Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Detection of Malicious Network Behaviour in Encrypted Network Traffic

Bc. Pavel Potoček

Thesis Advisor:
Ing. Martin Rehák, Ph.D.

Praha, 2017

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Pavel Potoček**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Detection of Malicious Network Behaviour in Encrypted Network Traffic**

Guidelines:

1. Study and describe the state-of-the-art algorithms for statistical anomaly detection and outlier detection, with emphasis on the algorithms used in the network security domain.

2. Improve the existing algorithms (or algorithms designed by team members) so that they can be used to identify the malicious traffic in ciphered channels, such as TLS.

3. Collaborate on software design, implementation and evaluation of new algorithms. Concentrate on the questions of parameter optimisation and empirical evaluation.

Bibliography/Sources:

Aggarwal, Charu C. Outlier analysis. Springer Science & Business Media, 2013.

Oppliger, Rolf. SSL and TLS: Theory and Practice. Artech House, 2014.

Chandola, Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM computing surveys (CSUR) 41.3 (2009): 15.

Diploma Thesis Supervisor: Ing. Martin Rehák, Ph.D.

Valid until the end of the summer semester of academic year 2016/2017

L.S.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-4-

prof. Ing. Filip Železný, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 18, 2016

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ……………...   …………………….
Podpis autora práce

Acknowledgments

My sincere thanks goes to my supervisor Martin Rehák for introducing me to the field of network security and providing me with an opportunity to work at the Cisco Cognitive Threat Analytics.

I am extremely grateful to Martin Grill and Jan Kohout for their guidance and help over the whole course of writing my thesis. Especially Martin Grill's contributions in later stages of the writing process were invaluable.

Abstract

Computer networks are facing threats of ever-increasing frequency and sophistication. With encryption becoming the norm in both legitimate and malicious network traffic, Intrusion Detection Systems (IDS) are now required to work efficiently regardless of encryption. In this thesis, we develop a method designed to improve the efficacy of the Cisco Cognitive Threat Analytics IDS system by sharing intelligence across a large number of enterprise networks. Intelligence sharing provides additional information to the intrusion detection process, which is much needed particularly for analysis of encrypted traffic with inherently low information content.

We experimentally evaluate our new method in four variants on real network traffic data, including a variant that employs a novel outlier ensemble normalization algorithm in presence of missing data. We show that our intelligence sharing method greatly improves detection efficacy for networks with bad baseline detection efficacy and slightly improves upon the average case. Robustness of the novel outlier ensemble normalization algorithm is also demonstrated. These improvements were measured on encrypted as well as non-encrypted network traffic.

Keywords: Intrusion Detection System, outlier ensemble model, anomaly detection, encrypted network traffic

Anotace

Počítačové sítě se potýkají s čím dál tím sofistikovamějšími a frekventovanějšími frebezpečnostními hrozbami. S tím jak se šifrování postupně stává normou, systémy pro detekci průniku (Intrusion Detection Systems, IDS) nyní musí efektivně fungovat i nad šifrovanou komunikací. Tato práce se zabývá návrhem metody pro vylepšení efektivity IDS systému Cisco Cognitive Threat Analytics s pomocí sdílení informací ve velkém počtu monitorovaných podnikových sítí.

Navrženou metodu experimentálně porovnáme ve čtyřech variantách, včetně varianty využívající nový algoritmus pro normalizaci detektorů anomálií v přítomnosti chybějících dat. Experimenty ukazují, že použitím navržených metod se výrazně zlepší efektivita detekce průniku v sítích se špatnou schopností detekce a mírně se zlepší efektivita v průměrném případě. Dále je demonstrována robustnost nového algoritmu pro normalizaci detektorů anomálií. Tato zlepšení byla naměřena nejen na šifrované, ale i na nešifrované komunikaci.

Klíčová slova: systém pro detekci průniku, detekce anomálií, šifrovaná síťová komunikace

Contents

1 Introduction **14**

1.1 Challenges in anomaly-based IDS 15

1.2 Thesis Outline 18

1.3 Cognitive Threat Analytics IDS 18

1.4 Thesis goals 21

1.5 Proposed architecture 22

2 Related Work **25**

2.1 Outlier Ensemble Models 26

2.1.1 Model normalization 27

2.1.2 Model combination 29

2.2 Dealing with missing values 30

3 Proposed Algorithm **32**

3.1 Global reputation model as an outlier ensemble 32

3.2 Normalization 33

3.3 Combination 35

3.4 Missing values 35

3.5 Pairwise optimization of normalization error 35

3.5.1 Derivation 35

3.5.2 Time Complexity 37

3.6 Reputation model usage scenario 37

4 Experimental evaluation **39**

4.1 Dataset description 39

4.2 Evaluation criteria 41

4.3 Global reputation table evaluation 41

4.4 Improvement of per-flow anomaly scores 44

5 Conclusions **49**

Bibliography **49**

A Convexity of the error function *E* **53**

B CD Contents **56**

List of Figures

1.1 Anomaly processing pipeline architecture 20
1.2 Global reputation model overview illustrated on three networks . . . 24

3.1 Horizontally and vertically sampled ensemble illustration 33
3.2 Error function decay illustration 34

4.1 Flow counts per network 40
4.2 Network intersection sizes 42
4.3 Global reputation table ROC curve comparison on the biggest 100 networks 43
4.4 Comparison of AUCs of per-flow anomaly scores 46
4.5 Network AUC score vs. network traffic size on all input data 48

List of Tables

1.1 HTTP(s) flow fields 19

4.1 Characteristics of the dataset used for experimental evaluation . . . 39

4.2 Global reputation table AUC score comparison on the biggest 100 networks 42

4.3 Comparison of AUCs of per-flow anomaly scores 47

List of abbreviations

malware	malicious software
IDS	Intrusion Detection System
C&C	Command & Control
CTA	Cognitive Threat Analytics
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over Transport Layer Security
URI	Uniform Resource Identifier
flow	proxy log entry
LAMS	Locally Adaptive Multivariate Sampling
EM	Expectation Maximization
CDF	Cumulative Distribution Function
ROC	Receiver Operating Characteristic
AUC	Area Under the ROC Curve
TPR	True Positive Rate
FPR	False Positive Rate

Chapter 1

Introduction

Computer networks are facing threats of ever-increasing frequency and sophistication. Recent years have seen an increase in both targeted and automated attacks, orchestrated by well-funded or even state-level adversaries. To mitigate those threats, sophisticated and layered defense mechanisms are needed. First line of defense is typically deployed on the network perimeter: a boundary of a network that is composed of various Internet-facing devices including boundary routers and firewalls. These are configured to block any easily-identifiable illegitimate traffic [1]. The second line of defense is an Intrusion Detection System (IDS): a system built to analyze events within the network for any evidence of ongoing malicious activities. Such evidence is reported to the network administrators so they can react accordingly [2]. IDSs can operate on various levels of computer system architecture and can be classified into two broad categories.

Host-based IDS monitors events occurring within a single host for suspicious activity. Characteristics such as network activity, system logs, running processes or application activity can be measured. A typical example of a host-based IDS is antivirus software preforming static analysis. Host-based IDSs are most commonly deployed on publicly available machines or machines containing sensitive information.

Network-based IDS monitors network traffic to detect malicious traffic flows, such as denial of service (DOS) attacks or traffic generated by malicious software (malware) on hosts in the network. Traffic within organization's network can be monitored, as well as traffic to and from external networks (e.g., the Internet).

From now on, we will discuss only network-based IDS and refer to them simply as IDS, since host-based IDS are outside the scope of this thesis.

Methodologies used by network-based IDSs can be categorized by the level of inspection (deep vs. shallow packet inspection) or by the method of detection (signature-based vs. anomaly-based techniques). While many systems use them in combination, different categories have different characteristics that will be discussed below.

First, categorization by the level of inspection (deep vs. shallow packet inspection):

Deep Packet Inspection exploits the data contained in packet payloads and operates on the Application layer of the network stack. Application-specific detection techniques can be deployed targeting, for example, JavaScript code in web pages [3] or DNS NXDomain responses [4]. With encryption becoming the norm [5], deep packet inspection is becoming non-effective since the packet payloads are inaccessible. Encrypted traffic can either be disregarded, lowering the recall, or a man-in-the-middle attack is performed, which may not be possible or desirable in various circumstances. Moreover, the sheer volume of transferred data often makes these techniques impractical: there may not be enough resources available to deeply analyze a significant portion of network traffic. Their advantage is, however, that they have more information available and e.g., signature-based detection may be largely impossible without deep packet inspection.

Shallow Packet Inspection exploits only information in packet headers and meta-information such as request-response delay. This method is much less granular than deep packet inspection, but it may be used on larger volumes of data and it's function is not impacted as much by encryption.

Categorization by the method of detection (signature-based vs. anomaly-based):

Signature-based detection systems search network traffic for predefined signatures indicating malicious behavior. Their advantages are low false positive and false negative rates and a comparatively simple design. However, they rely on expert-created signatures that are expensive and time-consuming to create, and can respond to new threats only with a non-trivial delay. While used extensively in the past, they are becoming less relevant due to increases in threat variability, pervasive use of encryption and other evasive techniques.

Anomaly-based detection systems rely on anomaly detection, which is the problem of finding patterns in data (called outliers or anomalies) that do not conform to expected behavior [6]. The major benefit of anomaly-based methods is the ability to detect previously unknown threats or whole classes of attacks based on broad behavioral characteristics rather than precise signatures. This often comes at the expense of higher false-positive rates and greater complexity. High false-positive rates cause a need for manual sifting through the data to extract actionable knowledge, which limits the utility of anomaly-based IDS in practice.

In this thesis we focus on improving an anomaly-based Intrusion Detection System, so we will describe the characteristics of such systems in greater detail in the following text.

1.1 Challenges in anomaly-based IDS

There are several common challenges that all anomaly-based IDSs face [7]. It is argued in [8], that due to these challenges, network anomaly detection is a fundamentally harder problem than classical machine-learning tasks such as classification.

These challenges heavily influence the design decisions of any such system. In order to understand the design and evaluation decisions in this thesis, we will discuss them in this chapter.

Outlier detection

Network anomaly detection is an outlier detection problem. However, machine learning algorithms excel much more at finding similarities and patterns rather than data that does not conform to those patterns. This problem can be side-stepped by viewing anomaly detection as a classification problem with two categories: background (legitimate) and anomaly (malicious). To learn efficiently, machine learning algorithms typically need examples of *all* categories, preferably comprehensively capturing each category's characteristics. We can learn using known malicious samples, but this leads to a scenario where our algorithm is well suited to finding variants of known behavior, not novel malicious activity [8]. If it is desirable to capture novel threats, then by definition, training samples of only one class are available. This leads to a situation where we are trying to find a class based on characteristics this class does *not* have, which is certainly not ideal. In order to do that, we need an accurate model of background behavior, which proves challenging to construct given the large diversity of network traffic discussed in the next paragraph.

Diversity of network traffic

Due to the fact that the Internet is a universal system for communication, there is an immense number of applications, sites, services and users, all contributing to the diversity of network traffic. When network behaviour is marked as anomalous, it may be because it is a rare (but legitimate) form of behavior observed for the first time instead of it being malicious. This can lead to high false alarm rates of IDSs, limiting their utility in practice. The impacts of this problem are exacerbated by class imbalance and the resulting high false-positive costs discussed below.

Class imbalance

The amount of background data is, by definition, much greater than the number of anomalies, and the difference is often many orders of magnitude. In order to have acceptable precision, due to the base-rate bias, the false positive rate of an anomaly detector must be kept extremely low [9]. Also, due to class imbalance, many classification algorithms are not directly applicable as they may require classes to be of comparable sizes. Smart sampling or cost weighting strategies may need to be employed to overcome this issue.

High cost of errors

The cost of classification errors is extremely high compared to the cost of errors in other fields of machine learning (character recognition, image classification, *etc.*). Alarms generated by an IDS are analyzed by human operators and each false alarm costs valuable time and resources. Even a very small rate of false positives can quickly render an IDS unusable [9]. On the other hand, false negatives have a potential to cause serious damage: even a single compromised system can severely undermine the integrity of an enterprise network.

Data volume

The volume of network traffic that needs to be analyzed is huge. This alone can make deep inspection of the traffic unfeasible, so only metadata such as connection type and timing is often exploited. Even in that case, the number of records can be large: The IDS developed by Cisco (see Section 1.3) reports more than 10 billion analyzed requests daily [10]. This places stringent limits on algorithmic complexity. IDSs are frequently built with a complex layered architecture, decreasing with each layer the amount of processed data and increasing with each layer algorithmic complexity.

Low quality training data

There is no recent and common training dataset for IDS [11, 8] and there are several reasons for it. High-quality training datasets for IDS development are typically very expensive to construct. It is possible to capture real-world traffic from a network and label parts of the traffic as malicious by hand, but this is a costly and laborious process and due to the complexity of the task and large data volume, the results may not be very reliable. Another option is to use real-world or simulated background network traffic and add malicious traffic gathered from infected virtual machines. However, it is difficult to gather realistic network traffic that does not already contain some malicious behavior. Moreover, the proportion and characteristics of artificially-introduced malicious traffic may not be realistic.

The biggest hurdle to publishing datasets in the security domain is, however, privacy concerns. Data may be sanitized by removing or anonymizing potentially sensitive information, but this may introduce artifacts and limit its utility for IDS development [12] and significant amounts of sensitive information may still be possible to extract [13].

Perhaps mainly due to these concerns, the availability of public training datasets is poor. This is best illustrated by the fact that a synthetic dataset KDD Cup '99 [14] created in the year 1999 is still widely used by the research community as shown in an extensive survey [11]. This is despite it's age and serious flaws that have been discovered over time [15].

1.2 Thesis Outline

The remainder of the thesis is structured as follows.

In the rest of **Chapter 1**, we look at the Intrusion Detection Systems in more detail. We start by describing the concrete instance of an IDS that we are working on: Cisco Cognitive Threat Analytics. We identify its weaknesses and propose an outline of a reputation model designed to improve its performance by integrating intelligence gathered from a large number of client networks.

In **Chapter 2**, we investigate relevant state of the art in relation to the design and requirements of the reputation model.

In **Chapter 3**, we propose several variants of the reputation model. One of the variants is a novel model designed for outlier ensemble normalization in presence of a substantial number of missing values.

In **Chapter 4**, we experimentally evaluate the proposed reputation model variants on real-world network traffic and discuss the results.

In **Chapter 5**, we summarize the results and conclude this thesis.

1.3 Cognitive Threat Analytics IDS

Cognitive Threat Analytics (CTA) [16], developed by Cisco Systems, Inc., is a cloud-based software-as-a-service product designed to detect infections on client machines. It functions as a network-based IDS, analyzing proxy logs produced by web proxies on a network perimeter. It is able to to discover data exfiltration, the use of domain generation algorithms[1], infections by exploit kits, malicious tunneling through HTTP(S) and command-and-control (C&C) communications [10].

CTA IDS focuses solely on analyzing traffic using the HTTP and HTTPS protocols. The rationale is that when malware communicates with a C&C server, it frequently does so using standard HTTP(S) protocols in order to blend in with the vast amounts of legitimate traffic that is typically generated in any network [17]. Moreover, communication protocols other than HTTP(S) are not universally available as they tend to be filtered out by networks.

The input of the CTA IDS anomaly detection pipeline is a proxy log. In the remainder of this thesis, we will call proxy log entries *network flows* or simply *flows*. Each flow contains various fields extracted from the HTTP(S) headers, such as time of the request, source and destination IP addresses, HTTP method, downloaded and uploaded bytes, user agent, *etc.* The full list is in Table 1.1. The content of the flows differs considerably for HTTP and HTTPS traffic. For HTTP, each log entry consists of a single HTTP request and response. For example, a typical web page triggers many flows of the GET method, because various assets need to be

[1] Domain generation algorithms generate an arbitrary number of domain names to avoid detection and blacklisting of hosts that provide malware.

Field name	**Field description**	**HTTPS**
x-timestamp-unix	timestamp	p
x-elapsed-time	elapsed time	p
sc-http-status	HTTP status	$\emptyset$
sc-bytes	bytes up	p
cs-bytes	bytes down	p
cs-uri-scheme	URI scheme	**Y**
cs-host	URI host	**Y**note1
cs-uri-port	URI port	**Y**
cs-uri-path	URI path	$\emptyset$
cs-uri-query	URI query	$\emptyset$
cs-username	user name	$\emptyset$
s-ip	server IP	**Y**
c-ip	client IP	**Y**
Content-Type	MIME type	$\emptyset$
cs-referer	HTTP referer	$\emptyset$
cs-method	request method	$\emptyset$
cs-user-agent	User-Agent	$\emptyset$

Table 1.1: HTTP(s) flow fields. Fields marked with "**Y**" are available in HTTPS, fields marked with "p" provide only partial information in HTTPS flows due to the fact that many requests may be grouped in one HTTPS flow, and the fields marked by $\emptyset$ are not available at all. Only 4 out of 17 fields are unaffected by the HTTPS protocol.
note1: The cs-host field is not available in HTTPS traffic, but the host name can be extracted from HTTPS certificate information.

downloaded separately (HTTP is a stateless protocol). On the other hand, HTTPS communication is hidden inside an encrypted tunnel which is typically left open for a time and may contain many HTTP requests. The whole tunnel is represented by a single flow with the method set to CONNECT. This means that not only is the information that is encrypted unavailable in HTTPS (such as the user agent string or the resource URI), also meta-information such as request size, request count and timing information is mangled. This makes the encrypted communication much less information-rich than non-encrypted and this fact is one of the key motivations for this thesis.

CTA uses multiple *anomaly detectors* to assign multiple anomaly values to individual network flows. Since each anomaly detector is (by design) a very weak classifier on its own, a layered *anomaly processing pipeline* is employed to combine the anomaly values into a single final per-flow *anomaly score*. The flows with high anomaly score are clustered, assigned labels and severities and finally sorted in the *post-processing* step. The results are displayed to a network operator so they can react accordingly. The main steps of this process are described in more detail in the following paragraphs. For a more in-depth treatment, see [7].

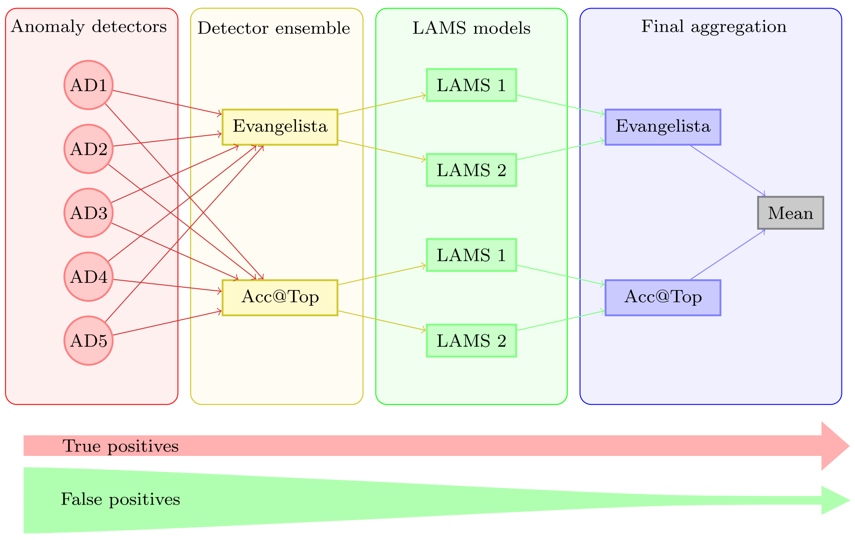

Figure 1.1: Anomaly processing pipeline architecture. The output of the Anomaly detectors is processed in three layers. The first layer aggregates the inputs using two functions: Evangelista and Acc@Top. The second layer uses LAMS models to smooth the detector ensemble outputs and decrease the false positive rate. Finally, the outputs from different LAMS models is once again aggregated by the Evangelista and Acc@Top functions and their results are averaged together to produce the final anomaly score. (image taken from [7]).

Anomaly detectors are used to compute anomaly scores from input flows by using information in a single flow as well as fusing information from the previously-encountered flows and aggregating it (in e.g. per-host traffic volume statistics). There were more than 40 separate anomaly detectors in CTA at the time of writing of this thesis. Each of them is a weak classifier with a real-valued output in the interval $[0, 1]$, zero is background and one is anomaly. The detectors are designed to be fast, single-purposed, diverse from one another and to have high recall (sensitivity) at the cost of low precision. Because they are diverse, false positives can later be filtered out in the anomaly processing pipeline, increasing the precision of the output.

Anomaly processing pipeline starts with the anomaly detector outputs and converts them into a single value with high classification strength. This process is illustrated in Figure 1.1. Detector outputs are processed in three layers. The first layer aggregates the results of anomaly detectors using two functions: the unsupervised Evangelista function and the supervised Acc@Top function. These functions are designed to increase the precision and preserve recall. The second layer uses LAMS models (described below) to smooth the anomaly values and decrease the false-positive rate. Finally, the outputs from different LAMS models are once again aggregated by the Evangelista and Acc@Top functions and their results are averaged together to produce the final anomaly score.

LAMS (Locally Adaptive Multivariate Smoothing) [18] is a method designed to lower false positive rates by smoothing the output of a detector. This way, accidental spikes in anomaly score are ignored while locally-repetitive patterns of malicious behavior are preserved. It works by replacing an output of a detector by a weighted average of itself over similar samples. Similarity is defined a Gaussian function of Euclidean distance in a predefined feature space[18]. Multiple different feature spaces are used in the pipeline in parallel to achieve better classification strength.

Post-processing of the final anomaly score is the process of converting the numerical output of the anomaly processing pipeline into actionable intelligence that can be presented to a network operator. It involves clustering the flows, attaching labels and severities, sorting by anomaly scores and displaying the output. It is not relevant to this thesis and will not be further discussed.

To sum up, Cognitive Threat Analytics IDS is an anomaly-based IDS operating on network traffic. Features are extracted from individual network flows and processed in an anomaly processing pipeline with the end result of displaying actionable intelligence to a network operator.

As noted above, network traffic using both HTTP and HTTPS protocols is analyzed, but the amount of information that can be extracted from HTTPS traffic is heavily reduced in comparison to HTTP traffic. With many companies and organizations advocating encryption and initiatives such as Let's encrypt decreasing adoption difficulties (for both malicious and legitimate entities), recent years have witnessed a dramatic increase in encryption pervasiveness. It is now a must to develop threat detection algorithms that perform well on encrypted traffic and to improve the performance of existing algorithms thereon.

1.4 Thesis goals

The main goal of this thesis is to improve the detection efficacy of the existing CTA anomaly-based network IDS (described in Section 1.3) on malware that uses network traffic encryption to avoid detection. To achieve this, we utilize information that is currently not utilized by the CTA IDS: inter-network data correlations. Although the CTA IDS is employed on hundreds of different networks, the existing anomaly processing pipeline makes no attempt to share information between individual networks and all the information that is used for anomaly detection is network-local.

We propose a novel way of improving the efficacy of anomaly detection by using threat intelligence gathered from a large number of enterprise networks that are all separately monitored by the CTA IDS. We create a global intelligence database that is shared among all the participating IDS systems and use it to improve their anomaly detection performance on both encrypted and non-encrypted network traffic.

There are several ways in which global intelligence sharing could improve detection capabilities of participating networks.

- There are networks that use the man-in-the-middle (MiTM) attack to inspect HTTPS connections and provide the full set of features for the individual HTTP requests that were sent through an HTTPS tunnel. Intelligence gathered from those networks can be used to improve detection efficacy on networks that do not use MiTM inspection.
- Some false positives might be specific to a single network and can be removed when using global intelligence. For example, there might be flows that are labeled as anomalous in the context of one network, but are labeled as normal in other networks, which would suggest that it is a case of a false positive. An example of this type of a false positive might be an access to the Yandex search engine that could be easily considered anomalous in American companies, but it is the most commonly used search engine in Russia.
- The detection efficacy on smaller, geographically-local networks may be poor, because there may not be enough data to meaningfully initialize the system and generate strong baseline of normal behaviour in the individual anomaly detectors. These networks can benefit importing information from well-initialized networks.

Apart from improving detection in participating networks, global intelligence can also be useful in its own right. For example, a global reputation list can be built to help network analysts assess threat severity, or the gathered intelligence may be used to better understand global statistics of network threat behavior (determine originating countries, scale of an attack, C&C mobility, *etc.*)

We set out three specific goals for this thesis.

1. Build a global intelligence sharing system in several variants on top of the CTA Intrusion Detection System.
2. Evaluate the performance of the system variants when used as global reputation lists.
3. Use the global intelligence to improve upon existing anomaly detection capabilities in individual networks and measure the results on both HTTPS and HTTP traffic.

1.5 Proposed architecture

In order to share intelligence pertaining to a part of observed traffic in a network, a way to identify similar parts of traffic in other networks must be devised. There are two ways to identify similarities in network traffic: remote identity identification or behavioral correlation.

Remote identity identification (using hostname[2], IP address, *etc.*) is a simple, crisp way to correlate traffic across networks. It can be used in a straightforward manner to assign reputation scores to different endpoints or form a blacklist of malicious entities. It is a standard and functional way to enhance network security. Identity identification offers an added benefit of reliably binning captured HTTPS traffic together with the corresponding traffic where an MiTM attack was used to inspect the contents. It can also be easily interpreted by a human operator. On the other hand, pervasive use of identity indentification has led to adaptation: attackers frequently change their originating hosts to make blacklisting less effective.

Behavioral correlation (using timing, request sizes, request methods, *etc.*) is able to overcome such evasive tactics as the resulting features are designed not to be tied to a single identity, but there is a steep cost to it: it is complex, fuzzy and unreliable. Moreover, behavioral correlation techniques must exploit a wide range of traffic characteristics and sharing these may raise privacy concerns in involved parties.

Because of its relative merits, we will use identity identification as the basis for our intelligence sharing system. There are several ways of identifying the remote party that are available for network flows: URI, hostname, IP address or the second-level domain. Out of these, we will use hostname as is the most specific identity information that is available for both encrypted and non-encrypted flows apart from the IP address. IP addresses are potentially more specific but also less meaningful; many IP addresses may be used to serve identical resources and be distinct just for the purpose of load balancing. Hostname is not always available; in cases it isn't, its value will be substituted by the IP address.

We propose a global intelligence sharing model, henceforth called the **global reputation model**, which uses information gathered in individual networks to globally associate anomaly values to individual hostnames. The overall reputation model structure is visualized in Figure 1.2. There are three parts to this reputation model (represented by blue arrows in the figure):

1. Aggregation of anomaly scores over hostnames, producing a set of *local reputation models*. This part can be implemented simply as an average of per-flow anomaly values (details in Section 3.1).

2. Combination of *local reputation models* to form a *global reputation model*. This part is much more involved than the others and presents difficult challenges such as normalization of individual local reputation tables. The remainder of this thesis mainly focuses on the design and implementation of this part of the overall intelligence sharing algorithm.

[2] A hostname is in the context of the Internet a fully qualified domain name that specifies the address of a host computer, e.g. *en.wikipedia.org*.

3. improvement of anomaly detection in individual networks using the global reputation model. This part can be implemented by replacing individual flow anomaly scores with hostname reputation values (details in Section 3.6).

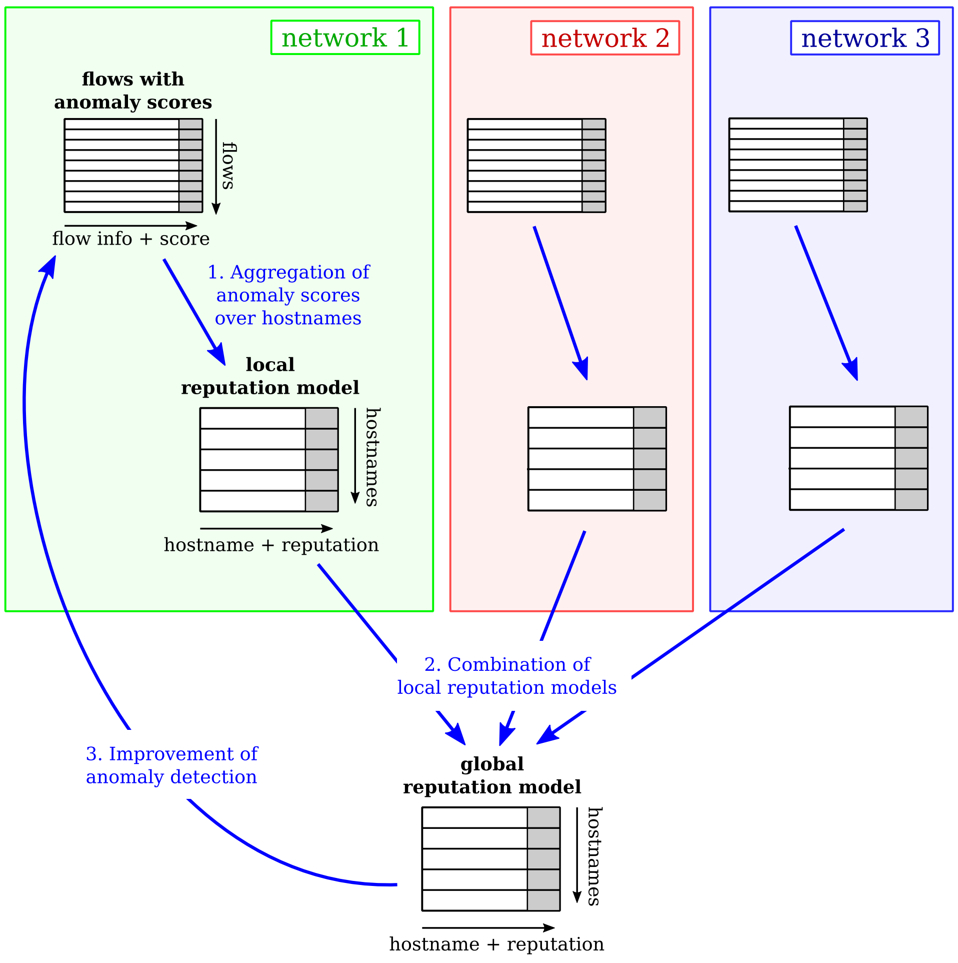

Figure 1.2: Global reputation model overview illustrated on three networks. The input is a list of flows with associated anomaly scores (produced by the anomaly processing pipeline of the CTA IDS), the output is a global reputation model. The global reputation model can be used in its own right or to enhance detection strengths in individual networks. Blue arrows represent parts of the global reputation model algorithm that are developed and evaluated as in this thesis.

Chapter 2

Related Work

In this chapter, we investigate the state of the art related to building a reputation model and discuss challenges in application to our domain.

It should be noted that the proposed reputation model is *not* an instance of a *reputation system.* Reputation systems let parties rate each other and use aggregated ratings about a given party to derive a trust or reputation score [19]. Notable examples of reputation systems include Google's PageRank [20] or user rating systems on many e-commerce sites. The main challenge faced by reputation systems is that the parties rate *each other;* trust is transitive and potentially malicious parties are not only rated, but they themselves also rate. We do *not* face this challenge: in our case, the rating parties (client networks) and rated parties (remote servers) are disjoint.

A better fitting abstraction to our problem is that of *outlier ensemble models.* In contrast to better-known classifier ensembles, outlier ensembles are designed to be applicable in an unsupervised setting and their design characteristics are much closer to our needs as stated in Section 1.1. Individual client networks can be viewed as individual outlier detectors and a global reputation model can be viewed as their ensemble. The output of each outlier detector (client network) is in this case a local reputation model, associating an outlier score to every observed hostname in each network[1].

One of the challenges is that individual outlier detectors observe different sets of hostnames. As argued in Section 3.1, this property can be treated systematically as an instance of missing data. In the following chapters, we investigate these ideas further. We start by introducing outlier ensemble models. Then, we explore relevant design decisions in outlier ensembles and discuss how they apply to our setting. Finally, we assess the impact of missing values on outlier ensembles and discuss algorithms dealing with this issue.

[1]We can arrive at the local reputation model (for example) by simply averaging the anomaly values of all flows corresponding to each single hostname.

2.1 Outlier Ensemble Models

In machine learning, ensemble models are the techniques of combining the outputs of multiple data-mining algorithms to produce a single algorithm [21, 22]. Because different algorithms make mistakes in different parts of the input space, this ensemble is often more accurate than any of its constituent parts [23, 24, 25]. Ensemble models are widely studied and successfully used in problems such as clustering and classification, but their usage is relatively sparse in the field of outlier detection. Two factors have been major impediments to the success of ensemble models in outlier detection: class imbalance and the unsupervised nature of the problem. Class imbalance refers to the issue that the number of outliers is (by definition) comparatively small. This makes it difficult to evaluate the success of the algorithm in a statistically robust way and presents a danger of over-fitting. The unsupervised nature refers to the fact that objective ground truth is often not available to be used to evaluate the quality of components in the ensemble, making it necessary to develop only simple algorithms with few qualitative choices. Nevertheless, outlier ensembles have been treated in literature a number of times and recent years have seen a number of significant advances in the topic [26].

Aggarwal [27] divides outlier ensemble methods into two categories:

Model-centered ensembles combine the outlier scores from different algorithms (models) built on the same data set. The major challenge is that the outputs of models are not directly comparable. It may even be the case that high outlier scores corresponds to high outlier probabilities in one algorithm and low outlier probabilities in another. Good normalization of the algorithm outputs is crucial for good ensemble performance.

Data-centered ensembles use only a single outlier detection algorithm, but apply it to different subsets or functions of the input data in order to produce an ensemble. Horizontal sampling (a sample of data points) or vertical sampling (a subspace of feature space) can be employed. The earliest formalized outlier ensemble method [28] called *feature bagging* falls into the category of data-centered ensembles, employing vertical sampling of random feature sub-spaces. Because only a single outlier detection algorithm is used, the outlier scores are not nearly as heterogeneous as is the case in model-centered ensembles and normalization does not play such a central role. To improve performance, the sampling procedure can be adjusted as well as the final combination function, and it can still be beneficial to employ normalization as seen in the original feature bagging article [28] or in [29].

Our reputation model can be viewed as a data-centered outlier ensemble, where each ensemble model is a network and data points are the observed hostnames. A typical outlier ensemble meta-algorithm contains three components that are used to arrive at the final result:

1. *Model creation.* This is the methodology used to create the individual algorithms (models) that form an ensemble. In data-centered ensembles, it may be, for example, a particular way of sampling input data points or feature dimensions.
2. *Model normalization.* Different models may produce outlier scores on different scales. In order for the scores to be comparable, they must be normalized.
3. *Model combination.* The normalized outlier scores are combined to form a single outlier score. A (weighted) average or maximum are common choices of combination functions.

The first component, model creation, is not relevant to this thesis, since the individual models are already given as individual network outputs. The next two steps, however, are necessary components of our reputation model and will be discussed in the following sub-sections.

2.1.1 Model normalization

The simplest approach to model normalization is not to normalize at all. This is possible for some data-centered ensembles where the models do not differ substantially in their output characteristics. For our purposes, this is certainly a viable approach since our ensemble is data-centered and it is unclear how, if at all, do the constituent model outputs statistically differ. Their differences may also be meaningful and not accidental, and normalization may in fact decrease the ensemble performance. That being said, normalization has been shown to be beneficial in the case of data-centered ensembles multiple times [28, 29]. Several approaches to model normalization have been proposed in literature and will be discussed below. A good summary of available normalization methods can be found in [30].

Two key properties of outlier scores for the purposes of model normalization are *regularity* and *normality.* An outlier score S is called *regular* if for any scored object o, $S(o) \geq 0$, and $S(o) \approx 0$ for an inlier and $S(o) \gg 0$ for an outlier. S is called *normal* if S is regular and $S(o) \in [0, 1]$ for any o. A process of making S regular, resp. normal while preserving the rankings of different data points is called *regularization,* resp. *normalization.* Normally, an outlier score is first regularized, then normalized. Normalization can be useful even if S is already normal: the requirements for normality are too lose to guarantee the possibility of meaningful outlier score combination in an ensemble.

Perhaps the simplest regularization method is simple linear scaling. A baseline value $base_S$ is chosen, equal to the expected inlier value. Then, the function S can be regularized as follows:

$$S_{reg}(o) = \max\left\{0, \pm_s \left(S(o) - base_S\right)\right\}.$$

The sign ($\pm_s$) depends on whether low outlier score $S(o)$ indicates that o is an outlier (plus sign is used) or conversely (minus sign is used). This regularization method is often used as a preprocessing step to make it easier to apply one of the normalization functions below. A similarly trivial *normalization* technique would be to scale S_{reg} to fit within the $[0, 1]$ interval in the following way:

$$S_{norm}(o) = \frac{S_{reg}(o)}{\max_o S_{reg}(o)}.$$

Since the maximum function is not a robust statistic, S_{norm} normalization is not robust either and it is useful more as a preprocessing step to reduce gross calibration differences than as a final normalization function. In our case, outlier scores are already normalized to begin with so the transformations above are not of any use to us.

A somewhat more involved approach to normalization is discarding precise outlier scores and using their ranks only (scaled to fit the range $[0, 1]$) [28, 30]. This neatly solves the normalization issue and makes the outputs of wildly different outlier detection algorithms comparable, but loses a lot of information in the process. The absolute differences between outlier scores are disregarded so there is no way for individual algorithms to express the amount of their confidence in a sample being an outlier or an inlier. Moreover, there is no way to express that there is no anomalous sample at all. In our case, nearly half of the networks do not contain any anomalies so this makes the rank-only normalization unsuitable.

Another option is to transform outlier scores into probability distributions. Two methods for achieving this goal are described in [29]. The first method assumes that the posterior probability of a sample being an outlier follows a logistic function in relation to its outlier score and learns the parameters of this function from the distribution of outlier scores. The second method models the outlier scores as a mixture of two distributions: exponential for the background and Gaussian for the outliers. Parameters of these distributions are learned and posterior probabilities are calculated using the Bayes's rule. In order to learn in an unsupervised manner, a generalized Expectation Maximization (EM) algorithm is used to jointly learn the distribution parameters and classifications of data points. Authors show modest improvements over the plain feature bagging [28] ensemble model. There are, however, several problems with this approach. It favors extreme values (0 and 1) in the resulting probabilities, which does not lend itself well to combination [30]. Moreover, it is not very stable due to the usage of the EM algorithm and the results can degenerate if there are only few outliers in a sample, which is frequently the case in the network security field. This means that this form of the EM algorithm is not applicable in our case.

Kriegel *et al.* [30] transforms outlier scores into probability values using a different approach. They assume that the outlier scores follow a chosen distribution, estimate its parameters and use its cumulative distribution function to arrive at the probability values. For example, for a normal distribution, an outlier score $S(\cdot)$ is transformed into a normalized score $Norm_1(\cdot)$ as follows:

$$Norm_1(o) = max\left\{0, 2 \cdot cdf_{gauss}(S(o)) - 1\right\},$$

where $cdf_x(\cdot)$ is a cumulative distribution function of an estimated distribution x. Scaling and maximum are used because we are interested only in a one-tailed probability value: an average case is assigned the outlier score of zero. Note that the probability $Norm_1(\cdot)$ can not be directly interpreted as a probability of the sample being an outlier; this is not needed since the goal is just making the values comparable. Instead of the Gaussian distribution, other distributions can be used in the same manner; the paper proposes also the use of uniform and gamma distributions. Although it is argued that the precise choice of a distribution is not very important, Gaussian scaling shows the best results in the majority of tests.

The last option we will discuss here is normalization without assuming a concrete probability distribution. Instead, the sample means can be normalized to zero and residuals to one by subtracting the sample mean μ and dividing by sample standard deviation σ [31]:

$$Norm_2(o) = \frac{S(o) - \mu}{\sigma}.$$

This is similar to the Gaussian normalization $Norm_1$ discussed above, but for the use of the cdf_{gauss} function. However, the result is not guaranteed to lie in the interval $[0, 1]$, and high values are not exponentially dampened by the cumulative distribution function that exponentially approaches one.

After being normalized by one of the techniques described above, the ensemble model outputs must be combined to form a single value. In the next section, we describe such model combination functions.

2.1.2 Model combination

There are several choices of the model combination function $f : \mathbb{R}^N \to \mathbb{R}$. The following paragraphs discuss the most common model combination functions.

Average is the obvious choice of a model combination function and as such, it has been used extensively in literature [30, 28]. It is quite robust to outliers and its estimation is unbiased regardless of the sample size (in contrast to e.g. maximum). One problem with Average has been identified though: if many models return irrelevant results (for example, when only the minority of models is expected to assign a high score to an outlier), the average could get diluted. In our model, we can't expect the average to be get diluted by this mechanism, but the converse may be true: many models could assign high anomaly scores to inliers (background traffic) due to them being observed for the first time. This may be a reason to consider other combination methods, but owning to its robustness, average is the obvious choice.

Maximum is useful for setups where the models in the ensemble display high precision and low recall. It is one of the most common combination functions and it is used particularly in parameter tuning scenarios: the individual models represent the same algorithm over identical data, but with different parameters. The ensemble then picks the strongest result with the maximum function. This choice was discussed in the classical Local Outlier Factor (LOF) paper [32] that implicitly used an outlier ensemble model. Maximum estimation is is meaningful for comparison only if all the data points have the same number of components in the ensemble. For our model, this is not true and neither do our ensemble models have high precision and low recall, so Maximum is not useful for us.

Minimum classifies a point as an outlier only if **all** the models in the ensemble classify it as such. Therefore, it could be useful for models with low precision and high recall. For vertically sampled ensembles this is not the case, because by vertical sampling, recall is diminished. On the other hand, for many outlier detection algorithms, horizontal sampling preserves recall and diminishes precision, so minimum can be a good choice. Horizontal sampling in outlier ensembles is not common so the minimum function is rarely (if at all) used. As is the case for maximum, all data points must have the same number of components in the ensemble, and this excludes Minimum from consideration as our combination function of choice.

Pruned average is a way to overcome the diluting effects of a simple average. Low anomaly scores are discarded (either by tresholding or by using only top n scores), and the remaining scores are averaged. The problems with this approach are that it contains parameters that must be tuned (which is difficult to do in an unsupervised setting), it is computationally expensive and some issues of comparability between the scores of different data-points arise. Despite the fact it combines the benefits of average and maximum, it is rarely used in practice [27].

Damped average applies a dampening function, such as a square root or logarithm, to the values before averaging them. This is used to prevent the result from being dominated by few extreme values. Geometric mean is a special case of damped average, where the dampening function is a logarithm. It does not seem necessary to implement dampening in our case, since the anomaly scores are already normalized and so very extreme values can not appear.

2.2 Dealing with missing values

When our reputation model is viewed as a horizontally-sampled ensemble (Equation 3.2), there are so many missing values (specifically, 99.2 %) that they require careful treatment. There are several conventional techniques used to deal with missing values [33]:

Complete-case analysis, also known as **listwise deletion**. In this method, the entire record is excluded from analysis if a single value is missing. This method is known to behave well even if values are not missing completely at random [34]. In our case that means that only the hostnames that are found in *all* of the networks would be used for analysis. This is difficult, because such a set of hostnames is actually empty. A way to get a non-empty set is to use only a subset of available networks. This, however, leads to a very small resulting dataset, which in turn leads to large estimation errors.

Available-case analysis, also known as **pairwise deletion**. For many models, the parameters of interest can be expressed using population mean, variance, correlation[2] and other population statistics. In available-case analysis, each of these is estimated using all of the available data for each variable or pair of variables. In our case, assuming we need the mean and standard deviation estimates, the entirety of the available data would be used, but the sets of hostnames that each network normalization is dependent on would be different. Intuitively, this would give better results than complete-case analysis due to the much bigger dataset available, but it also introduces additional bias if the probability of a value being missing is correlated to the anomaly score, and it introduces bias in estimating standard deviation due to the different sample sizes.

Data imputation. Many methods fall into this category; the common trait is that these methods guess or estimate the missing values before doing the analysis. A popular approach is simply to substitute means of existent values for missing values. Another method is to estimate the missing values using the maximum likelihood method, or draw the missing values from a chosen distribution multiple times and aggregate the results. If the substituted values are biased, the results of analysis based on them will also be biased. Due to the extremely large number of missing values in our case, data imputation seems not to be the right choice. The methods with strong theoretical guarantees (maximum likelihood and multiple imputation) need precise probabilistic models of the data which are unfortunately not available in our case.

In this chapter, we discussed the state of the art relevant to building a reputation model using an ensemble of outlier detectors. In the next chapter, we will use that knowledge to design several variants of the reputation model that can later be evaluated and compared.

[2]This is where the name *pairwise deletion* originates: to compute correlation, pairs of cases are needed.

Chapter 3

Proposed Algorithm

In this chapter, we propose several variants of the reputation model based on the considerations in the previous chapter. First, we show how the reputation model can be viewed as an outlier ensemble model. Then, we present our implementation choices of the three parts of the ensemble model that were identified in Section 2.1: normalization, combination, and the way of dealing with missing values. Lastly, we propose and analyze the properties of a novel normalization algorithm that is designed to suffer from missing values less than the state-of-the-art methods.

3.1 Global reputation model as an outlier ensemble

In order to use the outlier ensemble abstraction, we must first transform our input from the form of network flows with anomaly scores into a *local reputation mode*l (step 1. in Figure 1.2). We do this by treating each flow's anomaly score as an estimate of the corresponding hostname anomaly score. We then estimate the hostname anomaly scores by averaging the anomaly scores of all flows that communicate with said hostname in a single network. This simple procedure arrives at the local reputation model.

The *global reputation model* can be viewed as a data-centered outlier ensemble with horizontal sampling. Let H be the set of all existent hostnames and $H_n \subseteq H$ the set of all hostnames that were observed in the network $n \in N$. Let $f_n : H_n \to \mathbb{R}$ be the anomaly score of a hostname measured in the network n. Now, H_n is the horizontal sampling of H, and the sets of anomaly scores observed in individual networks form an ensemble E_h:

$$E_h = \{\{f_n(h) \mid h \in H_n\} \mid n \in N\}. \tag{3.1}$$

Horizontally sampled ensembles are routinely used in classification: classical examples are Bootstrap Aggregating [35] and Random Forests [36]. They have not, however, found use in outlier ensemble models. Intuitively, this is understandable: horizontal sampling leads to dilution of dense groups of data points which significantly hampers our ability to identify outliers. To be able to utilize the literature

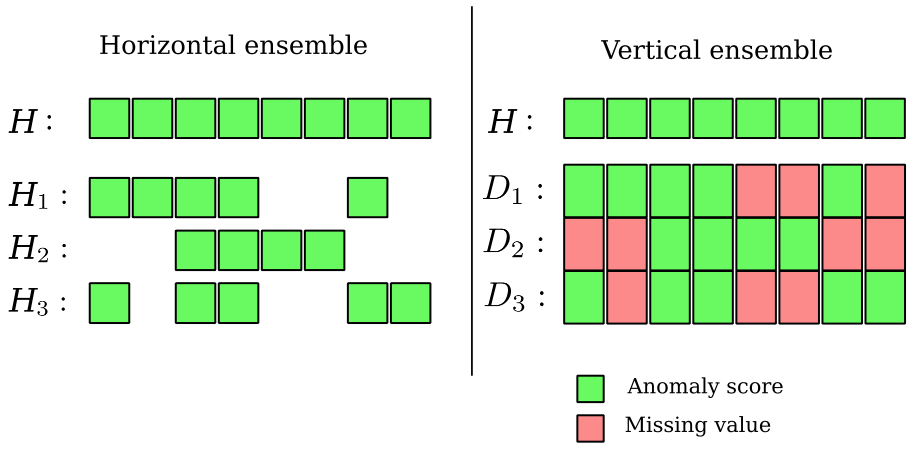

Figure 3.1: Horizontally and vertically sampled ensemble illustration. In the case of the horizontally-sampled ensemble (left), we treat each network as a sampling H_n of a set of hostnames H. In the case of the vertically-sampled ensemble (right), we treat each network as a dimension D_n in a feature space of hostnames. Sampling is then over the dimensions of the feature space, one dimension at a time, and values are missing whenever a hostname was not observed in a given network.

around outlier ensembles we re-define the reputation model as an ensemble with *vertical* sampling by allowing the existence of missing values. The ensemble is now

$$E_v = \{[f_n(h) \mid n \in N] \mid h \in H\}, \tag{3.2}$$

where $[f_n(h) \mid n \in N]$ is an ordered set of the anomaly scores of h in all networks. Since not all of the hostnames are observed in every network (the domain of f_n is a subset of H), there are missing values in the ensemble. The difference between these two representations is illustrated in Figure 3.1.

If we view the reputation model as a vertically-sampled ensemble, we must explicitly deal with missing values. The shortcomings of existing algorithms designed for estimation with missing values (discussed in Section 2.2) motivate the design of a novel ensemble normalization algorithm described later in this chapter.

3.2 Normalization

Based on the experiments conducted in [30], the best-performing normalization variant across a wide array of scenarios was Gaussian scaling:

$$Norm_{gauss}(o) = max\{0, 2 \cdot cdf_{gauss}(S(o)) - 1\}, \tag{3.3}$$

where o is a network flow, S is an outlier score cdf_{gauss} is the cumulative distribution function (CDF) of a Gaussian distribution with parameters estimated from the data.

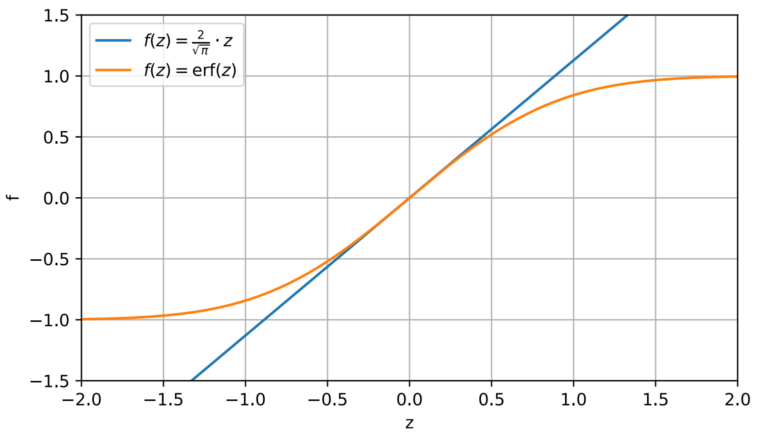

Figure 3.2: Error function decay illustration. The blue line is the first term of the Taylor expansion of $\mathrm{erf}(z)$. When used as a transformation function, the linear approximation better preserves values far from zero than the $\mathrm{erf}(z)$ function.

This can be rewritten using the Gauss error function erf:

$$Norm_{gauss}(o) = max\left\{0, erf\left(\frac{S(o) - \mu}{\sqrt{\pi}\sigma}\right)\right\}, \tag{3.4}$$

where μ, σ are the estimated mean and standard deviation. For our purposes, however, a change to this equation is warranted. As the models in our ensemble are identical (only the data is not) and we do not require the result to be inside the $[0, 1]$ interval, we can get rid of the exponential decay towards 1 of the erf function. This decay (illustrated in Figure 3.2) makes the scores of outliers less pronounced and could arguably lead to their dilution. When erf is substituted by its linear approximation around origin and the maximum function is removed[1], we get:

$$Norm_{lin}(o) = \frac{2}{\sqrt{\pi}} \cdot \frac{S(o) - \mu}{\sqrt{\pi}\sigma} \propto \frac{S(o) - \mu}{\sigma}. \tag{3.5}$$

This coincides with the standardization procedure which is a frequently used technique in ensemble outlier detection [31]. We will use this normalization function as one choice for experimental evaluation. Second choice will be the identity function: $Norm_{id}(o) = S(o)$. This will provide a baseline so we can evaluate how much (if at all) the results improve when normalization is employed.

[1]It does not have any purpose if we do not require the score to be normal (inside the $[0, 1]$ interval).

3.3 Combination

As discussed in in Section 2.1.2, only variants of the average function (average, pruned average, damped average) are suitable for our algorithm. Pruned average requires a pruning threshold to be set and damped average requires a dampening function. For neither of those, it is obvious how to choose these parameters in an unsupervised setting. Therefore, we will use a simple average as our combination function, consistently with much of the prior art on outlier ensembles.

3.4 Missing values

Among the methods useful for dealing with missing values described in Section 2.2, we choose for evaluation both complete-case analysis and available-case analysis. Because of the large proportion of missing values, we will not pursue data imputation approaches. To have enough complete cases for *complete-case analysis*, we will use only a subset of networks in experiments with this technique.

In addition to the standard methods of complete-case and available-case analysis, we propose a novel method that is designed to enable ensemble normalization with higher tolerance to missing values than either of the methods above. We call this method *pairwise optimization of normalization error* and we describe it in detail in the next section.

3.5 Pairwise optimization of normalization error

This method is similar to *pairwise deletion* (available-case analysis) when used to estimate a correlation matrix. In that case, pairwise deletion would use pairs of networks n_i, n_j to compute correlation coefficients $c_{i,j}$ and build a correlation matrix $[c_{i,j}]$. For each pair of networks n_i, n_j, a set of common hostnames would be used to compute the correlation; hostnames missing from either of them would be discarded. Much greater proportion of samples is utilized in this way than is the case with *complete-case analysis* and only identical sets of hostnames are ever compared together in contrast to *available-case analysis.* However, the resulting correlation matrix $[c_{i,j}]$ is not guaranteed to be positive-semidefinite.

Since our goal is not estimating the correlation matrix, but normalizing outlier scores, we build a normalization error function instead of a correlation matrix, but we do it in much the same pairwise manner. This error function encompasses the statistical differences of all pairs of networks and by minimizing it, we can estimate the parameters that minimize said differences.

3.5.1 Derivation

Let N be a set of all networks. Each network $n \in N$ contains a set of hostnames, H_n. Let $f_n : H_n \to \mathbb{R}$ be an anomaly score of a hostname $h \in H_n$ in network n. Let

$g_{\theta_n} : \mathbb{R} \to \mathbb{R}$ be a normalization function parameterized by θ_n. We wish to arrive at the parameters θ_n that normalize best the anomaly scores $f_n(\cdot)$. Using these parameters, it is possible to compute the normalized anomaly scores $q_{n,h}$:

$$q_{n,h} = g_{\theta_n}(f_n(h)),\ h \in H_n. \tag{3.6}$$

We can now use the least squares optimization method to find the best parameters θ_n. For every pair of networks n, m, the normalization error function $E_{n,m}$ is:

$$\begin{aligned} E_{n,m} &= \sum_{h \in H_n \cap H_m} \left(g_{\theta_n}(f_n(h)) - g_{\theta_m}(f_m(h))\right)^2 \\ &= \sum_{h \in H_n \cap H_m} \left(q_{n,h} - q_{m,h}\right)^2, \end{aligned} \tag{3.7}$$

where we substituted in the normalized anomaly scores $q_{n,h}$ for simplicity. By minimizing $E_{n,m}$ over θ_n, θ_m, we arrive at the normalization parameters θ_n, θ_m for networks n, m. To compute the global error function E, we can sum the errors $E_{n,m}$ over all pairs (n, m):

$$E = \sum_{n,m \in N,} E_{n,m} = \sum_{n,m \in N,} \sum_{h \in H_n \cap H_m} \left(q_{n,h} - q_{m,h}\right)^2. \tag{3.8}$$

This can be re-arranged to sum over hostnames last. If we do that, the descriptions of the summation sets would get quite hairy, so we will no longer print them and simply assume that a summand is present only if its value is defined. After re-arranging we get:

$$E = \sum_h \sum_n \sum_m \left(q_{n,h} - q_{m,h}\right)^2, \tag{3.9}$$

where according to the stated convention, n and m are summed only over the networks that observe the hostname h. It is now visible that the influence of a single hostname h is *quadratic* to the number of networks it is present in (there is a quadratic number of summands referencing this hostname). This is caused by using a pairwise error function instead of an error function comparing to a single baseline. This is certainly not desired; due to the nature of the dataset, we can expect the differences between the number of occurrences of different hostnames to be large. Were they quadratically exacerbated, the influence of less frequent hostnames would vanish. To alleviate this issue, we introduce a weighting parameter $w_h = |\{n | h \in H_n\}|$ (the number of networks that h is observed in) and use it to weight the error function:

$$E = \sum_h \sum_n \sum_m \frac{1}{w_h} \cdot \left(q_{n,h} - q_{m,h}\right)^2. \tag{3.10}$$

The resulting error function $E = E\left(g_{\theta_1}, g_{\theta_2}, \ldots, g_{\theta_{|N|}}\right)$ contains only sums and multiplications of g_{θ_n}, so it is differentiable with respect to g_{θ_n} and it is differentiable with respect to θ_n whenever g_{θ_n} is differentiable, so it can in principle be optimized by a first-order iterative optimization algorithm.

It should be noted that this error function may not be useful when the scale of the error function depends on parameters θ_n. For example, if we choose e.g. mean and scale parameters: $g_{\mu_n,\sigma_n}(x) = \frac{x-\mu_n}{\sigma_n}$, the optimal solution $E = 0$ would be for the degenerate case of $\sigma_n = 0,\ n \in N$. In cases like this, a more involved error function or a limited domain of parameters would be necessary to get useful solutions. For simplicity, we will use a shift normalization function $g_{\mu_n}(x) = x - \mu_n$ that does not suffer from these problems. This simple normalization function should eliminate the biggest calibration errors and improve the ensemble classification. We leave more involved normalization functions g_θ for future research.

After substituting for $q_{n,h}$ and using the shift normalization, we get the final normalization error function:

$$E(\boldsymbol{\mu}) = \sum_h \sum_n \sum_m \frac{1}{w_h} \left(f_n(h) - f_m(h) + \mu_m - \mu_n\right)^2 . \tag{3.11}$$

After differentiating with respect to μ_n, we get

$$\frac{\partial E}{\partial \mu_n} = \sum_h \sum_m \frac{4}{w_h} \left(f_m(h) - f_n(h) + \mu_n - \mu_m\right) . \tag{3.12}$$

This gradient function can be used to minimize E by any first-order optimization method such as gradient descend. In Appendix A, we prove that function E is convex, implying that convergence is possible and any minimum found will be global.

3.5.2 Time Complexity

In the worst case scenario where all the hostnames H are present in all the networks N, the gradient function ∇E has $|N|$ dimensions and $|H| \cdot |N|$ summands. Combined with the number of iterations I required to get to the minimum, the time complexity t of our algorithm is:

$$t \in O\left(|N|^2 \cdot |H| \cdot I\right) , \tag{3.13}$$

using the Big O notation. That is, complexity is linear to the number of hostnames and the number of iterations and quadratic to the number of networks in the ensemble. In practice, this complexity did not pose problems; for approx. 300 networks, 6.5 million unique hostnames and 50 iterations, optimization using gradient descent took approx. 2 minutes. In our case, the computation was not parallelized, so this solution is very scalable.

3.6 Reputation model usage scenario

In this section, we propose a way to use a global reputation model to improve detection in participating networks.

Firstly, enough data must be gathered to compute a meaningful global reputation model. If the time period is too long, the reputation model would suffer from modeling facts that may no longer be valid[2]. If the time period is too short, the estimate of a reputation model could be noisy. We propose and evaluate the use of three days worth of network data. After this time period elapses, a global reputation model is computed and distributed among networks. The networks can than use the reputation model to improve the subsequent network traffic anomaly detection. Simultaneously, they gather data to build a new global reputation model. After any set time period, the newly gathered data can be used to re-build the reputation model and the cycle continues. For simplicity, we evaluated the once-built reputation model for another three days in experiments. It would also be possible to refresh the system much more frequently to ensure maximum freshness of the reputation model.

The global reputation model in effect provides each network with a table of per-hostname anomaly scores. These anomaly scores can be used to improve anomaly detection in a network by performing a weighted average with the flow-specific anomaly scores. In our experiments, we strive to maximize the impact of the global reputation model on anomaly detection to achieve maximum contrast between the methods and to be able to better discern any differences between the two. Therefore, for every hostname for which we have a record in the reputation model, we replace its anomaly score with the one from the reputation model. In the case no record in the reputation model is present, we use the original anomaly score of the flow, normalized accordingly.

[2]Malicious actors could have for example moved their C&C servers to avoid blacklisting.

Chapter 4

Experimental evaluation

In this chapter, we evaluate the algorithms proposed in Chapter 3 on real-world network traffic captures. We start by describing the dataset and methods used for evaluation. Then, we use these to evaluate the proposed algorithms. Finally, we discuss the results.

We perform two experiments. The first one, **global reputation table evaluation**, is targeted at evaluating the classification performance of the global reputation models produced by the algorithms proposed in Chapter 3. The second experiment, **improvement of per-flow anomaly scores**, is aimed at evaluating possible improvements that can be attained by using the global reputation models to strengthen the anomaly scores in individual networks.

4.1 Dataset description

The dataset used for experimental evaluation consists of network communication of client networks that employ Cisco Cognitive Threat Analytics IDS. Data was gathered over the first week of January, 2017. There were in total 622 networks, but only 310 of these networks were chosen at random to produce a size-able dataset that simulates a realistic use-case while still being reasonably comfortable to work with (541 GB of raw data). Important characteristics of the dataset are shown in

Collected over	2017-01-03 ... 2017-01-10 (inclusive)
Number of networks	310
Total number of flows	12 932 245 944
Total number of malicious flows	394 951
Proportion of malicious flows	1 : 32 744
% of HTTPS connects in background flows	43.2 %
% of HTTPS connects in malicious flows	7.3 %

Table 4.1: Characteristics of the dataset used for experimental evaluation.

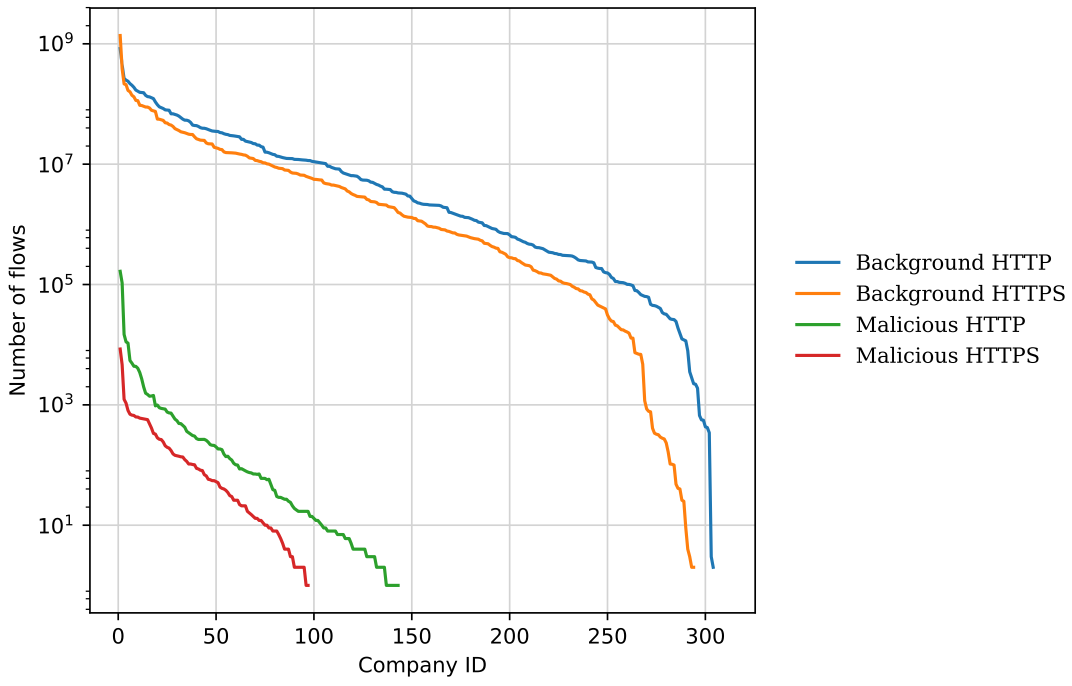

Figure 4.1: Flow counts per network. Networks are sorted independently for each line in order to make the lines monotonous. There are 310 companies in total, out of which 143 contain at least one malicious HTTP flow and 97 contain at least one malicious HTTPS flow. 6 networks contain zero HTTP flows, 16 contain zero HTTPS flows.

Table 4.1. The dataset entries were labeled by an expert at Cisco as being either legitimate or malicious. These labels are available both at the level of individual flows and at the level of individual hostnames. We will use these labels as ground truth in all of our experiments.

Companies vary wildly in the number of flows. To illustrate, 50 % of the biggest companies serve 99.0 % of the total number of flows. This is something that we need to pay attention to when we analyze the results. Detailed visualization of the company sizes is in Figure 4.1.

In order to test the deployment scenario described in Section 3.6, we must split our dataset chronologically into two parts. The first part is used to create and test a global reputation table while the second part is used to evaluate the possible improvements of per-flow anomaly scores using a reputation table. Coupled with the fact that there must be a warm-up period in which the anomaly detectors are learning the characteristics of network traffic and do not provide useful output, we arrive at the following split of the 7 days' worth of data:

- The first day is discarded as a warm-up period.
- The next 3 days are used to compute the global reputation tables.

- The last 3 days are used to evaluate the improvements of anomaly scores using the global reputation tables.

4.2 Evaluation criteria

We use Receiver Operating Characteristic (ROC) curves, and the Area Under ROC Curve (AUC) as our performance measurement methods. ROC curve is a graphical plot that illustrates classification characteristics of a classifier as its discrimination threshold is varied. It is created by plotting the true positive rate (TPR) against the false positive rate (FPR) for all possible values of discrimination threshold. This means that we do not have to choose any concrete threshold as the ROC curve captures all possible choices.

The AUC performance measure is simply the area under an ROC curve. It summarizes classifier performance in one number inside the $[0, 1]$ interval, where 1 is perfect classification and 0.5 is equivalent to the performance of random guessing. Another interpretation of AUC is that it is equal to the probability that a classifier would rank a randomly chosen positive sample higher than a randomly chosen negative sample. For convenience, AUC scores are often written as percentages.

4.3 Global reputation table evaluation

In this section, we evaluate and compare the classification performance of four global reputation tables produced by the four algorithms proposed in Chapter 3. All four of the algorithms use average as their combination function; their normalization functions and ways of dealing with missing values differ. The evaluated algorithms are listed below.

1. **average**: no normalization is performed;
2. **norm-isect**: $\frac{X-\mu}{\sigma}$ normalization on the intersection of hostnames (listwise deletion);
3. **norm-all**: $\frac{X-\mu}{\sigma}$ normalization on all observed hostnames (pairwise deletion);
4. **pairwise**: pairwise optimization of normalization error (proposed in Section 3.5).

From now on, we will refer to these algorithms by their identifiers that are typeset in bold above.

For the **norm-isect** algorithm, we need to have a meaningful intersection of all networks[1]. This is not the case when we use all of the networks in the dataset —

[1]In a sense that the set of hostnames observed from all networks is non-empty

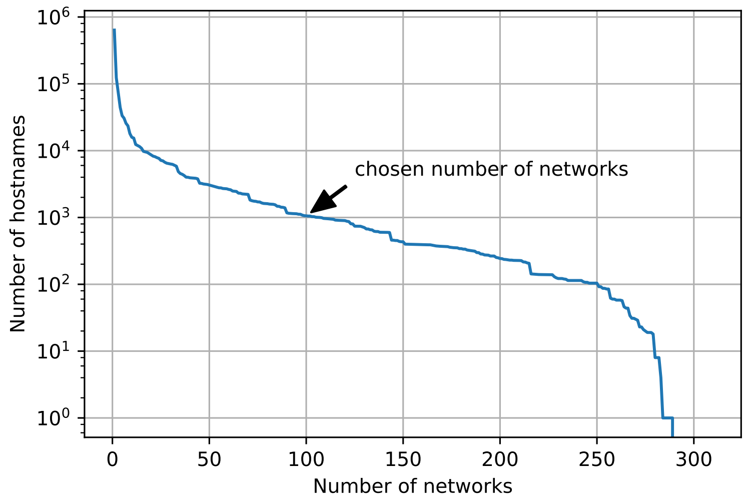

Figure 4.2: Network intersection sizes. Horizontal axis indicates the number of networks in an intersection (the networks with the largest number of hostnames are taken), vertical axis indicates the number of hostnames in the intersection on a logarithmic scale. We have chosen to use 100 networks, which correspond to the intersection size of 1054 hostnames.

Algorithm name	AUC score
average	86.6 %
pairwise	86.2 %
norm-all	80.8 %
norm-isect	76.8 %

Table 4.2: Global reputation table AUC score comparison on the biggest 100 networks. The algorithm **average** delivered the best performance and **pairwise** occupies a close second place. The other two algorithms performed significantly worse.

there are even networks that do not have *any* network flows in the training portion of the data-set, so the intersection is empty. Therefore, we need to use only a subset of networks for **norm-isect** to be applicable. A simple way to choose a reasonable set of networks is to choose the n networks with the largest numbers of unique hostnames. The sizes of intersections of the biggest n networks are visualized in Figure 4.2. Based on this figure, there is no immediately obvious number of companies that should be chosen. To strike a balance between calibration precision and utility, we quite arbitrarily choose to use 100 networks; their intersection contains 1054 unique hostnames. The same set of 100 networks is used for all four algorithms in order for their ROC curves to be comparable.

The resulting global reputation score ROC curves are shown in Figure 4.3 and the corresponding AUC values are listed in Table 4.2. The best-performing algorithms are **average** and **pairwise**, nearly tied in their AUC scores and ROC curve shapes.

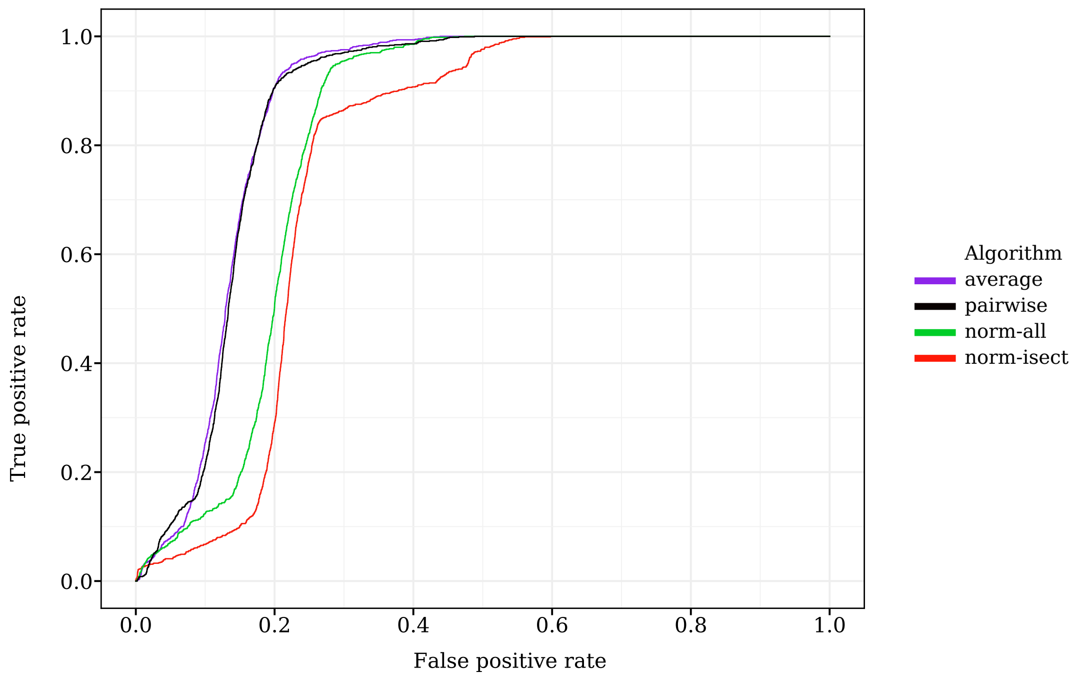

Figure 4.3: Global reputation table ROC curve comparison on the biggest 100 networks. The ROC curves were computed using the per-hostname anomaly values in the reputation tables.

Norm-all is in the third place and **norm-isect** performed the worst.

As expected, **norm-isect** didn't perform particularly well. It was able to exploit only a tiny fraction of available information to perform normalization: only 1054 hostnames were used compared to roughly 6.1 million hostnames utilized by **norm-all.** This probably caused substantial noise in the estimation of normalization parameters, degrading the ensemble performance. While **norm-all** exploits all of the available data, it suffers from a different problem: it standardizes every network output in isolation. The sets of observed hostnames are different for different networks, which then translates to different statistical characteristics of the individual network anomaly detection outputs. Even if networks were perfectly normalized (returning identical anomaly scores for each hostname), statistical differences between their outputs would exist and get erroneously corrected by the **norm-all** algorithm.

The **pairwise** algorithm utilized[2] 1.2 million hostnames (18% of total) and performed on par with the **average** algorithm. The **pairwise** algorithm could have improved upon **average** by performing normalization and thus removing *spurious* calibration differences. On the other hand, it could have decreased performance by removing *meaningful* calibration differences, or by making an inaccurate estimate of the normalization parameters due to noisy data. The positive and negative effects have canceled out to leave the performance of **pairwise** and **average** nearly

[2]All hostnames that were observed in at least two networks were utilized.

equivalent.

As we will see in the next section, there *are* benefits to using pairwise normalization over a simple average, but the positive effect manifests mainly for the worst-performing networks. This improvement likely got diluted in the mostly-unchanged performance of typical networks and did not manifest in the global reputation table AUC scores discussed above.

4.4 Improvement of per-flow anomaly scores

In this section, we explore the possibility of using a global reputation table to improve the anomaly scores in individual networks (the third arrow in Figure 1.2). We use the reputation model in the manner described in Section 3.6. There are two ways in which this procedure might improve classification.

1. **Smoothing.** Since many values are averaged together to arrive at the reputation score, much of the noise of the detectors will be filtered out.
2. **Information sharing.** Many different networks participate in the creation of the reputation table, which is beneficial as discussed in Section 1.4.

If smoothing is the main benefit of the reputation table, it may be possible to simply build a network-local reputation table and use that — no information sharing is necessary. Because of this, we must evaluate smoothing and information sharing separately to tell whether information sharing is beneficial. We evaluate three types of anomaly detection algorithms:

1. **plain**: the anomaly scores corresponding to individual flows; no reputation table is used.
2. **local**: anomaly scores are enhanced by a network-local reputation table. Since this reputation table is built using only one network, there are no normalization, combination or missing data issues and this reputation table is a simple per-hostname average of anomaly scores.
3. **global**: anomaly scores enhanced by a global reputation table. The global reputation table can be generated by either of the four different algorithms described in Section 4.3. Due to the poor performance of **norm-all** and **norm-isect**, we will not attempt to use them and will instead focus only on **average** and **pairwise.** Since we decided to discard the **norm-isect** algorithm, we can now use the full set of 310 networks. We recomputed the reputation models **global** and **pairwise** using all 310 networks and utilized these updated models in the following experiments.

The rationale behind evaluating **local** *versus* **global** is that most of the smoothing effect will be present in both of them; we can assess the strength of information sharing (mostly) alone by comparing their performance. If the **global** algorithms significantly improves classification strength over both **plain** and **local**, information sharing is beneficial.

In order to test the effect of information sharing on HTTP and HTTPS flows separately, we will test the performance of the algorithms on both in isolation as well as combined. We call the three resulting datasets **HTTP**, **HTTPS** and **All**.

This gives us four different classification algorithms over three distinct data-sets **per network**, for a total of 3720 distinct performance measures. This number of different results makes the ROC curves too granular; instead, we will use only the AUC scores to evaluate classification strengths. More than half of the networks do not contain even a single flow labeled as anomalous. For these, the AUC score is not meaningful and they are excluded from the comparison, leaving just 1432 performance measures.

We expect the classification performance of **plain** to be significantly worse for HTTPS traffic than for HTTP traffic because of lower information content discussed in Section 1.3. The reputation table should then improve on HTTPS traffic more than on HTTP traffic because of the sharing of information gathered by MiTM attacks in some networks (Section 1.4).

The performance of all tested algorithms is visualized as a box plot in Figure 4.4 and compared numerically in Table 4.3. It is visible from both the figure and the table that the largest differences between different algorithms are in the lowest percentiles. The 5^{th} percentile (displayed in the figure as lower whiskers) is a robust indicator of the behavior of networks with poor classification strength. In this percentile, both global reputation algorithms improve substantially upon the base case **plain** with **pairwise** offering the best performance. **Local** improves the 5^{th} percentile too, but not nearly as much as the global algorithms. Median does not differ very considerably or predictably between algorithms with the exception of HTTPS traffic where **local** and **average** give inferior results. Interestingly, the **pairwise** algorithm did not suffer from this effect nearly as much. Another interesting feature of the results is the distribution of outliers (displayed as circles in Figure 4.4). The **pairwise** algorithm eliminated *all* outliers with $AUC < 88\,\%$, but every other algorithm suffered from at least some networks with severely degraded performance. This indicates the high robustness of the **pairwise** reputation model.

Of all the tested algorithms, the algorithm **pairwise** exhibited the best AUC mean, minimum, 5th and 10th percentiles, and its standard deviation was smallest for all data sets. Other performance characteristics (including the global reputation table ROC curve) were never substantially worse than for other algorithms. These results suggest that **pairwise** is an effective algorithm for building a global reputation model.

Surprisingly, not much difference is immediately obvious between HTTP and HTTPS results despite the very different characteristics of their underlying data. We used the Wilcoxon signed-rank test [37] to test the significance of differences between all pairs of HTTP and HTTPS measurements: **plain**, **local**, **global** and **pairwise.**

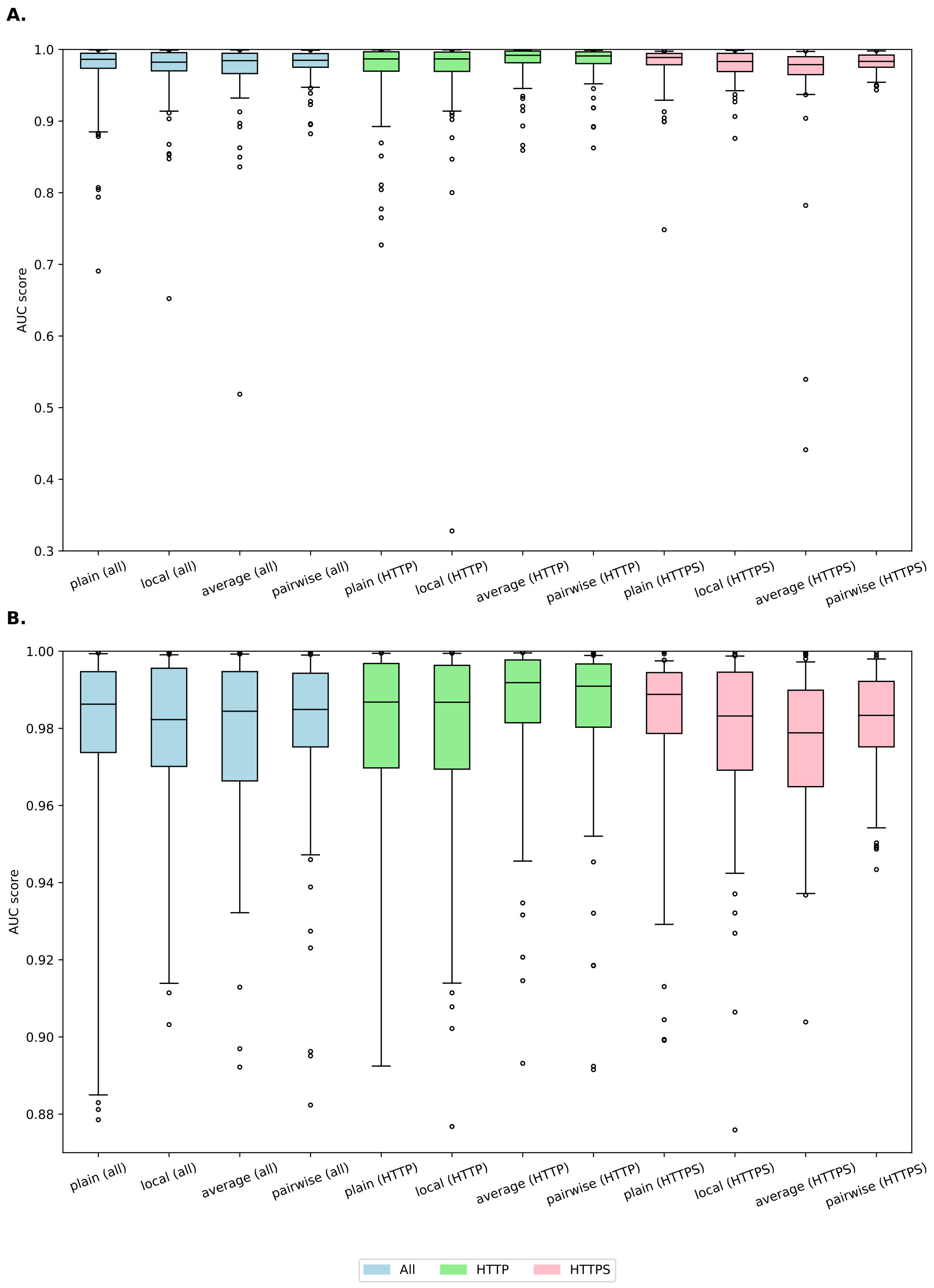

Figure 4.4: Comparison of AUCs of per-flow anomaly scores. Plot **A.** differs from plot **B.** only in the scale of the y-axis to highlight different parts of the plots. Boxes span between the 25th and 75th percentiles, whiskers span between the 5th and 95th percentiles, and circles represent all the networks outside the range of the whiskers. Median is marked by a horizontal line.

data	**All (HTTP + HTTPS)**			
algorithm	plain [%]	local [%]	average [%]	pairwise [%]
mean	97.3	97.4	97.3	**98.1**
standard deviation	4.4	4.0	4.8	2.0
minimum	69.1	65.2	51.9	**88.2**
5th percentile	88.5	91.3	92.8	**94.7**
10th percentile	94.0	94.3	94.5	**96.4**
25th percentile	97.4	97.0	96.6	**97.5**
median	**98.6**	98.2	98.4	98.5

data	**HTTP traffic**			
algorithm	plain [%]	local [%]	average [%]	pairwise [%]
mean	97.1	97.2	98.3	**98.4**
standard deviation	4.8	6.4	2.4	2.2
minimum	72.7	32.8	85.9	**86.3**
5th percentile	88.2	91.3	94.1	**94.9**
10th percentile	91.9	94.6	95.9	**96.3**
25th percentile	97.0	96.9	**98.1**	98.0
median	98.7	98.7	**99.2**	99.1

data	**HTTPS traffic**			
algorithm	plain [%]	local [%]	average [%]	pairwise [%]
mean	97.9	97.8	96.3	**98.1**
standard deviation	3.3	2.1	7.8	1.3
minimum	74.8	87.6	44.1	**94.3**
5th percentile	92.0	93.9	93.7	**95.2**
10th percentile	96.1	95.5	94.3	**96.2**
25th percentile	**97.9**	96.9	96.5	97.5
median	**98.9**	98.3	97.9	98.3

Table 4.3: Comparison of AUCs of per-flow anomaly scores. The AUCs are expressed as percents. The highest value in each row is bold with the exception of standard deviation. Note however, that the differences are very small in some rows (median) while being very large in others (minimum, 5th percentile). Higher percentiles are not displayed because the differences were mostly negligible for them. Maximum was in *all* cases equal to 100.0 %.

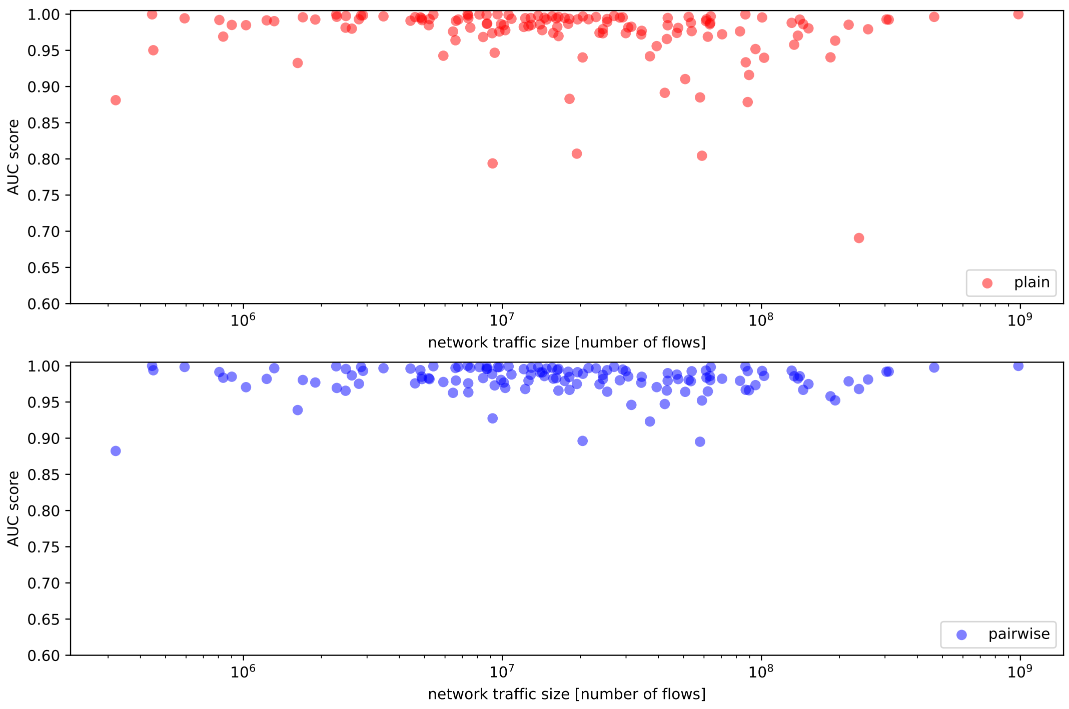

Figure 4.5: Network AUC score *vs.* network traffic size on all input data (HTTP and HTTPS). The top chart contains AUCs computed by the **plain** algorithm, the bottom chart contains AUCs computed by the **pairwise** algorithm. In neither of the cases, the AUC scores are not (obviously) linearly dependent on the amount of network traffic. Networks with both great and poor performance are found in all the parts of the size spectrum.

For neither of these were there any significant differences between their population mean ranks ($p < 0.01$).

Based on the considerations in Section 1.4, one would think that primarily small networks would exhibit low AUC scores: AUCs would be correlated with the amount of network traffic because small networks have less information to train their classifiers on. Surprisingly, this is not the case as illustrated in Figure 4.5. Similarly, AUC improvement by the reputation tables was not visibly correlated with network size. Even big networks can obviously exhibit bad classification characteristics and benefit from the improvement by global reputation tables.

Chapter 5

Conclusions

The main goal of this thesis was to improve the detection efficacy of the existing Cognitive Threat Analytics (CTA) anomaly-based network IDS (described in Section 1.3) on malware that uses network traffic encryption to avoid detection. Since the amount and quality of features that can be extracted from HTTPS network traffic is low, we proposed to utilize the fact that the CTA IDS is used in a large number of various networks and share threat information between them.

We created global threat intelligence and shared it among all the CTA IDS systems that are monitoring the individual enterprise networks to improve the anomaly detection efficacy on both encrypted and non-encrypted network traffic. Several variations of the method were developed, including one that incorporates a novel method for outlier ensemble normalization in presence of large numbers of missing values.

Experimental evaluation performed on a large amount of real-world network traffic showed that the proposed algorithm performs only marginally better than the state-of-the-art methods in average-case, but outperforms all of the the compared methods in the worst case scenario. Additionally to the efficacy improvements that were measured on the HTTPS communication, the proposed algorithm improved also the detection capability on HTTP network traffic.

Apart from improving detection accuracy in participating networks, the gathered intelligence can also be used on its own to help better understand global statistics of network threat behavior, to help network analysts assess threat severity, determine the scale of an attack or analyze command and control mobility patterns.

Bibliography

[1] Stephen Northcutt, Lenny Zeltser, Scott Winters, Karen Kent, and Ronald W Ritchey. *Inside Network Perimeter Security (Inside)*. Sams, 2005.

[2] K Scarfone and P Mell. NIST SP 800-94: Guide to Intrusion Detection and Prevention Systems (IDPS), Februar 2007.

[3] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An Automated Approach to the Detection of Evasive Web-based Malware.

[4] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou II, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *USENIX security symposium*, volume 12, 2012.

[5] Google. Transparency Report: HTTPS Usage. `https://www.google.com/transparencyreport/https/metrics/?hl=en`, 2017. Accessed: 2017-03-05.

[6] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[7] Martin Grill. *Combining Network Anomaly Detectors*. PhD thesis, Czech Technical University in Prague, 2016.

[8] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316. IEEE, 2010.

[9] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[10] Inc. Cisco Systems. Cisco Cognitive Threat Analytics Data Sheet. `https://www.cisco.com/c/en/us/products/collateral/security/cognitive-threat-analytics/datasheet-c78-736557.html`. Accessed: 2017-03-09.

[11] Atilla Özgür and Hamit Erdem. A review of kdd99 dataset usage in intrusion detection and machine learning between 2010 and 2015. *PeerJ PrePrints*, 4:e1954v1, 2016.

[12] Kevin S Killourhy and Roy A Maxion. Toward realistic and artifact-free insider-threat data. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 87–96. IEEE, 2007.

[13] Scott E Coull, Charles V Wright, Fabian Monrose, Michael P Collins, Michael K Reiter, et al. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *NDSS*, volume 7, pages 35–47, 2007.

[14] Information and Computer Science University of California, Irvine. KDD Cup 1999 Data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. Accessed: 2017-05-07.

[15] Information and Computer Science University of California, Irvine. KDD Cup 1999 Data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, 2007. Accessed: 2017-05-07.

[16] Inc. Cisco Systems. Cisco Cognitive Threat Analytics. `https://www.cisco.com/c/en/us/products/security/cognitive-threat-analytics/index.html`. Accessed: 2017-03-09.

[17] CIA Information Operations Center. Development Tradecraft DOs and DON'Ts. `https://wikileaks.org/ciav7p1/cms/page_14587109.html`. Accessed: 2017-03-09.

[18] Martin Grill, Tomáš Pevný, and Martin Rehak. Reducing false positives of network anomaly detection by local adaptive multivariate smoothing. *Journal of Computer and System Sciences*, 83(1):43–57, 2017.

[19] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.

[20] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[21] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[22] Thomas G. Dietterich. *Ensemble Methods in Machine Learning*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[23] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

[24] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, pages 231–238. MIT Press, 1995.

[25] Richard Maclin and David Opitz. An empirical evaluation of bagging and boosting. *AAAI/IAAI*, 1997:546–551, 1997.

[26] Charu C Aggarwal and Saket Sathe. *Outlier Ensembles: An Introduction.* Springer, 2017.

[27] Charu C Aggarwal. Outlier ensembles: position paper. *ACM SIGKDD Explorations Newsletter*, 14(2):49–58, 2013.

[28] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166. ACM, 2005.

[29] Jing Gao and Pang-Ning Tan. Converting output scores from outlier detection algorithms into probability estimates. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 212–221. IEEE, 2006.

[30] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. Interpreting and unifying outlier scores. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 13–24. SIAM, 2011.

[31] José Ramón Pasillas-Díaz and Sylvie Ratté. An unsupervised approach for combining scores of outlier detection techniques, based on similarity measures. *Electronic Notes in Theoretical Computer Science*, 329:61 – 77, 2016.

[32] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[33] Paul D Allison. *Missing data.* Sage Thousand Oaks, CA, 2012.

[34] Roderick JA Little. Regression with missing x's: a review. *Journal of the American Statistical Association*, 87(420):1227–1237, 1992.

[35] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[36] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[37] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[38] Olga Taussky. A recurring theorem on determinants. *The American Mathematical Monthly*, 56(10):672–676, 1949.

Appendix A

Convexity of the error function *E*

In this chapter, we prove that the error function E proposed in Section 3.5.1 is convex:

$$E\left(\boldsymbol{\mu}\right) = \sum_{h} \sum_{n} \sum_{m} \frac{1}{w_h} \left(f_n\left(h\right) - f_m\left(h\right) + \mu_m - \mu_n\right)^2, \tag{A.1}$$

where all the used symbols are defined in Section 3.5. Function's convexity is equivalent to positive-semidefiniteness of its Hessian matrix. We compute the Hessian matrix by differentiating E twice. We start from the first derivative (Equation 3.12):

$$\frac{\partial E}{\partial \mu_i} = \sum_{h} \sum_{m} \frac{4}{w_h} \left(f_m\left(h\right) - f_i\left(h\right) + \mu_i - \mu_m\right). \tag{A.2}$$

The second derivatives must be computed separately for two cases: differentiation by the same parameter twice or by two different parameters:

$$\frac{\partial^2 E}{\partial \mu_i^2} = \sum_{h} \sum_{m \neq i} \frac{4}{w_h} \cdot 1, \tag{A.3}$$

$$\frac{\partial^2 E}{\partial \mu_i \mu_j} = \sum_{h} \frac{4}{w_h} \cdot \left(-1\right), \quad i \neq j \tag{A.4}$$

This gives us a Hessian matrix of dimensions $N \times N$ (N is the number of networks in the ensemble):

$$H_{i,j} = \left[\frac{\partial^2 E}{\partial \mu_i \partial \mu_j}\right], \tag{A.5}$$

where Equation A.3 gives the diagonal elements and Equation A.4 gives the non-diagonal elements of $\boldsymbol{H}$. By Sylvester's criterion, $\boldsymbol{H}$ is positive-semidefinite if all its principal minors are non-negative. We show their non-negativity by first proving that all principal minors are *diagonally dominant* and their diagonal entries are non-negative, then proving that these properties imply their non-negativity.

A diagonally dominant matrix $\boldsymbol{a}$ is a matrix where the sum of moduli of its non-diagonal elements in every row is smaller than the modulus of the diagonal element in the same row:

$$|a_{i,i}| \geq \sum_{j \neq i} |a_{i,j}| \quad \text{for all rows } i. \tag{A.6}$$

A *strictly diagonally dominant* matrix is analogous, but a strict inequality (>) is required. By substituting A.3 and A.4 into A.6, we get

$$\sum_{h} \sum_{m \neq i} \frac{4}{w_h} \cdot 1 \geq \sum_{j \neq i} \sum_{h} \left| \frac{-4}{w_h} \right|, \tag{A.7}$$

$$\sum_{h} \sum_{m \neq i} \frac{4}{w_h} \geq \sum_{h} \sum_{j \neq i} \frac{4}{w_h}, \tag{A.8}$$

which holds and $\boldsymbol{H}$ is diagonally dominant. It is easy to see that the same is true for all principal minors of $\boldsymbol{H}$: the largest principal minor is $\boldsymbol{H}$ and for smaller principal minors, there are less summands on the right side of A.8, which only loosens the requirement. The diagonal elements of each principal minor are non-negative since they are a subset of diagonal elements of $\boldsymbol{H}$ which are themselves non-negative (equation A.3).

The last step is to show that diagonal dominance and non-negativity of diagonal elements together imply a non-negative determinant. Let $\boldsymbol{H'}$ be a principal minor of $\boldsymbol{H}$ and $\boldsymbol{D}$ be a diagonal matrix containing the diagonal entries of $\boldsymbol{H'}$. Let $\boldsymbol{M}(t)$ be a smooth transition between $\boldsymbol{D} + \boldsymbol{I}$ and $\boldsymbol{H'}$ ($\boldsymbol{I}$ is the identity matrix):

$$\boldsymbol{M}(t) = (t-1)(\boldsymbol{D} + \boldsymbol{I}) + t \cdot \boldsymbol{H'}, \quad t \in [0,1]. \tag{A.9}$$

$\boldsymbol{M}$ is strictly diagonally dominant, except maybe for the case where $t = 1$ ($\boldsymbol{M}(1) = \boldsymbol{H'}$). According to the Lévy-Desplanques theorem[1], a strictly diagonally dominant matrix has a non-zero determinant. Coupled with the positive determinant of $\boldsymbol{M}(0) = \boldsymbol{D} + \boldsymbol{I}$ (implied by the non-negativity of diagonal elements of $\boldsymbol{H'}$) and the intermediate value theorem, this shows that

$$\det \boldsymbol{H'} \geq 0, \tag{A.10}$$

which completes our proof of the convexity of the error function E.

Note that for the full matrix $\boldsymbol{H}$, both sides of the equation A.8 are equal. Moreover, a row sum $\sum_j H_{i,j}$ is equal to zero for every row i, implying $\det \boldsymbol{H} = 0$. Although

[1]This is a recurring theorem that has been independently proved a number of times. A very simple proof along with a brief history of the theorem is given in [38]. Lévy and Desplanques may have written some of the first articles about the theorem in 1881 and 1887.

$\boldsymbol{H}$ is positive-semidefinite, it is not positive-definite, and the minimum of E is not unique. This is not surprising, since for every minimum $E\left([\mu_1, \ldots, \mu_N]\right)$ there exist other minimums $E\left([\mu_1 + c, \ldots, \mu_N + c]\right),\ c \in \mathbb{R}$. It is not a problem in our case as we are not interested in the absolute value of E, but if it was, we could introduce a regularization term to the error function:

$$E' = E + \epsilon \sum_i \mu_i^2,$$

with some $\epsilon > 0$ close to zero. It is not difficult to show, analogously to the proof above, that E' is convex and its global minimum is unique.

Appendix B

CD Contents

All paths are expressed relative to the root. Due to the size of the datasets and due to privacy concerns, the data that was used in this thesis is not present on the CD.

`/thesis.pdf`	The final thesis in PDF format
`/report`	The sources of the thesis
`/report/thesis.lyx`	The main thesis project file written in LyX[1]
`/netflow`	A Rust[2] command-line program doing much of the computations needed for the thesis (ROC and AUC computations, data pre-processing, etc.)
`/npy-rs`	A Rust library for type-safe binary data interchange using `*.npy` files[3].
`/python`	A collection of data analysis and data visualization scripts using NumPy, Pandas, Matplotlib, Tensorflow and other packages.
`/python/viz`	Scripts used to generate many of the graphs in the thesis.
`/jupyter/`	Jupyter notebooks[4] used to generate many of the results (normalization weights, global reputation table evaluation, etc.) and several graphs.
`/remote-control`	Data traffic preliminary analysis using WireShark-collected data.
`/R`	A collection of data analysis and data visualization scripts in the R language. These are from the time before I switched to Python-based workflow.

[1] http://www.lyx.org/

[2] Rust is a systems programming language that prevents segfaults and guarantees thread safety without the need for a garbage collector. https://www.rust-lang.org/

[3] A simple format for NumPy Arrays. https://docs.scipy.org/doc/numpy/neps/npy-format.html

[4] The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. https://jupyter.org/

`/matlab` A collection of data analysis and data visualization scripts in the Matlab language. These are from the time before I switched to R-based workflow.

`/java/NpyExporter.java` A Java class for introspection-based binary export to the `*.npy` file format.