Project Mapyen

A network analysis tool to identify anomalous host behaviours

Valentine Mairet

Delft University of Technology, Department of Cyber Security

TU Delft

Project Mapyen

A network analysis tool to identify anomalous host behaviours

by

Valentine Mairet

to obtain the degree of Master of Science in Computer Science
Software Technology track
specialisation in Cyber Security,
to be defended publicly on February 14, 2018 at 10:00 AM.

Student number:	4141784	
Project duration:	June 1, 2017 – February 14, 2018	
Thesis committee:	Prof. Dr. P. Hartel,	TU Delft, chair
	Peter Cooper,	Adyen
	Dr. Ir. S. Verwer,	TU Delft, primary supervisor
	Prof. Dr. Ir. A. van Deursen,	TU Delft, secondary supervisor
	Maurício Aniche,	TU Delft, coach

TU Delft

Preface

"True glory consists in doing what deserves to be written; in writing what deserves to be read; and in so living as to make the world happier for our living in it."

— Pliny the Elder

The reality is that our world is moving towards a fully digital era if it cannot already be considered as such. The reason why I set my course for a career in Cyber Security is that I strive for a safe and secure environment for the generations of Tomorrow. I care about preservation of our values as the digital world develops, and I want to make a difference, regardless of how small my influence may turn out to be in the grand scheme of things. The significance related to my work involves the many and I sincerely hope my contributions can provide a bit of help to at least the few.

```
# ----- Acknowledgements -----

important_people = ['Peter Cooper', 'Sicco Verwer', 'Arie van Deursen', 'Mauricio Aniche', 'Huub
    van der Voort', 'Julienka Mollee', 'Daan Wagenaar', 'Paul Moreno', 'Albert Tresens', '
    Christopher Lawton', 'Adyen Catering Staff']


def thank(people):
    for awesome_person in people:
        print('Thank you so much for everything, %s.' % awesome_person)


thank(important_people)
```

```
# ----- Output -----

Thank you so much for everything, Peter Cooper.
Thank you so much for everything, Sicco Verwer.
Thank you so much for everything, Arie van Deursen.
Thank you so much for everything, Mauricio Aniche.
Thank you so much for everything, Huub van der Voort.
Thank you so much for everything, Julienka Mollee.
Thank you so much for everything, Daan Wagenaar.
Thank you so much for everything, Paul Moreno.
Thank you so much for everything, Albert Tresens.
Thank you so much for everything, Christopher Lawton.
Thank you so much for everything, Adyen Catering Staff.
```

Valentine Mairet
Amsterdam, February 2018

Contents

1 Introduction

We are slowly moving towards a world where weapons of mass destruction are but a few bytes in size and can potentially do more damage than an Apache gunship armada on full fire-power. This realisation struck us twice in a few weeks with the WannaCry [45] and Petya [36] outbreaks of May and June 2017, and later on with the Equifax and Deloitte data breaches [54]. While it may be that no one has died from these cyber attacks, the fact that such attacks can turn off medical systems or release our personal data proves the impact the digital world can have on our regular lives. Cyber Security has grown in importance over the years and awareness is slowly spreading across the major powers of this world. Not only is our data subject to this cyber threat but so are our people, our facilities, our reputation as companies, and our human right to privacy. With this ever-growing cyber threat landscape, it becomes crucially important for governmental institutions and companies to invest in security to safeguard their people and the rest of their assets.

The latest developments in cyber security management have delivered frameworks and policies to help such entities defend against the phantom menace. One example of this is the NIST framework [37], which provides guidance on how organisations can assess and improve their capabilities to prevent, detect, and respond to cyber attacks.

Cyber attacks are seldom on the scale of WannaCry or the Petya outbreak, or the Equifax and Deloitte data breaches, but the threat is obviously growing. Most of the time, though, malicious and low-risk incidents such as malware infections or successful phishing campaigns are the common hazards that populate the threat landscape. However, their impact must not be underestimated and can be very significant to the affected target. This is why organisations must be well-prepared and remain vigilant toward **each** cyber threat and have mitigation measures put in place in order to avoid, reduce, or recover from the resulting impact.

Governments, companies, and individuals all possess assets that can put them at risk of being the target of cyber attacks. Each company is exposed in its own way because of the nature of its function, its history, the kind of data it processes, its set goals, and its reputation in general. Employees of relatively sensitive companies may be the target of phishing campaigns to obtain access to confidential information specifically handled by the company. Malware, spyware, ransomware may end up on employee computers and lead to sensitive data being compromised or stolen, or even a company data breach.

1.1. Adyen

Adyen is a payment service provider that processes a large number of payments every day from all over the world. Adyen's merchants are scattered across the globe and the data handled by Adyen is often highly sensitive. To protect the privacy of shoppers and the integrity of merchants, Adyen is bound to privacy and data protection rules in order to remain a sustainable and trustworthy business. The relevance of security for Adyen is undeniable, especially when it comes to information security. How-

ever, not only Adyen's transactions are at risk of cyber attacks. Adyen employees, some of whom have specific access rights to the sensitive data handled, are also at risk of becoming targets of interest to attackers. Adyen's internal infrastructure is also potentially a high-value target for attackers, especially those with financial interests. Insider security is also important for Adyen, and security monitoring is a solution to preserve internal security.

1.1.1. The Adyen Project

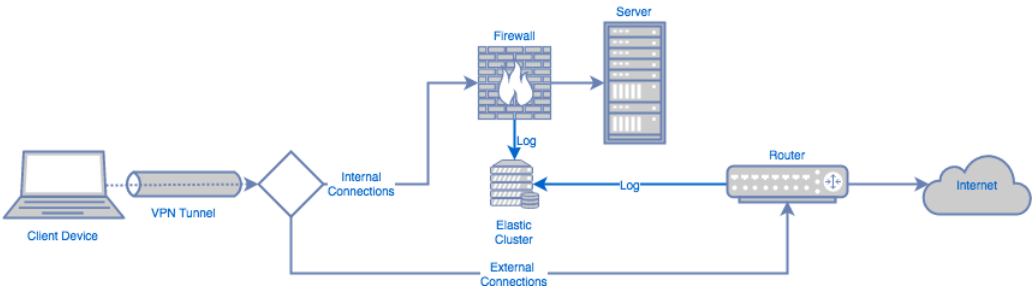
Security monitoring has risen in popularity as the cyber threat landscape has grown to an alarming size. Any company, especially in the case of companies that deal with extra sensitive information, is in dire need or has the obligation to perform dedicated monitoring to protect their assets as well as their employees. The goal of this project is to *investigate current security incident detection methods and uncover or identify those which suit the Adyen context best.* Current research to identify potential security incidents that appears very promising includes rule-based detection systems, statistical models, or anomaly detection [34].

1.1.2. Problem Description

Adyen is a payment service provider company, which means that substantial flows of transaction data are retrieved and processed by Adyen, and these can be highly sensitive for both merchants and shoppers. Adyen employees use, handle, and analyse this data every day. Adyen employees can be the target directed phishing attacks and be potentially vulnerable to malware infections that would cause inadvertent data breaches. In addition to that, people make mistakes both in analysis, assessment and communication, and of course a malware infection can be just a click away. More alarmingly, there always exists the insider threat of a bribed or disgruntled employee with malicious intent to deliberately cause an infection or leak sensitive data. This certainly poses a risk to the company, its assets, and its customers.

Adyen collects real-time network traffic metadata from the internal network and stores this data in a specific kind of noSQL datastore — an Elasticsearch cluster — which enables us to easily extract the data to perform analyses. Figure 1.1 illustrates a simplified example of the process used for logging of this data. The network traffic from an external device is either routed, partially or fully encapsulated within VPN tunnels. The detailed routing mechanisms are outside the scope of this paper. In the case of full-tunnel traffic, a routing decision takes place: connections to internal Adyen devices are directed through firewalls to Adyen servers, while outbound external connections go through routers and after that, through proxy servers towards the Internet. This next step is, for the sake of simplicity, not illustrated in the diagram. The logging of the metadata into the Elasticsearch cluster occurs at both the firewalls and routers.

Figure 1.1: A simplification of network traffic routing and logging of network traffic metadata in Adyen. Full-tunnel traffic goes through a decision process that takes connections to Adyen internal devices to firewalls, where the connections' metadata is logged into an internal Elasticsearch cluster, at the same time that the traffic itself is passed through to internal servers. Outbound connections, i.e., traffic intended for destinations outside of Adyen, are routed through routers to the Internet. The routers send the logs to the Elasticsearch cluster. Note that this diagram is a very simplified version of the actual network infrastructure.



Large companies that perform network monitoring to ensure the safety of their assets make use of anomaly detection systems such as DARKTRACE [13] or resource-intensive deep packet inspection

techniques. Such commercial products, however, are known to deliver a large number of false positives and may return false negatives — that is, miss unknown or unexpected anomalies, the Rumsfeldian unknown unknowns [9]. These two pathological failure modes can lead to a failure to respond effectively or in a timely manner to real hostile traffic. At Adyen, with a relatively large network and significant network traffic, finding the best method to perform network incident detection is challenging, as scalability and performance of the method to be used become serious issues that need to be taken into account.

We pose a problem that concerns finding a **well-founded solution to execute real-time network incident detection on lightweight network traffic metadata, which can provide a basis for future security monitoring research and development**. This mostly includes security threats that may put Adyen's inner functioning, data protection policies, or reputation at risk. We are using the term *lightweight* because the data collected just contains basic information about network connections made and does not go in depth into the communication. Solutions for security monitoring that use deep packet analysis may work better because they have access to much more information.

1.2. Context

When it comes to protecting their environments against cyber threats, companies usually implement their own security solutions or outsource it to specialised Managed Security Services companies. One of the first steps towards efficient cyber security management is the detection of cyber incidents [37]. Internal security monitoring tools, such as DARKTRACE [13], which is used in multiple large companies, or rule-based packet analysis tools such as Suricata [49] and Snort [46], or SIEM technology [50] have been developed to accomplish this step.

Advancements in latest research show techniques developed in the past seven years, such as Nfsight [10], CANDID [32], SANSR [21], and more. This spectrum of studies offers specific solutions that are more or less applicable to a large corporate context because of a certain gap between the academic and industry domains. For example, academic research is mostly performed on smaller, restricted datasets, often collected at the concerned universities themselves. Since companies would rather not share their internal traffic, academics performing research in security monitoring need to work on their own available data, which may not be representative of enterprise network behaviour.

With that being said, corporate and academic solutions work for the most part and are satisfactory to the large audience of both worlds. One crucial point worth mentioning is that current solutions for incident detection still deliver a high number of false positives [9]. It is in the hands of a security analyst to determine whether an incident is an actual cyber threat. The task of analysing the cause and impact of an incident can become a difficult one, as the analysis is often not so tractable. Furthermore, privacy concerns come into play and the European General Data Protection Regulation (GDPR) [39] coming up in May 2018 drastically increases these concerns.

1.3. Research Question

The aforementioned cyber incident detection technology works by performing rule-based checks (this event does not fit this rule), anomaly detection (this event does not correspond to the learned behaviour), or establishing statistical models (this new value does not match the model). Rule-based approaches usually, but not always, involve deep-packet inspection, which we are not able to do since the data available is lightweight network metadata. Anomaly detection, which can also non-extensively base itself on statistical models, is an interesting domain and the research concerning anomaly detection purely on network metadata is abundant.

In this context, we do not have a priori knowledge of the network; the analysis happens in an unsupervised environment, where the data is not labelled. Classical unsupervised anomaly detection for this problem generally starts with feature extraction on a dataset, where features can relate, for example, to connectivity behaviour, dispersion, or traffic content [10, 15]. This is traditionally followed by a clustering approach based on the features extracted to identify host behaviours and similarly behaving network assets [34]. Anomalous behaviour is then detected and reported.

Network security threats can be associated with anomalies as they derive from the average normal behaviour expressed by the hosts inside the network. The latest research provides threat detection solutions using anomaly detection with still relatively high false positive rates [34], nearing an average rate of 20%. Looking to introduce specific incident detection inside the Adyen network, our goal is to uncover the most appropriate anomaly detection method for the large-scale context of Adyen and extend upon the current research.

1.3.1. Statement

The lightweight network metadata provided by Adyen is called NetFlow data, which are quite commonly collected in companies [19]. NetFlow is a network protocol introduced by Cisco to collect metadata about network connections for the purpose of monitoring. For the sake of clarity, NetFlow data is referred to as *netflow* for singular instances or *netflows* for multiple in the rest of this paper. On average, Adyen collects approximately 240,000 netflow entries per minute every day, and this number increases throughout this research as the number of employees increases worldwide. We can now formulate the research question that we answer in this paper.

MQ — What clustering-based anomaly detection technique can we use to efficiently perform network incident detection in a large-scale industrial context?

We have to be critical and make it clear that current approaches may not perform as well when scaled up to the size of an industrial context such as Adyen, despite the relative simplicity of its network infrastructure. We are dealing with a company of roughly 700 employees scattered across the world, meaning different time zones, and all traffic going full or partial tunnel through the Adyen VPN. Related studies done on network metadata have been done on datasets such as DARPA or KDD-99 dating back from the late 90s [14, 26], which are no way comparable to the Adyen data size and recency. Moreover, these datasets have limitations in terms of data they hold and have been criticised for their generation procedures [34].

1.3.2. Decomposition

The primary research goal is to find an appropriate network anomaly detection approach. However, we are in an unsupervised environment inside an industrial-level network, with real active people and machines. This means that we must pose several subquestions that need to be answered to fulfil the research goal.

SQ1 — How is our research in network anomaly detection influenced by Ethics?

What aspects of our project must we consider in order to remain ethical in our analysis of network traffic? We want to find a network anomaly detection technique that is ethically correct and morally acceptable. Particular characteristics of internal network traffic analysis raise ethical and legal concerns that *need* to be taken into account when performing such analysis. It is not only our responsibility as academics to act ethically, but it is also part of the ACM Code of Ethics that provides ethical guidelines to software engineers and IT professionals in general [8].

SQ2 — On which features extracted from the available network metadata can we build clusters?

Unsupervised learning methods perform network host clustering to establish a baseline for general crowd behaviour and to detect outliers; hosts that derive from the baseline. These outliers are then flagged as anomalies, such as described in [10, 15, 18, 34]. Hence we take the latest advancements in anomaly detection on network metadata and tailor it to fit our problem. Some of the features used in behaviour clustering in the current research may or may not be redundant when the data is scaled to our context or may even have a negative effect on the analysis.

SQ3 — Which cost-effective clustering methods do we pick that fit the context best?

Among the numerous clustering methods, which ones do we pick to perform our behaviour clustering? Research such as [15] describes a clustering method to detect non-convex shaped clusters. Other

types of clustering, such as density-based clustering, uncover non-convex shaped clusters as well. Part of choosing an appropriate clustering method is looking for a balance between explainability and understandability, which are just as important as scalability and efficiency for Adyen. This is what makes the problem require a solution that is cost-effective in these four regards. Exhaustively testing **all** clustering methods is impossible, so we have to choose a subset of clustering techniques from the available body of knowledge, keeping in mind the solution requirements.

SQ4 — What is an anomaly?

Once we have the baseline for crowd behaviour established, we are able to differentiate host behaviours inside the network based on the chosen features. We then need to move forward towards the anomaly detection step and clearly define what makes an anomaly and which anomalies we want to detect. The survey from [34] provides us with assumptions made in the context of clustering-based anomaly detection in order to characterise anomalies, but how does that fit into our model?

SQ5 — Are cyber threats of interest detectable? To what extent can lightweight network data be used for the purpose of detecting these cyber threats?
The cyber threat landscape is large and impossible to exhaustively include in our anomaly detection engine. Adyen has a specific subset of cyber attacks that are more import than others, as they have more business impact than other cyber attacks. These attacks are internal port scans, exfiltration of data, and malware infections. But can this subset of attacks be detected by the methods offered by current research? Moreover, can these attacks be detected solely through the analysis of lightweight data or should more information be provided? We hope to uncover a novel approach in order to increase our chances of successful detection, even with metadata that does not contain much information.

1.4. Ethical Process

Adyen must obviously try to maximise the protection of assets and employees by putting in place security measures to achieve this goal while obeying the law and continuing to conduct business efficiently. It may be argued that purely legalistic criteria for operating a business are not sufficient, and so we address ethical concerns around data analytics for security monitoring purposes in this section.

The task at hand concerns the execution of a data analysis of collected network metadata, including data from people, to detect potentially suspicious activities. Particular characteristics of this task may raise ethical and legal concerns. Specifically, the issue of concern is the collection of significant volumes of data about, amongst others, real people's activities using computers on the network and analysis of this activity, the possible justifications for that collection and analysis for security purposes, and therefore the privacy implications and employees who may be the subjects of collection and analysis for this project.

To our knowledge, this may be one of the first Cyber Security master's thesis project at this university that executes a formal ethics review, after one supervisor raised a question about the ethical basis for the research. The ethics review process was iterative and involved translating the project concerns into terminology suitable for laymen.

We consult with the TU Delft Human Research Ethics Committee to agree upon guidelines to follow through the entirety of this project, in order to remain legal and ethical with regards to the people's data handled. The main point of concern from the committee is the analysis of individual IP addresses that can potentially identify real people inside the network. Anonymity is, therefore, a crucial must that this research must take into account when performing the extraction, processing, and analysis of the network metadata. The following guidelines are delivered as a conclusion to the Ethics Committee project application:

- Employees are made aware of the collection and analysis of the internal network metadata for the purpose of security monitoring.
- The host IP addresses in the network metadata must be anonymized for the analysis.

- If deanonymization of IP addresses is required, this should be done by authorized Adyen personnel only.
- Any call to action required after the discovery of an incident should be done by the authorized personnel.

As of October 23, 2017, the project has received full approval from the TU Delft Human Research Ethics Committee to pursue and finalise this project.

1.5. Outline

As to answer the second research subquestion **SQ2**, we dedicate a chapter to explain the mathematical concepts that are used in this paper. Chapter 2 then describes the latest research in network analysis and anomaly detection research. This chapter also answers part of **SQ3** and **SQ4**. The methodology to present the dataset used and guide the execution of this research is introduced in Chapter 3. It describes the lightweight netflow clustering procedure to follow to fully answer **SQ3**, and the profiling and anomaly detection steps to fulfil **SQ4**. The lightweight netflow clustering analysis and execution are explained in Chapter 4. The host behaviour profiling and anomaly detection processes are described in Chapter 5. The combination of the three of concepts of clustering, profiling, and detecting anomalies into one system for Adyen is elaborated upon in Chapter 6, where a module-by-module explanation of the product is done. **SQ5** is answered in the following Chapter 7, which evaluates the system created by testing it against synthetic data. This subquestion is further discussed in Chapter 8 with a reflection on the research challenges faced throughout this study. Finally, we conclude this paper in Chapter 9 and open the doors to the possible future work.

2

State of the Art

*Recent research netflow analysis provides insights on how to address the posed problem and also on what to do or what **not** to do. We have already briefly mentioned a few studies conducted in this domain in the earlier sections. This section dives deep into the research that is useful for our topic of interest.*

The research question is divided into subquestions or work items that allow us to focus our study more thoroughly. These work items include host behaviour clustering and network anomaly detection. They provide guidance to lead this research and answer the main research question.

Before diving into the literature on host behaviour identification and clustering for the purpose of network anomaly detection, there are a few key concepts that require definition. This is done in Section 2.1, where we define notions such as clustering, network traffic metadata, and other ideas that are further applied in this study. The interesting recent research that relates to identification of a clustering-based anomaly detection technique for Adyen is described in Section 2.2. This is where we explain the current work in clustering of network behaviours and anomaly detection for network monitoring purposes.

2.1. Background Informaton

A few terms that are mentioned and used in the next chapters require definition in this section for better understanding of the research. These concepts are independent from this project and may be very applicable in other contexts. The following sections attempt to explain these key concepts.

2.1.1. Mathematical Concepts

In this report, we use different mathematical terminology that requires explanation in more detail. We have two important concepts on which we focus: information entropy and the notion of distance.

Entropy

A term that is used in this research is the concept of *entropy*, which is a metric for disorder or uncertainty in information theory, introduced in 1948 [44]. Entropy can be used to quantify the unpredictability of a piece of information, meaning how much of the information is potentially random. Another possible extension of the use of entropy is the measurement of how noisy information can be, and in this paper, we particularly make use of Shannon's Entropy, which is expressed by the following formula:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

We take a long sequence of the same numbers, namely $[1,1,1,...,1,1,1]$. In the formula above, p_i represents the probability of occurrence of symbol i and n is the total number of symbols. In this case, we only have 1 symbol, namely the value 1, so $n = 1$ and $p_1 = 1$. Following the formula:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i = -\sum_{i=1}^{1} p_1 \log_2 p_1 = -1 \log_2 1 = -0 = 0$$

In this case, the entropy $H(X) = 0$, because all the symbols are the same and there is, therefore, no randomness. Let us now take the sequence of numbers $[1,3,3,1,1,2,1,1,3]$. We solve the formula:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i = -[p_1 \log_2 p_1 + p_2 \log_2 p_2 + p_3 \log_2 p_3]$$

$$H(X) = -\left[\frac{5}{9}\log_2 \frac{5}{9} + \frac{1}{9}\log_2 \frac{1}{9} + \frac{1}{3}\log_2 \frac{1}{3}\right] = -[-1.35164] \approx 1.35$$

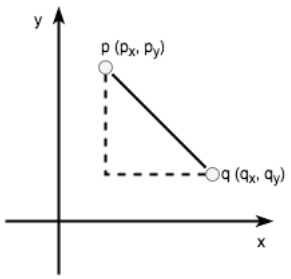
An entropy of approximately $H(X) = 1.35$ still means that the entropy is relatively low, which makes sense because we only have three values. It does, however, indicate that this sequence is more random than a much larger sequence containing just the same numbers.

Distance Metrics

Distance is the measure of how far apart objects are. In machine learning particularly, distance is often used as dissimilarity metric, to measure to what extent data differs. There are many different types of distance and this paper focuses on four main distance metrics, namely Euclidean, cosine, Manhattan, and Mahalanobis distance.

Euclidean

Figure 2.1: A graphic illustration of the concept of Euclidean distance. The Euclidean distance between point p and point q is denoted by the length of the line between them. The Euclidean distance relates to the Pythagorean theorem by being the calculation of length of the hypotenuse of a right triangle from points p and q in 2D space and 3D space.



Euclidean distance is also referred to as the *ordinary straight line* distance. The Euclidean distance between points p and q is the length of the line segment connecting them. It is the most common distance metric in geometry and is expressed by:

$$\sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

A commonly known problem with Euclidean distance is that it loses its power in higher dimensions. Euclidean distance becomes unreliable when moving towards dimensions higher than 3 [7].

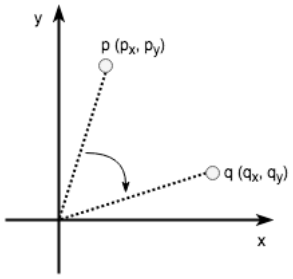
Figure 2.1 illustrates the concept of Euclean distance. Since we are in a 2-dimensional graph, i in the formula is equal to 2. Dimension 1 is x, and dimension 2 is y. So, the formula becomes:

$$\sqrt{\sum_{i=1}^{2} (q_i - p_i)^2} = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}$$

Cosine

Cosine distance is closely related to cosine similarity by being its complement in positive space. It is an angle measurement instead of being the measurement of a line between two points, which makes it a

Figure 2.2: A graphic illustration of the concept of cosine similarity. The similarity between point p and point q is the measure of the angle between $\boldsymbol{p}$ and $\boldsymbol{q}$.



measurement of orientation instead of magnitude. Cosine similarity takes two vectors and calculates their dot product, and then normalises the result by the product of the vector lengths. Output close to 1 indicates high similarity. The formula for the cosine distance between p and q is:

$$1 - cos(p,q) = 1 - \frac{\boldsymbol{p} \cdot \boldsymbol{q}}{||\boldsymbol{p}|| ||\boldsymbol{q}||}$$

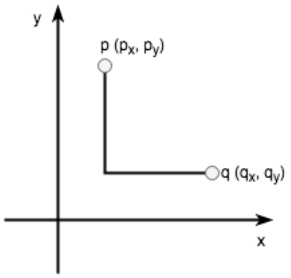
Taking the complement of cosine similarity returns the cosine distance. Values close to 1, therefore, indicate high dissimilarity, distance. Cosine distance seems to work better than Euclidean distance in high dimensions [7].

Figure 2.2 shows a simple representation of cosine similarity between points p and q. If we calculate the cosine *distance* between p and q according to the formula, we obtain:

$$1 - cos(p,q) = 1 - \frac{\boldsymbol{p} \cdot \boldsymbol{q}}{||\boldsymbol{p}|| ||\boldsymbol{q}||} = 1 - \frac{p_x q_x + p_y q_y}{\sqrt{p_x^2 + p_y^2}\sqrt{q_x^2 + q_y^2}}$$

Manhattan

Figure 2.3: A graphic illustration of the concept of Manhattan distance. The distance between point p and point q is the distance along the right angle.



Manhattan distance, also referred to as City Block Distance, is similar to Euclidean distance but it measures the distance between two points p and q along a right angle. It basically computes the distance between these two points as if it followed a grid-like path. The formula for this distance metric is:

$$\sum_{i=1}^{n} |q_i - p_i|$$

Figure 2.3 depicts the concept of Manhattan distance. In this case, there are two dimensions, so i in the formula is 2. The first dimension is denoted by x, and the second by y. So, the formula becomes:

$$\sum_{i=1}^{2} |q_i - p_i| = |q_x - p_x| + |q_y - p_y|$$

Mahalanobis

As opposed to the previously mentioned distance metrics, Mahalanobis distance computes the distance between a point p and a distribution D of a dataset. It generalises the idea of computing how many standard deviations away point p is from the mean of D [31]. It is, however, possible to compute the Mahalanobis distance between two points u and v by including the inverse of the covariance matrix V of the entire dataset in the formula. The formula for the Mahalanobis distance is:

$$\sqrt{(\boldsymbol{u}-\boldsymbol{v})V^{-1}(\boldsymbol{u}-\boldsymbol{v})^T}$$

We describe an example of Mahalanobis distance calculation. Let points u and v have respective 2D coordinates (u_x, u_y) and (v_x, v_y), and be taken as two vectors

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Suppose these points are taken from a larger dataset with many 2-dimensional points, distributed randomly. This dataset has a covariance matrix, also known as dispersion matrix, which generalises the notion of variance of the dataset to two dimensions. This matrix is represented by:

$$\begin{bmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{bmatrix}$$

The number denoted by $\sigma(x,y)$ is the covariance between the two vectors x and y given as parameters. The covariance represents how two vectors vary with one another. Note that, if there is no correlation between the x and y dimensions, $\sigma(x,y) = \sigma(y,x) = 0$. By taking these values into account, the Mahalanobis distance between u and v is then:

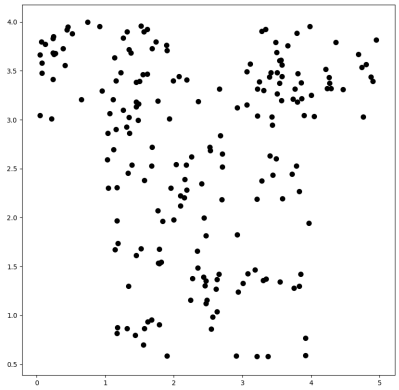
$$\begin{bmatrix} u_x - v_x \\ u_y - v_y \end{bmatrix} \frac{1}{\sigma(x,x)\sigma(y,y) - \sigma(x,y)\sigma(y,x)} \begin{bmatrix} \sigma(y,y) & -\sigma(x,y) \\ -\sigma(y,x) & \sigma(x,x) \end{bmatrix} [u_x - v_x u_y - v_y]$$

The Mahalanobis distance basically computes the Euclidean distance between two points but includes the inverse covariance matrix of the entire dataset in the calculation. Note that, when the data is highly correlated, the covariance matrix converges to the identity matrix and becomes non-invertible, so the Mahalanobis distance is undefined.

2.1.2. Clustering

Humans tend to categorise entities into clusters, or classes. Each cluster is characterised by common attributes specific to each entity. It is a natural way to distinguish between the similar and the different. In machine learning and pattern recognition, clustering comes into play when class labelling of the training set is not available [51], generally speaking in an unsupervised learning context. There exist different types of clustering, which seem to be categorised differently across the literature [25, 51]. We use four distinct types of clustering in this paper, which are commonly said to fall into different categories.

Figure 2.4: A set of datapoints. This set is generated for the purpose of showing how clustering algorithms can differ, also in terms of result.



For each clustering type, we show the results of the clustering applied to the dataset presented in Figure 2.4. This dataset is generated for the purpose of showing how clustering outcomes can differ depending on the method for clustering used.

Connectivity Clustering

This is a class of distance-based clustering techniques that hypothesise that points closer to each other in space are more similar than points further away. One example of connectivity clustering is hierarchical clustering, also known as agglomerative clustering. Connectivity clustering starts by creating clusters out of every single point in the dataset and slowly aggregates these clusters by merging and splitting smaller and larger clusters based on distance between points inside these clusters.

Figure 2.5: A small example of agglomerative clustering with average linkage to create three clusters. Average linkage is when the distance between two clusters is defined by the average distance between data points in the first cluster and points in the second clusters. Clusters with the smallest average distance are merged together. In this plot, we have three visible clusters that are established by the agglomerative clustering with average linkage algorithm. Each colour represents a different cluster, assigned by the algorithm. The dataset on which the clustering was applied was randomly generated by us, for the purpose of illustrating the different clustering techniques.

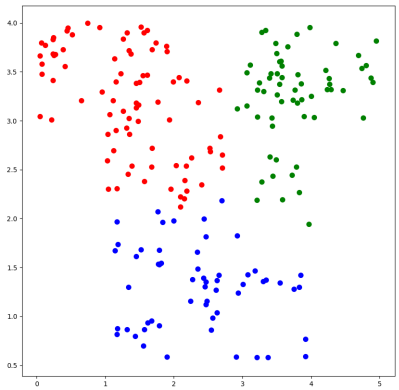


Figure 2.5 shows an example of what connectivity clustering is able to do, particularly agglomerative clustering with average linkage. Because we wanted a specific amount of clusters, the algorithm

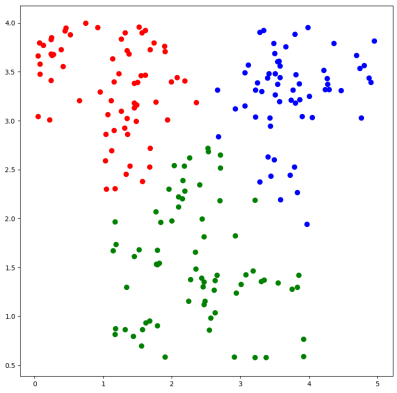
returned the three clusters it finds. However, more or less clusters are possible. In agglomerative clustering with average linkage, clusters are created for each datapoint and aggregate based on the average distance between two points. The distance from a point to a cluster is defined by the average distance from this point to all the points in that cluster.

The other types of agglomerative clustering are agglomerative clustering with single or maximum linkage, or with Ward's algorithm. Minimum Spanning Tree (MST) clustering is the same algorithm as agglomerative clustering with single linkage.

Centroid Clustering

The most well-known centroid clustering algorithm is K-means clustering. These iterative approaches designate centroids in the dataset and grow clusters around these picked centroids by absorbing neighbouring points. It does this iteratively to find the local optima, the best solution within the neighbour solution set.

Figure 2.6: A small example of K-means clustering. K-means clustering is a centroid-based algorithm that aims at creating k clusters from a given dataset. It starts by choosing k random points and grows clusters around these points, purely based on distance. The closest point to a chosen *centroid* point gets absorbed into a cluster with that point. The different colours in the plot indicate different clusters.

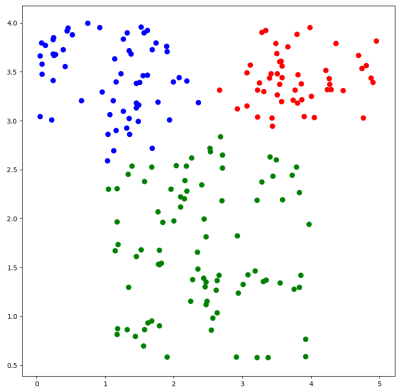


We show an example of centroid clustering in Figure 2.6. This is the result of K-means clustering applied to the same dataset as in Figure 2.5, and the colours represent the different clusters. Despite yielding similar results, there are differences that are worth noting. K-means starts with a set K and creates a solution from this number of clusters while agglomerative clustering starts from many clusters to reduce the number of clusters to a set amount.

Distribution Clustering

Distribution clustering interprets the data as a mixture of distributions. Instead of calculating distance between points, it measures how probable a point is to belong to a specific distribution, a specific cluster. In essence, it computes the distance between a point and a distribution, which returns a probability. It is assumed that different clusters have different distributions and this is how one can separate them.

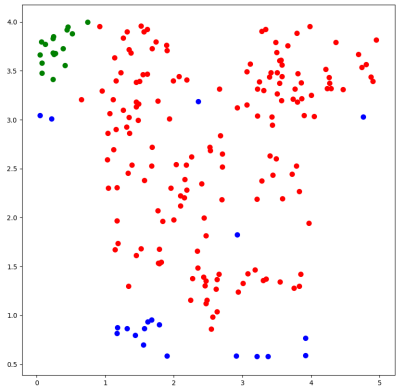
Figure 2.7: A small example of a distribution clustering algorithm, namely EM Clustering. In this example, the different colours represent the different clusters assigned by the algorithm. EM works by identifying a number of distributions inside the data and fits a point into a cluster by calculating the distance between this point and that distribution.



One example of this is Expectation-Maximisation (EM) clustering, which takes the data as a mixture of Gaussian distributions. It is assumed that datapoints generated emerge from a mixture of k Gaussian distributions with unknown parameters. Simply put, the algorithm attempts to represent the feature space as distinct normal distributions, which eventually respectively represent each cluster. Figure 2.7 shows an example of EM clustering, applied to, again, the same dataset as in the previous figures. We can see the EM clustering results differ slightly from the previous algorithms, though the clusters remain similar overall.

Density Clustering

Figure 2.8: An example for density clustering. Density-based clustering, e.g., Density-Based Spatial Clustering of Applications with Noise (DBSCAN), is used on this example. It works by distinguishing between areas of high and low density in the dataset. Again, a different colour means a different cluster. In this example, we see that the clusters do not really make sense. There is one large condensed cluster identified by the algorithm, but the other clusters contain but a few points and their positions are not self-explanatory.



Density models take the feature space as a contrast between areas of high and low densities. Points in areas of high density are assumed to belong to the same clusters, and areas of low density separate the different clusters. An example of a density clustering algorithm, namely Density-Based Spatial Clustering of Applications with Noise (DBSCAN), is shown in Figure 2.8.

2.1.3. N-Grams

In this research, N-grams are used to establish profiles for host network behaviours. The real reason behind this is explained in Chapter 4. Here, we explain the concept of n-grams and where they are regularly used.

An N-gram model is a probabilistic model used in text prediction [48]. N-grams are used in communication theory or computational linguistics for natural language processing. The goal of n-grams is to analyse successions of letters, establish a probabilistic model of letter occurrence, and predict what letter should be where next. The applications can either be for text recognition, text generation, or text categorisation.

An N-gram is an n-character sequence of a longer piece of text. The `n` parameter ranges from 1 to 5 consecutive letters. To build an N-gram model, one extracts all the possible n-grams in a piece of text, which we call the *training* text, and determines the probability of occurrence of each n-gram, based on their occurrence in the text. The probabilistic representation of the notion of N-gram for this example would be to predict the probability of occurrence of character c_i at position i based on $c_{i-(n-1)},...,x_{c-1}$, n being the length of the character sequence used. This probability is expressed by $P(c_i|c_{i-(n-1)},...,x_{c-1})$. This is the conditional probability of occurrence of character c_i after the previous sequence $c_{i-(n-1)},...,x_{c-1}$.

Take the following piece of text as example:

```
wherever people pay
```

For this sequence of 19 characters, it is possible to generate multiple n-grams. We choose 3-grams for this example. The possible 3-grams are whe, her, ere, rev, eve, ver, er, rp, pe, peo, eop, opl, ple, le, ep, pa, pay, including the spaces and stopping at the end of the sequence. The probabilistic model that can be built through these 3-grams holds for the first sequence the following probability of occurrence:

$$P(c_3|c_1,c_2) = P(e|wh)$$

These probabilities can be computed by taking the whole set of 3-grams. In this case, each sequence appears just once in the text, so they have the same probability of occurrence. Hence:

$$P(e|wh) = 1/17$$

Based on the generated model, it is possible to create new text by predicting where the different 3-sized sequences should appear. It also becomes possible to check if a new *test* text is predictable. This is the crucial idea we use for the building of network profiles, to identify profiles that are unexpected according to the learnt model. More on this is explained in Chapter 4.

2.2. Related Work

The core message to be retained from the **MQ** in 1.3 is that we are looking for a solution for clustering-based anomaly detection in an industrial context. The subsequent questions posed concern the specifics of clustering and the definition of worthwhile anomalies. We first need to define which features we can extract from the data at hand on which to perform clustering. Then, we look into the clustering approaches used in the literature in order to select candidate algorithms for our own clustering problem. After this, we focus on how the current research connects network incident detection to an anomaly detection problem.

2.2.1. Host Behaviour Identification

Host behaviour identification is the initial step towards anomaly detection to be able to learn a model for host behaviour and detect hosts that derive from that model. What is to be understood from the term *host behaviour* is the concept of type of network activity witnessed. A highly active host displays a high connectivity rate and sends large amounts of data to its peers. A less active host may make less

connections and send less data over the network. Since we are in an unsupervised context, we focus our literature study on unsupervised methods for behaviour identification. These methods all involve an initial step for feature extraction.

From the literature, we notice a different techniques applied but with quite similar sets of features, which seem to revolve around connectivity information relating to source and destination addresses, content of data transmitted, and connectivity range. The following subsections describe studies executed in this domain that are relevant to our goals.

Basic Heuristics

Looking at behaviour identification through statistical analysis, we find research that was done to create the Nfsight tool [10] for network awareness. This tool delivers, among other capabilities, the ability to differentiate between client and server activity by using heuristics based on features extracted from netflows.

Nfsight: netflow-based network awareness tool

Researchers from AT&T labs developed a tool in 2010 called Nfsight [10], which provides knowledge about how hosts use the network and how network events are related to each other. Nfsight uses a set of heuristics to differentiate between client and server. These heuristics are important to note, as they can be used for a basic network infrastructure mapping. Below is a subset of these heuristics, which can be used specifically for this project.

H.1 Port number - two port numbers associated with a bidirectional flow; the endpoint with the smaller port number is likely a server.

H.2 Port number with a threshold at 1024 - an endpoint with a port number lower than 1024 is likely a server.

H.4 Number of distinct end ports related to a given endpoint - if two or more different end port numbers are associated with the same endpoint, this endpoint is likely a server.

H.5 Number of distinct end IP addresses related to a given endpoint - identical to H.4 but counting end IP addresses instead.

H.6 Number of distinct end tuples (IP address, port number) related to a given endpoint - identical to H.4 but counting end tuples instead.

Because individual heuristics can provide inaccurate results due to the quantity of the netflow data, Nfsight uses statistical analysis techniques to combine the different heuristics and get a better estimate. In the end, they evaluated their approach to be quite accurate, with results attaining over 99% accuracy in differentiating between clients and servers in their dataset.

Clustering

The aforementioned research [10] delivers a set of features that have proven useful for later research. From this study stems a collection of more recent research in clustering of netflows to perform host behaviour clustering, for the sole purpose of network awareness or anomaly detection.

Unsupervised host behaviour classification from connection pattern

Looking at unsupervised learning approaches, which are relevant for our research, we touch upon a study of classification of host behaviours as a preliminary step toward traffic classification and anomaly detection in network communication [15]. This is quite relevant for the following steps, as part of our research is to isolate different host behaviours and perform anomaly detection on clustered devices, without relying on a priori knowledge of the network infrastructure. This paper's proposed solution is based on a 9-dimensional feature space and an MST clustering technique, which does not require any supervised learning. An overview of these features is presented below.

1. **Network connectivity.**

(a) The number of peers that is, the number of distinct destination IPs, involved with one IP address, expressed by:

$$\sum peer$$

(b) The number of source ports, divided by the number of peers, expressed by:

$$\frac{\sum port_s}{\sum peer}$$

(c) The number of destination ports, divided by the number of peers:

$$\frac{\sum port_d}{\sum peer}$$

2. **Connection dispersion.**

(d) The ratio of Shannon entropies of the second and fourth bytes of destination IPs to measure the distribution dispersion in the network, expressed by:

$$\frac{H(bytes_2)}{H(bytes_4)}$$

(e) The ratio of Shannon entropies of the third and fourth bytes of destination IPs, expressed by:

$$\frac{H(bytes_3)}{H(bytes_4)}$$

3. **Host traffic content.**

(f) The mean number of packets per flow, expressed by:

$$\frac{\sum packets}{N_{packets}}$$

(g) The percentage of small-size packets (≤ 144 bytes) in emitted traffic, expressed by:

$$\frac{\sum packets_{<=144}}{N_{packets}} \cdot 100$$

(h) The percentage of large-size packets (≥ 1392 bytes) in emitted traffic, expressed by:

$$\frac{\sum packets_{>=1392}}{N_{packets}} \cdot 100$$

(i) The entropy of the distribution of medium-size packets (between 144 and 1392 bytes) in emitted traffic:

$$H(packets_{>144} \cup packets_{<1392})$$

These nine features provide a 9-dimensional feature vector to accurately and efficiently characterise host behaviour. They are then also normalised and scaled between 0 and 1. Computing the ration of entropies of the second and fourth and the third and fourth bytes of destination IP addresses helps obtain insights on communication dispersion in networks and subnetworks. The paper does not give a motivation behind this choice of packet sizes 144 and 1392 bytes. However, we can assume that it is based on knowledge about regular network traffic and the common amplitudes of connections. Classification is based on an unsupervised MST clustering approach and is cross-validated against classical port-based classifiers or transport layer-based procedures.

Situational awareness of network system roles (SANSR)

The previous study and others that followed gave birth to the SANSR tool in 2017 [21]. The goal of this proof-of-concept tool is to allow security analysts to detect consequential changes in the network and be able to initiate incident response plans. The researchers base their feature selection on the previous

papers and solely focus on internal IP ranges. The tool performs categorisation against ground-truth vectors and different types of clustering combined with ground-truth vectors to cluster similarly behaving hosts together. The purpose of introducing ground-truth vectors in the analysis is to have a way to label clusters made, but this is not made explicit in this paper. Clusters are built according to these ground truth vectors, each representing a different cluster. New datapoints are compared against these ground truth vectors and get associated with the one which they resemble the closest using angular distance, presumably cosine distance - this is not specified. The researchers tested their system against a dataset containing roughly 80,756,737 netflows, which is notably significantly smaller than the Adyen dataset. The performance of the algorithms was significantly higher than for the previously mentioned study [15]. This tool is interesting for our current research because it places itself in a company context with real company data and provides corporate guidance for a project such as ours. We find it noteworthy that there is no verification of method described in the paper.

Clustering and profiling IP hosts based on traffic behaviour
This paper [22] comes from research executed in 2015 on the clustering and profiling of host behaviours, based on features inspired by the previously mentioned research [15]. As opposed to other research, this research executed the clustering algorithm on significantly more data, for period of over an hour as compared to 5 to 15 minutes in previous work. Longer periods of time are assumed to provide more stable behaviours. In this study, the researchers first reflect upon the correct choice of features for clustering to characterise and categorise host behaviours, adjusting the features from the previous paper [15] slightly. For clustering, they make use of the DBSCAN algorithm, which is a density-based clustering algorithm. Their second contribution is a way to distinguish IP addresses from one another, by extracting more significant IP nodes. They do this by looking at IP addresses that account for more than 10% and 20% of the flow data collected. The paper, however, does not concretely justify why this specific choice of IPs is made. The results presented are promising but are not further analysed. The paper does not focus on the true and false positive rates.

2.2.2. Anomaly Detection

Anomaly detection is complementary to host behaviour identification as it identifies hosts that are behaving like no other in the network. It is part of a crucial data analysis step in security monitoring because it is useful for identifying network intrusions or other types of security threats.

There are four major approaches for anomaly detection, namely classification, statistics, information theory, and clustering [34]. Classification is executed in a supervised domain, with a classifier that learns on training data. Since we are in an unsupervised environment, the classification approach does not apply. Furthermore, since the **MQ** stated in 1.3 focuses on clustering methods, we solely focus on anomaly detection through clustering methods.

Traffic anomaly detection using K-means clustering
A relatively early study on machine learning for netflow anomaly detection in 2007 [35] principally looks into using K-means clustering, a centroid-based clustering algorithm, for the purpose of detecting anomalies in netflow data. The feature set here is very simple; they take the total number of packets sent per host, the total number of bytes, and the distinct source and destination port pairs. The distance metric used on the clustering is Euclidean distance, which is a standard for K-means. The goal of this clustering application is to separate normal from anomalous clusters. The paper does not evaluate the correctness of the algorithm but does suggest improvements to be made, such as using Mahalanobis distance instead of Euclidean.

Self-configuring netflow anomaly detection using cluster density analysis
A group of researchers recently released a framework to ameliorate anomaly detection theory on netflow data in real time environments [18]. The approach developed is stated to be easily extensible and makes use of time-based correlations with historic data to generate a normalised view of activity on the network under study. This means that events over time in the network are correlated with one another in order to establish a baseline of activity, which makes it possible to run this process on streaming data. They are essentially creating time series from the input data, making a series of correlated points indexed in chronological order. They make use of polynomial regression techniques,

which are machine learning techniques for fitting and classification, on cluster densities to detect abnormal behaviour. This algorithm has potential but still requires to be tested in real-time environments. The paper mentions key anomalies to have been found in their dataset of netflows, however, there is no record of statistics of the validity of the results obtained, such as false positive versus true positive rates. Furthermore, the Adyen context may be too large for this algorithm to remain performant. The algorithm was tested on about 150,000,000 netflows, which is relatively small compared to the 240,000 netflows per minute of Adyen.

Comparing unsupervised learning approaches to detect network intrusion using NetFlow data
In the domain of network intrusion detection using netflow data, many techniques have been developed over the years. J. Zhang et al. [56] survey current research in unsupervised learning approaches to specifically detect network intrusion events, focusing on anomaly-based detection instead of signature-based or rule-based detection. Rule-based approaches identify intrusions using a comparative approach with predefined patterns of previous intrusion events. One disadvantage is that rule-based approaches do not necessarily automatically identify new classes of intrusions. Anomaly-based intrusion detection detects outliers deviating from a baseline of normal behaviour. One disadvantage is that anomaly-based approaches have a higher false positive rate than signature-based intrusion detection systems. This survey provides an evaluation of how well popular unsupervised anomaly detection approaches perform with regards to the detection of significantly different and rare anomalies. The conclusion of this research is that, while some methods in anomaly detection apparently worked better than others, the researchers were not subject-domain experts and their study had its own limitations in terms of model used. They note the need for an expert to categorise anomalies the evaluation of each approach.

A survey of network anomaly detection techniques
In a research conducted in 2016, a survey of network anomaly detection techniques is executed [34]. This paper summarises common network anomaly detection techniques in both supervised and unsupervised contexts. Concerning our topic, it focused on clustering methods used for host clustering based on regular and density-based clustering, and co-clustering, describing the advantages and shortcomings. Additionally, this paper criticised certain aspects of the common network datasets for their limitations. The key result of this research concerning our field of study is that three main assumptions are made when it comes to anomaly detection through outlier detection via clustering. These three assumptions are:

A.1 New data that does not fit well with clusters is considered to be anomalous without necessarily being malicious. This means that anomalous data is not necessarily a characteristic for anomalous behaviour. An anomalous datapoint may or may not be a sign of malicious activity.

A.2 When a cluster contains both normal and anomalous data, anomalous data is found far away from the cluster centroids. The assumption here is that more normal data is closer to the centroid than less normal to anomalous data.

A.3 Smaller and further away clusters can be considered anomalous. In a cluster crowd, this assumption says that further away clusters that are small and sparse may or may not be anomalous clusters, meaning clusters solely containing anomalous datapoints.

These assumptions are supposedly accepted as true in the network anomaly detection community, according to this paper.

2.2.3. Topics of Interest
The work related to our research helps us for our research by giving an indication as to what features can we use for network behaviour identification, what clustering techniques have been used and have proven to be effective, and how we can use this to perform anomaly detection. Additionally, it provides us with information on what has not been done before, at least what has not been published in the past.

Feature Extraction

To perform clustering on our own data, the feature set mentioned by G. Dewaele et al. [15] is quite promising for our research. Further work that has drawn inspiration from the latter have used identical or very similar feature sets in order to perform their classification [22, 27]. We then decide to opt for the features enumerated in the research conducted by G. Dewaele et al. for their simplicity and the fact that they are still used in the latest literature.

Clustering Techniques

From the literature, we notice research touching upon the four kinds of clustering techniques mentioned in Section 2.1.2. MST clustering, density-based clustering, k-means clustering - these are all being used in current research, with several adjustments each time. Because no clear clustering algorithm outperformed the other, we decide to test their performance for ourselves. We take the four types of clustering algorithm and test their workings on the Adyen network metadata. These clustering methods are sometimes used in the literature to incite a distinction between client and servers. However, in real-world networks, this difference in behaviour is ambiguous as polling servers can display client-like behaviours. The interpretation of these two different categorisations is erroneous and not necessarily reliable.

Anomaly Detection

Complex models as described in [18] seem to render the problem more complex than what it is. As a start proof of concept, we retain the knowledge gained from the survey of anomaly detection techniques for network monitoring [34], because this paper provides us with an overview of the latest research in this particular domain. Furthermore, the large context of Adyen pushes us to opt for a strategy that is both time-efficient and effective.

Cluster Evolution

The time-based analysis described in [18] inspires us to look into the idea of keeping a time-based record of netflow clustering. Keeping a record of inter-cluster evolution per host in the network is something that we have not encountered in the literature in network anomaly detection. In general machine learning research, time-based cluster evolution analyses is applied to identify similar clusters and behaviour patterns, based on a centroid movement analysis over time [6]. Here, we specifically want to analyse the cluster evolution relative to the hosts themselves, not their clusters. We have the ability to cluster hosts together according to a choice of features. We can then build a record over time of clusters to which hosts have been assigned. By performing host-relative inter-cluster evolution analysis, we want to uncover new ways to detect anomalous hosts in the network, using the network metadata provided.

3

Methodology

The latest research provides solutions for network anomaly detection that make use of various machine learning techniques in both supervised and unsupervised domains. We are here in an unsupervised environment, which requires us to veer towards methods that can be applied to unlabelled data. Clustering-based methods have proven their accuracy and efficiency in current research. While they still deliver a noticeable amount of false positive in general, they seem to be the best we can do.

The main research goal is to deliver a solution for network anomaly detection, which works effectively in the large-scale Adyen context with the network infrastructure at hand, and can be used to justify a call to action. To fulfil this goal, we design and implement a proof of concept for network behaviour profiling, which is made specifically to identify and record behaviours. The methodology to develop this proof of concept is divided into three stages: the first phase concerns the design of the behaviour profiling pipeline, the second phase describes the productisation of the solution according to Adyen requirements, the third phase describes the process of evaluation of the proof of concept. The following sections explain each stage of the methodology in detail.

3.1. Dataset

Adyen collects metadata of internal network connectivity, namely netflows. A simplification of the specifics of the data collection is shown in Chapter 1 in Figure 1.1.

Figure 3.1: A sample of netflow entries, how they are stored in Adyen clusters. The **timestamp** is date and time when the flow was collected. The **host** is the firewall where the flow was collected. The **source IP** is where the conversation started. The **destination IP** is the destination of the conversation. The **protocol** is the protocol used, either TCP, UDP, or ICMP. **Source bytes** shows the number of bytes sent from the source and **destination bytes** the number of bytes sent from the destination.**Source packets** and **destination packets** represent the number of packets sent respectively from the source and back from the destination. The **state** of the conversation is either ongoing (NEW) or finished (DESTROY). Finally, the **port** tuple is the port used at the source and the port used at the destination.

Time	host	ip	ip	proto	bytes	bytes	packets	packets	state	port
11/01 09:05:59				udp	69	123	1	1	DESTROY	39529,53
11/01 09:05:59				udp	69	123	1	1	DESTROY	39811,53
11/01 09:05:59				tcp	22393	11462	36	25	DESTROY	58379,8080
11/01 09:05:59				tcp	611	3032	7	6	DESTROY	53853,80
11/01 09:05:59				tcp	611	2980	7	5	DESTROY	53854,80
11/01 09:05:59				udp	61	177	1	1	DESTROY	27432,53
11/01 09:05:59				udp	63	127	1	1	DESTROY	28219,53

Netflows are stored onto an Elasticsearch [16] cluster and can be accessed codewise via the Python Elasticsearch client [41]. Figure 3.1 shows a sample of netflow entries in the database. The timestamps, host, source and destination IPs are anonymised in this paper. They are simplified in such a way that request and response netflows become abstracted as one bidirectional connection entry in the database. On average, Adyen collects approximately 240,000 netflow entries per minute every day, and this number is increasing.

When segmenting the provided data to create our own dataset for our research, we focus on entries of the transport layer. The transport layer records packets and bytes transferred, which we need for our analysis. The reason is explained in the further sections when choosing features for analysis. Because the area of interest is insider threat detection and the protection of Adyen employees, we decide to extract data about *client* devices themselves. Employee laptops are, by design, configured on a static IP subnet range, with some server on other subnet ranges. We decide to take the entire site range for our analysis. However, our entire analysis can be applied to another range; we just want to include Adyen employees in our analysis. In the end, these employees are all operating computer nodes, and a node remains a node like any other device in the network.

Adyen netflow logs are bidirectional connections, meaning the logging concatenates requests and corresponding responses together. However, for this study, we interpret netflows as unidirectional, because we are picking a subset of the dataset that does not hold extensive information on responding hosts. So, analysing responses of hosts may be flawed because we do not have their entire activity. Note that not all connections coming from hosts are recorded. There needs to be an actual connection made to log that connection, meaning any failed connection is not logged. This can pose problems for the detection of port scans, for example, as executing a port scan also results into a large portion of failed connections. An alternative logging source contains failed connections but this was not made available for this project.

3.2. Proof of Concept

We are dealing with the development of a proof of concept for a network behaviour profiling tool that is to act on the available netflow data. The purpose is to essentially identify behaviours that do not fit with the general trend of the crowd or previously recorded profiles for a specific host. The term *crowd* refers to the large populations of similar hosts inside clusters. The proof of concept itself requires being divided into several execution goals. Based on the literature review in Section 2, we specify the following items that make up a pipeline for the network behaviour profiling and anomaly detection system:

1. Identify distinct classes of behaviours inside the host crowd.
2. Extract information on behaviour change per host.
3. Build individual profiles according to the extracted behaviour change information.
4. Detect anomalous behaviour with regards to the crowd **and** a host's profile.

The first step requires the use of a behaviour identification technique on the network metadata provided to identify distinct classes of behaviours. Since we are in an unsupervised context as the data we have is not labelled, we choose to opt for clustering algorithms to extract distinct patterns of behaviour inside the network. The related work in Section 2 describes multiple instances where clustering has been used to perform host behaviour profiling and/or anomaly detection specifically on netflows, in unsupervised environments.

Once concrete distinct clusters of behaviour are formed, we can begin the second step and observe host movements across clusters. Some hosts may display a constant behaviour and remain in the same cluster over time, while some may move to another cluster from time to time. By recording switches of clusters, we can establish a profile of host behaviour change, which takes us to step 3.

Traditional network anomaly detection methods focuses on distances between hosts and centroids, or relative positions of hosts in space [34]. We take this further and build a *time-based record of inter-cluster movement per host*, essentially how hosts move between clusters over time. Behaviour changes relative to one host are recorded as a sequence of corresponding clusters over time.

We take on the fourth step and detect behaviours that do not match the patterns observed or barely fit the clustering model. By combining the approaches in step 2 and 3, our goal is to uncover anomalies that may not be identified through the latest techniques and reduce the number of false positives

overall. The set goal is to bring time-based inter-cluster movement into the equation to provide more information to make a substantiated decision regarding whether or not a host is behaving maliciously.

3.3. Implementation

"At the end of the day, I want a set of IP addresses that require attention, with enough information justifying the need for a call to action."

— Peter Cooper, Information Security Officer at Adyen

The proof of concept described above in Section 3.2 is intentionally not production-ready but does meet the functional requirements set by Adyen. A system that can be used by Adyen requires a further development phase to turn the proof of concept into a usable component of the Adyen monitoring environment.

The development of a prototype at the industry level comes with an implementation phase that unites the different pieces of the implemented proof of concept into one shippable product. This is where this research focuses on the *engineering* of a piece of software. The product is expected to act as a basis for network anomaly detection in Adyen. This system must still fulfil requirements set by the company, which indicate what the end result of the system should be, how it can be used, and to what extent it can be used.

Figure 3.2: The system's pipeline. We are looking at a machine learning problem for host behaviour identification, so the initial `begin` state is at the behaviour clustering module, which extracts information from the available database and identifies the different behaviours inside the network logs. Based on the clusters assigned to each host, a time-based record of inter-cluster activity is built to represent individual host profiles. The temporal aspect of this next step is illustrated by the hourglass symbol. The final step is the anomaly detection engine that acts both on clustering results and individual profiles and takes us to the `end` state.

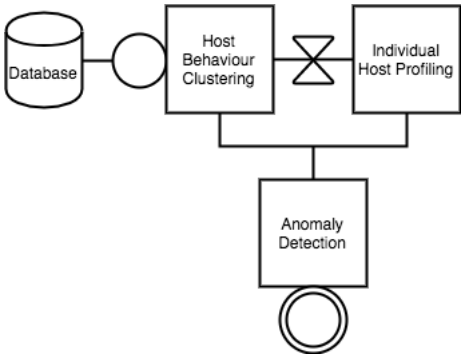


Figure 3.2 illustrates the expected prototype product to be delivered to Adyen. Netflow data is extracted from the *Database* and is analysed by the *Host Behaviour Clustering* process. Clusters are made and a time-based record is built over time for each host. An *Anomaly Detection* phase takes place using the information from *both* the clustering and profiling processes.

Keeping in mind what P. Cooper said, we can set our primary requirement to the eventual delivery of anomalous IP addresses. Because we are talking about justifications of calls to action when anomalies are detected, we need to think about how to present such evidence to a user of the system. A user of the system would be, for example, a member of the Adyen security team.

We primarily need to establish what information from our analysis that can substantiate a call to action is relevant to show. Before the system can be run, it requires to be run on a specifically chosen dataset, to build clusters and profiles. With that being stated, the following decision concerning the end result is made:

The system displays a list of IP addresses of hosts that it has found to be anomalous. To explain the choice of IP addresses, the system shows where specifically the anomaly was detected; whether it is with regards to the clustered crowd or the host's own behaviour profile, or both. Additionally, the record of the host's profile is given, along with an explanation of what the clusters mean in terms of network activity. The user can then view the same profile records for a host, for the previously recorded time frame, in order to view whether this particular host has displayed this behaviour before. The user can then make their own decision on whether to raise a call to action with the help of the information given by our proof of concept.

3.4. Evaluation

Our research goal is to detect anomalies inside the Adyen network. More specifically, anomalies that characterise a malicious security incident. We want to achieve this by building network behaviour profiles and, therewith, identify profiles that do not fit the accepted model. However, not all anomalies are relevant. There are in fact but a few anomalies that would actually raise a call to action.

To assess the performance of the designed proof of concept, we describe malicious scenarios that Adyen wants to detect. By playing out these scenarios and, therefore, generating enough malicious data samples, we can test our system.

Out of the vast threat landscape that surrounds the payment industry and actors that handle large chunks of highly sensitive data, there are three specific scenarios that make up the basic *must-haves* of this project:

1. Internal port scans
2. Data exfiltration incidents
3. Malware infections

The performance is tested by computing the precision and recall of our detection system with regards to each scenario. The specifics of the scenarios concerning their nature and the synthetic data generation are mentioned later in Chapter 7. We test our system on a number of observations, each being either a regular or malicious self-generated sample. The precision is the ratio of correctly detected scenarios, or true positives, n_{TP}, to the number of detected anomalies $n_{detected}$, expressed by the following formula:

$$P = \frac{n_{TP}}{n_{detected}}$$

Computing the precision of the system corresponds to finding out of all detected scenarios, how many scenarios are actually malicious.

The recall is the ratio of correctly detected scenarios $n_{correct}$ to the total number of scenarios that should be detected $N_{malicious}$:

$$R = \frac{n_{TP}}{N_{malicious}}$$

The recall of a system indicates how many malicious scenarios out of all malicious scenarios it is able to detect.

Additionally, we take into account the false positive rate, which is the ratio of the number of detected scenarios $n_{detected}$ minus the number of true positives n_{TP} to the total number of observations N minus the total number of malicious scenarios $N_{malicious}$:

$$FP = \frac{n_{detected} - n_{TP}}{N - N_{malicious}}$$

The false positive formula takes the ratio of total number of detected scenarios that are incorrect to the total number of non-malicious observations.

Precision, recall, and false positive ratio values give insights on the number of true positives, false positives, and false negatives, which Adyen deems to have the most business value. With these, we draw conclusions with regards to the performance of our system.

4

Lightweight NetFlow Clustering

Providing Adyen with a solution to perform anomaly detection on the internal network is our main goal, and to achieve this goal, we develop the proof of concept as described in Chapter 3. In this chapter, we describe in detail the approach of the proof of concept pipeline for behavioural identification and clustering.

We execute the first stage of the proof of concept that is illustrated in Figure 3.2, i.e. the identification of distinct classes of behaviours a crowd of hosts. We select a specific portion of the data on which to *train* the building of clusters. Section 4.1 describes the process of choosing features for behaviour identification. Section 4.2 describes how these features are normalised. Section 4.3 presents the entire clustering phase, with a section on feature extraction and normalisation. To decide on which clustering method to use, we test our clustering with different algorithms and see what results make the most sense. This is also explained in the latter section. Furthermore, in Section 4.4, we discuss specifics about the clustering and the results obtained.

Throughout the lightweight netflow clustering phase, we make use of Python and scikit-learn [40] to build our proof of concept. The implementation specifics, from an engineering perspective, are developed in Chapter 6.

Our research takes place in an unsupervised context, so we operate with unsupervised classification techniques to make sense of the data we have. Looking at the survey on network anomaly detection techniques from [34] in 2016, we opt for a clustering solution. Clustering refers to a set of unsupervised learning algorithms that act on unlabelled data to split it into *clusters* of seemingly similar data, based on predefined features. Multiple clustering techniques have been used in the past to perform network anomaly detection, such as centroid-based or density-based clustering, as mentioned in Chapter 2. In our research, we focus on four essentially different clustering approaches, which are described in detail in the following sections. It is argued that, since we do not exactly have an idea as to what the clusters may look like, it is better to use clustering methods that can identify non-convex clusters, such as MST clustering [15]. Here, we pick both convex and non-convex clustering methods, because we see no reason why we should choose convex over non-convex, for the exact reason that we do not exactly know what clusters are supposed to look like.

4.1. Choice of Features

In order to cluster, we need to have a set of features on which to cluster. The features we picked are based on previous research [15]. Research that has stemmed from this previous research have used identical or very similar feature sets in order to perform their classification [22, 27]. Based on the research from [15], which came up with a set of features that gave satisfactory results overall, we extract the following features per host in our dataset. The term *feature extraction* used here is to describe the choice of features and should not be confused with the machine learning term, which is the transformation of data in high-dimensional space to fewer dimensions.

4.1.1. Network Connectivity

(i) Number of peers
A host's number of peers is the number of distinct destination IP addresses involved in communications with that particular host.

(ii) Number of source ports per peer
This rough estimate is the total number of source ports from one host divided by its total number of peers.

(iii) Number of destination ports per peer
This rough estimate is the total number of destination ports from one host divided by its total number of peers.

4.1.2. Connection Dispersion

(iv) Ratio of Shannon entropies: second and fourth bytes of destination IP addresses
Computing the ratio of entropies of the second and fourth bytes of destination IP addresses allows us to obtain a value that characterises communication dispersion in a typical enterprise or large network. In standard IPv4, the first and second bytes of IP addresses generally correspond to locations or organisations managing IP addresses, while the fourth byte nearly always indicates variations of hosts inside the same subnetwork. High values of entropy of the second byte but low values of the fourth byte of destination IP addresses show hosts that are communicating with very different entities but specific subnets. The inverse pattern would show hosts communicating with few entities but with many hosts in the subnetwork, which, if detected, would be a typical pattern for scanning behaviour [15].

(v) Ratio of Shannon entropies: third and fourth bytes of destination IP addresses
Computing the Shannon entropy of the third byte of destination IP addresses show communication dispersion inside a specific subnetwork often inside the same organisation. Measuring the ratio of entropies of the third and fourth byte of destination IP addresses associated to a specific host can show a more specific dispersion angle for that host. It shows how a host communicates inside the organisation's network. High entropy values of the third byte but low values of the fourth byte indicate that just a small portion of the subnet is visited.

4.1.3. Host Traffic Content

(vi) Mean number of packets per flow
We extract, for one host, each number of packets recorded and average that number by the number of entries for that host, meaning the total number of communications.

(vii) Percentage of small-sized packets
We calculate the percentage of packets ≤ 144 bytes in emitted traffic.

(viii) Percentage of large-sized packets
We calculate the percentage of packets ≥ 1392 bytes in emitted traffic.

(ix) Entropy of medium-sized packets
The entropy of medium-sized packets (between 145 and 1391) bytes in emitted traffic tells us how many different types of mid-sized connections we have for one particular host. High entropy in the medium-sized packet distribution tends to suggest web traffic, while low entropy may indicate traffic with fixed-packet sizes, such as P2P traffic [15].

4.2. Feature Normalisation

Some clustering or visualisation methods rely on distance between features to establish dissimilarity. However, the features we have are not yet on the same scale. To accomplish this, we normalise each individual feature such that all values are between 0 and 1. After each feature value is normalised for each host in our dataset, we store all values onto a Python Pandas dataframe [38]. A Pandas dataframe can be interpreted like a database. Each row is indexed and represents a host with its IP address and its feature vector. The first column in the dataframe represents the host IP addresses, and the rest of the columns are the values per host of each feature. The feature columns are ordered in the way we enumerate each extracted feature in the previous section.

4.2.1. Features: (i), (ii), (iii), (vi)

The researchers in [15] normalise this subset of features according to a non-linear transform to balance out the feature set. The transform is as follows:

$$F_n : f_n = (2/\pi)\arctan(F_n, R_n)$$

F_n represents the normalised value for feature n. R_n is a reference parameter specific to a feature f_n. For feature (i), R_i is set to the total average number of peers. R_{ii} and R_{iii} are both set to the respective, not normalised, number of peers, and feature (vi) is scaled by $R_{vi} = 100$.

4.2.2. Features: (iv), (v), (ix)

It is stated that feature (iv) and (v) are naturally normalised, as they are ratios [15]. However, empirically, some values we obtain are greater than one. This is logical: if the denominator is smaller than the numerator, the result will always be greater than 1. The formulas for feature (iv) and (v) shown in Section 2.2 are:

$$\frac{H(bytes_2)}{H(bytes_4)}$$

$$\frac{H(bytes_3)}{H(bytes_4)}$$

There are cases where $H(bytes_4) \leq H(bytes_2)$ or $H(bytes_4) \leq H(bytes_3)$. In this case, the ratios would be ≥ 1. To normalise these two features such that they scale between 0 and 1, we use a simple normalisation method:

$$x_{normalised} = \frac{x - min(x)}{max(x) - min(x)}$$

This formula subtracts the minimum value of the entire set of values for a feature from the current value and divides this by the maximum value minus the minimum value. The result is a normalised value between 0 and 1. This normalisation method is also used on feature (ix). The [15] paper states that this value is also naturally normalised, from the Shannon entropy formula. However, we observe values that are higher than 1, because we use a base 2 logarithm, which is common standard.

4.2.3. Features: (vii), (viii)

Both these features are percentages, so dividing values by 100 gets us values between 0 and 1.

4.3. Clustering

To identify the different classes of behaviours in the crowd of hosts that make up our dataset, we apply clustering. Clustering provides a way to distinguish between types of behaviours. Based on the literature, we decide to test different clustering algorithms on our data and pick the one that works best.

4.3.1. Training and Test Sets

To build clusters, we need to select a specific training set from our data, on which the clustering is applied. Together with the Adyen security team, we decide on the selection of a day of data, divided into 30-minute chunks. This is because we want to record every 30-minute long activity on this day. This way, we can have a seemingly accurate summary of a regular day at Adyen. As we want to record every single 30-minute behaviour for each host throughout the day, these 30-minute iterations are spliced at the end in order to have one individual feature set on which to train clusters. We select a random day of data, namely the **1st of November 2017 at 00:00 CET until the 2nd at 00:00 CET**. Please note that, for the clustering execution, we follow the ethical guidelines suggested by the TU Delft Human Research Ethics Committee and purposefully ignore host IP addresses.

Once clusters are built based on our selected training set, we cluster a new dataset according to the learnt model. This test set is constructed by selecting a random 30-minute chunk of data on the day that follows the day of our training set. The resulting test set is a 30-minute long record of activity on **November 2 from 10:30 until 11:00**.

4.3.2. *Clusterability*

First and foremost, before we start clustering, we need to assess that our data is *clusterable*. A way that seems natural for us to accomplish this is to calculate the variance in each feature column, to see how spread out our sample space is. From the variance σ^2, we can deduce the standard deviation σ, which expresses by how much feature values of one feature column differ from the mean value for the entire column. Table 4.1 shows the values obtained after computing the variance and deducing the standard deviation of each feature column in our training set.

Table 4.1: Variance and Standard Deviation values per feature vector of the training set.

Feature	Variance	Std. Deviation
(i)	0.019	0.136
(ii)	0.020	0.141
(iii)	0.010	0.099
(iv)	0.028	0.170
(v)	0.032	0.180
(vi)	0.032	0.179
(vii)	0.042	0.206
(viii)	0.036	0.190
(ix)	0.057	0.239

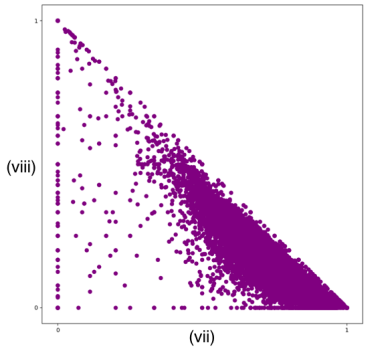
At first glance, the variance values are not that alarmingly high. However, since our entire feature space is scaled between 0 and 1 and the standard deviation values range from 0.099 to 0.23, these values show that most values in each feature vector range from $1/10^{th}$ to $1/5^{th}$ away from the mean. These numbers tell us that values are not all concentrated at the mean and are spread across sufficiently to be able to distinguish clusters, regardless of how similar the clusters may be.

4.3.3. Feature-by-Feature Representation

In the previous section, we assess that our training dataset can indeed be clustered. We first visualise our training set feature-by-feature in order to see if we can make sense of our data prior to clustering, in a 2D feature space. Because we have 9-dimensional feature vectors, visualisation of the entire feature space is not possible with standard geometry. Additionally, visualising the data feature by feature can help us uncover correlations among the data that have not been mentioned in [15].

Figure 4.1 shows the scatter plot of feature vii on the x-axis and feature viii on the y-axis. This is the sole feature-by-feature representation that shows meaningful results. The descending diagonal of points indicates a clear negative correlation tendency. However, there are still many datapoints that are positioned away from the diagonal and the two features do not account for the total possible packet sizes. For these reasons, we decide that this correlation is not significant enough to conclude anything

Figure 4.1: Sample of the feature-by-feature representation. On the x-axis, feature vii, the percentage of packets ≤ 144 bytes. On the y-axis, feature viii, the percentage of packets ≥ 1392 bytes. A negative correlation can definitely be noticed, which is logical because the percentage of large packets is directly related to the percentage of small packets. If there is a high percentage of small packets, there is certainly a low percentage of high packets. However, because there are still many datapoints away from the obvious diagonal, these points should still be taken into account. We decide to keep both these features despite the high correlation.



notable meaningful.

Despite figure 4.1 being the only plot with interesting results, the other feature-by-feature representations are still worth mentioning. Appendix A contains the rest of these representations. The positive result from this visualisation is that all our features are uncorrelated. Correlated features mean some require to be discarded, which is not the case here.

4.3.4. Clustering Algorithms

Out of the multiple clustering algorithms we can choose, we select algorithms of types of clustering that have been proven suitable for network anomaly detection in the literature [10, 15, 29, 34]. We start with scikit-learn's Python implementation of K-means clustering, as proposed in previous research [10], followed by the Gaussian Mixture approach. Gaussian Mixtures in scikit-learn implement the EM clustering algorithm, which is used in a previous study [29] to perform anomaly detection on IDS data. The survey of current network anomaly detection techniques mentions the use of density-based clustering that is suitable for unsupervised anomaly detection [34]. We make use of scikit-learn's density-based clustering method DBSCAN on our training data. Finally, we use the clustering method proposed in the study on which we base our features [15]. This study uses MST clustering, or agglomerative or hierarchical clustering with single linkage. There are more clustering techniques used in the literature, but we argue that with these four algorithms, we cover a great part of the types of clustering possible, and increase our chances of finding one that fits our data. The following sections describe the different clustering methods according to the scikit-learn clustering documentation [43].

We do not go deep into the mathematics and algorithms behind these clustering methods. Instead, we explain their workings practically and show some examples of the results that are to be expected. Each example shows each algorithm applied to the same set of points, which was randomly generated by us, specifically for this research.

K-means Clustering

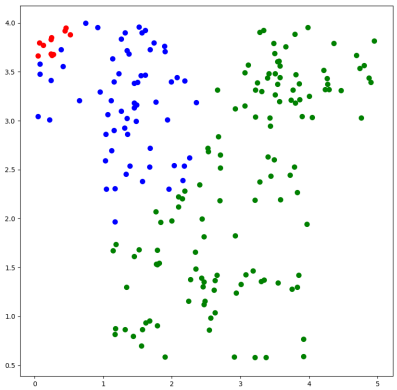
K-means clustering is a centroid-based algorithm that attempts to separate data into k groups of equal variance. K-means clusters are ellipsoidal in nature. The algorithm is executed in three steps: the first step selects k random centroids inside the feature space, the second step assigns to a centroid the nearest datapoint it can find, the third steps computes a new cluster centroid by averaging the values of the two datapoints that are assigned to one another. K-means uses Euclidean distance to find the nearest point. The second and third steps are executed repeatedly until all datapoints are assigned to a specific centroid, which means that all datapoints are clustered.

When a new dataset is introduced and the new datapoints need to be clustered based on the trained model, the K-means `predict` method computes the Euclidean distance between a new datapoint's feature vector and the predetermined centroids. That point is assigned to the nearest cluster centroid. An example of K-means clustering is shown previously in Chapter 2, Section 2.1, in Figure 2.6.

EM Clustering
EM clustering, in scikit-learn, is implemented through Gaussian Mixture models. The concept behind the Gaussian Mixture probabilistic model is that it is assumed that datapoints generated emerge from a mixture of k Gaussian distributions with unknown parameters. To initialise the parameters of the model, one common method is to use K-means to obtain centroids and covariance matrices. The other way is to initialise the parameters using a random centroid distribution. Each datapoint is scored with a probability value of that datapoint belonging to one of the distributions, to one of the clusters.

An example of EM clustering results on a small set of points is previously shown in Figure 2.7, and shows a representation of the effect of EM clustering on the small dataset, with **K-means** initialisation. In Figure 4.2, we show the results for EM clustering, this time with a random centroid initialisation step.

Figure 4.2: An example of EM clustering, this time with **random** initialisation. We can now see that the results are significantly different. The reason behind this difference is not looked into for this research. We are still dealing with the same set of points, but the fact that the centroid distributions are randomly initialised clearly influences the end result of the algorithm.



When a new dataset needs to be clustered according to the learnt model, the algorithm makes use of the Mahalanobis distance, to compute the distance between a new point and each Gaussian. It calculates how many σ away from the mean of each distribution that particular point is. The smallest Mahalanobis distance determines to which distribution the new point belong; therefore, to which cluster.

Density-based Clustering
DBSCAN was previously introduced in Chapter 2 in Section 2.1, through Figure 2.8. In this figure, we can see that the algorithm identified one large condensed cluster. As opposed to K-means clustering, we explore the possibility to make use of a type of density-based clustering, namely scikit-learn's DBSCAN algorithm. Density-based clustering interprets the data as high-density areas separated by areas of low density. DBSCAN introduces the concept of *core samples*, which are subsets of the data in areas of high density. A cluster is represented by multiple proximate core samples, in terms of a given distance metric, and non-core samples that are close to a core sample. Density-based clustering is a method that can identify non-convex shaped clusters. DBSCAN requires two parameters, namely ϵ and $min_{samples}$, which are respectively how close datapoints should be to fall into the same cluster and the minimum number of neighbours a point should have to be included in a cluster.

In order to cluster new data according to the learnt model, DBSCAN regenerates the same number of clusters on the new data and adapts the model. The clusters do not have centroids, as centroids are concepts for convex-shaped clusters. DBSCAN has the ability to identify non-convex clusters, which do not have centroids. So, computing the distance of new points to existing centroids cannot occur.

The DBSCAN algorithm does not start with a pre-set number of clusters. Instead, it creates clusters depending on the two aforementioned parameters. Determining ϵ involves the application of the k-nearest neighbours (kNN) algorithm on the training data, which helps create a distance plot that in its turn determines the best ϵ value [17]. The k in the kNN algorithm is the value for $min_{samples}$. This approach is explained further in 4.3.7.

Agglomerative Clustering

Agglomerative, or hierarchical clustering, groups a specific family of clustering algorithms that builds clusters by merging or splitting them successively. Clusters have a hierarchical, tree-like representation. The root of the tree is the unique cluster and, at the start, each datapoint is a leaf of the tree. The algorithm works its way up the tree by merging leaves together according to a given strategy and a given linkage distance metric. In scikit-learn, the possible linkage strategies are:

- **Ward** minimises the variance of the clusters being merged - only Euclidean distance can be used.
- **Complete** or maximum linkage - compares distances by taking the respective farthest points of two clusters.
- **Average** compares distances by taking the average distance of two clusters.

What is missing from the scikit-learn library is the *single linkage* strategy, which is the same as MST clustering, used in [15]. Single linkage compares distances by taking the closest points of two clusters, as opposed to complete linkage. It is argued that single and complete linkage both have shortcomings that average linkage seeks to balance out. Agglomerative clustering with single linkage can return clusters that are too spread out and not compact enough, while complete linkage finds compact clusters, but they are often not far enough apart [52]. For this specific reason, we opt for agglomerative clustering with average linkage and Ward-based clustering.

Figure 4.3: An example of agglomerative clustering with average linkage using the Manhattan distance metric. The same dataset as the previous clustering examples is used. Each colour represents a different cluster, assigned by the algorithm.

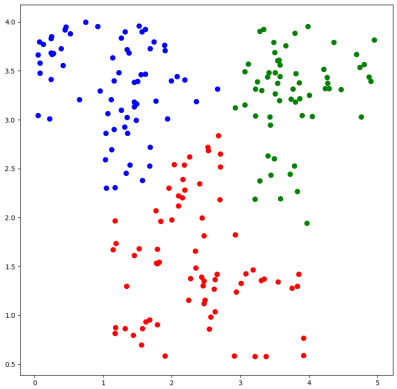


Figure 4.4: An example of agglomerative clustering with average linkage using the cosine distance metric. The same dataset as the previous clustering examples is used. Each colour represents a different cluster, assigned by the algorithm.

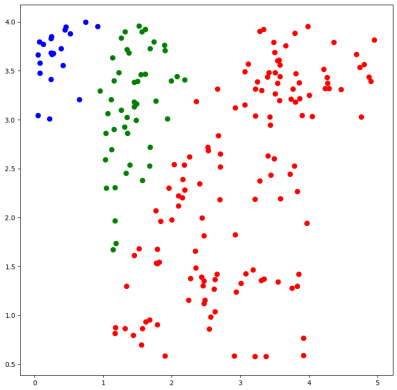
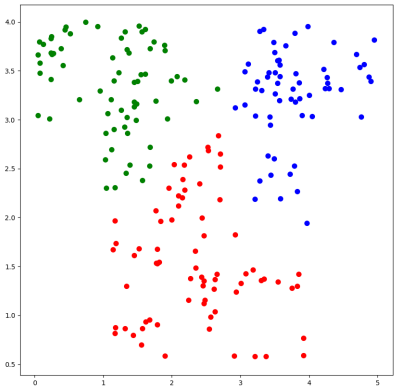


Figure 4.5: An example of agglomerative clustering using Ward's algorithm. Ward's algorithm is an optimisation method concerning the variance between clusters. It merges clusters according to the variance between them. The lower the variance, the more likely these clusters belong together. The higher the variance, the easier we can distinguish two clusters.



Just as DBSCAN clusters, these clusters can be non-convex and, therefore, do not have centroids. Clustering new data based on the learnt model in Agglomerative clustering occurs by refitting the model to the new data.

We already show an example of agglomerative clustering, with average linkage, in Figure 2.5 in Chapter 2, in Section 2.1. In this example, the distance metric used is Euclidean distance. Figure 4.3 and Figure 4.4 show the same algorithm, this time executed with the Manhattan and cosine distance metrics respectively. Finally, 4.5 shows the result of agglomerative clustering using Ward's algorithm. Each clustering method was applied to *the same set of datapoints* and it can clearly be seen that they deliver different results.

To recapitulate, the literature describes how these categories of clustering algorithms are being used in the state-of-the-art. We choose four different main types of clustering algorithms, namely K-means clustering, EM clustering with different parameters, DBSCAN, and agglomerative clustering with the different distance metrics described in Chapter 2.

4.3.5. Distance Metrics

The distance metrics we use for this research are explained in detail in Chapter 2, in Section 2.1. For K-means, the distance metric used is always Euclidean. For EM clustering, the default distance metric is Mahalanobis. DBSCAN can be run with different distance metrics. However, the most common one is Euclidean distance [43]. Since DBSCAN starts with a kNN phase, which also most commonly makes use of Euclidean distance [43], we decide to use Euclidean distance on the DBSCAN algorithm. For agglomerative clustering, it is possible to use different distance metrics, but not Mahalanobis distance in scikit-learn. In a 9-dimensional space, the concept of distance becomes intricate and complex to grasp. What a good metric is to use in this case is something that cannot be determined easily. Euclidean may be rendered almost useless and no longer be the appropriate distance to use, so we decide to try other metrics as well. For our experiment with agglomerative clustering, we use three distinct distance metrics: **Euclidean**, **Manhattan**, also known as City Block Distance, and **cosine** distance. This way, we can observe the differences in clustering results, depending on the metric used. We also use **Mahalanobis** distance, specifically for EM clustering.

4.3.6. The Optimal K

The next step is to derive the appropriate number of clusters from our training set. One simple statistical method to accomplish this is the *Elbow* method [24]. The Elbow method takes the total within-cluster sum of square (WSS), which represents the total intra-cluster variation, as a function of k. When clustering data, the statistical goal of clustering is to minimise the total WSS. An Elbow plot, in theory, shows when the WSS no longer significantly varies, regardless of the increased number of clusters. The *elbow* in the plot, which is the location of a bend in the curve, is generally considered to be the indicator of the appropriate number of clusters.

DBSCAN, on the other hand, is the only algorithm here that does not accept k as parameter. Instead, it automatically generates an optimal number of clusters, fitting the provided training data, according to ϵ and $min_{samples}$ values that represent the maximum distance between two samples for them to be considered in the same cluster and the minimum number of neighbouring points to make up a cluster [43]. We go deeper into finding the optimal ϵ and $min_{samples}$ in the subsequent section.

For each of the clustering method implemented, except for DBSCAN, we run the Elbow algorithm for $k \in [1,20]$.

Figure 4.6: Elbow plot of K-means clustering for $k \in [1,20]$. The elbow of the curve represents where the k value is most optimal, with regards to the total intra-cluster variation. We can say that this value is between 6 and 10, so we pick $k = 8$.

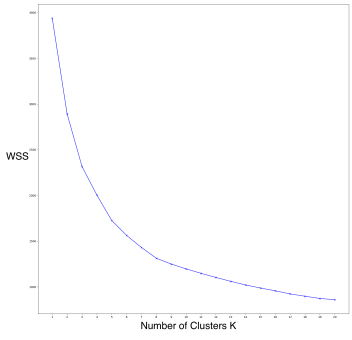
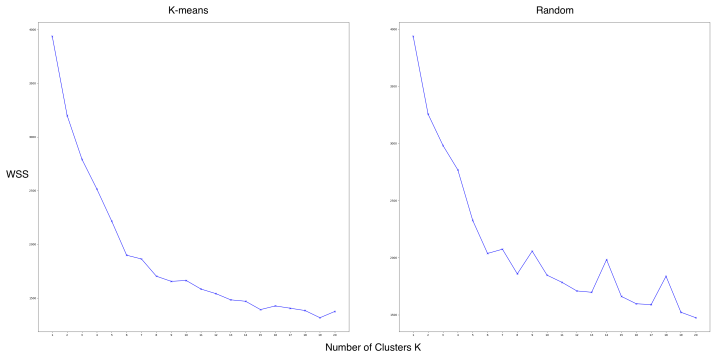


Figure 4.6 shows the Elbow plot for K-means clustering and the curve has an inverse exponential behaviour. Identifying the *elbow* is, of course, approximate and is probably not completely accurate, but it gives an indication that we can deem as *good enough*. In the K-means clustering case, we choose $k = 8$.

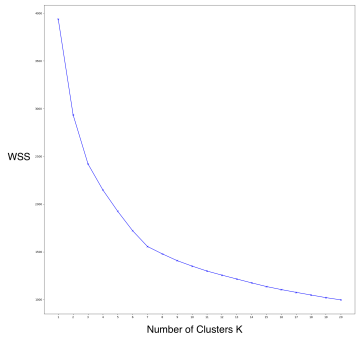
Figure 4.7 shows the Elbow plot for EM clustering, for both *k-means* and *random* parameters We pick

Figure 4.7: Elbow plot of EM clustering for $k \in [1,20]$. We choose $k = 6$ for K-means initialisation because it seems to be an appropriate value for k. The Elbow curve when randomly initialising the algorithm is quite unstable. However, the WSS significantly slows down after 6 clusters, so we choose $k = 6$ for this one as well.



$k = 6$ for EM with K-means initialisation technique despite the slight increase at $k = 10$, because it is the k that makes the most sense to us. Regarding the odd behaviour on the Elbow plot for random initialisation, we cannot exactly explain what is going here but we expect it to be an artefact of the algorithm. We settle with $k = 6$ for EM clustering with random initialisation as well.

Figure 4.8: Elbow plot of agglomerative clustering with Ward linkage for $k \in [1,20]$. The intra-cluster variation seems to relatively stabilise after $k = 7$, so we choose 7 for our k value.



For agglomerative clustering with Ward's algorithm, we heuristically pick $k = 7$. The curve slowly descends after $k = 7$, meaning the WSS no longer changes significantly. The Elbow plot can be found in Figure 4.8

Figure 4.10 shows the Elbow plots for agglomerative clustering with average linkage for each distance metric. Namely: Euclidean, Manhattan, and cosine distances. For Euclidean distance, we pick $k = 7$, because the curve then descends slower. We do notice a dip in the curve at $k = 15$, but we do not find this significant enough to choose another value for k. For Manhattan distance, we pick $k = 9$ and for cosine distance, we pick $k = 7$, where the curve stabilises for a while.

4.3.7. DBSCAN's Optimal ϵ and $min_{samples}$

The parameter ϵ can be determined by first applying the kNN algorithm to the training data. The k in kNN represents the $min_{samples}$ value. This k is determined by applying the Elbow method on K-means clustering [42], which we happen to do in the previous section. Therefore, the $min_{samples}$ parameter is equal to 8.

Once we have computed all distances in the data through the kNN algorithm, we sort the distances

Figure 4.9: Elbow plot of agglomerative clustering with Average linkage for $k \in [1,20]$. Despite these three plots being relatively unstable in some parts, we can still identify *elbows* in the different curves. We settle with $k = 7$ when using Euclidean distance, $k = 9$ when using Manhattan distance, and $k = 7$ again when using cosine distance.

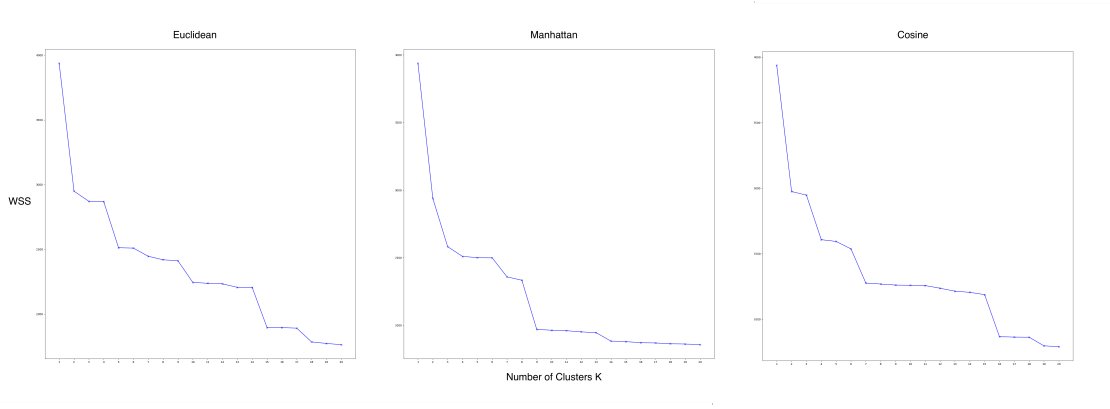
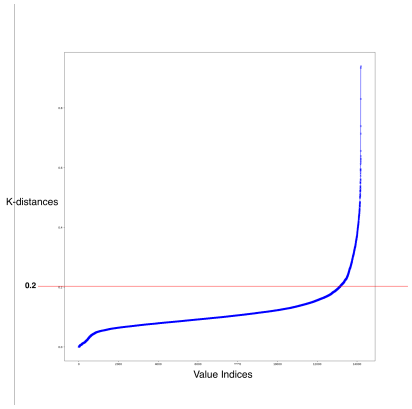


Figure 4.10: K-distance plot to determine DBSCAN's optimal ϵ. The distances obtained by the kNN algorithm applied to the training data are sorted in ascending order and plotted with their values on the y-axis and value indices on the x-axis. The red line indicates the beginning of an *elbow* and corresponds to the optimal ϵ, which can be found on the y-axis. This value is equal to 0.2.



obtained with $k = 8$ in an ascending order and plot them with the value indices on the x-axis and distances on the y-axis. The value for ϵ is determined by looking at the position when the curve takes an *elbow* shape, as for the Elbow method [17]. Here, we conclude that $\epsilon = 0.2$.

4.4. Results

By computing each clustering algorithm's optimal parameters, we are able to execute these algorithms on our training set. We use K-means clustering with $k = 8$, we use EM clustering with $k = 6$, for both initialisation algorithms, and Ward clustering with $k = 7$. For agglomerative clustering with average linkage, we use $k = 7$ for Euclidean distance, $k = 9$ for Manhattan distance, and $k = 7$ again for cosine distance. Finally, we use $\epsilon = 0.2$ and $min_{samples} = 9$ for the DBSCAN clustering algorithm. Table 4.2 summarises the experiments executed on our training set with each clustering algorithm and the corresponding parameters.

All K-means, EM, Ward, and agglomerative with average linkage methods give us results that appear to show different clustering. By just looking at the resulting clusters and their if-applicable centroids or content, we cannot conclude anything that indicates that our clusters are valid. However, DBSCAN clustering immediately returns a set of labels for our training set with a large amount of negative values ($\geq 40\%$). Negative values in DBSCAN are scikit-learn's way to express outliers, meaning values that do not fit in any clusters. Therefore, the data is too noisy for DBSCAN to handle. DBSCAN is not the right

Table 4.2: Summary of the experiments executed on the training dataset with each clustering method. This table groups the different distance metrics and parameters used per algorithm. K-means clustering does not come with a specific parameter, so the cell is left blank. The same goes for DBSCAN, which does not take a specific k to dictate the number of clusters.

Clustering Type	Distance Metric	Number of Clusters	Special Parameters
K-means	Euclidean	8	
EM	Mahalanobis	6	K-means Initialisation
EM	Mahalanobis	6	Random Initialisation
DBSCAN	Euclidean		$\epsilon = 0.2$ and $min_{samples} = 9$
Agglomerative	Euclidean	7	Average Linkage
Agglomerative	Manhattan	9	Average Linkage
Agglomerative	Cosine	7	Average Linkage
Agglomerative	Euclidean	7	Ward

approach to identify clusters according to our training set.

4.4.1. Cluster Validity

The concept of cluster validity is defined as *our clusters make sense to us humans*. This means that two hosts whose feature vectors are mathematically comparable should consequently be clustered together. Datapoints have intrinsic motivation whether or not they belong to the same cluster. Cluster validity is difficult to verify with no ground truth. We cannot simply validate whether this host belongs to that cluster because we do not have a priori knowledge about this host.

In order to understand the clusters made, we attempt to visualise the 9-dimensional datapoints of our training set as scatter plots in two dimensions. Dimensionality reduction is an important step here [51], and we can make use of Principal Component Analysis (PCA) [51] or t-Distributed Stochastic Neighbour Embedding (t-SNE) [3]. Because of the nature of our training set, this visualisation does not bring anything useful to the analysis of cluster validity. The datapoints of our training set are not represented as clear separate clusters in two dimensions. Two-dimensional scatter plot visualisation does not impact our research. However, we explain the entire process of visualisation in Appendix C.

There is one self-explanatory method that we can use to verify the validity of our clustering approaches, namely generating heatmaps out of our clustered data. Heatmaps are useful plots to visualise multidimensional feature vectors in two dimensions. We apply each clustering method, except for DBSCAN as we discard it before further analysis, to the randomly selected 30-minute chunk that is our test set. Then, for each clustering method, we sort the hosts according to their attributed cluster and create a 2-dimensional matrix where each row represents a host's feature vector. The 2-dimensional matrix is plotted and the value at a specific position $[i, j]$ is represented as a square of colour along the `hot` colour map provided by the Matplotlib framework. This colour map is actually a linear gradient starting from black at value 0, going through red and yellow until the white colour at value 1. If the clustering algorithm identifies the right clusters, this is visible through the heatmap, as similarly looking row vectors are clustered together and a clear distinction can be seen across clusters.

Figure 4.11 shows a piece of the heatmap of the clusters obtained through K-means clustering. Figure 4.12 shows for EM clustering, both with K-means and random parameter. Finally, Figure 4.13 shows for Agglomerative clustering, for Ward and Average linkage with the different distance metrics. Each piece of the heatmap respectively shows the difference between hosts of cluster G and hosts of cluster H for Figure 4.11, cluster E and cluster D for EM clustering with K-means parameter cluster C and cluster D for random parameter in Figure 4.12, and cluster A and cluster B for Ward clustering and for average linkage with Euclidean distance in Figure 4.13, cluster C and cluster D for average linage with Manhattan distance and cluster B and cluster C for average linkage with cosine distance also in Figure 4.13. Note that the cluster labels for each algorithm mean different clusters, since the algorithms all cluster differently. Cluster A for K-means clustering is not the same as cluster A for agglomerative clustering.

Figure 4.11: Heatmap sample for hosts clustered using K-means clustering. The sample shows hosts from cluster G, in green, and hosts from cluster H in blue. The colouring used follows a linear gradient starting from black at value 0, going through red at value 0.5, then towards yellow until white at value 1. It is noticeable that these hosts show very similar patterns. Distinguishing these clusters is not intuitively possible.

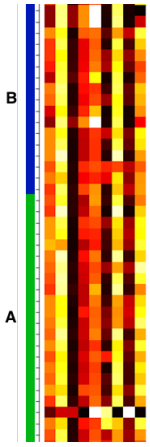
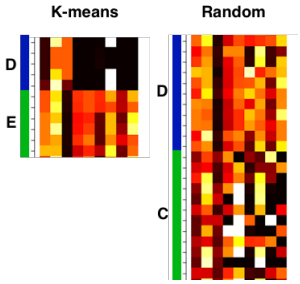


Figure 4.12: Heatmap samples for hosts clustered using EM clustering, with K-means and random initialisation. For K-means initialisation, the sample shows hosts from cluster E, in green, and hosts from cluster D in blue. The colouring used follows a linear gradient starting from black at value 0, going through red at value 0.5, then towards yellow until white at value 1. For random initialisation, the sample shows hosts from cluster C, in green, and hosts from cluster D in blue. This time, we can definitely notice a physical difference between clusters. Hosts in one cluster do not appear to have the same *heat* as hosts in the other cluster, for EM clustering in general, regardless of the initialisation parameter.



What can immediately be seen is that, except for EM clustering, the differentiation between clusters made by the clustering algorithms does not make sense. Both clusters look alike to the human eye. However, when it comes to EM clustering, especially with K-means initialisation, a clear distinction can be seen between cluster E and D. This method essentially shows us which clustering method works best on our data, for our problem. It is not yet clear precisely why EM clustering delivers subjectively better results. Because this clustering algorithm can make the clearest distinction between hosts, we pick **EM clustering with K-means parameter** to build the clusters for our problem. The entire heatmaps generated from our test set can be found in Appendix B. There, the differences in clustering ability can be seen on a larger scale.

4.4.2. Behaviour Classes

The six clusters we built on the training data with EM clustering and K-means initialisation represent a generalisation of the observable behaviours inside the Adyen network. Because we do not have any ground truth about the nature of network clusters on which to base our analysis so that we can label our clusters, we have to evaluate the centroid values for each cluster. Each cluster has a centroid with specific values for each feature. Table 4.3 shows the centroid values we obtain through clustering on our training data.

Figure 4.13: Heatmap samples for hosts clustered using agglomerative clustering with Ward and average linkage, with the three distance metrics we use. These samples show two different clusters each time, denoted by the distinct colour blue and green. The colouring used follows a linear gradient starting from black at value 0, going through red at value 0.5, then towards yellow until white at value 1. For Ward's algorithm and average linkage with the Euclidean distance parameter, we show samples of cluster A and B. For average linkage with Manhattan distance, we show samples of cluster C and D. For cosine distance, we show cluster B and C. We can clearly see that we do not have a proper way to distinguish these clusters with the naked eye. The clusters contain hosts that all look alike, or all look different, and no pattern can be seen.

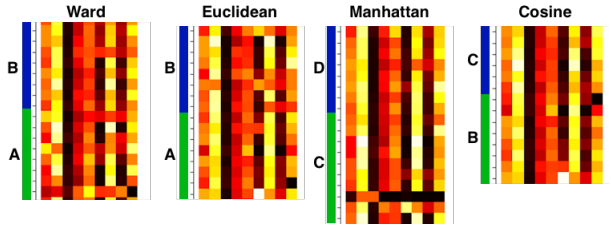


Table 4.3: The training data is clustered according to the EM clustering algorithm with a K-means initialisation step. Each cluster receives a cluster name, which in our case is a letter from the alphabet, and a centroid with a different value for each feature used for clustering. Note that the values are normalised and scald between 0 and 1.

Cluster	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)
A	0.35	0.78	0.16	0.47	0.80	0.02	0.85	0.04	0.28
B	0.44	0.80	0.08	0.35	0.61	0.02	0.85	0.05	0.55
C	0.25	0.57	0.29	0.28	0.46	0.31	0.19	0.65	0.10
D	0.14	0.81	0.42	0	0	0.01	0.93	0.00	0
E	0.53	0.73	0.06	0.35	0.52	0.27	0.66	0.21	0.61
F	0.54	0.84	0.05	0.36	0.51	0.10	0.80	0.13	0.69

We now proceed to label our clusters according to their centroid values, by making assumptions on the network behaviour observed, based on research from [15] and our own experience in computer networking. Note that each centroid value is normalised and scaled between 0 and 1.

- **Cluster A** The numbers indicate that cluster A has a high number of source ports and a relatively low number of peers and destination ports. This means that cluster A connects many times to specific servers for specific services. Furthermore, cluster A sends a high number of small packets. Cluster A could then be a cluster of hosts intensively performing very specific tasks. Judging by the high entropy value for dispersion in network subnets, we can say that cluster A contains **hosts that browse through a lot of specific subnets**.

- **Cluster B** The numbers for the centroid of cluster B are close to A's centroid. Cluster B also has a high rate of small-sized connections. However, it seems that hosts in cluster B connect to more servers than in A, but less dispersed in the network. This could be a cluster of **hosts performing a lot of different tasks inside the network**.

- **Cluster C** This cluster has relatively low numbers for most of the features, meaning there are not many peers, not many connections, and the connections are not so dispersed. What we do notice is that hosts in cluster C tend to make medium to large-sized connections, and the destination port count indicates multiple services being visited. This could be a cluster of **hosts uploading data to few specific machines**.

- **Cluster D** Hosts in cluster D make an obviously large amount of small connections to just one server. Both entropy values are 0, meaning hosts connect to just one server. It could be one internal proxy server, exposing Internet browsing behaviour, or more generally, a cluster of hosts performing just one task. The value for the percentage of large packets is extremely small that it is almost 0, meaning **hosts make significantly few large connections**.

- **Cluster E** The relatively high amounts of peers and source ports indicate that hosts inside cluster E communicate with many servers for many services. The numbers that concern packet size show that the connections made vary in size. There is about the same amount of small-sized and

medium-sized packets being sent, and some of large sizes. Cluster E could be a cluster of **hosts performing many different tasks**, more variate than in cluster B, from uploads, downloads to updates and browsing behaviour.

- **Cluster F** Here is the same story as for cluster E, but the connections are much smaller on average. This could be a cluster of **hosts displaying high activity in the network**, but with fewer uploads than in cluster E.

We can only speculate about the nature of the different clusters because we have no absolute ground truth. These speculations are based on our own knowledge and experience of computer networking and network behaviours as explained in [15]. It is possible to go through the individual netflow logs of each host and analyse their respective activity to establish true labels, but this task falls out of the scope of this research and would take a significant amount of time given the quantity of data we have. Therefore, we cannot yet give concrete labels to the clusters obtained through EM Clustering, only a short description of their respective behaviours.

5

Host Behaviour Profiling

We could halt here and perform standard anomaly detection on our unsupervised clusters. Instead, we introduce a technique to look for anomalies over time, inside network behaviour profiles. Each host in the network is clustered using EM clustering with a K-means initialisation step. The cluster assigned to each host is recorded and an inter-cluster profile builds over time.

The novel aspect of this research is the time-based analysis of host cluster evolution using N-grams. To our knowledge, this has not been done before in the domain of unsupervised network anomaly detection. Research in general machine learning mentioned in Chapter 2 looks into cluster evolution based on cluster centroids [6]. In this research, the concept is that we take a look at inter-cluster movement over time **relative to hosts**; the individual cluster-switching behaviour of hosts. It is suggested that, in theory, hosts display similar behaviours daily, or at least a behaviour that is not that different from the established baseline. The hypothesis here is that significant cluster change could be a suggestive indicator of compromise. At the end, we merge standard anomaly detection technique on clusters with the detection of anomalous profiles over time.

5.1. Building Profiles with N-Grams

From the training data, we derive each host's succession of clusters at 30-minute intervals and interpret this as a sequence of characters, each character corresponding to one cluster. The training sequence is then read into a 3-gram builder, which is an N-gram of size 3. A 3-gram is an extension of bigram, which is the equivalent of a Markov chain. As defined in Section 2.1, n-grams are a way to probabilistically model character prediction in a given text. Roughly put, a 3-gram takes a subsequence of three characters, such as `AAB`, and returns the probability of `B` occurring after two `A`s, based on a training 3-gram.

Each host receives its own 3-gram, which can be referred to as a *profile* for inter-cluster movement. Once the 3-grams are generated based on the training data, we obtain a probabilistic model of character occurrence, of cluster changes over time per host. We can now feed each 3-gram new sequences of characters and obtain the probability of occurrence per character in the new sequence.

Let the following sequence be a sample record of inter-cluster evolution for an arbitrary host.

$$[A, B, A, A, B, A]$$

In this sequence, there are two occurrences of sequence $[A, B, A]$, one occurrence of sequence $[B, A, A]$, and one occurrence of sequence $[A, A, B]$. The following probability model is established:

$$P(A|[A, B]) = \frac{1}{2}$$

$$P(A|[B, A]) = \frac{1}{4}$$

$$P(B|[A,A]) = \frac{1}{4}$$

Using this probability model, we can now compare a new sequence of characters against this learnt model. We choose to use 3-grams instead of bigrams because we expect, purely based on experience, that unexpected patterns of behaviour happen over a longer period of time than as expressed in bi-grams.

5.2. Unexpected Cluster Change

Because we have a model that can tell us whether or not a specific inter-cluster movement has less probability to occur, we can identify cluster changes that are unexpected. An arbitrary host theoretically has a regular pattern of clusters in which it sits throughout the day. If it suddenly switches to a cluster which it never visited before, it could mean that unusual activity is happening at that moment of the cluster switch. This cluster switch is to be detected by looking at the probability of occurrence of the new cluster, which is based on the trained 3-gram for that host. A high probability suggests the cluster was to be expected, while a low probability indicates an unexpected change.

Consider the example sequence above:

$$[A,B,A,A,B,A]$$

Let this sequence be used as inter-cluster evolution training profile for an arbitrary host. Now, let us say we introduce a new cluster change inside this sequence and we obtain:

$$[A,B,A,A,B,A,B]$$

To check the predictability of this newly introduced cluster change, we compare it against the learnt model. Here, it is the introduction of a B after a sequence $[B,A]$. According to the model:

$$P(A|[B,A]) = \frac{1}{4}$$

Whether we deem this to be an unexpected cluster change ultimately depends on the threshold we set.

5.3. Anomaly Detection

All the work done for this research thus far is to enable the possibility to detect anomalous behaviour with regards to the crowd **and** the hosts' own profile. There are two aspects of the proof-of-concept we build: a clustering engine for host behaviour identification and the building of individual host profile fingerprints to detect inter-cluster changes. When it comes to detecting anomalous behaviour, we need to perform some kind of anomaly detection on both branches of the system and *fuse* the results into a concrete incident alert. The motivation behind the *fusion* is that one result from one aspect may not be significant enough, but combining the two outcomes may give us stronger motivation to raise a call to action.

5.3.1. Outlier Detection

Anomaly detection, when it comes to clustering, amounts to outlier detection [34]. Hosts that are far away from their cluster's centroid, or that do not fit into any cluster, are flagged as anomalies. Clusters that are far away from a cluster crowd may also be considered anomalous. However, in our case, clusters are rather condensed together, so looking for faraway clusters does not make much sense for us.

To perform outlier detection, we take each cluster of hosts and compute the Mahalanobis distance between each host and their cluster's centroid. We pick the Mahalanobis metric because it is the metric used in EM clustering and because it has the advantage to work better in multi-dimensional space than other regular distance metrics [28]. The next step is to establish a threshold, from which we classify outliers. Any host further away from their cluster's centroid than that threshold is flagged as an outlier. The threshold is dynamically assigned a value, depending on the host-to-centroid distances in

their corresponding clusters, and changes depending on the cluster. For each cluster, we sort the host-to-centroid distances of hosts in that cluster and take the 90th percentile value as threshold value. We choose to dynamically compute a threshold for each cluster because it may as well be possible that some clusters are sparser than others. If we choose a fixed threshold, a sparse cluster could be filled with outliers according to our outlier detection because a lot of hosts would be relatively far from the centroid.

In our system, hosts are always clustered. This is why our outlier detection solely bases itself on host-to-centroid distances.

5.3.2. Cluster Unpredictability

When it comes to host behaviour change over time, anomaly detection amounts to detecting unpredictable cluster changes. We observe an individual host's inter-cluster movement and build a new sequence of clusters that we compare to the host's training 3-gram profile. The result obtained is a sequence of probabilities, each probability p_i corresponding to the probability of this host being put into that cluster at time t_i. Since the probabilities are normalised, the total sum in the sequence of probabilities is equal to 1.

Because we cannot determine easily what probability should be significant enough to be interpreted as an indicator of compromise, we, once again, dynamically assign a threshold value for anomaly detection. We need to isolate probabilities that are significantly lower than others. We accomplish this by taking the median in a sorted sequence of probabilities and flagging anything below half the value of the median as an anomaly. Substantially justifying this choice is difficult. The threshold was empirically selected, and we suspect it to be relevant nonetheless, especially when combined with the outlier detection.

5.3.3. Fusion

We combine outlier detection and cluster unpredictability determination by fusing the two computations. A host flagged as an outlier may be enough proof to instigate a call to action. A host put into a cluster that could not have easily been predicted may also be enough to raise a call to action. When a host is flagged as an outlier *and* is deemed to be assigned to a cluster that does not fit the known pattern, it is much stronger evidence of an indicator of compromise. There must be something truly abnormal at hand for both parts of the system to have raised an anomaly on this host in question. The goal here is to lower the false positive rate and the corresponding theory here is that the *fusion* step can help us achieve this.

6

Implementation of a Product: *Mapyen*

An important part of this research is the material contribution of a piece of software to Adyen. The research done up until now is joint together into one product. The proof of concept is divided into several execution items, from the clustering of network behaviour to the building of individual host profiles. Each piece of the proof of concept is hence assembled at the implementation phase.

The development of a product is done on the available Adyen network metadata and has the name *Mapyen*. Mapyen is implemented on a 3.1 GHz Intel Core i7-based MacBook Pro with 16 GB of RAM. In conformance with internal security policies, data analysis on the laptop, instead, is required to be executed on a QEMU-KVM virtual machine running on 3.8 GB of RAM with 2 GB of swap space, on a Red Hat Enterprise Linux 7.4 (x86_64) server, in a protected segment of the Adyen network.

6.1. Software Packages

Mapyen is implemented in Python 3.0, accompanied by several Python packages. As mentioned before, we make use of the Python Elasticsearch Client, version 1.10.0, to fetch the data from the Elasticsearch cluster. For the general data handling, we make use of the Python data analytics framework Pandas [38], version 0.20.3. Pandas is an open source data analysis library that can be used to efficiently store and manipulate data. There are many more statistical analysis capabilities provided by Pandas, but we do not use it that extensively in our project.

For the host behaviour clustering, we make use of the previously-mentioned effective machine learning and data analysis tool scikit-learn [40], version 0.19.0. Scikit learn is open source and can be found on the GitHub open source development platform. It was developed by a selection of researchers in the domains of machine learning, pattern recondition, and artificial intelligence. Additionally, we include NumPy, version 1.13.1, and SciPy, version 0.19.1, for matrix operations and scientific calculations. Finally, we use Matplotlib [33], version 2.0.2, for data visualisation, which is also an open source framework available on GitHub. Some packages we use may be from older versions. That is because we require everything to be compatible with one another. In other words, we need to downgrade some packages so they can work with other packages that only work with older versions.

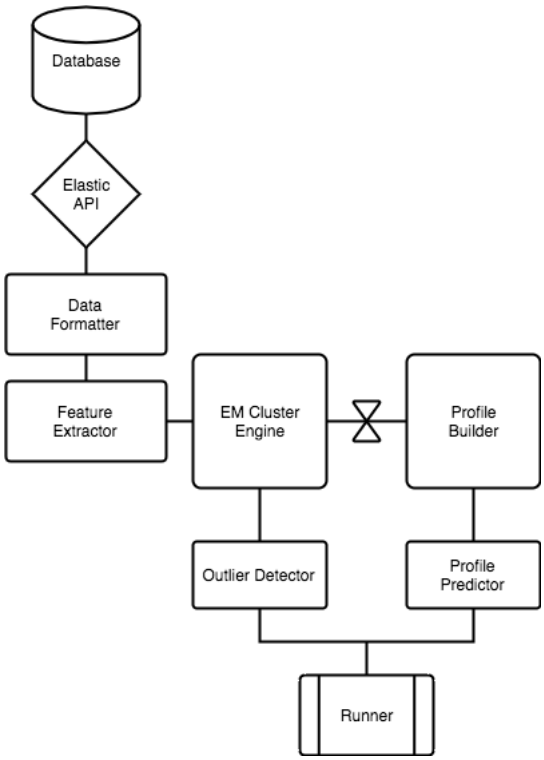
6.2. Mapyen

The Mapyen product is based upon the pipeline introduced in Figure 3.2. The system itself is divided into components, each of which is described in the following subsection. At the current state, Mapyen cannot be run on streaming data. We describe in the next subsection how Maypen can effectively be used and what makes the most sense with regards to how to use it.

Of course, Maypen is a proof of concept and it consequently has limitations, primary the fact that it cannot yet be run on streaming data. In subsection 6.2.3, we describe what Mapyen cannot do at this point and why design choices led us to one point instead of another.

6.2.1. System Decomposition

Figure 6.1: Mapyen's component diagram. Each component is used for a single task and the assembled system performs anomaly detection on the Adyen network metadata. The Adyen netflow clusters are accessed through the Elasticsearch API interfacing component, formatted properly, processed, and analysed to finally return a list of anomalous hosts with a justification as to why they are deemed anomalous.



The system described in Figure 6.1 is a specification of the pipeline described in Figure 3.2. It is a simple representation of the Mapyen system in terms of components of the system. Each component has a specific responsibility and all components work together to perform anomaly detection on the Adyen network metadata.

- **Database and the Elasticsearch API** The Database component represents the available Elasticsearch clusters on which the network metadata is stored at Adyen. Elasticsearch clusters, in general, can be accessed and queried through Python, using the Python Elasticsearch Client, which is represented by the Elasticsearch API component. This interface component is specific to Mapyen, and has to fetch the data in a proper format from the Elasticsearch netflow cluster for further processing by Mapyen.

- **Data Formatter** This component is used by Mapyen to format the data in such a way that deriving features from the dataset for clustering becomes easier. The specific information contained in the database entries is not relevant for the feature set, so this information is not stored by the Data Formatter. This component returns a Pandas dataframe containing all hosts inside the analysed part of the network with the relevant information for further processing.

- **Feature Extractor** The choice of features is done by the Feature Extractor component, which takes the dataset returned by the Data Formatter and creates a new datastructure containing the hosts with their features derived from this dataset. Another Pandas dataframe is returned by the Feature Extractor component, containing each host analysed and their corresponding feature vectors.

- **EM Cluster Engine** The dataframe delivered by the previous component is processed by the EM Cluster Engine, which creates EM clusters from the extracted sets of features. It clusters hosts with the EM Clustering algorithm, with k-means initialisation, and returns a dataframe of

clustered hosts with hashed IP addresses for privacy preservation, and the different features of the computed cluster centroids. If the engine is used for training, all recorded features for each iteration are spliced to create one single dataframe containing all behaviours to classify. For testing, features are clustered at each iteration. The Cluster Engine also analyses each cluster to give an explanation on their nature and characteristics.

- **Outlier Detector** Once cluster vectors are built, the Outlier Detector component can *detect outliers*. By examining the distance between hosts inside clusters and their corresponding cluster centroids using a dynamic threshold, a list of outlying host IDs is returned by this component.
- **Profile Builder** The Cluster Engine passes information about hosts and their clusters to the Profile Builder, which begins to create profiles for each host. The Profile Builder consumes frames of data until it is able to create a reasonable profile. Since profiles are built over time, they only begin to make sense after a few iterations of a sequence of dataframes. The Profile Builder returns, for each host, an N-gram profile that gives statistical information about occurrences of the host inside the different clusters.
- **Profile Predictor** Each host network profile passed through by the Profile Builder to the Profile Predictor is analysed, and profiles that turn out to be unpredicted are retained. The Profile Predictor module returns a list of hosts and their *anomalous* profiles, with markers at the specific timestamp of an unpredicted profile. It additionally makes use of the known cluster characteristics previously analysed by the EM Cluster Engine to show from which cluster to what cluster a particular host makes a switch.
- **Runner** The final component takes care of running the entire system, by providing a command line interface. It combines the Outlier Detector and Profile Predictor into one module and notifies the user when a host is, at a specified timestamp, both an outlier with regards to cluster crowds, and has an anomalous profile. The individual results for each anomaly detection component are also shown, but scoring on both component makes the case stronger for a call to action.

6.2.2. Usability

The Methodology in Chapter 3 briefly mentions how the system can be used by the Adyen security team. In this section, we provide the specifics of the system's usability and additionally go deeper into the training and test phases of Mapyen. A user can run the system through the command line and obtain a list of anomalous host IDs. It does not return IP addresses for immediate privacy preservation. However, if the user deems the evidence provided by Mapyen's Runner component to be significant enough for a call to action, it is possible to retrieve the IP address from the host's identifier, as a mapping from hashed identifier to IP address is available.

In order for Mapyen to build clusters and profiles, a training phase needs to occur. The extent and duration of this training phase are in the user's hands, but we advise them to keep consistency between the training phase and test phase. If a user decides to analyse hour-long chunks of netflow data throughout an entire day, we suggest to similarly train Mapyen on hour-long chunks during the previous day or the corresponding day in the previous week. Training on 5 minutes and testing on 24 hours would not make much sense because 5 minutes of data do not hold all possible behaviours observable in 24 hours, and vice versa. The user can, however, input the duration of the training independently from the duration of the testing phase. An example of Mapyen running query is shown:

```
python Mapyen -trd8 '2018-02-14' -trD 720 -trG 30 -ted8 '2018-02-15' -teD 240 -teG 30 -r ? -k ?
```

The example command shown above is used to train Mapyen on February 14, 2018, specified by the argument `-trd8`. The *duration* of the training phased is specified in minutes by `-trD` and is 12 hours in our case, and is split into chunks separated by 30-minute *gaps*, specified by the argument `-trG`. The date of the testing phase is specified by `t-ed8` and is set here on the next day. The duration of the training phase `-teD` is 4 hours and is split into 30-minute long chunks of data with `-teG`.

Additionally, there are a few aspects of Mapyen that are not completely predetermined, namely the IP range on which to execute Mapyen and the number of clusters to be used for clustering. The IP range

can be given by using the argument `-r`, for range. The next issue concerns the number of clusters given by `-k`, on which to train Mapyen. In Chapter 4, we describe ways to determine the optimal number of clusters, namely using the Elbow method. On the specific range we performed our research, the optimal number of clusters turned out to be $k = 6$, but this can evidently change when adjusting the IP range. Before being able to run Maypen on a range of IP addresses, the optimal number of clusters would have to be determined using the Elbow method again. Only then does it become possible to give a value for argument `-k`.

6.2.3. Limitations

In its current state, Mapyen is only able to handle static data. In order to run Mapyen on streaming data, the system has to be adjusted to take in streaming data and build clusters dynamically. This is an engineering challenge that falls out of the scope of this research.

Mapyen's biggest limitation is performance. The step that takes the largest amount of time is the data retrieval step, as it needs to run through the Elasticsearch data bulk for each host and extract the necessary data. On average, executing our data formatting step on the specific IP range we used for our research for 30 minutes, which contains ±630 hosts at the time, takes about **12 minutes on average**. Considering that there are on the order of ±4000 hosts inside the Adyen network, extracting data from the entire Adyen network for, say, an entire day split into 30-minute chunks would take ±30.5 hours, assuming the duration increase over time is linear. To solve the issue of performance, the formatting of the data needs to be rearranged in a way that we can parallelise the task on multiple threads. This would significantly increase the speed of data formatting.

Another limitation is the fact that, if a new host is introduced to Mapyen at the testing phase but was not present in the training data, Mapyen does not have a known profile for that host. At this point, Mapyen just notifies the user that a profile is non-existent. Additionally, profiles may be incomplete as a host could be practically inactive during the training phase. This also becomes counterproductive for the testing phase, because if the host is active then, comparing profiles does not make sense.

7

Evaluation of Mapyen

Anomaly detection techniques have been proven quite effective in detecting incidents such as botnet, command and control behaviour, DDoS attacks, port scans, P2P traffic, etc. [15, 18, 34] and more. From the literature, anomaly detection for the purpose of network security monitoring works well enough but still runs into some problems when it comes to false positive rates. By combining the detection of anomalous host in clusters and the analysis of anomalous inter-cluster behaviour profiles, the goal is to potentially reduce the number of false positives and increase our changes of incident detection.

On the request of Adyen, we focus on specific incidents the company wants to have monitored and detected. These classes of incidents can be considered applicable to the insider threat domain.

1. Data Exfiltration
2. Command and Control Traffic (Malware-like Behaviour)
3. Internal Port Scans

In order to test whether Maypen can detect these attacks, the Adyen security team focuses on a synthetic data generation process. Each attack is independently simulated. However, there are not enough resources to generate extensive attacks, so the following analysis may not be entirely statistically relevant. In any case, it enables us to already build hypotheses as to why Mapyen can hold potential for future developments in network incident detection.

7.1. Synthetic Data Generation

Internal port scans, data exfiltration incidents, and malware-like behaviour is what Adyen wants to primarily detect. These classes of incidents are of serious concern and current tools both issue false-positives and false-negatives, leading to significant resource usage and possibly missed threats.

7.1.1. Internal Port Scans

When a malicious internal person, or an external attacker is already inside the internal network and decides to find out what open services and vulnerabilities hide within the network, they can execute a port scan. Internal port scans are a quick and dirty way to evaluate the security of services inside a network. A general tool for port scans is Nmap for more extensive scanning, which are both vulnerability scanning tools. Port scans can also be done through Netcat.
We generate data for a collection of nine port scans. These are Nmap port scans, which occur at a very fast pace. The rest is done through Netcat connections, which is much slower and can reach port scanning rates of ten ports per hour. Table 7.1 summarises the executed port scans.

7.1.2. Data Exfiltration

Data exfiltration involves unauthorized transmission of sensitive data to an outside party. Adyen is very interested in detecting data exfiltration incidents due to the sensitive nature of the data handled

Table 7.1: Summary of the port scans executed by the Adyen security team on the internal network.

Label	Duration	Type
p1	22 minutes	Nmap
p2	4 minutes	Nmap
p3	19 minutes	Netcat
p4	12 minutes	Netcat
p5	22 minutes	Netcat
p6	30 minutes	Netcat
p7	30 minutes	Netcat
p8	30 minutes	Netcat
p9	30 minutes	Netcat

and the difficulty of detection of such events in a relatively open network and computing environment, even though use of cloud-based storage technology such as Dropbox is a strict policy violation. We decided to simulate data exfiltration by uploading specifically-sized files regularly to Dropbox with explicit permission. Because Internet requests to external parties are routed through Adyen proxy servers, the destination IP address does not identify the actual destination, since it will go through the same proxy servers no matter the external storage service being used. While these proxies log traffic destinations, the use case is for arbitrary and as-yet-unidentified cloud services. Realistically, a malicious insider would be likely to upload dumps of Adyen sensitive databases, that perhaps contain customer information.

To create a temporary file in Linux or MacOS, we use the following command:

```
dd if=/dev/zero of=test.img bs=1024 count=0 seek=\$[1024*s]
```

In this command, the `*s` at the end determines the size of the file.

Table 7.2: Summary of data exfiltration simulations run by the Adyen security team.

Label	Duration
e1	3 minutes
e2	9 minutes
e3	2 minutes
e4	9 minutes
e5	12 minutes
e6	12 minutes

With the resources at hand, the team is able to generate a small set of six data exfiltration samples. These samples are summarised in Table 7.2.

7.1.3. Malware-like Behaviour

A possible approach to malware simulation, despite the lack of systematic characterisations of malware and botnet traffic types on the Internet, is to send outbound requests at a regular pace over a period of time. It is generally understood that common malware regularly connects to a command and control server at consistent time intervals. Using a script that regularly sends DNS requests to a random server outside the Adyen network, the team simulates four possible instances of common malware behaviour.

Table 7.3 summarises the four malware simulations that are done by the Adyen security team. These simulations vary in duration, which is interesting to have tested to tell us whether time influences our detection system.

Table 7.3: Summary of malware infection simulations run by the Adyen security team.

Label	Duration
m1	14 minutes
m2	22 minutes
m3	30 minutes
m4	7 minutes

7.2. Malicious Dataset

To create the malicious dataset, we embed each attack in a 30-minute chunk of activity and take the two previous and following blocks. Some attacks were, however, happening concurrently. For example, the team runs a port scan that lasts for about an hour, which we then split into two consecutive 30-minute chunks. For each isolated attack, we end up with a record of five 30-minute chunks of activity, having the attack at the centre of this record. For each consecutive attacks, we take two 30-minute chunks prior to the start of the attack and two 30-minute chunks posterior to the last 30-minute attack activity. We need this record over time because we need to build network profiles for each host over time. These profiles need to be long enough to be able to compare them to the training profiles.

Table 7.4: Summary of the entire malicious dataset construction. Isolated attacks are joined by prior and posterior 30-minute chunks. Consecutive attacks are joined by prior 30-minute chunks to the first sequence and posterior chunks to the last 30 minutes of malicious activity. We refer to each chunk as *data block*. The prior and posterior data blocks are part of the analysis solely for the building of profiles. The anomaly detection does not run on these data blocks but only on the attack blocks. Attack data blocks contain both the concerned attack and non-malicious data. Note that some attack blocks may be occurring concurrently or even simultaneously.

Prior	Attack Block	Posterior
$p1_{-2}$, $p1_{-1}$	p1	$p1_{+1}$, $p1_{+2}$
$p2_{-2}$, $p2_{-1}$	p2	$p2_{+1}$, $p2_{+2}$
$p3_{-2}$, $p3_{-1}$	p3, p4	$p4_{+1}$, $p4_{+2}$
$p5_{-2}$, $p5_{-1}$	p5, p6, p7, p8, p9	$p9_{+1}$, $p9_{+2}$
$e1_{-2}$, $e1_{-1}$	e1	$e1_{+1}$, $e1_{+2}$
$e2_{-2}$, $e2_{-1}$	e2	$e2_{+1}$, $e2_{+2}$
$e3_{-2}$, $e3_{-1}$	e3	$e3_{+1}$, $e3_{+2}$
$e4_{-2}$, $e4_{-1}$	e4, e5	$e5_{+1}$, $e5_{+2}$
$e6_{-2}$, $e6_{-1}$	e6	$e6_{+1}$, $e6_{+2}$
$m1_{-2}$, $m1_{-1}$	m1, m2	$m2_{+1}$, $m2_{+2}$
$m3_{-2}$, $m3_{-1}$	m3, m4	$m4_{+1}$, $m4_{+2}$

The evaluation of the performance of Maypen is done by testing our system against the malicious data, solely on the malicious attack blocks. The prior and posterior data blocks are part of the analysis solely for the building of profiles. The anomaly detection does not run on these blocks. Note that attack data blocks contain both the attack in question and non-malicious data. Maypen is trained on the training data mentioned in Section 3.1 and tested on the data blocks shown in Table 7.4.

7.3. Performance of Mapyen

In order to evaluate the performance of Mapyen according to the Methodology as described in Chapter 3, we take the precision, recall, and false positive ratios for each attack type in our malicious dataset. Note that some blocks in our dataset may be occurring concurrently or even simultaneously.

We take the results in Table 7.5 and calculate the precision and recall values, and false positive ratios for all attack scenarios, for each type of attack. We choose to individually evaluate each attack scenario, so port scans, data exfiltration incidents, and malware simulations, to see the difference of performance of each Mapyen module with regards to each attack.

Table 7.5: The results of Mapyen run on the malicious dataset. For each data block, we indicate whether the corresponding attack was detected by the Outlier Detector or Profile Predictor systems, or both. This is indicated by a *v*. Non-detection is indicated by a blank cell. When both components detect the attack, the Fusion module returns a *v*. For each detection component, we indicate how many *anomalies* $n_{detected}$ were detected by the system. Finally, we show the total number of hosts N at the moment of analysis of the attack block.

Block	Outlier Det.	$n^O_{detected}$	Profile Pred.	$n^P_{detected}$	Fusion	$n^F_{detected}$	N
p1	**v**	54	**v**	64	**v**	6	501
p2	**v**	55	**v**	51	**v**	10	520
p3		51		53		6	491
p4	**v**	45		93		5	429
p5		59		26		3	571
p6		58		28		4	552
p7		56		19		4	528
p8		53		44		7	507
p9		52		71		6	483
e1		50		63		6	465
e2		49		82		10	464
e3		52		70		7	486
e4		55		51		10	520
e5		53		64		9	491
e6		48		61		6	450
m1		55	**v**	51		10	520
m2		53	**v**	64		9	491
m3		55	**v**	41		5	520
m4		54	**v**	55		5	501

Table 7.6: The different notations used to compute the precision, recall, and false positive rates of Mapyen. The first three rows describe the notation for the respective numbers of true positive port scans, data exfiltrations, and malware simulations detected by the Outlier Detector, the Profile Predictor, and the *fusion* of the two modules. The second set of three rows describes the same notation but for the total number of outliers for each attack block detected by the different parts of the system. The last two rows describe the notation for each attack scenario concerning the total number of scenarios for each attack type and the total number of hosts during when the attack is executed. Since the number of hosts is different per block but the numbers are relatively similar, we average the total number of scenarios and total number of hosts for port scan, data exfiltration, and malware simulation blocks to obtain single values for $\bar{n}_{detected}$ and $\bar{N}$.

	Port Scans	Data Exfiltration	Malware Simulations
Outlier Detector	$n^O_{TP-PS} = 3$	$n^O_{TP-E} = 0$	$n^O_{TP-M} = 0$
Profile Predictor	$n^P_{TP-PS} = 2$	$n^P_{TP-E} = 0$	$n^P_{TP-M} = 4$
Fusion	$n^F_{TP-PS} = 2$	$n^F_{TP-E} = 0$	$n^F_{TP-M} = 0$
Outlier Detector	$\bar{n}^O_{detected-PS} \pm 54$	$\bar{n}^O_{detected-E} \pm 51$	$\bar{n}^O_{detected-M} \pm 54$
Profile Predictor	$\bar{n}^P_{detected-PS} \pm 50$	$n^P_{detected-E} \pm 65$	$\bar{n}^P_{detected-M} \pm 53$
Fusion	$\bar{n}^F_{detected-PS} \pm 6$	$\bar{n}^F_{detected-E} \pm 8$	$\bar{n}^F_{detected-M} \pm 7$
Total number of scenarios	$N_{malicious-PS} = 9$	$N_{malicious-E} = 6$	$N_{malicious-M} = 4$
Total number of hosts	$\bar{N}_{PS} \pm 509$	$\bar{N}_E \pm 479$	$\bar{N}_M \pm 508$

Table 7.6 describes the notation used to compute the different values for precision, recall, and false positive rates of the Mapyen system. These values are treated individually for port scans, data exfiltrations, and malware simulations. Each module is individually tested; the Outlier Detector, Profile Predictor, and the fusion of the two, when if both modules return the same anomaly, the fusion of the two returns that anomaly, like an `AND` query between the two modules.

For each type of attack, we compute the average precision values for the Outlier Detector, Profile Predictor, and Fusion combined. The different notation for the values used is described in the Table 7.6 in question. Hence, the formulas for each precision value are:

$$\bar{P}_{PS} = \frac{\frac{n^O_{TP-PS}}{\bar{n}^O_{detected-PS}} + \frac{n^P_{TP-PS}}{\bar{n}^P_{detected-PS}} + \frac{n^F_{TP-PS}}{\bar{n}^F_{detected-PS}}}{3} = \frac{\frac{3}{54} + \frac{2}{50} + \frac{2}{6}}{3} \pm 0.14$$

$$\bar{P}_{E} = \frac{\frac{n^O_{TP-E}}{\bar{n}^O_{detected-E}} + \frac{n^P_{TP-E}}{\bar{n}^P_{detected-E}} + \frac{n^F_{TP-E}}{\bar{n}^F_{detected-E}}}{3} = \frac{\frac{0}{51} + \frac{0}{65} + \frac{0}{8}}{3} = 0$$

$$\bar{P}_{M} = \frac{\frac{n^O_{TP-M}}{\bar{n}^O_{detected-M}} + \frac{n^P_{TP-M}}{\bar{n}^P_{detected-M}} + \frac{n^F_{TP-M}}{\bar{n}^F_{detected-M}}}{3} = \frac{\frac{0}{54} + \frac{4}{53} + \frac{0}{7}}{3} \pm 0.03$$

Once more, we average these values to obtain a total precision $\bar{P}$:

$$\bar{P} = \frac{0.14 + 0 + 0.03}{3} \pm 0.06$$

For each type of attack, we now compute average recall values for each part of the system, still basing ourselves on the notation in Table 7.6.

$$\bar{R}_{PS} = \frac{\frac{n^O_{TP-PS}}{N_{malicious-PS}} + \frac{n^P_{TP-PS}}{N_{malicious-PS}} + \frac{n^F_{TP-PS}}{N_{malicious-PS}}}{3} = \frac{\frac{3}{9} + \frac{2}{9} + \frac{2}{9}}{3} \pm 0.26$$

$$\bar{R}_{E} = \frac{\frac{n^O_{TP-E}}{N_{malicious-E}} + \frac{n^P_{TP-E}}{N_{malicious-E}} + \frac{n^F_{TP-E}}{N_{malicious-E}}}{3} = \frac{\frac{0}{6} + \frac{0}{6} + \frac{0}{6}}{3} = 0$$

$$\bar{R}_{M} = \frac{\frac{n^O_{TP-M}}{N_{malicious-M}} + \frac{n^P_{TP-M}}{N_{malicious-M}} + \frac{n^F_{TP-M}}{N_{malicious-M}}}{3} = \frac{\frac{0}{4} + \frac{4}{4} + \frac{0}{4}}{3} \pm 0.33$$

Averaging these values gives us a total recall $\bar{R}$ of:

$$\bar{R} = \frac{0.26 + 0 + 0.33}{3} \pm 0.20$$

We do the individual false positive calculations for each attach and average the false positive rates obtained by each component of the system to get one false positive value for the whole system:

$$\bar{FP}_{PS} = \frac{\frac{\bar{n}^O_{detected-PS} - n^O_{TP-PS}}{\bar{N}_{PS} - N_{malicious-PS}} + \frac{\bar{n}^P_{detected-PS} - n^P_{TP-PS}}{\bar{N}_{PS} - N_{malicious-PS}} + \frac{\bar{n}^F_{detected-PS} - n^F_{TP-PS}}{\bar{N}_{PS} - N_{malicious-PS}}}{3} = \frac{\frac{51}{500} + \frac{48}{500} + \frac{4}{500}}{3} \pm 0.07$$

$$\bar{FP}_{E} = \frac{\frac{\bar{n}^O_{detected-E} - n^O_{TP-E}}{\bar{N}_{E} - N_{malicious-E}} + \frac{\bar{n}^P_{detected-E} - n^P_{TP-E}}{\bar{N}_{E} - N_{malicious-E}} + \frac{\bar{n}^F_{detected-E} - n^F_{TP-E}}{\bar{N}_{E} - N_{malicious-E}}}{3} = \frac{\frac{51}{473} + \frac{65}{473} + \frac{8}{473}}{3} \pm 0.09$$

$$\bar{FP}_{M} = \frac{\frac{\bar{n}^O_{detected-M} - n^O_{TP-M}}{\bar{N}_{M} - N_{malicious-M}} + \frac{\bar{n}^P_{detected-M} - n^P_{TP-M}}{\bar{N}_{M} - N_{malicious-M}} + \frac{\bar{n}^F_{detected-M} - n^F_{TP-M}}{\bar{N}_{M} - N_{malicious-M}}}{3} = \frac{\frac{50}{504} + \frac{49}{504} + \frac{3}{504}}{3} \pm 0.03$$

When we average each value to obtain an average false positive rate, we obtain:

$$\bar{FP} = \frac{0.07 + 0.09 + 0.03}{3} \pm 0.06$$

Despite the low false positive rate, the precision and recall values are also low, as we obtain an average precision of 6% and an average recall of 20%. When looking at the result, we do notice that the Outlier Detector works on rapid port scans done through Nmap, but slow port scans that act at rates of 10 ports per hour are undetected. We also notice that solely the Profile Predictor module is able to detect all malware simulations. There are only four examples, but it is a worth-mentioning start. Concerning the data exfiltration incidents, the results are very disappointing, not only research-wise but also from a business value perspective. In Chapter 8 in Section 8.2, we analyse the low precision and recall results and open a reflective discussion on the results obtained.

For this research, we did not adjust the detection thresholds to our data, because this proof of concept is to serve as a basis for future developments. Fine-tuning is an expected step that needs to occur and finding better thresholds is to be done as future work. Additionally, it is possible to apply Mapyen to the training data itself, in order to see what kinds of outliers and supposedly malicious profiles are detected.

8 Discussion

The research conducted through this paper leads us to several aspects that spark doubts and lead to discussions. Primarily, this study holds a few research challenges that we address and openly discuss in this chapter. Current research delivers new solutions on which we build the proof of concept that is Mapyen. However, some aspects of previous research are questionable in terms of decisions that are made and evaluation soundness. In this chapter, we describe the challenges encountered when looking at the latest research and our own research path.

In addition to the research challenges we face, the proof of concept developed, seems to work less well than hoped. However, when looking at the results, there appears to be potential for future developments, whereby the performance both in processing performance and in terms of our correctness criteria: low false positives, low false negatives, and high true positives. Analysing the results in Section 8.2, we expose the aspects of Mapyen that prove to hold significance and are open to valuable extensions.

8.1. Research Challenges

This section unveils the research challenges that we run into throughout the process of this master's thesis. Throughout the literature survey, we find aspects of current research that are worth discussing in the following subsections. Furthermore, this section makes some remarks regarding the application of seemingly black-box machine learning techniques in the domain of security monitoring, particularly regarding netflow traffic analysis.

8.1.1. Publication Aspects

Parts of the research we examine apply several machine learning techniques regarding behaviour clustering in order to perform host identification in computer networks, for the purpose of situational awareness in networks or incident detection [10, 15, 18, 21, 22, 27, 34]. As we see in these studies, different types of clustering are applied to either private or public datasets, and the results, in general, prove to be successful, while still delivering a high rate of false positives [34].

The analyses are done on datasets that have been criticised for their nature and generation procedures [34]. Furthermore, there is, to the best of our knowledge, no newer public dataset than the KDD-99 or DARPA sets from the late 90s [14, 26]. This makes it hard for current research to move forward in network anomaly detection on netflow as the public resources in this domain are scarce and outdated.

We suspect that in-depth developments in network intrusion systems that provide highly efficient solutions used in the industry remain in the realm of unpublished industrial research. It may as well be possible that large companies have enough resources on which to train and test their systems and would rather keep the data, the end results, and possible productisation to themselves. Within Academia, research that delivers insufficient or unsatisfactory results may remain unpublished, which is referred to as the issue of publication bias [47].

8.1.2. Lack of Validation

An issue worth considering regarding the abundant research done in security incident detection using network metadata is the lack of in-depth analysis of the machine learning techniques used. In most papers reviewed, a concrete decision process for an appropriate clustering method is missing. The machine learning techniques are used in a black-box manner, without further explication of the inner-workings of the algorithms that influence the results on the specific data handled.

When going through the latest advancements in network anomaly detection, we discover that some researches base their validation steps on statistics about true positive and false positive rates, or alternatively, general *speed* of the system. General research aimed at uncovering new solutions for this problem evaluate them by testing against the previously mentioned old datasets. However, as mentioned before, the lack of a validation step of the clustering algorithms being used means that it is difficult to understand whether the data is clustered in the right way.

In this study, we validate our clusters by plotting their *heatmap* representations. Heatmaps provide a fast and subjective way for humans to evaluate effectiveness of clustering. So far, we have used this technique to identify one particular algorithm that appears to work best in our context. But how can we be sure this would work in another industrial context, where the network setup and netflow collection practice is different? One possible way would be to test it against the available public datasets, which would provide us with more cyber attacks to test Mapyen.

The main problem concerning this is that we do not have, in our domain, a standardised method for validation of our classifications. The reason for this is the absence of ground truth in our netflow datasets. Understanding clusters to validate the clustering results is a common machine learning problem and in our case, it becomes very hard to exhaustively generate labelled data, as our datasets are very large and might be considered Big Data. Despite the research mentioning ground truth vectors used as cluster centroids [21], these vectors are not representative of an exhaustive set of network activity. KDD-99 and DARPA datasets contain labelled attacks, but when it comes to regular non-malicious traffic, we do not have labelled data. A consideration for the state-of-the-art would be to encourage researchers to take the time to create new, up-to-date non-malicious traffic datasets and label network activity for future work in the domain of network anomaly detection.

8.2. Reflection on Mapyen

As seen in Chapter 7, the performance of Mapyen is poor. With an average precision score of $\bar{P} \pm 0.06$ and an average recall score of $\bar{R} \pm 0.20$, it tells us that Maypen struggles at identifying the synthetic threats we created. However, the numbers hide the fact that some threats were completely detected by Mapyen individual detection components, and an interesting aspect of the undetected attacks prove that Mapyen holds potential for future successful developments.

In this section, we provide an analysis of the detected and undetected attacks, discussing as to why they were detected or not. Furthermore, we look into the average false positive rate of $\bar{FP} \pm 0.06$, which is very low compared to the 20% in current research. However, this result may be biased by the fact that Mapyen performs poorly at detection in general. As a reminder, Mapyen has *three* anomaly detection components:

- **Outlier Detector** This component calculates the distance between hosts and their cluster centroid and returns any host that is further away than a specific threshold.
- **Profile Predictor** This component looks for unpredictable clusters in a host's new profile, based on a profile on which the component was trained.
- *Fusion* Mapyen combines the results of the Outlier Detector and Profile Predictor modules by means of an `AND` query. If both components return one host, that host is returned at *fusion*.

8.2.1. Detected Attacks

Mapyen is successful at detecting instances of rapid port scans done by Nmap, while slow port scans tend to slip through the detection modules. Rapid port scans, regardless whether the duration was 22

or 4 minutes, as given in Table 7.1, are successfully detected by both the Outlier Detector and Profile Predictor modules. That is because a host suddenly switches to an unusual cluster and is a far outlier to that cluster. Slow port scans, however, seem like normal traffic to Mapyen, but that is to be expected if only 10 ports are visited in one whole hour. We expect slow port scans to also slip through other detection systems which base themselves on the number of destination ports for their clustering. One slow port scan, namely $p4$ is detected solely by the Outlier Detector component.

The Profile Predictor module is successful at detecting all cases of malware simulations. When looking at the behaviour profiles recorded, we notice a consistent change of behaviour at the start of the simulation. The simulating host namely executes a cluster switch from **cluster F** to **cluster C**, from a highly active cluster in the network to a cluster more likely to contain hosts uploading data to specific targets in the network. This makes sense to some extent, as we expect malware behaviour to show consistent outbound upload patterns, which in Adyen's case go through the internal proxy server, hence the specific targets in the network.

8.2.2. Undetected Attacks

We previously mention the fact that slow port scans are not detected by Mapyen. This is easily explainable by the fact that we end up with hosts that connect to an expected number of ports per hour, so the port scans slip through the detection modules. We do expect attackers to run rapid port scans in general, as attackers are under time pressure to identify new targets before they are themselves identified and their foothold eradicated. Furthermore, the fact that the logs do not contain netflows of failed connections also influences the difficulty in detecting port scans from these logs. Future work on this can attempt to find a way to compensate for the missing information.

Looking at data exfiltration incidents, it is actually to be expected that these are difficult to detect, especially since we have no information as to where the data is going in the current state. Since the outgoing traffic goes through the proxy, we cannot see beyond proxy connections in our training data. One would need to correlate netflow entries from our training data with entries going out of the proxy, and this is a hard thing to do when many connections are happening at once. If we were able to add the destination IP address to our analysis, we could filter on the most likely targets for data exfiltration, namely Dropbox, which is against policy at Adyen, as is Google Drive, WeTransfer, or any cloud storage program.

We take a glance at an example of a data exfiltration incident, namely $e6$, in order to see what the evolution of clusters tell us. This incident is not detected as an outlier, probably because uploading behaviour is something that happens regularly inside the company, and most often for non-malicious purposes. Employees are constantly downloading and uploading data from and to the Web, and internal servers do the same. Perhaps looking at the inter-cluster evolution can give an indication as to what we can do in order to have indications of data exfiltration. The training inter-cluster record during working hours for the host responsible for the data exfiltration is as follows:

$$[F, F, F, F, E, F, None, F, F, F, F, F, F, F, F, F, E, E]$$

The inter-cluster record for this same host for $e6_{-2}$, $e6_{-1}$, e6, $e6_{+1}$, and $e6_{+2}$ is the following:

$$[F, E, F, F, F]$$

It now makes sense why the behaviour seems normal, as the patterns observed here fit the training profile to an extent. Patterns $[F, E, F]$ and especially $[F, F, F]$ are known to the training N-gram built on this training sequence of clusters. Unfortunately, data exfiltration detection does need more than netflow data in our case. As the outgoing network traffic is routed through proxy servers, it is not possible to immediately check the destination IPs for *forbidden* websites. Future work on Mapyen should come up with smarter ways in order to detect such incidents, such as time-correlating with connections going out of the proxy.

8.2.3. False Positives

Mapyen scores low in terms of false positive ratios. However, this could be explained by the low detection success of Mapyen, especially regarding the data exfiltration incidents. We do deem it interesting, though, to look into the false positives detected and figure out as to why they are classified wrongly.

We randomly select 5 hosts that are falsely classified as anomalous and denote them respectively by h_1, h_2, h_3, h_4, and h_5. These are five hosts identified by both anomaly detection modules, the *fusion* of which, as anomalous.

h_1

This host is detected as an outlier by the Outlier Detector because the activity stopped halfway through the analysis. If a host stops its activity in the middle of the timeframe under analysis, much less data is recorded. It is then understandable that this host appears to be anomalous because significantly fewer data has been recorded. For the Profile Detector, the training profile for this host shows an inactive host that becomes active in the middle of the night, so little activity is recorded. We give a small excerpt of the training sequence of clusters for that host, right before it becomes active:

$$[None, None, None, None, E]$$

During the day of the synthetic attack, the host was highly active, and even in the timeframe when the host goes offline, it is still clustered in the same cluster as the previous one. Below is a sample of the test activity.

$$[F, E, E, E]$$

Mapyen detects an anomalous profile for h_1 at every cluster record because it learns a profile containing mostly *None* entries, as the host is practically inactive at the training phase. The activity recorded at the testing phase differs much from the training phase. This is an example of Mapyen's shortcoming: Mapyen cannot deliver a reliable analysis if the training data concerning a host is lacking.

h_2

As opposed to h_1, the host is highly active during the training phase and rather inactive at the moment of the synthetic attack. This means that the training profile of that host is quite extensive, but the compared profile is only showing an inactive host. Below is a sample of the training profile and a sample of the test profile at the same time the day of the attack block under analysis:

$$[F, F, E, F]$$

$$[E, C, C, E]$$

From Section 4.4.2, we expect cluster C to be a cluster of low activity. If the activity is low enough, this host may appear as outlier, as it is then much less active than other hosts in that cluster. Perhaps this indicates anomalous behaviour; one can, however, not be sure that this is the case since the training profile could be erroneous because the host was not very active at that time. In any case, further research into the activity at the time of training is required in order to determine whether this is a real false positive.

h_3

Here, we are dealing with a host constantly switching between **cluster B** and **cluster E**. These two clusters are quite similar, except for **cluster E** potentially displaying more variate traffic. If this host is standing at the limit of these two clusters each time, it makes sense that it is detected as an outlier with regards to the clusters' centroids, because it is always at the limit.

h_4

When looking at h_4's profile, we can see a complete switch in behaviour from its training profile. Figure 8.1 shows a sample of the netflow traffic observed at the time h_4 was identified as an anomaly. We can witness a large download happening, which would explain the change in behaviour and the fact that

Figure 8.1: A sample of h_4's netflow traffic at the time it was detected as an anomaly. From left to right, the values indicate the protocol used, destination port, bytes sent at the source, and bytes received from the destination. We can see a large download occurring.

tcp	80	441	7748
tcp	80	571	3952
tcp	80	717	3292
tcp	80	507	3292
tcp	80	507	3240
tcp	80	596	6630

h_4 becomes an outlier. This is HTTP traffic and after some research, we figured the traffic was going through one of the internal proxy servers.

h_5

This is another example of a host of which the profile is not known because it was barely active at training time. There is but one record at a specific timestamp in the profile on which h_5 was trained. Furthermore, this host does a cluster switch when flagged as an outlier and does not change cluster after. We suspect the host to be slowly moving towards a different cluster and, at the time of measurement, the host was still on the outlying delimitations of the new cluster.

Overall, the Mapyen proof of concept delivers promising results but is also affected by specific issues regarding performance and precision. There are many possible advancements to be done on Mapyen and these are explained in the next chapter, in Section 9.3.

9 Conclusion

We take a moment to glance back at the original problem, which was the finding of a ***well-founded solution to execute real-time network incident detection on lightweight network traffic metadata, which can provide a basis for future security monitoring research and developments****. Section 9.1 summarises the answers that we bring to the five subquestions and the next sections reflect upon the key findings and possible future improvements of this research. The main research question that sparks from the posed problem,* ***what clustering-based anomaly detection technique can we use to efficiently perform network incident detection in a large-scale industrial context?****, is subsequently answered.*

We deliver Mapyen, the outcome product of this research, which provides Adyen with a proof of concept for further security monitoring developments. Mapyen is a clustering-based approach for anomaly detection, to detect network incidents that are relevant for Adyen's secure functioning. Mapyen does not perform as well as hoped for, but holds potential for future work and can surely be extended to something of higher performance.

Mapyen serves as a basis for network monitoring, but the implicit results of this study for Adyen go beyond Mapyen. Adyen now has knowledge about what clustering method is best to use in their context. It is provided with an ethical framework that, besides the issue of network security, relates to data analytics in general. There is also a clear indication of what not to do in the future; the failures of this research are gains for the further development of Adyen.

9.1. Outcomes

SQ1 — How is our research in network anomaly detection influenced by Ethics?

To answer this question, we describe the ethical process done to establish guidelines on how to pursue this project in an ethically correct manner. consult with the TU Delft Human Research Ethics Committee in Chapter 1. The research was approved on October 23, 2017, and the guidelines agreed upon with the committee are followed through the entire analysis.

SQ2 — On which features extracted from the available network metadata can we build clusters?

The choice of features and reason behind this choice is explained in both the literature review in Chapter 2 and Chapter 4. The features we picked are proposed by [15], which are still being used to an extent in the latest research. These features are extracted per host and relate to network connectivity, connection dispersion, and information about traffic content.

SQ3 — Which cost-effective clustering methods do we pick that fit the context best?

By testing different clustering algorithms in Chapter 4 from the four main types of clustering methods explained in Chapter 2, we are able to choose the optimal clustering method for Adyen, namely EM

clustering with a K-means initialisation step. This result comes from the different *heatmap* visualisation of the clustering results, which reveals that EM clustering delivers the best results. We do not have a clear indication why this is so, but this opens the possibility for further research in this matter.

SQ4 — What is an anomaly?

The research from [34] explains how we can interpret anomalies with regards to unsupervised learning techniques for network anomaly detection. In Chapters 3 and 5, we explain how we choose to perform anomaly detection and what we deem to be flagged as anomalous. We introduce a time-based profiling module that creates host profiles based on clustering results over time and is able to evaluate new profiles to determine whether these profiles fall outside the known patterns. The answering of this question leads to the creation of Mapyen.

SQ5 — Are cyber threats of interest detectable? To what extent can lightweight network data be used for the purpose of detecting these cyber threats?

In Chapter 7, we establish the cyber threat landscape Adyen wishes to have covered and evaluate the anomaly detection system proposed in the chapters that answer **SQ4**. It is found that Mapyen does not perform well as is, but shows promise potential for future developments and holds the potential for significantly lower amounts of false positives. It is found that, with the network infrastructure at hand and the nature of our features, certain threats are not detectable through sole netflow analysis. This is discussed in Chapter 8.

9.2. Contributions

Performing research is for the intrinsic purpose of bringing something new to the body of Science. Doing what deserves to be written and writing what deserves to be read, striving to make our world a happier place in the end. My research, in the end, has contributed to Science in relatively small proportions, but the core meaning behind the findings is part of a bigger scheme that aims at creating a more secure digital society.

9.2.1. Ethical Guidelines

Ethics is a delicate subject that needs to be approached with care. We obtained full approval from the TU Delft Human Research Ethics Committees to pursue and finalise this project on October 23, 2017. The consulting of the Ethics Committee delivers a set of guidelines to follow for this project, primarily concerning the anonymity of the data analysed.

9.2.2. An Approach for Network Behaviour Profiling

Finding the optimal solution for clustering at Adyen allows us to extend upon current research in network anomaly detection with the inter-cluster movement recording over time. To our knowledge and based on the literature reviewed in Chapter 2, such host profile modelling has not been done before in the network anomaly detection domain. Recording each host's cluster at regular moments in time takes us to the building of network behaviour profiles, and we are now able to probabilistically model changes of clusters that are unexpected through the use of N-grams. Unexpected cluster change detection is proven to hold potential for security incident detection and is a source for reasonable evidence with regards to when a call to action should be raised.

9.2.3. Mapyen's Evaluation

The proof of concept we deliver under the name of Mapyen is evaluated against synthetic data containing netflows of different cyber threats that Adyen wishes to detect. These three threats are:

1. Internal port scans
2. Data exfiltration incidents
3. Malware infections

The generation procedures for these attacks are explained in Chapter 7. Mapyen does not perform as well as hoped and a large portion of attacks of a specific type still remain undetected, despite the addition of inter-cluster evolution analysis. The results, though, are promising, and there is certainly more to be done in this domain.

9.3. Future Work

Prior to this, we mention the many possible extensions for Mapyen and how they could be significant for Adyen. However, besides industrial implications of future work possible, there are further research possibilities that stem from this research. In our study, the clustering of the netflow dataset is done based on features that each have the same impact on the algorithm. Introducing feature weights may reveal to be more effective. This is discussed in Section 9.3.1. Furthermore, throughout this report, we encounter specific unanswered questions that fall outside the scope of this study, which can be great opportunities for future research. This is elaborated upon in Section 9.3.2. Besides these additional questions, there is also future work to be done on Mapyen itself, described in Section 9.3.3. Lastly, Mapyen has the potential to go beyond anomaly detection, as explained in Section 9.3.4.

9.3.1. Weighted Features

Perhaps an interesting study to lead after this current research would be to perform host behaviour clustering using weighted features. Because of the nature of Adyen's network infrastructure, all outgoing traffic is routed from host to the internal proxy servers. Therefore, the number of peers each host remains relatively constant, especially when hosts are actively browsing the Web. It is not possible to differentiate, at this stage, hosts that do more intensive browsing than others. By introducing higher weights to features such as those that relate to traffic amplitude or destination port counting, it could create more significant clusters that we can use for behaviour profiling.

9.3.2. Additional Questions

Road-Not-Taken Opportunities

We discuss the road not taken in Appendix D. This section mentions small side experiments that are done in this research but do not go further to answer the main research question. These experiments seem to hold potential for a future extension of Mapyen. The road not taken is an indication for Adyen of what it can now do with Mapyen in hand.

The Higher Performance of EM Clustering

We investigate four main types of different clustering techniques and discover that EM Clustering works best with the data at hand. But the mystery lies in the question of why this is actually the case. We speculate that it works better because of the relatively low variance of the 9-dimensional data, which makes it so that a distribution-based clustering algorithm is more powerful than distance-based. Perhaps it could mean that EM Clustering is more appropriate when it comes to Big Data analysis. However, these are just speculations that require further research to be confirmed.

The Drawbacks of t-SNE

Previous studies in dimensionality reduction show that t-SNE can be a powerful technique to represent data in lower dimensions [3]. However, one commonly known drawback of t-SNE is the fact that this technique can make mistakes. When we try to visualise our data using t-SNE, we can see hosts that are clustered together in nine dimensions far apart from each other in two. This is an artefact of mapping in two dimensions using t-SNE and further research would be required to uncover the reasons behind the mistakes t-SNE makes and how to tweak t-SNE parameters such that they effectively work.

More Extensive Cyber Threats

Mapyen performs poorly on the current malicious dataset we construct. However, it holds potential to detect more cyber threats when its components are executed individually. Detecting outliers from clusters seems to be the most efficient way to detect rapid port scans while looking at behaviour profiles seems to help us identify malware infections. The next step would be to test Mapyen against public netflow datasets that contain a variety of cyber incidents. Doing so would help make more

extensive performance calculations and perhaps reveal that Mapyen performs better than on our synthetic data.

9.3.3. Extend upon Mapyen

The proof-of-concept we build to answer our research question opens doors to possible future work to be done. The precision and recall of Maypen itself is not yet acceptable but it definitely shows that this system holds potential for neflow analysis. Mapyen is able to identify the rapid port scans generated and shows that the profiling part can identify the malware simulations run. The nature of the network infrastructure does not permit us to easily view exactly where outbound connections go beyond the internal network, in our training dataset, which would be useful to detect data exfiltrations to cloud services such as Dropbox or Google Drive. We also lack a record of failed connections, as these are not logged into the database. Future research into what additional information, features, we can use to compensate for this lack of information can be done, and should very well be done.

In addition to the possibility of enhancing Mapyen with the aspects mentioned in Appendix D, the building of profiles itself could contain more information useful for detection. Recording the distance of hosts to their centroids over time can enhance the profiles themselves by providing new insights on host movements in space and time.

Another room for improvement concerns the sizes of training and test sets. Our training set contained an entire day of 30-minute chunks, but whether this covers the optimal set of behaviours we want to observe is unsure. Future work should focus on testing Mapyen with different training sizes and see which ones suit the network activity at hand best. Furthermore, not only training and test set sizes can be adjusted, but also the anomaly detection thresholds, as mention at the end of Chapter 7.

For Adyen itself, real-time security monitoring should be Mapyen's end use. Right now, Mapyen trains itself on static moments in time and can be tested against fixed new data. Being able to execute Mapyen on streaming data would enable the possibility for real-time monitoring. This aspect is an interesting item for future developments within Adyen.

Mapyen's thresholds for anomaly detection can and should also be adjusted in the future. Additionally, Mapyen gives the anomalies detected by the Oulier Detector, Profile Predictor, and the *fusion* of the two. The *fusion* here behaves like an `AND` query between the two modules. This could significantly be improved in future work, especially if we start basing the *fusion* on previous information about one host or on other heuristics.

9.3.4. Beyond Anomaly Detection

Looking at areas of interest beyond anomaly detection, the proof of concept we built has components that can be used for different purposes. The host behaviour clustering example can be used for device identification inside the network. Current research done in network situational awareness [21, 32] offers the possibility for Adyen to extend upon the EM Cluster Engine and use it to create a mapping of their internal network.

Furthermore, the Profile Predictor engine provides us with a new way to monitor devices, solely by monitoring their cluster changes. By using clusters with specified characteristics, such as *upload/-download cluster* or *Internet browsing cluster*, security experts can monitor device activities inside the network, while also monitoring when devices are active. Perhaps even classifying network devices inside network can be possible through the inter-cluster evolution analysis.

One final note, Mapyen is designed to work on features extracted from network metadata. However, the system can work on any type of features, beyond network analysis. As long as there is a time-based measurement component, Mapyen is, in theory, able to run on any feature set.

Bibliography

[1] Wetboek van strafrecht. *Wet- en regelgeving*, 1881. URL http://wetten.overheid.nl/BWBR0001854/2018-01-01.

[2] Wet op het financieel toezicht. *Wet- en regelgeving*, 2006. URL http://wetten.overheid.nl/BWBR0020368/2018-01-03.

[3] Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[4] The prioritized approach to pursue PCI DSS compliance. *PCI Security Standards Council*, 2016. URL https://www.pcisecuritystandards.org/documents/Prioritized-Approach-for-PCI_DSS-v3_2.pdf.

[5] PSD2. *Nederlandse Vereniging van Banken*, 2017. URL https://www.nvb.nl/veelgestelde-vragen/bank-maatschappij/2730/psd2.html.

[6] Cluster evolution analysis: Identification and detection of similar clusters and migration patterns. *Expert Systems with Applications*, 83:363 – 378, 2017.

[7] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. *Proceedings of the 8th International Conference on Database Theory*, pages 420–434, 2001.

[8] R. E. Anderson. ACM code of ethics and professional conduct. *Commun. ACM*, 35(5):94–99, 1992.

[9] ASM. CISSP intrusion-detection systems (IDS). *ASM Educational Center*, 2016. URL https://asmed.com/cissp-intrusion-detection-systems-ids/.

[10] R. Berthier, M. Cukier, M. Hiltunen, D. Kormann, G. Vesonder, and D. Sheleheda. Nfsight: Netflow-based network awareness tool. *Proceedings of LISA '10: 24th Large Installation System Administration Conference*, 72:119, 2010.

[11] G. Bottazzi, G. F. Italiano, and G. G. Rutigliano. Frequency domain analysis of large-scale proxy logs for botnet traffic detection. *Proceedings of the 9th International Conference on Security of Information and Networks*, pages 76–80, 2016.

[12] M. Chessel. Ethics for big data and analytics. *IBM Big Data and Analytics Hub*, 2014. URL http://www.ibmbigdatahub.com/sites/default/files/whitepapers_reports_file/TCG%20Study%20Report%20-%20Ethics%20for%20BD%26A.pdf.

[13] DARKTRACE. URL https://www.darktrace.com/.

[14] DARPA. URL https://www.ll.mit.edu/ideval/data/.

[15] G. Dewaele, Y. Himura, P. Borgnat, K. Fukuda, P. Abry, O. Michel, R. Fontugne, K. Cho, and H. Esaki. Unsupervised host behaviour classification from connection patterns. *International Journal of Network Management*, 20:312–337, 2010.

[16] Elasticsearch. URL https://www.elastic.co/products/elasticsearch.

[17] M. T. H. Elbatta and W. M. Ashour. A dynamic method for discovering density varied clusters. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 6(1), 2013.

[18] K. Flanagan, E. Fallon, A. Awad, and P. Connolly. Self-configuring netflow anomaly detection using cluster density analysis. *19th International Conference on Advanced Communication Technology (ICACT)*, 2017.

[19] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys Tutorials*, 16(4):2037–2064, 2014.

[20] H. Honerkamp. Corporate values and hypocrisy. *Creating Good in the Workplace*, 2013. URL https://goodworkplace.wordpress.com/2013/08/28/corporate-values-and-hypocrisy/.

[21] K. M. T. Huffer and J. W. Reed. Situational awareness of network system roles (sansr). *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, (8), 2017.

[22] A. Jakalan, J. Gong, Z. Weiwei, and Q. Su. Clustering and profiling ip hosts based on traffic behavior. *Journal of Networks*, 10(2), 2015.

[23] M. Josephson. Ethical responsibilities in the employer-employee relationship. *The Josephson Institute - Exemplary Leadership and Business Ethics*, 2016. URL http://josephsononbusinessethics.com/2010/12/responsibilities-employer-employee-relationship/.

[24] A. Kassambara. Practical guide to cluster analysis in R: Unsupervised machine learning. 1, 2017.

[25] S. Kaushik. An introducton to clustering and different methods of clustering. *Analytics Vidhya*, 2016. URL https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/.

[26] KDD-99. URL http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[27] B. Li, M. H. Gunes, G. Bebis, and J. Springer. A supervised machine learning approach to classify host roles on line using sflow. *Proceedings of the first edition workshop on High performance and programmable networking*, pages 53–60, 2013.

[28] S. Z. Li and A. Jain. Mahalanobis distance. *Encyclopedia of Biometrics*, pages 953–953, 2009.

[29] W. Lu and T. Hengjian. Detecting network anomalies using cusum and em clustering. *Advances in Computation and Intelligence*, pages 297–308, 2017.

[30] K. Macnish. Surveillance ethics. *The Internet Encyclopedia of Philosophy*, 2011. URL http://www.iep.utm.edu/surv-eth/.

[31] P. C. Mahalanobis. On the generalised distance in statistics. *Proceedings National Institute of Science*, 2(1):49–55, 1936.

[32] S. Marshall. Candid: Classifying assets in networks by determining importance and dependencies. 2013. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-64.html.

[33] Matplotlib. URL https://matplotlib.org/.

[34] A. Mohiuddin, M. Abdun Naser, and H. Jiankun. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, (60):19–31, 2016.

[35] G. Münz, S. Li, and G. Carle. Traffic anomaly detection using k-means clustering. *In GI/ITG Workshop MMBnet*, 2007.

[36] L. H. Newman. A scary new ransomware outbreak uses wannacry's old tricks. *WIRED*, 2017. URL https://www.wired.com/story/petya-ransomware-outbreak-eternal-blue/.

[37] National Institute of Standards and Technology. Cybersecurity framework. *NIST*, 2014. URL https://www.nist.gov/cyberframework.

[38] pandas. URL https://pandas.pydata.org/.

[39] The European Parliament and the Council of 27 April 2016. Regulation (EU) 2016/679. *In EUR-Lex - Access to European Union Law*, 2016. URL http://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX%3A32016R0679.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] Elasticsearch Python. URL https://elasticsearch-py.readthedocs.io/en/master/.

[42] R. M. Rahul. How can I choose the best k in knn (k nearest neighbour) classification? *Quora*, 2017. URL https://www.quora.com/How-can-I-choose-the-best-K-in-KNN-K-nearest-neighbour-classification.

[43] scikit learn. URL http://scikit-learn.org/stable/modules/clustering.html.

[44] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, 2001.

[45] A. Simon-Lewis. This heat map shows how wannacry spread around the world like an epidemic. *WIRED*, 2017. URL http://www.wired.co.uk/article/how-wannacry-spread-around-the-world.

[46] Snort. URL https://www.snort.org/.

[47] F. Song, S. Parekh, L. Hooper, Y. K. Loke, J. J. Ryder, A. Sutton, C. Hing, C. S. Kwok, C. Pang, and I.M. Harvey. Dissemination and publication of research findings: An updated review of related biases. *Health technology assessment (Winchester, England)*, 14, 2010.

[48] C. Y. Suen. n-gram statistics for natural language understanding and text processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 164–172, 1979.

[49] Suricata. URL https://suricata-ids.org/.

[50] D. Swift. A practical application of SIM/SEM/SIEM automating threat identification. *SANS Institute InfoSec Reading Room*, 2006.

[51] S. Theodoridis and K. Koutroumbas. Pattern recognition. *Academic Press*, 2008.

[52] R. Tibshirani. Clustering 2: Hierarchical clustering. *R. Tibshirani's lecture on Data Mining: 36-462/36-662*, 2013. URL http://www.stat.cmu.edu/~ryantibs/datamining/lectures/05-clus2-marked.pdf.

[53] M. G. Velasquez. Business ethics concepts and cases. *Pearson Education, Inc*, 2012. URL http://silabus.feb.unila.ac.id/etika-bisnis/bahan/Ethics%20and%20the%20Employee%20-%20Velasquez_C8.pdf.

[54] J. Wallenstrom. Exposure of your sensitive data isn't a bug, it's a feature. *TechCrunch*, 2017. URL https://techcrunch.com/2017/09/25/exposure-of-your-sensitive-data-isnt-a-bug-its-a-feature/.

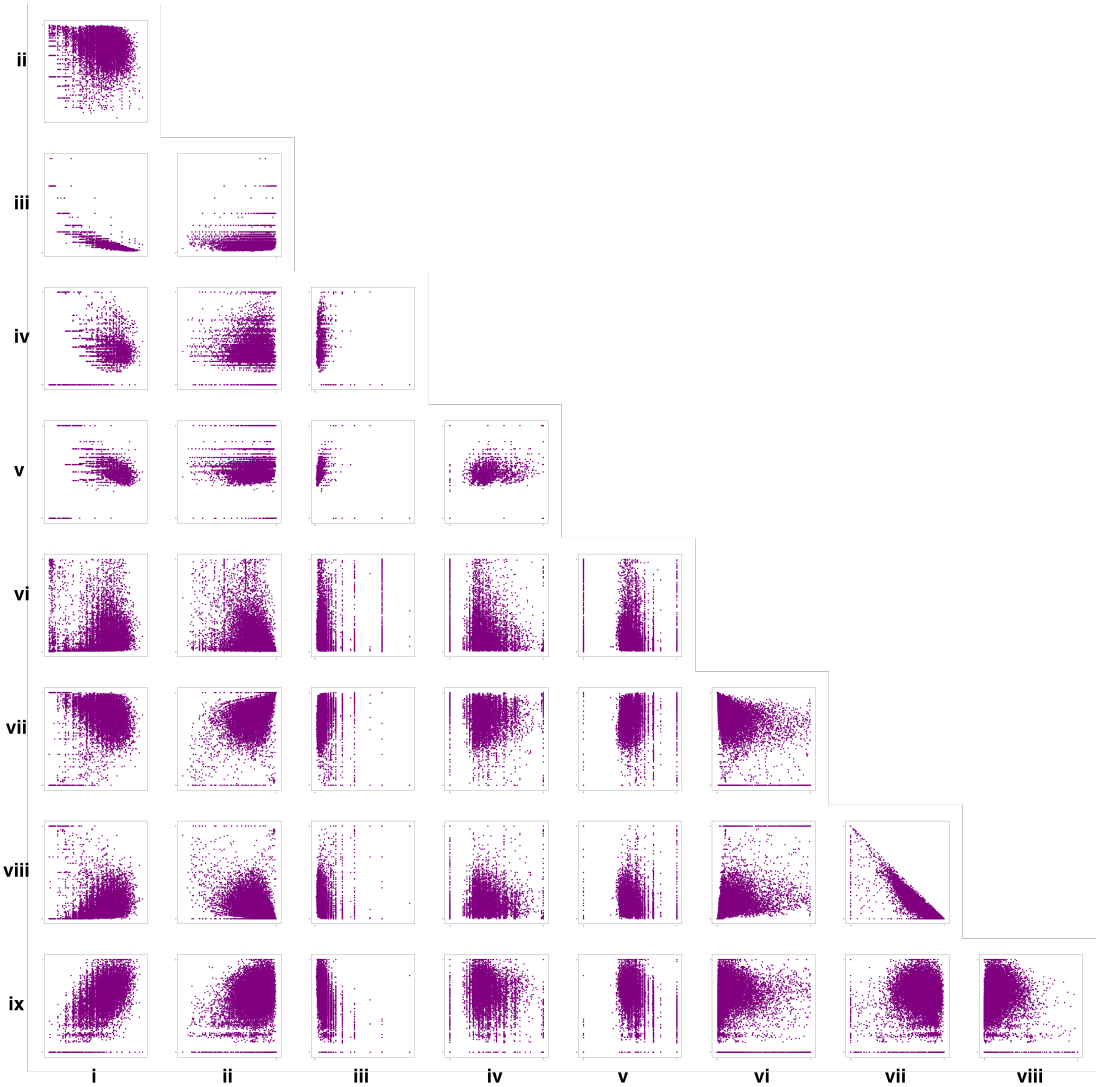
[55] WorkTime. European union (EU) employee monitoring laws: What are employers allowed and not allowed doing in the workplace? *Respectful Employee Monitoring - Legal Aspects*, 2016. URL https://www.worktime.com/.

[56] J. Zhang, K. Jones, T. Song, H. Kang, and D. E. Brown. Comparing unsupervised learning approaches to detect network intrusion using netflow data. *Systems and Information Engineering Design Symposium*, 2017.

A

Feature-by-feature Representation

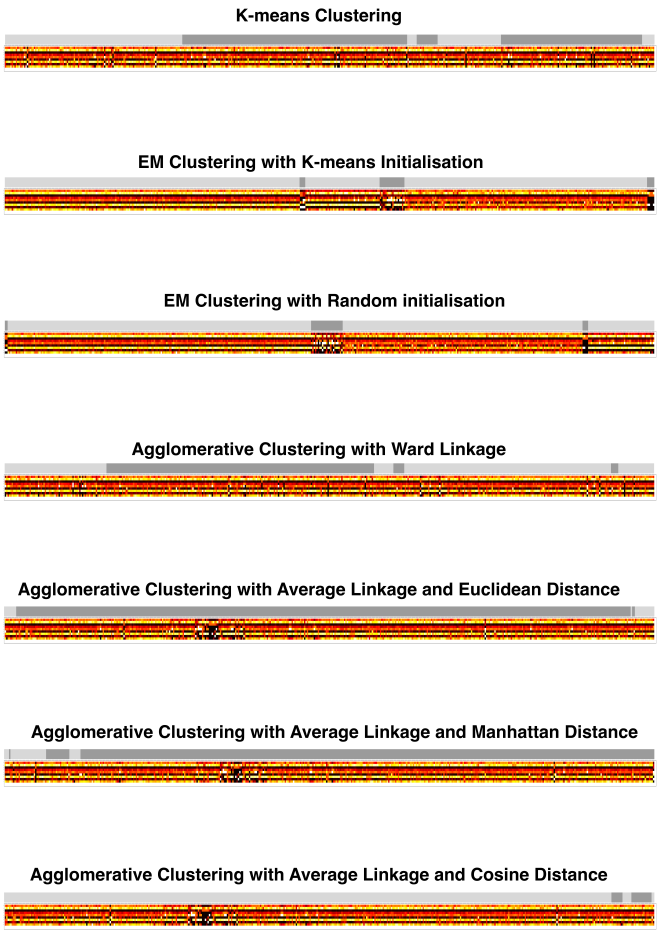
Figure A.1: The full feature-by-feature representation of the training dataset. Each feature is numbered according to the nomenclature expressed in 4.1.



B

Cluster Validity: Heatmaps

Figure B.1: The plotted heatmaps for each clustering method. The gray areas indicate the different clusters created.



C

Visualisation

We attempt to visualise our test data described in Chapter 4 using scatter plots. However, the results do not provide useful information for us to look further into this. We do deem it relevant enough to mention it our observations here, in the Appendix.

Having found the best clustering algorithm for our problem, namely EM clustering with K-means initialisation, we originally want to find a solution to visualise our clusters. The reason for this is that we would like to see if scatter plot visualisation can display anything intuitive where conclusions can be made on cluster shapes, cluster variation over time, and outliers. Our data spans to nine dimensions, and visualising nine dimensions is not something easy for humans. The first step to perform data visualisation is, therefore, a dimensionality reduction step.

C.1. Dimensionality Reduction

Geometrically, visualising nine dimensions is not something we can achieve. The observable world is in three dimensions, so we have to reduce the dimensions of our feature sets before being able to visualise it. Machine learning offers different techniques to perform dimensionality reduction, such as feature extraction, which identifies a set of principal variables in multi-dimensional space and reduces the dataset to these principal variables.

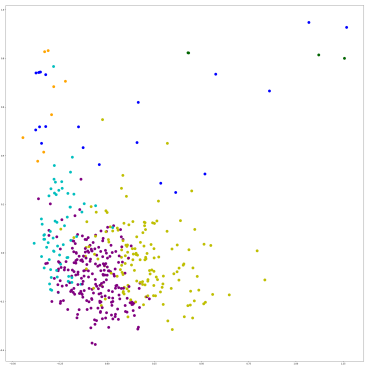
The reason why we do not apply dimensionality reduction *before* clustering is because we are not sure whether or not we would lose crucial information in the process. Additionally, a 9-dimensional feature space is not considered to be too large. It is then *safer* and without much overhead to cluster directly on the nine dimensions.

C.1.1. Principal Component Analysis

PCA is a linear feature extraction technique that looks for a specific number of *principal components* in the data [51]. It performs a linear mapping to lower dimensions in a way that the variance of the dataset is maximised. By doing so, it reduces the feature space but still maintains enough variance for the data to be *clusterable*, preserving the original variance of the dataset.

Figure C.1 shows the 2D scatter plot representation of PCA applied to our test data, after clustering with the EM algorithm and K-means initialisation step. What is noteworthy is that there seems to be one main crowd of three clusters and a few of outlying hosts belonging to the other three clusters. The purple, light green, and cyan clusters make sense to some extent. The blue cluster, however, does not appear to be a reasonable cluster in two dimensions. We suspect that, because the three outlying clusters are very small in comparison to the clusters in the crowd, we do not have enough hosts in these clusters to plot a sensible cluster. Perhaps with more hosts in these clusters, the visualisation would make more sense. In any case, the blue cluster still seems too sparse, and it may be that PCA loses information which would be useful for visualising a clearer distinction between our clusters.

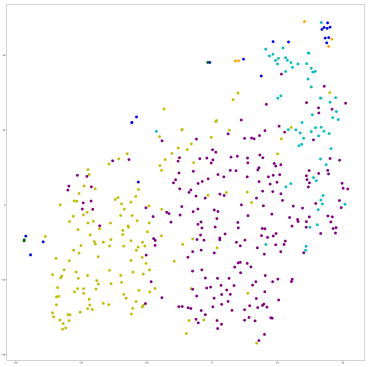
Figure C.1: Visualisation of the 9-dimensional test data reduced to two dimensions through Principal Component Analysis. Each dot represents a host and its colour corresponds to the cluster to which it belongs. In this plot, we can see one main crowd of clusters containing many hosts, while outlying clusters are more sparse.



C.1.2. T-Distributed Stochastic Neighbour Embedding

The t-SNE method was introduced to perform dimensionality reduction on a set of features in order to visualise multi-dimensional data [3]. T-SNE is an enhanced version of Stochastic Neighbour Embedding and is said to be much easier to optimise. This method calculates the similarity score between data points and tries to recreate this similarity score in smaller dimensions. Note that t-SNE may sometimes not be able to recreate this similarity and may have to make mistakes.

Figure C.2: Visualisation of the 9-dimensional test data reduced to two dimensions through the t-SNE method. Each dot represents a host and its colour corresponds to the cluster to which it belongs. Some parts of clusters in this plot are distinguishable, but most of it is just a big blob of hosts that do not seem to be *clusterable* in two dimensions.



We apply t-SNE to our test data and visualise our clusters using scatter plots, in two dimensions. Figure C.2 shows the t-SNE representation. We notice that some hosts may be clusters together through EM Clustering, but they are plotted far apart in their t-SNE representation. This could be an artefact from t-SNE when mapping nine dimensions to two. It would be interesting for further research to look into why t-SNE behaves this way with such data. One could look into the adjusting of t-SNE parameters to see which ones would fit best, or find out whether t-SNE is useful at all for this problem.

C.2. Setbacks

The visualisation of the test data in two dimensions through scatter plots has a few setbacks. We clearly see that PCA and t-SNE, despite their known effectiveness at reducing dimensions to be able to

visualise multi-dimensional data, do not bring us significant results to move further into the matter. We want to detect anomalous hosts among the network crowds and we would expect anomalies to look like outliers to our clusters [34]. We want to find a way to categorise a host as anomalous, and visualising our clusters of hosts does not seem to be the right approach.

D

The Road Not Taken

We considered different solutions to help us solve the problem and parts of these solutions have proven to be promising for future work, especially when looking into sole peer analysis and signal processing.

The first option we consider whether we can conclude anything about network activity, purely based on connectivity information on IP addresses to which the hosts in our dataset connect. The technique we briefly use is described in Section D.1. Furthermore, by interpreting connections between peers in the form of signals, with the amplitude of the connection over time, we explore the possibility to use signal processing for our research. This is further explained in Section D.2.

D.1. Peer Analysis

We want to verify whether we can say anything useful for network analysis, purely based on the IP addresses to which hosts connect. Note that, because of the nature of the Adyen network infrastructure, we do not see to where internal hosts connect when it comes to outbound connections. We only see connections made from hosts to internal servers.

To execute a brief peer analysis, we take a small sample of the original dataset described in 3. This sample is the network traffic generated by the laptops from Adyen security team members, on January 16, 2018, from 13:00 to 13:30. For each host and for each peer of that host, we extract two features from this sample set:

1. The total number of connections to that peer p is expressed by C_p.
2. The average volume v of connections to that peer p is expressed by v_p and calculated by:

$$\frac{\sum bytes_p}{\sum packets_p}$$

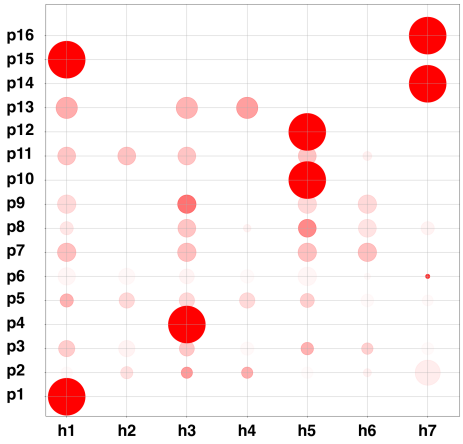
The following is an example of how the features are computed. Let h denote an arbitrary host with multiple peers $p1, ..., pn$. Host h communicated three times with $p1$ and sent the following amount of packets and bytes tuples: $[11 \rightarrow 112, 12 \rightarrow 8, 2 \rightarrow 20]$. Therefore:

$$C_{p1} = 3$$

$$v_{p1} = \frac{112+8+20}{11+12+2} = 5.6$$

We visualise this through *fingerprints* of connections per host to their peers in Figure D.1. Each entry on the x-axis is a host from sample set and on the y-axis are all the peers to which each host has connected. The results given by the two features for each host and for each of its peer are expressed by the size and colour of the circle at the corresponding host and peer coordinates.

Figure D.1: Fingerprints of connections per host for all peers. Each host inside the sample set is represented on the x-axis. These hosts are the laptops from each Adyen security team member. On the y-axis, we represent the peers that all hosts on the x-axis have visited in this 30-minute timeframe. Each circle represents the connectivity nature of the corresponding host on the x-axis to the corresponding peer on the y-axis. The size of the circle is proportional to the number of connections C_p to that peer. The intensity of the colour of the circle is proportional to the average volume of connection to that peer v_p.



The size of the circle is proportional to the total number of connections to a peer C_p, normalised by the total number of connections made to that peer by all hosts in the sample set. The intensity of the colour of the circle is proportional to the average volume of connections to that peer v_p, normalised by the total average connection volume to that peer by all hosts in the sample set. By normalising these two features this way, we obtain a relative view of the connectivity information to each peer for all hosts in the sample set.

Looking at Figure D.1, we can immediately notice the connection dispersion of each host. Some hosts, such as $h1$, $h3$, and $h5$ connect to a lot of different peers and are quite active in different subset of these peers. Hosts such as $h2$, $h4$, and $h6$ have very lightweight connections to few peers. Host $h7$ on the other hand has very few connections to otherwise often frequented peers and two very large and reoccurring connections to unvisited peers $p14$ and $p16$.

This observation could either mean that $h7$ is performing very specific tasks during this timeframe or it could be sending large amounts of data to a server unknown by the rest of the community of hosts. Looking at $p16$, we discover that it is, in fact, the server on which the development of Maypen happens, meaning $h7$ is mostly busy with development.

D.2. Signal Processing

In 2016, novel research for the detection of botnet behaviour focuses on frequency domain analysis, by interpreting network connections as signals [11]. This research inspires us to attempt interpreting peer-to-peer communication over different periods of time as discrete signal and perform frequency domain analysis to uncover different communication patterns.

We extend the sample set by taking the following four consecutive 30-minute chunks of network activity for the same hosts. The sample set now contains five iterations of 30-minute network activity of the Adyen security team. This is done such that we can build discrete signals over time for each host. Each signal represents a duration of communication of a specific amplitude at different time intervals from a host to its peer. We define the amplitude A of a signal s for host h to peer p at timestamp t as:

$$\frac{\frac{\sum bytes_p^t}{\sum packets_p^t}}{C_p^t}$$

For each host, we build a signal profile, consisting of all signals representing each communication from this host to its peers.

Figure D.2: Plot of a subset of communication signals for $h3$ over the extended sample set. The intervals of time are shown on the x-axis and the amplitude is plotted on the y-axis. Each signal is a curve of a specific colour, different from the rest.

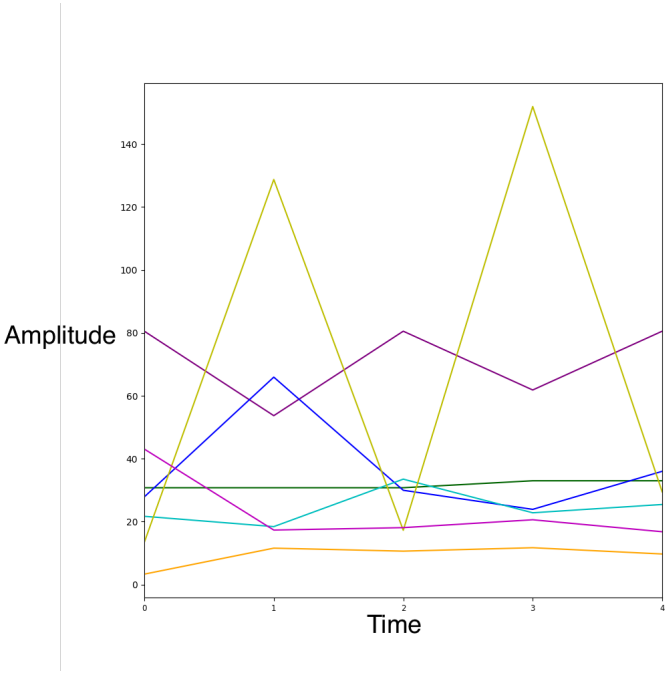


Figure D.2 shows a plot of the signal profile for host $h3$ in Figure D.1. We plot a subset of all communication signals in order not to crowd the representation. Each signal has a different colour.

Two signals, light green and purple, have a much higher amplitude than the ones that seem relatively constant at the bottom. They respectively correspond to $p8$ and $p4$ in Figure D.1. Based on Figure D.1, we can see that $p8$ is a peer that is visited by practically all hosts in our sample set. This perhaps means that the higher amplitude can be explained by the popularity of this destination IP address.

When looking at to which peer these communication signals correspond, we find that $p8$ is in fact a proxy server, through which outbound requests go. If $h3$ is, at these specific time periods, highly active on the Internet, then this observation makes a lot of sense. Peer $p4$ is a storage server. The high amplitude indicates that heavyweight connections were made to that peer, meaning that $h3$ is probably uploading large files to Adyen the storage server.

Despite the interesting and promising results of these analyses, they did were not enough related to the core problem of clustering for us to pursue the research in this domain. However, these results could prove useful for future work, especially when added to the main analyses done for this thesis. Frequency domain analysis using these signals could be the next step to take. The signal profiles yield useful information for more extensive host behaviour profiling.