחקר המרכזיים שלך. תודה על העידוד, ההכוונה ועל תשומת הלב גם לפרטים הקטנים ביותר. תודה על שהיית זמינה עבורי בכל זמן, לעיתים גם בשעות פחות קונבנציונאליות. זכיתי במנחה מדהימה, הן מבחינה מקצועית והן מבחינה אישית, וברצוני להודות לך על כך שגרמת לתקופה הזאת להיות כ"כ נעימה עבורי. אני רוצה להודות גם למנחה השותף שלי, פרופ' שאול מרקוביץ', על כך שהצליח להקדיש לי זמן ומחשבה גם בזמנים עמוסים ולחוצים. תודה על נקודות המבט הנוספות שסיפקת לי, על הרעיונות החדשים ועל החשיבה החדה. תודה על חלקך הגדול בעבודה זו. כמו כן, ברצוני להודות לדר' גבי נקיבלי. תודה על ההכוונה והחשיפה לתחום שאינני שולטת בו. תודה על הנכונות לפנות לגופים שונים במטרה לתרום לאיכותו של המחקר. תודה על שיתוף הפעולה ועל העזרה והייעוץ במהלך המחקר. מחקר זה נתמך בחלקו על ידי המרכז לאבטחת סייבר ע"ש הירושי פוגיווארה שבטכניון וברצוני להודות לו על כך. אני רוצה להודות לידידי הראל קין. תודה על הזמינות שלך, על הרצון להשקיע מחשבה וזמן בשביל לקדם את המחקר. תודה על העצות הקטנות והגדולות שנתת לי לאורך הדרך. הידיעה שיש לי מישהו שבשמחה יירתם לעזרתי בכל בעיה, קטנה או גדולה, נתנה לי תחושת ביטחון ורוגע. תודה לך על התחושה הטובה והעזרה הרבה, גם בזמנים פחות שגרתיים. ברצוני להודות לחבריי שהיו שותפים איתי במסע הזה. תודה על כך שהייתם שם גם בזמנים לחוצים יותר. תודה על העידוד, התמיכה והעזרה בעת הצורך. בלעדיכם הדרך הזו הייתה קשה הרבה יותר. לבסוף, ברצוני להודות למשפחתי. להורים שלי, מנחם ושושי, לאחותי מיכל, לאחי וגיסתי – אורי ורעות, וכן לשלושת האחיינים שלי – אריאל, יואב ושחר. תודה על ההנאה והשמחה שהכנסתם לתקופה הזאת. תודה על עזרתכם, על הליווי, התמיכה והאהבה האינסופית. תודה על כך שתמיד הייתם שם בשבילי. תודה להוריי על שלימדו אותי לכוון גבוה, להשקיע ולא להתייאש, גם כשעדיין לא רואים את האור שבקצה המנהרה. תודה על הדחיפה וההירתמות לעזור בכל זמן. בלעדיכם זה לא היה קורה. תודה מיוחדת לאחותי מיכל על העזרה המקצועית הרבה. אני אסירת תודה לך על שתרתם לי מהידע הרב ומהחשיבה היצירתית שלך, בצורה נעימה ואוהבת. אומרים שמשפחה לא בוחרים, אבל לו הייתה לי היכולת לבחור – הייתי בוחרת בכם. תודה לכם.

הכרת תודה מסורה לטכניון על מימון מחקר זה.



# **שימוש בלמידה חישובית לצורך זיהוי התקפות בזמן אמת**

**חיבור על מחקר**

לשם מילוי חלקי של הדרישות לקבלת התואר  
מגיסטר למדעים במדעי המחשב

**נורית דביר**

הוגש לסנט הטכניון – מכון טכנולוגי לישראל  
אדר א' התשע"ט    חיפה    פברואר 2019



# שימוש בלמידה חישובית לצורך זיהוי התקפות בזמן אמת

נורית דביר