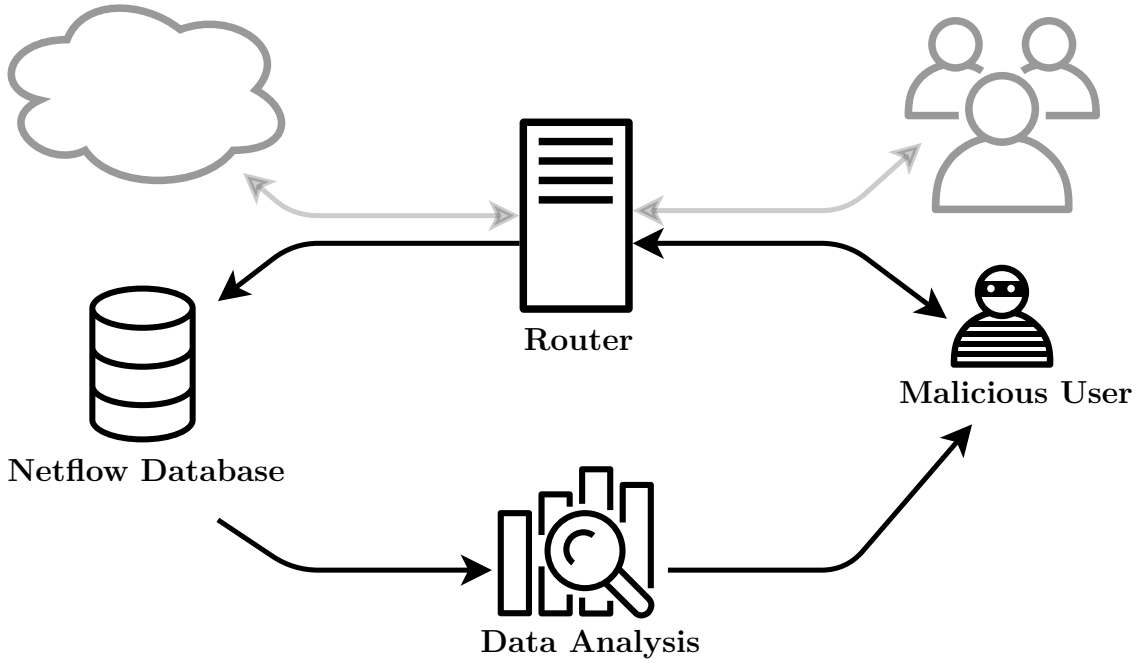
CHALMERS
UNIVERSITY OF TECHNOLOGY



Finding a Needle in a Stack of Logs

A survey of network anomaly detection techniques and a proof of concept for an unsupervised model applicable to large datasets

Master's thesis in Complex Adaptive Systems

JERRY LIU, MARTIN ERIKSSON

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Finding a Needle in a Stack of Logs

A survey of network anomaly detection techniques and a proof of concept for an unsupervised model applicable to large datasets

Jerry Liu, Martin Eriksson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2019

Finding a Needle in a Stack of Logs
A survey of network anomaly detection techniques and a proof of concept for an unsupervised model applicable to large datasets
Jerry Liu, Martin Eriksson

Supervisor: Staffan Truvé, Recorded Future
Co-supervisor: Sanil Cohan, Recorded Future
Examiner: Rebecka Jörnsten, Mathematical Sciences

Master's Thesis 2019
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Schematic view of the process for finding a malicious user using Netflow logs

Typeset in LaTeX
Gothenburg, Sweden 2019

Finding a Needle in a Stack of Logs
A survey of network anomaly detection techniques and a proof of concept for an unsupervised model applicable to large data sets
JERRY LIU, MARTIN ERIKSSON
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Anomaly detection in networks can provide invaluable information to the network administrator or forensic information for network security analysts. We review the viability of a large set of methods for Netflow anomaly detection using knowledge from statistics, information- and graph-theory. We asses how well these methods fit into our objective of running an unsupervised method in real time to extract anomalies in a network producing over 100 million flows each day. A sketch based method that fits these requirements is selected for further improvements. The method is unsupervised, and relies on pseudo-random projections of the feature-space into multiple significantly smaller spaces. The distribution in each space is continuously monitored and any activity that significantly changes these distributions will trigger a detection. We show how this method can be run in real-time with a low false-positive rate and high detection rate.

Keywords: Netflow, network, anomaly detection, entropy, Kullback-Leibler divergence, sketch algorithm, histogram detection.

Acknowledgements

We would like to extend our appreciation to Sanil Cohan, Staffan Truvé and the wonderful people at Recorded Future for supporting us with their insights throughout the writing of this Master's thesis. We'd also like to extend our deepest gratitude to Rebecka Jörnsten for going beyond her duties as an examiner in helping us throughout this project.
Further, we are grateful for the opportunity of working on our project at the Recorded Future office. The company has welcomed us with open arms, and invited us to their social events. Working in this environment at an exciting and growing company has been an amazing experience.
We would also like extend a thanks to our friends, our families and our respective partners. Your support through these past months has been invaluable.

Jerry Liu, Martin Eriksson, Gothenburg, December 2019

Contents

List of Figures

List of Tables

1
Introduction

- Background
- Aim
- Limitations
- Related Works

The internet is an integral part of modern everyday life, both for people and organizations. Following the increased reliance on and usage of internet services are also the attempts to exploit this online activity. Malicious activity comes in many forms, with widely different goals; influencing operations, denial of service, information extraction, etc. [2]. Preventing this activity is far from trivial, it's a cat-and-mouse game of exploitation versus security. Which is why knowledge of this activity, and the patterns of exploitations, is crucial.
The internet also provides opportunities to learn about current threats. Recorded Future specializes in threat intelligence, scraping the internet of data to catalog and discover both physical and online threats. This thesis aims to explore a previously unused data source from which to gain knowledge of possible malicious actors. With the use of network traffic logs - Netflow[1] - we implement a method to discover threats present in a network.

1.1 Background

Many of the different types of malicious activity have been categorized and tracked by different organizations within cyber security. One such example is MITRE [3], which has a fairly comprehensive list of different exploitations and attacks. Developing methods to detect and identify these attacks is an always ongoing battle against time and the inventors of the next attack. The use of Netflow logs to detect attacks against networks has been widely studied, as shown by Marnerides et al. in a survey of some of the most successful techniques [4]. Netflow is a widely used standard, and many network routers have options to log Netflow out of the box. Relative to other logging options, such as packet inspection[2], its storage requirements are relatively low. Even so, the logs often exceed millions of entries per day for a moderately sized network. Another advantage of Netflow over packet inspection is its nonintrusive nature. For a non-encrypted connection, packet inspection could potentially reveal private information of the network user, where Netflow will not.
A common working hypothesis is that the detection of suspicious or malicious activity can be restated as an anomaly-detection problem. In Netflow, a single flow is

[1] Netflow: explained further in section 2.1.5
[2] Packet Inspection: analysis and handling of individual packet payload contents.

very rarely indicative of an anomaly, unless one of the involved addresses is a known malicious address. The aggregation and larger scale activity of one, or a group of, actors may however let one draw conclusions about whether their activity constitutes normal behaviour or not. Finding the tiny fraction of relevant information which paints the picture of malicious activity requires advanced analysis methods. To date, a large number of methods from different fields have been proposed, ranging from Machine Learning to Digital Signal Processing.

1.2 Aim and Limitations

This thesis aims to implement and test a method for running anomaly detection on Netflow data in real time. The goal is to find a method which would find potential threats or suspicious activity, while ignoring normal high-volume traffic. The method should be unsupervised[3], able to handle large quantities of Netflow each day and require as little manual analysis as possible. Starting with a wide search of previous endeavours described in section 1.3, several different methods are implemented and tested against the available data. From these, one method is chosen to study in depth, to analyze its capabilities.

1.3 Related Work

Anomaly detection is a wide field with a multitude of different applications. In a generic sense, it constitutes detecting a change in some monitored metric significant enough to be of interest. Time-series monitoring and outlier detection are two common methods of working with the problem. Applications of anomaly detection in networks is a widely studied field, and the methods range from many disciplines such as statistics, digital signal processing, information theory, graph theory and machine learning, with many using a combination of these fields to achieve the best result. Another comprehensive survey has been done by Marnerides et al. [4], where results of the more influential works are discussed in detail.
A notable project in the general field of Netflow analysis is the BLINC project developed by Karagiannis et al. [5]. It does not explicitly deal with anomaly detection, but is rather an incredibly ambitious attempt at classifying network traffic, using only Netflow. The model uses uses graphlet[4] matching and rule-based classification to match observed Netflow traffic to a set of predefined usage categories. Packet inspection is used to create a ground truth model, to which the BLINC classifications can be compared. It is shown that classification of flows into the usage categories yields a high accuracy. Additionally, the model include the anatomy of known attacks, and show how these can be detected. The authors describe the model as extendible, but also point out that one has to carefully consider additions, to not disturb the sequential structure of the classification model.
Looking at network flow from statistical point of view, some authors propose the usage of a Netflow trace from an earlier time period to build a baseline. Muraleed-

[3]Unsupervised: does not require labelled data to tune the method.
[4]Graphlet: a small connected subgraph of a larger network.

haran et al. [6] propose using the Chi-squared distance test to compare future flows to the reference period. Another example is Yaacob et al. [7] where the authors employed Auto-regressive Integrated Moving Average (ARIMA), using its predictive power to detect anomalies when the predictive error was too large. This kind of approach assumes previous knowledge, and a consistent behavior over time which is not guaranteed for a dynamically changing network where the malicious actors could assume different identities.

More behaviour dependant detection systems have been applied in the form of Hidden Markov Models (HMM). Ourston et al, Kumar et al. and Chen et al. [8, 9, 10] all have proposed the usage of this method. HMM assume hidden states within a network device and gives a probability of being in this state depending on the previous state i.e. a probability for the observed behaviour. This makes it particularly useful in cases where attacks are carried out with different detectable phases. The drawbacks of this approach are highlighted in an article by Ye et al. [11] where they show its lack of robustness against noise and performance degradation as noise increases. Modelling hidden states of many devices is a computationally heavy task, which means HMM is not a good fit for larger sets of data.

A commonly studied field in computer science are graphs which network data naturally represents. Franccois et.al. [12] proposed using Pagerank on Netflow with additional clustering using DBSCAN[5] to detect P2P[6] botnets. In actuality what they describe in their article is the Hyperlink-Induced Topic Search (HITS) algorithm which differs from Pagerank. A thorough comparison between the two algorithms is done by Devi et al.[13]. The only two features: hubs and authority score, are used for clustering. A honeypot[7] is used to generate labels for known P2P botnet nodes prior to clustering, since similarly scored nodes don't necessarily have to be P2P bots. The proposed method is limited to detecting P2P botnets, relying heavily on their interconnection. Additionally, the use of a honeypot further limits the detection to only nodes related with the honeypot, which in itself isn't trivial to set up. Another approach by Chowdhury et al. [1] used intrinsic graph attributes as features for clustering with self-organizing maps. These features are inbetweeness, local clustering, eigenvector centrality, weighted and normal in-/out-degree. A similar issue as with HMM occurs with many of these graph-based algorithms where computational costs increases non-linearly as the dataset grows.

Another field in computer science, information theory, has inspired many methods. On of the most common tools used is the information-theoretic formulation of entropy as proposed in 1948 by C.E. Shannon [14]. Detection of anomalies by monitoring the entropy of different flow features is a method commonly found in research, one such example is the entropy-based detection method described by Berenzinski et al. [15], where several different formulations of entropy were tested to detect artificially injected attacks in normal traffic data. Another entropy based method was published by Lakhina et al. [16], where peaks of the empirical sample entropy

[5]DBSCAN: a clustering algorithm, explained further in A.4

[6]Peer-to-Peer (P2P): a distributed network that shares the task between its peers. In botnets this architecture is used to relay information/commands between peers.

[7]Honeypot: used in computer science as a baiting mechanism that acts as a legitimate user but is actually isolated, controlled and monitored.

within a time window is used to flag time windows as containing an anomaly.
Closely related to Entropy is the Kullback-Liebler divergence [17], which can be seen as the similarity between two different distributions. This was for example used by Brauckhoff et al. [18] to determine when the volume distribution of a Netflow feature experiences a significant change.
Another common perspective to general anomaly detection is time-series monitoring and analysis using techniques from Digital Signal Processing (DSP). Kalman filters is one such method which has been used by Soule et al. [19] to generate a predictive model that's compared with observed data for detection and extraction of anomalies. Barford et al, Huang et al, Lu et al [20, 21, 22] all have implemented similar systems, but instead used wavelets to isolate signals into their time-frequency domain. This allows studying each component with a resolution matching its scale. These methods also rely on building a reference profile.
Artificial Neural Networks (ANN) have, unsurprisingly, also found their way into network anomaly detection. Sheikan et al. [23] propose an alternative training method for ANN in network anomaly detection using the gravitational search algorithm (GSA) and compares it against particle swarm optimization (PSO) and backpropagation. While not directly applied to Network Anomaly Detection, Li et al. [24] have proposed using Generative Adversarial Networks (GAN) to find anomalies in multivariate time series. Since Netflow contains mostly discrete variables, GAN may be difficult to apply directly. However, applying this in a similar fashion to DSP models to network anomaly detection could be a feasible endeavour.
A Netflow data point exists in a massive multidimensional space consisting mostly of discrete variables. In addition to this, both port number and IP addresses have limited to no local similarity. Several groups, such as Brauckhoff et al, Dewaele et al. and Casas et al. [18, 25, 26] have applied a scheme where hash functions are used as random projections into multiple smaller feature spaces. After analyzing, the results can be recombined to figure out the exact participants in the anomaly. The idea behind this method is described at greater length in chapter 3.
Lakhina et al. [27, 16, 28] made several important contributions to the field. In their 2005 paper [16] they use entropy to detect the temporal location of anomalies. In the time window of a detected anomaly they apply the Multiway Subspace method described in another article by the same authors [27]. This method uses Principal Component Analysis (PCA) to approximate a normal state-space such that anomalies can be identified by their residual-space component. Finally, they use this residual space to cluster anomalies into categories.
For this thesis, the method used is based on an article by Brauckhoff et al. [18]. The method fits well within the scope of our project and it is capable of extracting the IP addresses involved in an anomaly.

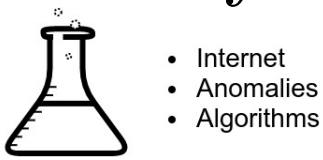
1.4 Contributions

We present an analysis of the methods implemented by Brauckhoff et al. [18] with minor modifications. After implementing the model we evaluate how well it performs on our network data. Additionally, we discuss the strengths of the method, as well

as its shortcomings. Finally, we discuss the problem in general and explore some future ideas for work that could be done in Netflow Anomaly Detection.

2

Theory



2.1 Internet

With its ever increasing usage, the internet has become critical infrastructure in today's society. The International Telecommunications Union (ITU) estimates that 53.6 percent of the global population or about 90 percent in first world countries use the internet [29]. Tasked with the open-source development of this gigantic framework the Internet Engineering Task Force (IETF) lies behind most of the subjects discussed in this section.

2.1.1 Internet Protocol Address

Internet Protocol address (IP address) is a numerical identification label for each device connected to a network which contains information regarding its network host and location. This information can be used to leverage the knowledge about a communication in the network.
There are two address specifications used: IPv4 and IPv6. Currently IPv4 is the most commonly used standard, defining each address as a 32-bit unsigned integer [30], a notation decomposition is shown in figure 2.1. This brings the total available address space to 4.3 billion, less than earth's human population. Due to the growth of the internet, IPv4 exhaustion has been a growing concern. This was however already predicted in late 90s which gave birth to the IPv6 standard with 128-bit

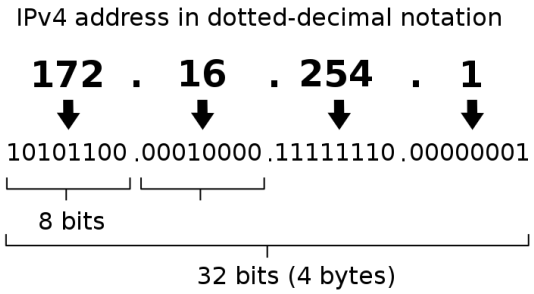


Figure 2.1: IPv4 address decomposition from human-readable dotted-decimal notation to its binary value.

addresses [31], yet adoption has only taken speed in recent years. To understand the size of the IPv6 address space one could imagine assigning a unique address to every single cell in every person on earth, which would require about one billionth of the possible IPv6 space [32].

2.1.2 Ports

A port in network communication is a 16-bit unsigned number (0 to 65,535) that's always coupled with an IP address and transfer protocol. It functions as an identifier to a specific process or network service on a computer. Some specific ports are often reserved for common services: SSH, VPN, HTTP etc. which allows an arriving packet to be easily forwarded to the correct application. Historically ports below 1024 identify the most commonly used services [33].
A new outgoing communication will often be assigned a set of ports referred to as ephemeral ports. Since they initiate connection, they do not need to have a common application port but can wait for a response to the same port. Employing this fact one is able to often distinguish the client and service of a network communication. Since only the service port has any significance by being application bound, while the client port can be any unassigned port. This can be used to for example determine which IP is a user browsing Twitter.

2.1.3 Network Transport Protocol

Communication protocols are the systems that govern communications in the network. The basic requirements of each communication such as data formats, address mapping, acknowledgements, among many, can be discussed in great depth but is out of scope for our purpose. It suffices to know the three most observed protocols in our datasets and their general use [30].

TCP (Transmission Control Protocol)	Web traffic
UDP (User Datagram Protocol)	Large data transfer
ICMP (Internet Control Message Protocol)	Error-response

2.1.4 Packets

To transfer information across a network the information is packaged into packets for shipment. These packets have varying size depending on the transport protocol. The content within each packet can be analyzed by deep packet inspection revealing information about its path and payload, however this isn't captured by Netflow. The only content information being recorded by Netflow is the byte-size [30].
Direct inspection of the packet size doesn't relay much information. Depending on the transfer protocol there are certain requirement of information for their headers[1] e.g. a TCP header could be between 20 to 60 bytes while headers for UDP has only 8 bytes per packet. Using this knowledge, rough guesses can be done on the content size within each packet or wether or not there was any content at all in the packet.

[1]Header: the package label of each packet, with information such as source/destination port.

2.1.5 Netflow

NetFlow is a logging standard designed by Cisco used to collect and monitor network communication data through routers [34]. There are some variations between versions of this standard, but all share a common 5-tuple of timestamp, source IP, destination IP, source port, destination port. The standard used in this thesis has some additional fields: number of bytes, number of packets and protocol as seen in table 2.1. The NetFlow used is also unidirectional meaning two flow records are generated for one connection as shown in the table. This is due to the nature of routing where one interface handles inbound/ingress traffic and outbound is on a separate interface. These flow pairs only occur for data over the TCP protocol, UDP and ICMP do not share the same structure.
If the interface allows, Netflow can also be collected in a bidirectional standard defined by the most recently adapted RFC[2] by Trammell et al. [35]. As discussed by Garcıa [36], using bidirectional flows instead of unidirectional can often be a better choice for analysis. While you give up some information granularity in the number of packets and bytes sent in each direction you gain the information of which address is the server and which is the client, or more generally: which device initiated the connection. There are ways to build an approximation of bidirectional flows from a unidirectional log. A method described by Sommer and Feldmann [37] is to iteratively build the bidirectional representation.

Table 2.1: Example of two unidirectional flow records used in this thesis. The first eight bits of the addresses are left out.

Timestamp	Source IP	Src Port	Destination IP	Dst Port	Bytes	Packets	Protocol
11:00:29,53	XXX.32.84.118	6881	YYY.162.236.67	80	126	2	TCP
11:00:30,13	YYY.162.236.67	80	XXX.32.84.118	6881	355	6	TCP

2.2 Network Anomalies

It is important to not confuse network anomalies with malicious behaviour. An anomaly is not inherently malicious, however it is often a reasonable assumption that malicious behaviour implies anomalous records within some metric. Network anomalies can be divided into four different categories used by Marnerides et al. [4]

- *Malicious Attacks:* attempts to disturb normal service by external means such as Denial of Service attacks (DoS).
- *Abnormal network usage:* legitimate but irregular traffic deviating from normal, such as flash crowds.
- *Measurement anomalies:* data collection issues, for example hardware failure of collection router.
- *Misconfiguration and failures:* software issues due to bugs or human error leading to improper service configuration.

[2]Request for Comments (RFC) explained in more detail in appendix A.1

2.2.1 Malicious Anomalies

The last two points may be of interest to network operators. However for this thesis the goal is to be able to detect bad actors participating in a network. Some common malicious anomalies we aim to detect are listed here.

Denial of Service

Denial of Service describes a targeted effort to prevent people from using a specific service. In practice this often means overloading a service such as a web server with more requests than it can handle, such that real users cannot reach it or have large waiting times. Many different methods are employed, but it is often done using botnets [38].

Scanning

Port scanning is the practice of probing the ports of a host for open ports or running services [38]. While often used by attackers to gain knowledge about the target it is not an inherently malicious practice and can also be used by system administrators to detect vulnerabilities in their own system. Port scanning is easily identified by many low-payload connections to one host on a large set of the host's ports.

Botnets

A common form of malicious activity detected through inspection of network traffic are botnets. These are devices that have been compromised with malware and show different kind of behaviours depending on the infection state [3]. Every strain of malware is different, but as a rule of thumb the behaviour of a botnet agent can be roughly summarized in three different states.

- *Dormant:* the agent sleeps, waiting for a pre-programmed time or commands from a C&C node for activation. [39]
- *Passive:* similar to dormant state the agent awaits commands by pinging known services or it's C&C node. Although low activity, still detectable.
- *Activated:* carrying out the commands instructed by C&C node e.g. port scan or DDoS. High activity, very noticeable traffic.

These describe some of the most commonly observed behaviours and in no way represent the whole picture. Of these three states the last one represents most of the observable traffic. Although it is possible to detect passive botnets through looking at longer time spans it is often very difficult, as the number of flows is very small.

2.3 Entropy

Entropy has proven a useful metric when working with network traffic analysis. The discrete entropy of some probability mass function $P(X)$ with possible values $x_1, ...x_n \in X$ is defined by:

$$H(X) \equiv -\sum_{i}^{n} P(x_i) \log P(x_i). \tag{2.1}$$

Within information theory it is commonly defined with the base 2 logarithm [14], but from a mathematical standpoint any common base is acceptable. This entropy conveys some information of the underlying distribution. For example in the limit $H(X) \to 0$, x_i can only take a single value. Maximal entropy is given by $\log n$ only when $P(x_i) = \frac{1}{n}$ for all possible i. This means entropy can be used as a measure of concentration or dispersion of a distribution.

2.3.1 KL divergence

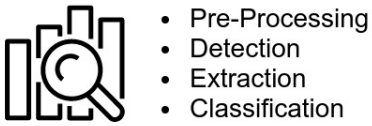
Another, more direct way to compare distributions is the Kullback-Leibler divergence (KL divergence) [17], defined as

$$D_{KL}(P||Q) \equiv -\sum_{x_i}^{n} P(x_i) \log \frac{Q(x_i)}{P(x_i)}, \tag{2.2}$$

where $P(X)$ and $Q(X)$ are two distributions with identical support $x_1, ...x_n \in X$. In practice, the definition is often extended to include the case when $P(x_i) = 0, Q(x_i) \neq 0$, which is 0 per the limit $\lim_{x+\to 0} x \log x = 0$. Similarly, the case $Q(x_i) = 0$ is undefined because of the non-identical support, and for the purposes of this thesis, this is also taken as 0. When $D_{KL} = 0$, the two distributions P, Q are the same, and with increasing values the distributions are less similar. The measure is not a symmetric metric: $D_{KL}(P||Q) \neq D_{KL}(Q||P)$, and has been proven to be useful in information theory [17].

3

Methods



This chapter describes the process from data input to anomaly extraction. Additionally, the experimental setup to evaluate method performance is also presented. A pseudocode representation of the detection and extraction steps is shown in algorithm 1.

3.1 Data

As noted by Malowidzki et al. [40], there are very few quality datasets publicly available of anomalies in Netflow records. Even fewer of those are labelled, making automatic classification a difficult undertaking. Beyond this, the usage of different applications and services within networks is ever changing, which could mean that knowledge from older traces could grow obsolete over time. Considering this, our project works with non-labelled data, and uses an unsupervised approach. The dataset used are described below.

3.1.1 Corporate dataset

This data source continuously collects Netflow logs sampled from a few outward-facing machines in a medium-sized corporate network. This data is saved for 30 days after collection. The sampling points produce around 100 million records per day of unidirectional Netflow. Unless stated otherwise, the experiments and results of this report were conducted using subsets of this dataset.

3.2 Pre-Processing

It is often beneficial to pre-process Netflow logs to highlight the desired information.

3.2.1 Filtering

Table 3.1 shows the distribution of protocols in the corporate dataset over a period of two months. Since the usage patterns differ between protocols, analyzing their respective flows may be interesting. However, methods such as Entropy or Sketch that rely on flow distribution would make changes in ICMP traffic near invisible

Table 3.1: Protocol occurrence in corporate dataset measured over two months.

Protocol	Count	Percentage
TCP	8 230 058 840	85.90%
UDP	1 342 980 141	14.02%
ICMP	6 322 066	0.066%
IPv6-ICMP	1 659 923	0.017%
Other	15 583	0.000%

in the scope of the entire set. This makes a case for separating the Netflow data by its protocols TCP, UDP and ICMP (both IPv4 and IPv6) and analyzing them separately.

3.2.2 Netflow directionality

As described in section 2.1.5 using bidirectional Netflow can often be beneficial. However, the proposed stitching method by Sommer and Feldmann [37] proved to be very time inefficient. Running both the stitching method and the detection method in real time is currently not possible within the scope of this project.
Instead, to gain some more granularity of the IP features, we use knowledge of where in the network the sampling points are located. We know that there are two sets of IP addresses, within the network and outside the network. As displayed in figure 3.1 this means that the sampled traffic can have 3 classes, incoming/outgoing and inside traffic. Since inside traffic constitutes only a small fraction of the captured flows, the vast majority of flows will have one internal and one external address. Annotating every flow with these two fields yields a more granular view than using Source/Destination IP, which are practically identical due to the nature of unidirectional Netflow. In the cases of inside traffic, we have chosen to annotate Internal with the source address, and External with the destination.

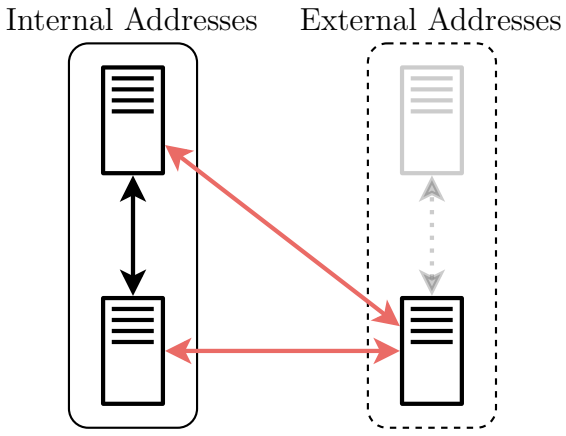


Figure 3.1: Network communication graph between internal and external addresses. The arrows denote the possible communications between these sets of addresses. The red arrows show incoming/outgoing traffic which crosses the network boundary.

3.3 Detection

The detection algorithm is the core of the implementation. The model we chose to implement and evaluate is a slightly modified method presented by Brauckhoff et al. [18]. The method is run on a stream of data, discretized into time windows of length T. Using the Sketch algorithm detailed below, a histogram of the flow distribution in each window is built. Using the KL divergence, as explained in section 2.3.1, we estimate how much the flow distribution changes between consecutive time windows. The various parameters used throughout this section are summarized in table 3.2.

Table 3.2: Description of model parameters and typical value ranges.

Symbol	Description	Typical values
k	Number of different hash functions	4-32
m	Number of bins per hash function	128-1024
T	Time window length (minutes)	5-30
τ	Threshold value for detection	$2 - 4\hat{\sigma}$
v	Voting threshold	1-k

3.3.1 Sketch algorithm

The space of possible combinations of IP addresses and port numbers are categorical and offers limited local similarity. It is also difficult to draw any conclusions from their raw representation. The sketch algorithm provides randomized projections which allows a detection algorithm to gain multiple views of the data, while also providing a way to extract the anomalous feature value. The original idea is inspired by the count-min sketch method proposed by Cormode et al. [41] and has been implemented in several works in network anomaly detection by Brauckhoff et al., Deweale et al. and Casas et al. [18, 25, 26].
The sketch algorithm uses a set of k different hash functions[1] $h^k(\cdot)$ which all map any input to some set of discrete values $h^k(\cdot) \rightarrow \{1, .., m\}$. All k functions provide a unique pseudo-random approximately uniform mapping from the feature space to the desired range of values. Figure 3.2 demonstrates how the original data is copied into multiple views, one for each hash function. The histograms are constructed by the number of flows mapped into every bin. Other examples of aggregations include total number of packets or total data transferred. In total, the algorithm produces k histograms with m bins which are used in the histogram detection.

3.3.2 Histogram detection

Many anomalies are detectable by a change in the distribution of flows. We achieve this by comparing histograms from adjacent timesteps, similar to the method previously proposed by Stoecklin et al. and Kind et al. [42, 43]. The same method is employed by Brauckhoff et al. [18].

[1]Hash function: a map from any input to a discrete set of outputs. Further explained in appendix A.3

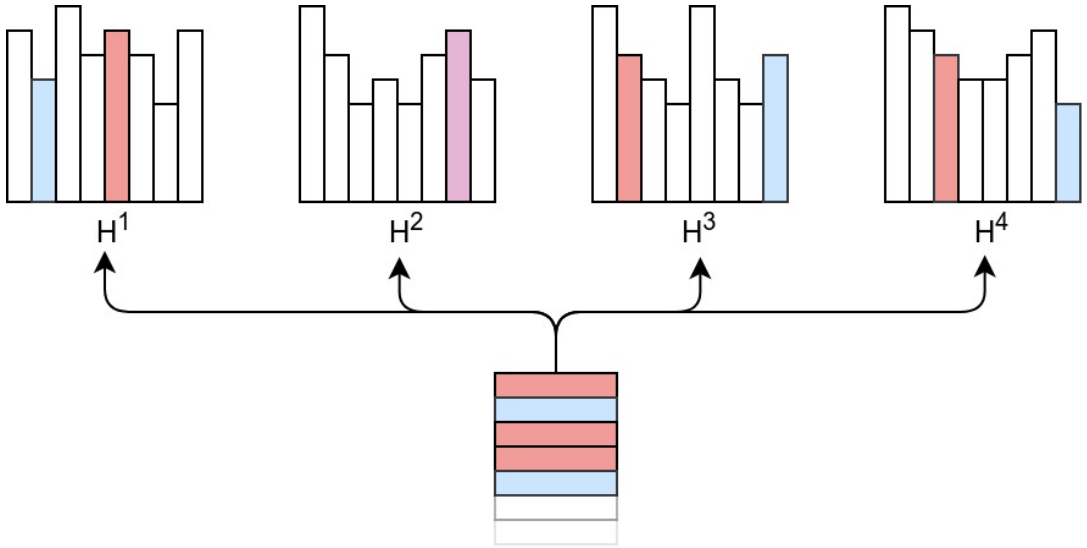


Figure 3.2: Sketch hashing process for a set of data points with 4 hash functions and 8 bins per hash. The bin height reflects the number of existing items in each bin. Each data point is hashed and put into the resulting bin. An example of hash collision occurs in the second bin which maps both the blue and red values into the same bin.

For every time window the sketch algorithm gives us a set of histograms $H^i(t), i \in [1..k]$, each with a set of bins $B^i_j(t), i \in [1..k], j \in [1..m]$. For any two adjacent time steps, $(t-1), t$, we compute the KL divergence between those histograms

$$\begin{gathered} D_{KL}(H^i(t-1)||H^i(t)) \\ i \in [1..k] \end{gathered} \tag{3.1}$$

By repeating this procedure in every subsequent time step, we can produce a time series of the divergence of each hash function. Brauckhoff et al. observed that the first difference of this time series is approximately a Gaussian distribution with 0 mean and $\hat{\sigma}$ standard deviation. They use an empirically calculated value $\tau = 3\hat{\sigma}$ as a detection threshold [18].
The detection rule is a comparison of the first difference of the KL divergence time series to a threshold value

$$\begin{gathered} \left\| D_{KL}(H^i(t-1)||H^i(t)) - D_{KL}(H^i(t-2)||H^i(t-1)) \right\| > \tau \\ i \in [1..k]. \end{gathered} \tag{3.2}$$

This results in a set of detections $I \subseteq [1..k]$. We are interested in which bin contributed to every detection. We also note that Brauckhoff et al. [18] use an identical detection as above, but only for the positive case, not the absolute case. For the detected histogram indices $\hat{i} \in I$ at time t, the bin to flag as anomalous is chosen as the bin which has the largest absolute difference between time windows:

$$\underset{j}{\operatorname{argmax}} \left\| B^{\hat{i}}_j(t-1) - B^{\hat{i}}_j(t) \right\|. \tag{3.3}$$

Every detection is represented by a unique tuple $(\hat{i}, \hat{j})$. A illustration of this is shown in figure 3.3, small changes are noticed between the time windows but only a single bin changed large enough for detection.
To facilitate the possibility of multiple bins being anomalous for the same hash function, the detection rule 3.2 is checked against the same histograms $H^{\hat{i}}$, but with all anomalous bins removed (count set to 0). This is repeated until the detection rule is no longer fulfilled.

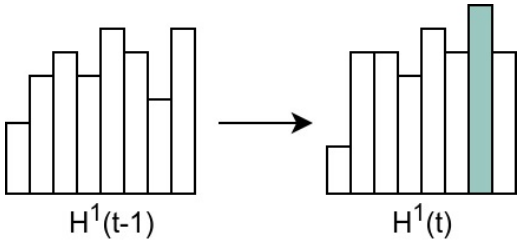


Figure 3.3: Sketch detection step for a single hash with eight bins between two time steps. Slight changes occur in time but only a single bin had a change large enough to warrant detection.

3.4 Extraction

Since hash-functions are not bijective we cannot trace an anomaly in some set of bins $B^{\hat{i}}_{\hat{j}}(t)$ to a specific feature value. That is one of the reasons to use k different hash functions. By keeping track of every feature value which maps into which anomalous bin $(\hat{i}, \hat{j})$, one can use the intersection (or voting) of these sets to trace a specific feature value (IP address) to a set of anomalous bins. This is demonstrated in figure 3.4. This method, while completely deterministic, can misidentify feature values due to hash collisions. Approximating the hash function as a uniform probability distribution, we can see that the probability of two values colliding through all hashes is $\frac{1}{m^k}$ which is generally a very small value.
In practice, the voting scheme used here and also by Brauckhoff et al. [18] is tuned by the voting parameter v and determines the lower bound for how many times a value has to occur in an anomalous bin to warrant extraction. This means that if $v = k$, the voting procedure produces the intersection of all detections. If $v = 1$ the extraction will instead retrieve the union of detections. Brauckhoff et al. [18] also provide simulations of how to tune the voting threshold v to the total number of hash functions to avoid false detections. Another, very similar scheme is implemented by Casas et al. [26]. Here, they also provide an empiric study in how the parameters k, m need to be tuned to provide a unique answer.

3.5 Experimental setup

The experiments done to achieve the results in the next chapter were run on a virtual machine. Using the external address of the flows we verify the parameters suggested by Brauckhoff et al. [18].

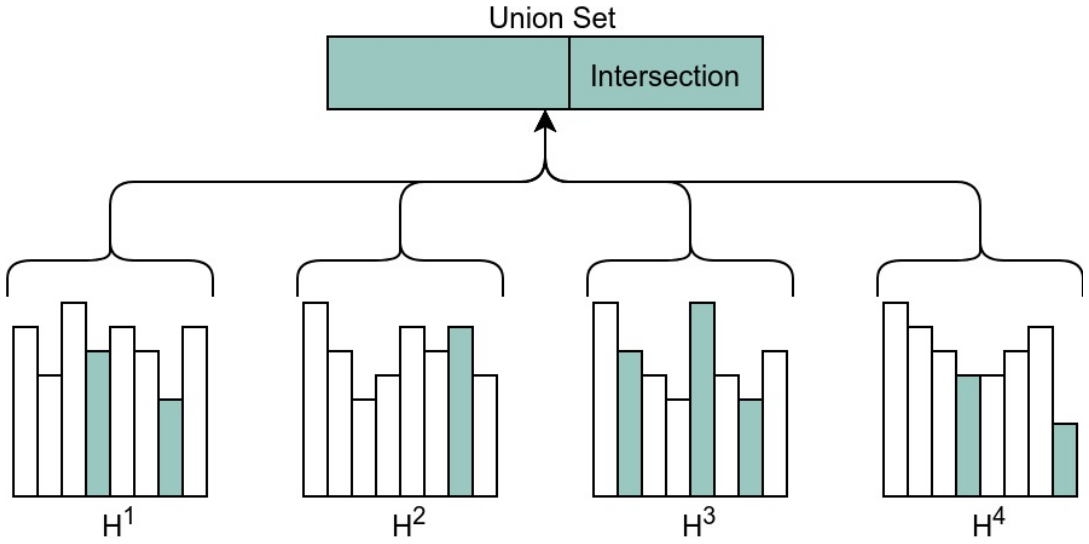


Figure 3.4: Extraction visualization through histogram detection with 4 hashes and 8 bins per hash. The colored bins in the histograms represent bins which have been flagged by the detection rule. A union set containing all the values which map into the selected bins is constructed. The (complete or partial) intersection of this union set represents the anomalous values.

3.5.1 Time series inspection

Before running any detection, the KL divergences for each time step are exported. As discussed in 3.3.2, the time series is used to find a set of potential threshold values for the detection rule, equation 3.2. The experiment was run with a one-week training period, resulting in a time series of the KL divergence.

3.5.2 Threshold

Based on the results of the time series inspection, we choose a set of threshold values. They are based on the variance of the first difference of the KL divergence, as Brauckhoff et al. [18] also used. From this we analyze the resulting detections for the different thresholds.
Detections which represent unexpected traffic within the network are marked as anomalous, and thus True Positive. Detections which show traffic with services commonly used in the network are marked as false positives. This step is done by manual inspection of the Netflow data. Additionally, Recorded Future's service includes some enrichment to IP adresses, such that geolocation and owner organization is known.

3.5.3 Time window size

The next operating parameter is the time window size. Brauckhoff et al. [18] suggested windows from 5 to 30 minutes, and observed that a shorter time window will produce more detections. We run the same dataset for a set of different time

windows to see how it affects the detection.

Algorithm 1: Detection function for one time window. Input a list of feature values from the time window.

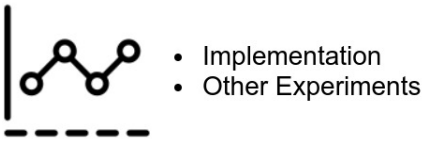
```
for seed in hash_seeds do
    hash all flows with seed, keep a map of (value, hashed value)
    histogram ← distribution of hashed flows
    div ← KL-divergence between histogram, old_histogram
    while div < threshold do
        bin ← argmax |histogram - old_histogram|
        flag (seed, bin) as anomalous
        histogram(bin), old_histogram(bin) ← 0
        div ← KL-divergence between histogram, old_histogram
s ← set of all unique flows
keep score for every value in s (initial 0)
for every anomalous flag do
    for every value mapped into (bin, seed) do
        add 1 to score for value
detections ← values where score > voting_threshold
old_histogram ← histogram
return detections
```

3.6 Other Experiments

As seen in the extensive search of related works, there are plenty of methods to consider and verify. The methods differ greatly in use-case, scalability, and accuracy. In the early stages of this project, small scale versions of a few methods were implemented before settling on the final Brauckhoff-inspired solution. Some general results as well as some analysis of their suitability for our usage are presented at the end of the Results chapter, in section 4.3.

4

Results



One of the key focus points of this project is to build an Anomaly detection system which is agnostic to the kind of network it operates on. In this chapter, the results of testing our implementation based on the paper by Brauckhoff et al. [18] is presented. In section 4.3, experiments with other implementations are presented, as well as short discussions to why the work on those was not continued.

4.1 Implementation

The method chosen for detection in Brauckhoff et al.'s implementation [18] is a threshold value for the KL divergence between two subsequent time windows. They observe that the first difference of the time series of the divergences is approximately Gaussian, and use $3\hat{\sigma}$ as a detection threshold. In figure 4.1 the same experiment is recreated for our corporate data. Our experiments agree in that the first difference is a zero-mean series. To verify the statement that the first difference conforms to a Gaussian distribution we show figure 4.2. We conclude that this is not the case, but the variance can still be used.

4.1.1 Features

In figure 4.2 we also observe the different structure of the Internal and External features. This indicates that the creation of these features provide a better view of the data than the Source and Destination features commonly used. These observations are consistent with the statements by Garcia et al. [36], that bidirectional flows contain a large amount of redundant information. Instead of Source and Destination being redundant versions of approximately the same information, Internal and External instead provide another view of this information.

4.1.2 Parameter estimation

As is commonly the case in anomaly detection there is a tradeoff between detecting true positives (TP) and false positives (FP). In Brauckhoff et al. [18] the ROC[1] curve is presented for whether the histogram detector detects an anomaly.

[1]ROC: Receiver operating characteristics, describes the relationship between true positive rate and false positive rate.

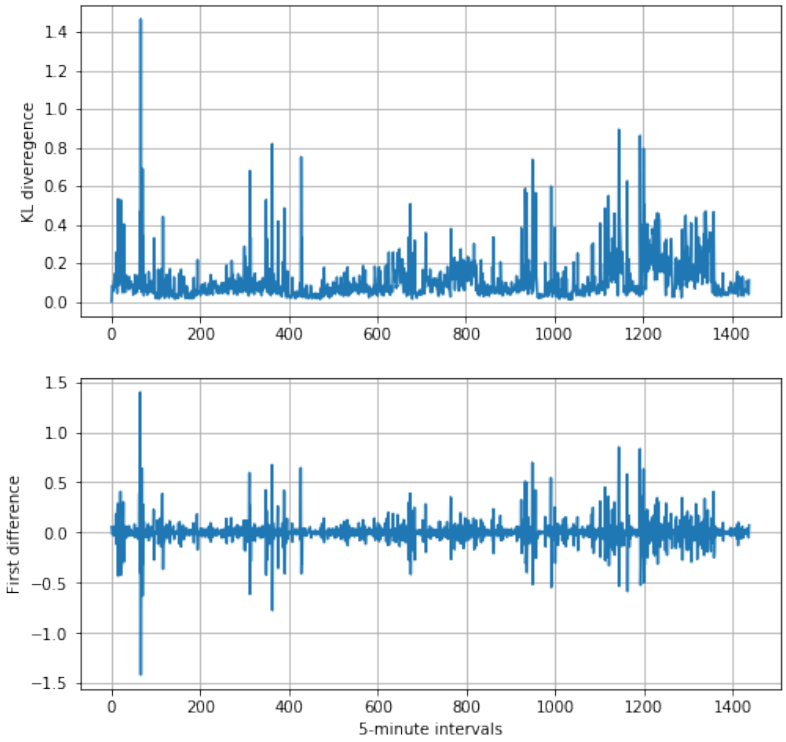


Figure 4.1: Top: KL divergence between subsequent timesteps of the Source IP address feature. Bottom: First difference of KL divergence timeseries.

For our experiment, we used the full detector rather than only the histogram detector. It would have been infeasible to verify histogram detections, as the anomalies are previously unknown. We ran the detector at the following operating point: 8 hash functions, voting threshold 6, 1024 bins, for time windows of 5, 10 and 15 minutes. The threshold value for the detector described in Eq. 3.2 was varied over a set of values based on the $\hat{\sigma}$, around the $3\hat{\sigma}$ operating point suggested by Brauckhoff et al. [18]. The value $\hat{\sigma}$ is unique per time window size and is derived from calculating the KL divergence of the previous week. The resulting detections were manually classified and are presented in table 4.1. All of the true positives are portscan activity. Many false positives are known services with high volume.
Just as Brauckhoff et al. [18] observe, we see that the number of detections decrease for longer time windows. However, as displayed in table 4.2, many detections are duplicates of the same address. Additionally, a comparison between the set of detections from different time windows shows that the overlap is limited. The different time windows provide substantially different detections.

4.1.3 Stability

When working with the sketch algorithm, it is crucial to tune the number of hash functions k and voting parameter to keep detections possible, while minimizing the chance of hash collisions. Empirical results displaying this property is shown by Dewaele et al. [25], while Brauckhoff et al. [18] instead used simulations of different k and v-values to estimate the probability of missing a true positive or causing a

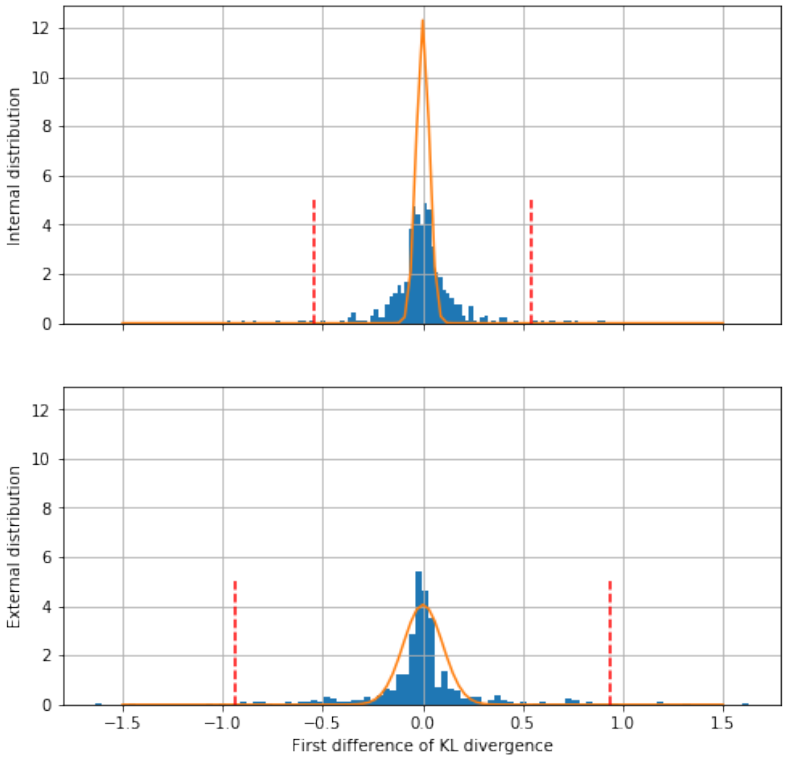


Figure 4.2: Distributions of the first difference of a KL divergence timeseries. The curve is a Gaussian fit to the data and the dashed lines denote $3\hat{\sigma}$. Top: Depicts the Internal feature. Bottom: The External feature.

Table 4.1: Manually classified detections for different threshold values from data spanning 4 days. The detection was run with several different time windows T, and a set of threshold values τ for each time window. Many of the false positives are known, high-volume services. One set of detections from the (5 min, $2.5\hat{\sigma}$) detector was omitted because of a large amount of hash collissions.

	5 min		10 min		15 min	
Threshold	TP	FP	TP	FP	TP	FP
$2.5\hat{\sigma}$	10	73	6	54	2	51
$2.75\hat{\sigma}$	9	57	6	39	2	42
$3\hat{\sigma}$	7	47	4	25	2	32
$3.25\hat{\sigma}$	4	36	4	22	1	29
$3.5\hat{\sigma}$	3	31	3	13	0	7
$3.75\hat{\sigma}$	4	19	2	13	0	5
$4\hat{\sigma}$	4	17	1	11	0	4

Table 4.2: Total number of detections for different time windows in companion with comparisons of number of overlapping detections between windows is shown.

Window	Detections	Unique	Observed in 5 min	Observed in 10 min
5 min	86	26 (30%)	-	-
10 min	64	35 (55%)	41 (64%)	-
15 min	54	30 (56%)	28 (52%)	38 (70%)

false positive.
In addition to mapping the detection capabilities of different parameter sets, the experiments which resulted in table 4.1 also pointed to other difficulties in tuning the parameters. In one case, for the lowest threshold value we had thousands of detections. This is a known problem with the method. Using a too low threshold value when one or several high-volume anomalies occur, will result in a significantly increased number of extracted values in a single time window. This is caused by too many flagged bins, which causes the extraction of IP addresses to extract too many values, as there are too many hash collisions. This can be solved by running the detection with multiple thresholds.

4.2 Runtime

For anomaly detection in networks it is important that a solution can be run in real time. The implementation of this method runs at approximately 60 seconds on a 15 minute time window. This is measured on a quad-core CPU clocked at 4.0 GHz in a laptop configuration. With parallel processing adding more detectors does not increase runtime significantly. However, the processing time increases quickly with the number of hash collisions, sometimes increasing to the order of hours. This problem is discussed at length in section 5.5.

4.3 Other experiments

This section outlines implementations of several methods that were tested as a part of investigating related works. These methods were discarded because they do not fulfill the requirements of anomaly detection in networks. The methods are presented with the specific reason for why they were not used in the final implementation.

4.3.1 Entropy

Entropy has been successfully used in network anomaly detection in several cases, for example by Lakhina et al. and Berezinski et al. [16, 15]. In our experiments, we confirmed that sample entropy often highlighted the time windows of interesting points and anomalies. However, extracting the responsible flows, or responsible IP addresses, are a challenge. In articles by Lakhina et al. [16, 27], the multiway subspace method based on PCA is used. This is useful in the case of OD flows (Entry/Exit point pairs for a flow passing through the network), where the number

of unique pairs is the number of sampling points squared. This kind of analysis is not feasible for source and destination IP pairs.
As seen in figure 4.3, the entropy measure highlights anomalies which can also be seen in the total volume of flows. It also shows anomalies which are not directly connected to the number of flows, but rather the distribution of the external addresses. This is also noted by Lakhina et al. [16], that entropy highlights a partially overlapping set of anomalies compared to direct flow volume monitoring.

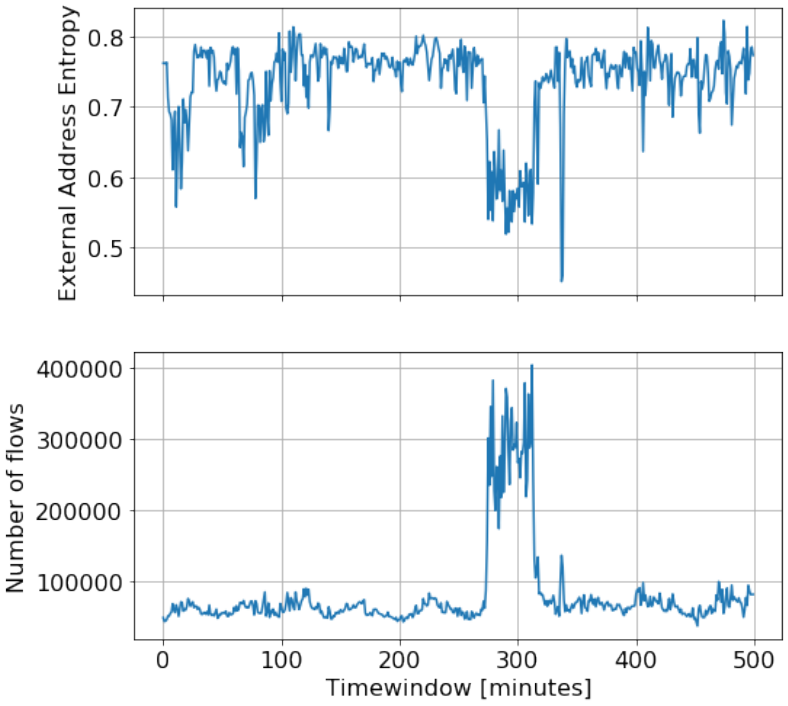


Figure 4.3: Top: Normalized entropy of the external address field. In this case, the anomalies are represented by sudden decreases in entropy. Bottom: Number of total flows in every one-minute time window. Only some anomalies are visible in this representation.

With the challenge of extracting knowledge from a detection we draw the conclusion that the methods that directly measure entropy are better suited for when analyzing the large-scale patterns in a network.

4.3.2 Non-Gaussian detection

Dewaele et al. [25] uses the sketch algorithm in the same way as Brauckhoff et al. [18]. However, for detection, they instead use the parametrizations of Gamma distributions of flows aggregated in time windows of different sizes. The idea is to be able to capture both long- and short-term correlation structures, rather than purely volume dependant structures.
This was implemented and run on the corporate dataset. Figure 4.4 shows the aggregated time series as well as gamma fittings to the distribution, which is consistent with the observations by Scherrer et al. and Dewaele et al. [44, 25].

With the proposed detection using Mahalanobis distance it often detected normal, but heavy traffic as anomalous. Our initial testing also pointed towards it having a low detection rate.

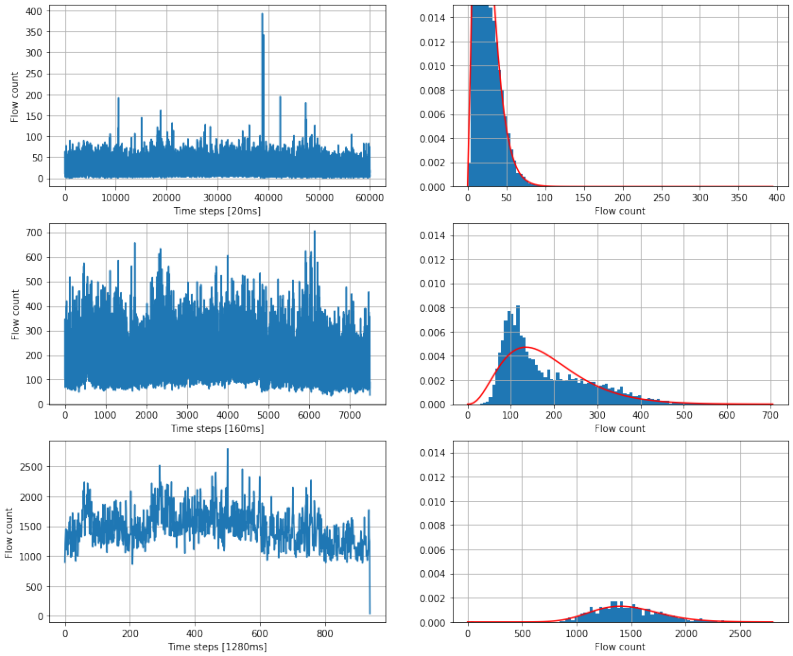


Figure 4.4: Aggregations and fits of 20 minutes of flows. Left column: Total flow count in every time step for different aggregation levels. Right column: The distribution of flow counts for different aggregation levels. A gamma distribution is fitted to each histogram.

Dewaele et al. [25] also describe an anomaly going on for several months. This raises the question of whether this is actually an anomaly, or an odd traffic pattern such that it triggers the anomaly detection. Regardless of whether it *should* trigger the rule or not, there is also the problem of a long-standing anomaly overshadowing smaller anomalies. The authors do not discuss this problem.

4.3.3 Graph methods

Franccois et al. and Chowdhury et al. [12, 1] used clustering on various topological graph features in their articles. However many of the algorithms used to generate them are non-linear in time complexity[2], for example the Inbetweeness has a worst case time complexity of $\mathcal{O}(V^3)$. This is seen with a subset of the corporate dataset with around 6 million flows (daily flows around 100 million). For this subset the algorithm for HITS, inbetweeness, local clustering and eigenvector centrality failed to converge within a reasonable time.
With an even smaller dataset of less than 500 thousand flows the HITS algorithm successfully converged, seen in figure 4.5. By further inspection, all points with non-zero valued authority score were the collection routers. Thus the remaining points,

[2]Time Complexity: Explained further in A.2.

the point of interest exists only on the hubs axis as their authority score is 0. These two clusters are so concentrated that no further conclusions can be drawn.

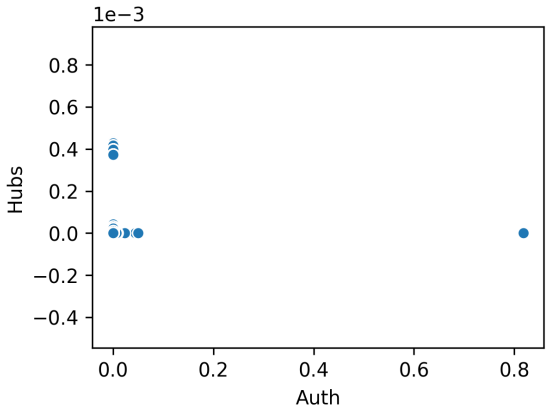
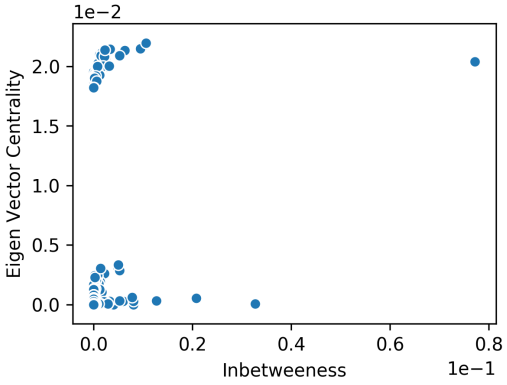


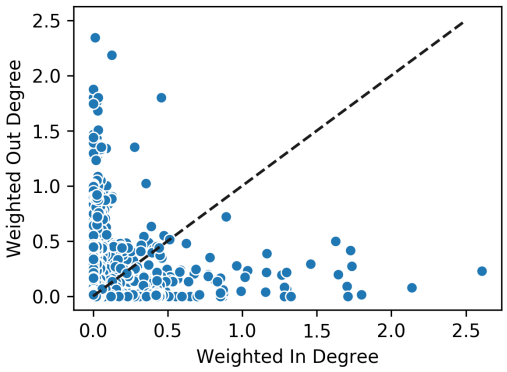
Figure 4.5: Hubs and authority score on a dataset with 500 thousand flows. All non-zero valued auth identified as the collection points.

Using the same dataset for Chowdhury et al.'s [1] method shows figures with the collection points excluded. We found by manually looking at the outliers in inbetweeness, figure 4.6a some TOR-nodes.[3] However the other features presented little knowledge by themselves. The authors used Self-Organizing Maps[4]. With regard to it being an older algorithm not readily available in Python, we instead chose DBSCAN as our clustering algorithm. Using the elbow method to find the ϵ for DBSCAN the clusters didn't provide any more knowledge than the features did by themselves.
In the end our tries at implementing these two papers didn't give any satisfactory results, there are multiple factors which could have caused this. One main factor can be attributed to the difference in datasets used, it may simply be that these kind of P2P communications that both these papers look for doesn't exist in our dataset. Additionally Franccois et al. [12] uses data from a large internet operator, we speculate that these kind of data has a larger variety of addresses. This is apparent when comparing our results for the HITS algortihm vs Franccois et al. who had much more diffusion in clusters [12].

[3]TOR-node: a router related to the TOR-network used to anonymize your network usage.
[4]Self-Organizing Map: a kind of neural network used to produce two dimensional data.



(a) Inbetweeness vs. Eigenvector Centrality



(b) Weighted In- vs. Out Degree

Figure 4.6: Four of the features proposed by Chowdhury et al. [1]. Graphs are shown without collection routers.

5

Discussion

- Result Analysis
- Implementation & Desgin
- Future Improvements

The Discussion chapter begins with analysis of the results aquired by the Brauckhoff-inspired method. Further, we discuss the problem of anomaly detection in general.

5.1 Experimental results

The results of the varying thresholds in section 4.1.2 show that there is a disparity between our observations and the results presented by Brauckhoff et al. [18]. First of all, this experiment ran over 4 days, which is a short time span for these kinds of experiments. For comparison, Brauckhoff et al. present a two-week test.

Secondly, our datasets seem to differ quite a lot, Brauckhoff et al. use an internet backbone link with an hourly flow volume of 90 million, which is comparable to the volume we observe daily in our data. The nature of the network is a plausible factor in explaining the disparity. Many methods presented in section 1.3 are tested on high-traffic links of internet backbone infrastructure. Both Lakhina et al. and Brauckhoff et al. [16][18] use backbone data, which is likely a more stable and consistent source of data than a private network, because of the high number of constantly connected machines.

This disparity is also visible when analyzing the time series of KL divergences, as described in section 3.5.1. 5 days of data were used. The empirically calculated standard deviation $\hat{\sigma}$ can vary greatly from week to week. It was observed to change by a factor of 3 for a 15 minute time window between two subsequent weeks. Since this parameter determines the detection threshold, consistently finding a good operating point is difficult. Brauckhoff et al. [18] do not present the time period for which they calculate this parameter. They do however show a two-day sample of the divergence time series. Comparing that to our samples, such as in figure 4.1, we see that our samples have a much higher level of apparent noise. This again suggests that the different datasets might be a reason for the disparity.

In section 4.1.2, we make the observation that increasing the time window size tends to reduce the total number of detections. This is in line with the results of Brauckhoff et al. [18]. We also see that the number of duplicate detections (the same address for multiple time windows) is higher for short time windows.

5.2 The volume problem

Anomalies which are invisible in some metric can be seen in others. This is suggested from the results in section 4.3.1 where volume data is compared to external address entropy, and also shown by Lakhina et al. [16]. The method we implemented strikes a balance between these, measuring the volume and uses an entropy-inspired view for detection. However, a significant number of flows are still required for a detection. This is observed in our tests, and strongly suggested by Brauckhoff et al. in their use of a flow count threshold in the association rules.

Through manual inspection of the Netflow belonging to our detections we find that these are exclusively anomalies with high volume. Whether other anomalies exist is unknown. This begs the question of which anomalies are at all detectable with our method. As specified in section 2.2, we are primarily interested in malicious or potentially malicious activity. This is still a very wide definition. For example, a botnet agent which contacts its host a handful of times every week would certainly be of interest to detect. With our method however, this activity would never accumulate enough flows to warrant detection.

5.3 Consistency

Brauckhoff et al. [18] mention that larger time windows produces subsets of smaller time windows. Looking at table 4.2 we see that this isn't necessarily true, with the number of unique detection being the highest in percentage for the largest time window. Furthermore the results in table 4.2 show that different time window sizes find different anomalies, which is problematic. Since the anomaly flow count is often on the same order of magnitude as the other highest contributors, an anomaly which was cut off into two time windows will naturally be harder to detect. This suggests that an ensemble of time windows may be an effective strategy.

5.4 Behavioural Analysis

A field where manual inspection is commonly used is behaviour based detection. An experienced eye is able to identify communication patterns that might elude detection methods focused on a larger scope such as ours. These kind of close inspection is very time consuming and requires a certain intuition on where to start.

As shown in our review of related works, there are plenty of methods for anomaly detection and many of them set out to detect different kinds of activity. Either implicitly or explicitly. For example, the BLINC architecture by Karagiannis et al. [5] would only detect malicious anomalies which match their graphlets, the HMM or graph methods search for structures rather than volume and entropy methods such as Lakhina et als. [16] look at feature distributions.

5.5 Implementation issues

This project started with the idea that our implementation should be able to run on a live data pipeline. With the data being stored on a ElasticSearch server we built our application to query from the server. We realized later that it is much faster to keep the files locally and then run the detection algorithms on it. This change reduced the run-time of a detection sweep over a week of Netflow from three days to a couple of hours. With better experimental planning from the start we could have saved a lot of time.
Running the detection algorithm we have encountered issues where the detection algorithm would freeze at certain time windows. This occurs more frequently for lower thresholds which leads to the conclusion that the execution gets stuck on time windows where too many bins are passed from the detection, leading to a high number of hash collisions. The extraction part (see Algorithm 1) of the algorithm, which keeps track of the voting and voting thresholds, was written in a brute-force manner and scales badly with the number of detections, and number of feature values. For a real world implementation, the extraction algorithm would have to be implemented in a better way, or a timeout which preserves the state (histograms) could be used.
Hash collisions also poses a conceptual problem. If coinciding anomalies can force the algorithm to produce a high number of collisions, it can mask the actual anomalies. This problem is counteracted by the size of anomalies required to achieve this. These anomalies would easily be detectable by other, volume-dependent methods.
From the start, a goal was to implement an anomaly detection method which was unsupervised and agnostic to network structure. The method by Brauckhoff et al. [18] proved one of the best fitting methods. While it has proven to highlight interesting traffic and find anomalies invisible to other methods it has also been difficult to find a good operating point for our specific network. As discussed previously, changes to the network over time can change the characteristics of the divergence time series. Tracing the observed characteristic changes to real world network changes would be near impossible.

5.6 Experimental design

In hindsight there are many things we would have liked to explore in experiments, but have not been able to. A contributing factor has been that the combined search space of parameters is large, and it is not directly obvious which parameters need to be tuned. This, combined with the prohibitive running time has made our experiments shorter than desired. The difficulty to pinpoint good values for $\hat{\sigma}$ also required us to restart experiments multiple times.
Another issue in the experimental design is that we are simply working in the dark. As Malowidzki et al. [40] note, high quality annotated data sources are very rare. Other authors, such as Berezinski et al. [15] have instead opted for synthetic attacks in their anomaly detection. For any real conclusions to be drawn on robustness and detection capabilities, a possible approach would have been to simulate attacks.

Instead of tuning the threshold by historical data, it could likely be chosen as a function of the number of bins and average number of flows in a time window. This should give a more robust estimate for a good threshold value.

5.7 Future Improvements

The following points are concrete ways in which to improve the method in addition to fixing the initial implementation issues, either by performance or usability.

5.7.1 Whitelist

When analyzing the results (Table 4.1) we noted that the majority of the false positive detections stemmed from a handful of high-volume services that we know will have a significant amount of traffic in the network. This suggests that a filtering approach with a whitelist could reduce the total amount of flows, making anomalies more visible. The whitelist would not have to be a trust-or-not-trust system but rather a filter with known high-volume services.

5.7.2 Post-detection analysis

Brauckhoff et al. [18] use association rule mining to identify combinations of feature values frequent enough to be noteworthy for a system administrator. This is applied after the detection and extraction steps, to summarize the results. Initially, we intended to use unsupervised methods such as clustering as a post-processing step to group different types of anomalies. This could e.g. be some form of DBSCAN [45] or a neural network [23], making this a completely autonomous anomaly detection system. However, due to time-limitations we instead focused on tuning and evaluating the detection algorithm by itself and opted for manual inspection of detections. It should be noted that even without this step, the method produces a low number of detections which can be manually analyzed.

6

Conclusion

- Future Work
- Recommendations

In this thesis, we have surveyed a large amount of articles investigating the analysis of netflow data. We have also implemented and presented a slightly modified version of the detection algorithm first developed by Brauckhoff et al. [18]. Our results show that even with limited tuning it can detect some anomalies. However, tuning the parameters to find a good operating point has proven a challenge, and robustness and consistency has not been adequately obtained.

6.1 Future work

Drawing conclusions from Netflow data is a nontrivial task, and there are many different ways to tackle the problem. Some ideas that could be investigated in further projects are listed here.

Autoencoder

Autoencoders have previously been implemented in anomaly detection in othe fields, such as Sakurada et al. [46]. The model could be adapted to learn network behaviour, and anomalies could be detected by the reproduction error. The main challenges of this approach would be how to encode the Netflow data into a representation usable by a neural network, and to make it robust to problems like services resolving to different IP addresses.

Traffic classification

Another interesting possibility is a framework similar to the work done in the BLINC architecture [5]. In some of our evaluations, we observed behaviours (packets/bytes sent/recieved) for sets of IP pairs that clearly distinguished different services from each other.

6.2 Recommendations

The possible general applications of Netflow data for Recorded Future can roughly be divided into two categories, *forensic* and *anomaly detection*.

Our work towards the forensic application has been mostly limited to graph methods in section 4.3.3. We believe developing tools that incorporate Netflow into the forensic workflow could be of great interest, but have not found a particular direction for this idea.
For anomaly detection, we cannot directly recommend the method implemented in this thesis. While it does detect some anomalies, it is sensitive to changes in the network and difficult to tune when anomalies are in the same size as normal network traffic.
In conclusion, we thank Recorded Future for the opportunity to work on this project and to develop our understanding in network anomaly detection.

Bibliography

[1] Sudipta Chowdhury, Mojtaba Khanzadeh, Ravi Akula, Fangyan Zhang, Song Zhang, Hugh Medal, Mohammad Marufuzzaman, and Linkan Bian. Botnet detection using graph-based feature clustering. *Journal of Big Data*, 4(1):14, 2017.

[2] Recorded Future: Disinformation service campaigns. `https://www.recordedfuture.com/disinformation-service-campaigns/`. [online; accessed 2019-12-03].

[3] MITRE ATT&CK: Softwares. `https://attack.mitre.org/software/`. [online; accessed 2019-11-04].

[4] Angelos K Marnerides, Alberto Schaeffer-Filho, and Andreas Mauthe. Traffic anomaly diagnosis in internet backbone networks: A survey. *Computer Networks*, 73:224–243, 2014.

[5] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM computer communication review*, volume 35, pages 229–240. ACM, 2005.

[6] N Muraleedharan, Arun Parmar, and Manish Kumar. A flow based anomaly detection system using chi-square technique. In *2010 IEEE 2nd international Advance computing conference (IACC)*, pages 285–289. IEEE, 2010.

[7] Asrul H Yaacob, Ian KT Tan, Su Fong Chien, and Hon Khi Tan. Arima based network anomaly detection. In *2010 Second International Conference on Communication Software and Networks*, pages 205–209. IEEE, 2010.

[8] Dirk Ourston, Sara Matzner, William Stump, and Bryan Hopkins. Applications of hidden markov models to detecting multi-stage network attacks. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, pages 10–pp. IEEE, 2003.

[9] Pardeep Kumar, Vivek Sehgal, Kinjal Shah, Shiv Shankar Prasad Shukla, Durg Singh Chauhan, et al. A novel approach for security in cloud computing using hidden markov model and clustering. In *2011 World Congress on Information and Communication Technologies*, pages 810–815. IEEE, 2011.

[10] Chia-Mei Chen, Dah-Jyh Guan, Yu-Zhi Huang, and Ya-Hui Ou. Anomaly network intrusion detection using hidden markov model. *Int. J. Innov. Comput. Inform. Control*, 12:569–580, 2016.

[11] Nong Ye, Yebin Zhang, and Connie M Borror. Robustness of the markov-chain model for cyber-attack detection. *IEEE Transactions on Reliability*, 53(1):116–123, 2004.

[12] Jérôme François, Shaonan Wang, Thomas Engel, et al. Bottrack: tracking botnets using netflow and pagerank. In *International Conference on Research in Networking*, pages 1–14. Springer, 2011.

[13] Pooja Devi, Ashlesha Gupta, and Ashutosh Dixit. Comparative study of hits and pagerank link based ranking algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, 3(2):5749–5754, 2014.

[14] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[15] Przemysław Bereziński, Bartosz Jasiul, and Marcin Szpyrka. An entropy-based network anomaly detection method. *Entropy*, 17(4):2367–2408, 2015.

[16] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM computer communication review*, volume 35, pages 217–228. ACM, 2005.

[17] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[18] Daniela Brauckhoff, Xenofontas Dimitropoulos, Arno Wagner, and Kavé Salamatian. Anomaly extraction in backbone networks using association rules. *IEEE/ACM Transactions on Networking (TON)*, 20(6):1788–1799, 2012.

[19] Augustin Soule, Kavé Salamatian, and Nina Taft. Combining filtering and statistical methods for anomaly detection. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 31–31. USENIX Association, 2005.

[20] Paul Barford, Jeffery Kline, David Plonka, and Amos Ron. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 71–82. ACM, 2002.

[21] Chin-Tser Huang, Sachin Thareja, and Yong-June Shin. Wavelet-based real time detection of network traffic anomalies. In *2006 Securecomm and Workshops*, pages 1–7. IEEE, 2006.

[22] Wei Lu and Ali A Ghorbani. Network anomaly detection based on wavelet analysis. *EURASIP Journal on Advances in Signal Processing*, 2009:4, 2009.

[23] Mansour Sheikhan and Zahra Jadidi. Flow-based anomaly detection in high-speed links using modified gsa-optimized neural network. *Neural Computing and Applications*, 24(3-4):599–611, 2014.

[24] Dan Li, Dacheng Chen, Jonathan Goh, and See-kiong Ng. Anomaly detection with generative adversarial networks for multivariate time series. *arXiv preprint arXiv:1809.04758*, 2018.

[25] Guillaume Dewaele, Kensuke Fukuda, Pierre Borgnat, Patrice Abry, and Kenjiro Cho. Extracting hidden anomalies using sketch and non gaussian multiresolution statistical detection procedures. In *Proceedings of the 2007 workshop on Large scale attack defense*, pages 145–152. ACM, 2007.

[26] Pedro Casas, Johan Mazel, and Philippe Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772–783, 2012.

[27] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM computer communication review*, volume 34, pages 219–230. ACM, 2004.

[28] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 201–206. ACM, 2004.

[29] International Telecommunication Union. `https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx`. [online; accessed 2019-11-26].

[30] Jon Postel et al. Rfc 791: Internet protocol. *Internet Engineering Task Force*, 1981.

[31] Steve Deering and Robert Hinden. Rfc 2460: Internet protocol, version 6 (ipv6) specification. *Internet Engineering Task Force*, 9, 1998.

[32] Julian Onions. A historical perspective on the usage of ip version 9. *Internet Engineering Task Force*, 1994.

[33] Internet Assigned Numbers Authority. Port service names. `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt`, 2019. [Online; accessed 2019-10-29].

[34] Benoit Claise, G Sadasivan, V Valluri, and M Djernaes. Rfc 3954: Cisco systems netflow services export version 9. *IETF http://www. ietf. org/rfc/rfc3954. txt*, 2004.

[35] BH Trammell and E Boschi. Rfc 5103: Bidirectional flow export using ip flow information export (ipfix). *Internet Engineering Task Force (IETF), Tech. Rep*, 2008.

[36] Sebastián Garcıa. Identifying, modeling and detecting botnet behaviors in the network. *Unpublished doctoral dissertation, Universidad Nacional del Centro de la Provincia de Buenos Aires*, 2014.

[37] Robin Sommer and Anja Feldmann. Netflow: Information loss or win? In *Internet measurement workshop*, pages 173–174. Citeseer, 2002.

[38] Internet Engineering Taskforce. Internet security glossary. `https://tools.ietf.org/html/rfc2828`, 2000. [online; accessed 2019-11-28].

[39] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[40] Marek Małowidzki, P Berezinski, and Michał Mazur. Network intrusion detection: Half a kingdom for a good dataset. In *Proceedings of NATO STO SAS-139 Workshop, Portugal*, 2015.

[41] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[42] Marc Ph Stoecklin, Jean-Yves Le Boudec, and Andreas Kind. A two-layered anomaly detection technique based on multi-modal flow behavior models. In *International Conference on Passive and Active Network Measurement*, pages 212–221. Springer, 2008.

[43] Andreas Kind, Marc Ph Stoecklin, and Xenofontas Dimitropoulos. Histogram-based traffic anomaly detection. *IEEE Transactions on Network and Service Management*, 6(2):110–121, 2009.

[44] Antoine Scherrer, Nicolas Larrieu, Philippe Owezarski, Pierre Borgnat, and Patrice Abry. Non-gaussian and long memory statistical characterizations for internet traffic with anomalies. *IEEE Transactions on Dependable and Secure Computing*, 4(1):56–70, 2007.

[45] Tran Manh Thang and Juntae Kim. The anomaly detection by using dbscan clustering with multiple parameters. In *2011 International Conference on Information Science and Applications*, pages 1–5. IEEE, 2011.

[46] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4. ACM, 2014.

[47] Internet Engineering Taskforce: Rfc. `https://ietf.org/standards/rfcs/`. [online; accessed 2019-12-01].

[48] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

A

Terms in Computer Science

A.1 Request for Comments (RFC)

Request for Comments (RFC) are technical and organizational documents that describe many aspects of computer networking used by the Internet Engineering Task Force (IETF) [47]. These documents are authored and submitted by engineers and computer scientist for peer review, to convey new concepts, meeting notes, opnions and sometimes humour e.g. Onions RFC on the history of IPv6 [32]. The IETF can choose to adopt a RFC as a internet standard.

A.2 Computational Time Complexity

Time complexity is a measurement used in computer science to measure the running time of an algorithm. It is estimated by counting the number of elementary operations, as a single elementary operation is considered constant in time. An algorithm is said to take linear time if the computational time grows at most linearly with the size of the input. Thus a non-linear algorithm could grow for example at logarithmic, polynomial or exponential time.

A.3 Hash function

A hash function is a function which maps any input to a discrete set of output values. A useful property of a hash function is the approximate uniformity of its output distribution. Note that a hash function by definition is deterministic, and the uniformity is a result of the mapping rather than randomness.

A.4 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm proposed by Ester et al. [48]. It is a non-parametric algorithm that generates clusters using the nearest neighbour method. This means that high density points are clustered together while low density points is classified as outliers.