h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbmn
FACHBEREICH MATHEMATIK
UND NATURWISSENSCHAFTEN

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt

Fachbereiche Mathematik und
Naturwissenschaften & Informatik

Anomaly Detection in Data Network Traffic with Machine Learning

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M. Sc.)
im Studiengang Data Science

vorgelegt von

Thomas Buck

Referent	:	Prof. Dr. Martin Stiemerling
Korreferent	:	Prof. Dr. Sebastian Döhler

Ausgabedatum	:	19. Juni 2020
Abgabedatum	:	4. Dezember 2020

DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 4. Dezember 2020

Thomas Buck

ABSTRACT

The detection of anomalies within network traffic has been a hot topic in the network security research community for the last decades. The immense growth of the internet and the ever-deeper reaching integration in our lives have emphasized the importance of network security. Significant effort has been put into the research of unsupervised anomaly detection due to the ever-changing threat landscape. However, despite the many proposed solutions, they are rarely adopted by the industry.

Network anomaly detection faces many different obstacles, such as lack of representative datasets, high cost of error, and the ever-changing nature of the traffic itself. While the last years have seen a surge of new public available datasets, the question of how to adapt to the dynamic nature of network traffic and security threats is rarely answered. This thesis explores the possible application of a dynamic sketching algorithm, the Robust Random Cut Forest, on current network traffic data. This unsupervised anomaly detection algorithm does not need prior training and can adapt to concept drifts within the data. With the utilization of hyperparameter-optimization, we investigate the detection capabilities in an emulated real-world scenario. The algorithm shows promising detection performance as it was overall capable of distinguishing between anomalies and normal instances and outperformed a comparable static model. However, the long runtimes and many false alarms restrict the application as a standalone solution. To tackle these issues we provide an outlook on possible adaptions where the dynamic anomaly detection capabilities can be utilized in the context of network anomaly detection.

ZUSAMMENFASSUNG

Die Erkennung von Anomalien im Netzwerkverkehr war in den letzten Jahrzehnten ein heißes Thema in der Forschungsgemeinschaft für Netzwerksicherheit. Das immense Wachstum des Internets und die immer tiefer greifende Integration in unser Leben haben die Bedeutung der Netzwerksicherheit unterstrichen. Aufgrund der sich ständig verändernden Bedrohungslandschaft wurden beträchtliche Anstrengungen in die Forschung zur unbeaufsichtigten Erkennung von Anomalien unternommen. Trotz der vielen Lösungsvorschläge werden sie jedoch nur selten von der Industrie übernommen. Die Erkennung von Netzwerkanomalien stößt auf viele verschiedene Hindernisse, wie z.B. den Mangel an repräsentativen Datensätzen, hohe Fehlerkosten und die sich ständig ändernde Natur des Datenverkehrs selbst. Während in den letzten Jahren eine Flut neuer öffentlich verfügbarer Datensätze zu verzeichnen war, wird die Frage, wie man sich an die dynamische Natur des Netzwerkverkehrs und die Sicherheitsbedrohungen anpassen kann, nur selten beantwortet.
In dieser Arbeit wird die mögliche Anwendung eines dynamischen Skizzenalgorithmus, des Robust Random Cut Forest, auf aktuelle Netzwerkverkehrsdaten untersucht. Dieser Algorithmus zur Erkennung unbeaufsichtigter Anomalien erfordert kein vorheriges Training und kann sich an Konzeptabweichungen innerhalb der Daten anpassen. Mit Hilfe der Hyperparameter-Optimierung untersuchen wir die Erkennungsfähigkeiten in einem emulierten Realwelt-Szenario. Der Algorithmus zeigt eine vielversprechende Erkennungsleistung, da er insgesamt in der Lage war, zwischen Anomalien und normalen Instanzen zu unterscheiden und ein vergleichbares statisches Modell übertraf. Die langen Laufzeiten und viele Fehlalarme schränken jedoch die Anwendung als Standalone-Lösung ein. Für diese Probleme, bieten wir alternative Lösungen an, da wir glauben, dass die intrinsischen Eigenschaften des Algorithmus im Zusammenhang mit der Erkennung von Netzwerkanomalien eine wertvolle Hilfe sein können.

CONTENTS

I THESIS

1 INTRODUCTION 2
- 1.1 Motivation . . . 2
- 1.2 Goal of the Thesis . . . 2
- 1.3 Structure . . . 3

2 FOUNDATION 4
- 2.1 Anomaly . . . 4
- 2.2 Anomaly Detection . . . 6
  - 2.2.1 Challenges . . . 6
  - 2.2.2 Approaches . . . 8
- 2.3 Metrics . . . 12
  - 2.3.1 Receiver Operating Characteristics (ROC) . . . 16
  - 2.3.2 Precision Recall Curve (PRC) . . . 17
- 2.4 Network Fundamentals . . . 18
  - 2.4.1 Network Topology . . . 18
  - 2.4.2 Network Traffic . . . 19
- 2.5 Data . . . 22
  - 2.5.1 Packet-level . . . 22
  - 2.5.2 Flow-level . . . 22
  - 2.5.3 Packet-level vs. Flow-level . . . 23
- 2.6 Intrusion Detection Systems . . . 24
  - 2.6.1 Overview . . . 24
  - 2.6.2 Components . . . 26
- 2.7 Related Work . . . 27
  - 2.7.1 Shortcomings . . . 28

3 EXPERIMENT 29
- 3.1 Benchmark Data . . . 29
  - 3.1.1 UNSW-NB15 . . . 31
  - 3.1.2 CIC-IDS2017 / CSE-CIC-IDS2018 . . . 32
  - 3.1.3 Summary . . . 33
- 3.2 Algorithm . . . 35
  - 3.2.1 Detection by Isolation . . . 35
  - 3.2.2 Isolation by Binary Trees . . . 35
  - 3.2.3 Robust Random Cut Forest . . . 38
- 3.3 Concept . . . 43
- 3.4 Data Preperation . . . 45
  - 3.4.1 Missing Values and Irregularities . . . 45
  - 3.4.2 Categorical Features . . . 45
  - 3.4.3 Data Overview . . . 47
- 3.5 Hyperparameters . . . 49
  - 3.5.1 Model Hyperparameters . . . 49

3.5.2 Scaling Hyperparameters . . . 49
3.5.3 Dimension Reduction Hyperparameters . . . 49
3.6 Hyperparameter Optimization . . . 51
3.7 Testbed . . . 53
3.7.1 Hardware . . . 53
3.7.2 Software . . . 53
3.8 Experiment Execution . . . 54
3.9 Runtime Performance Benchmark . . . 55
3.9.1 Single-Tree Benchmark . . . 55
3.9.2 Forest Benchmark . . . 57
3.10 Evaluation . . . 58
3.10.1 Hyperparameters . . . 58
3.10.2 Performance . . . 60
4 CONCLUSION AND FUTURE WORK 63

II APPENDIX
A UNSW-NB15 FEATURES 68

BIBLIOGRAPHY 71

LIST OF FIGURES

Figure 2.1 Two dimensional Anomaly Example. 5
Figure 2.2 Decision Threshold 14
Figure 2.3 Perfect Discrimination 15
Figure 2.4 Threshold maximizing True Positives 15
Figure 2.5 Receiver Operating Characteristics (ROC) 16
Figure 2.6 Precision-Recall Curve 18
Figure 2.7 Star Topology 19
Figure 2.8 TCP/IP model 20
Figure 2.9 UDP-Header 21
Figure 2.10 TCP-Header [16] 21
Figure 2.11 Host Intrusion Detection System 24
Figure 2.12 Network Intrusion Detection System 25
Figure 2.13 IDS components 26
Figure 3.1 UNSW-NB15 Testbed [42] 31
Figure 3.2 CIC-IDS2017 Testbed [29] 33
Figure 3.3 CSE-CIC-IDS2018 Testbed [30] 34
Figure 3.4 Binary Tree example Isolation of a Normal Instance . . 36
Figure 3.5 Binary Tree example Isolation of an Anomaly 37
Figure 3.6 Average Path Lengths over 1 to 500 trees 38
Figure 3.7 Tree Probability Example 40
Figure 3.8 Outlier Masking 43
Figure 3.9 Moving Average of Flows per Second (Window=60 sec) 47
Figure 3.10 Distribution of Transaction Protocols and Services . . . 48
Figure 3.11 Avg. Processing Times per Data Point (Raw Data) . . . 56
Figure 3.12 Avg. Processing Times (Preprocessed Data) 56
Figure 3.13 Total times (1000 insertions) for different Forrest Sizes. 57
Figure 3.14 Max PR/ROC AUC over 250 optimization Itereations . 58
Figure 3.15 ROC and PR Curve 61
Figure 3.16 Normal and Anomaly CoDisp D istribution 61
Figure 3.17 Attack Distributions 62
Figure 4.1 Example Architecture 65

LIST OF TABLES

Table 2.1 Confusion Matrix 13
Table 2.2 Protocol Examples 20
Table 3.1 Categorical Features 46
Table 3.2 Attacks during Test Timeframe 48
Table 3.3 Model Parameters 49
Table 3.4 Scaling Parameters 50
Table 3.5 Dimension Reduction Parameters 50
Table 3.6 Hardware Configuration 53
Table 3.7 Optimization Studies Sample (PR AUC) 54
Table 3.8 Optimization Studies Sample (ROC AUC) 55
Table 3.9 Baseline Comparison 60

Part I

THESIS

1 INTRODUCTION

We live in a time in which more and more parts of our lives are digitalized. We connect with various devices such as computers, smartphones, or tablets and are increasingly dependent on online services, as many social interactions, business matters, and financial transactions are handled over computer networks. The ongoing events only emphasize this trend as more and more people work from home and generally spend more time online [68]. The dependence on the digital infrastructure highlights the importance of network security in the private and public sectors as vulnerabilities or malicious activity can harm a network's integrity and compromise sensitive data. Of particular interest in the field of network security is the detection of anomalies. The anomaly detection is usually discussed in the context of network intrusion detection systems (NIDS). Despite the substantial research effort put into network anomaly detection, most adopted solutions still rely on the signature-based techniques [61]. This discrepancy can be explained by multiple factors, such as the high error rate and the associated costs, the lack of publicly available realistic network traffic data as well as the dynamic nature of network traffic. Especially the dynamic nature of network traffic is an issue that is often overlooked [63]. Network traffic can not be regarded as static as it changes over time and between networks. In this thesis, we investigate the possible use of a dynamic model that, in theory, can conquer those issues by adapting to the ever changing nature of network traffic.

1.1 MOTIVATION

This thesis is part of the research project ANEMO (Automatisiertes Netzwerkmonitoring). The Hessen Agentur GmbH supports the project within their funding program LOEWE3. In this project, the Institute for Computer Science of the University of Applied Science Darmstadt cooperates with the DICOS GmbH [13] to combine concrete requirements from the private sector with the latest academic knowledge in the fields of network technology and machine learning to provide an automated network monitoring solution.

1.2 GOAL OF THE THESIS

There has been substantial research effort in the field of network anomaly detection in the last decades. Most of the research has been primarily focused on machine learning approaches. However, the approaches have been rarely adopted by the industry [63]. Anomaly detection faces multiple challenges which we describe in Section 2.2.1 in further detail. However, due to the reviewed research, we have concluded that two of the main challenges

are the lack of transferability and the ever changing nature of network traffic. To tackle those issues, we investigate the application of a dynamic sketching algorithm, the Robust Random Cut Forest [24]. This unsupervised algorithm does not need prior training and has the ability to adapt to concept drifts. This thesis investigates the anomaly detection capabilities of the algorithm on realistic network traffic data and provides the groundwork for further possible applications in the problem domain of network anomaly detection.

1.3 STRUCTURE

Chapter 2 covers the necessary foundations for the network anomaly detection task. Anomalies are defined and the groundwork for Anomaly Detection is explained along with the metrics used for evaluating the performance of an anomaly detection system. Section 2.4 provides the necessary network basics followed by the standard ways to represent network data.In Section 2.6 we describe the intrusion detection systems, primarily focused on network anomaly detection followed by an overview of the current research along with the identified shortcomings.
Chapter 3 starts with the presentation of the reviewed benchmark data, followed by the description of the used anomaly detection algorithm, the Robust Random Cut Forest. Section 3.3 explains the concept of the conducted experiments followed by a more detailed discussion about the components. Section 3.8 describes the process of executing the experiments along with the encountered problems followed by a closer investigation of the main issue, the long runtimes. Section 3.10

2

FOUNDATION

2.1 ANOMALY

The term anomaly has its etymological roots in the ancient Greek word "anomalos" and is translated to uneven or irregular [80]. Today the term is widely used to describe patterns which do not conform to normal or expected behavior and is utilized in various application domains such as biology, astronomy, geology, medicine, and many more. Anomalies are also often referred to as outliers, aberrations, irregularities, or novelties. Outliers and anomalies are the terms most used in the context of machine learning. The process of identifying such deviations from the norm is referred to as anomaly detection or outlier detection, respectively. While those terms are often used interchangeably, the process of outlier detection is more often associated with the data cleansing process where the goal of eliminating anomalous samples is to improve model performance, while anomaly detection is more often used when the anomalies themselves are the point of interest, their nature, and the possible causes. The possible causes are of particular interest as they can be of critical nature where interference is necessary, such anomalies could be caused by malicious intent (credit card fraud, network intrusion detection), system failure of critical systems (e.g. predictive maintenance), or in the medical context (malignant tumors shown in mri scans) [11]. Because of this reason, Anomaly detection is a critical part of various fields such as fraud detection concerning credit cards or insurance, fault detection in critical systems, or intrusion detection in the context of cyber-security.

The definition was extended with two, especially in the context of data mining and machine learning, important characteristics [23]. The two characteristics are:

- Anomalies are different from the norm with respect to their features.
- They are rare in the dataset compared to normal instances.

The second characteristic already opens up another question, what is considered rare? There is no set in stone threshold, but a rule of thumb says that there should not be more than 5% anomalous samples in the data [22]. Sticking to the definition of anomalies as patterns that do not conform to normal behavior, we can differentiate between different anomalies on an abstract level. To further clarify, we have picked examples in the context of network traffic. Those examples are chosen for simplicity and might not be considered an anomaly of interest in a real-world use case.

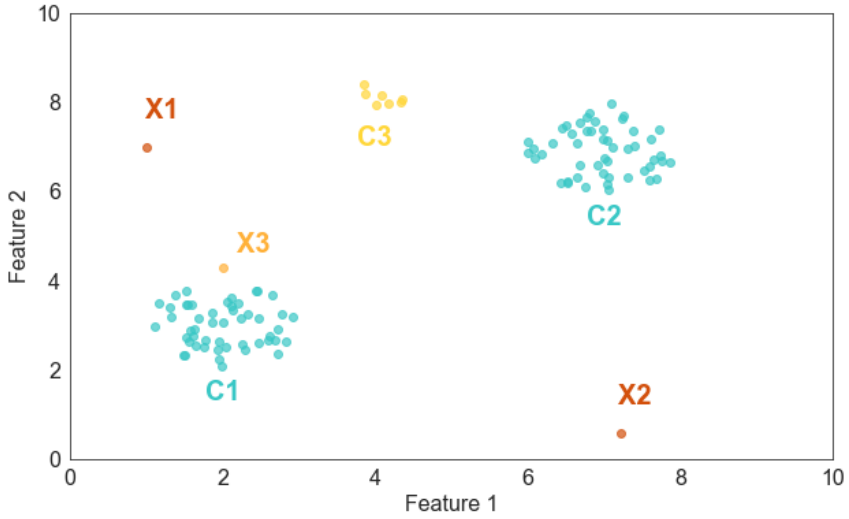

Figure 2.1: Two dimensional Anomaly Example.

Figure 2.1 shows the different kinds of anomalies illustrated in a two-dimensional dataset with Feature 1 and Feature 2. Without knowing any background information about the nature of the data at hand, we can define the normal regions in cluster C1 and C2 solely on the basis that most data points lie within them. With the normal regions defined, we can see the following anomalies within the dataset:

POINT/GLOBAL ANOMALIES are single instances that show clear deviation from the norm concerning their features, Fig. 2.1 shows two examples, namely X1 and X2. Most of today's anomaly detection algorithms are primed on identifying point anomalies [23]. In network traffic, a point anomaly could be any network packet with malicious intent, such as an unauthorized login attempt.

LOCAL ANOMALY is a kind of anomaly which appears to be a part of the normal instances when viewed in relation to the whole dataset. However, if focused on the normal cluster it belongs to, it can be categorized as an anomaly within that cluster. We can see such an anomaly, as shown in Fig 2.1 by point X3 and cluster C1. If a local anomaly should be regarded as an anomaly of interest depends on the task at hand.

COLLECTIVE ANOMALY/MICRO CLUSTER introduces another level of complexity to the anomaly detection task, as anomalies are no longer limited to single instances. One such cluster is shown in Fig. 2.1 labeled C3. Other than the illustration suggests, the single point in a collective anomaly does not need to be anomalous themselves, but the connection between those instances form the anomaly. To further illustrate this with an example, a single request to one port of a host should

not be considered an anomaly, but many of those requests to different ports could be a sign of an ongoing port scan attack and thus form a collective anomaly.

CONTEXTUAL ANOMALIES can not be illustrated, as they are not anomalous without the right context. Considering the network of an office. A high number of HTTP requests during a lunch break can be considered normal. The same number of requests in the middle of the night is not normal and might constitute an anomaly. The normal behavior was made anomalous with the right context, in this example, time. The necessary context needs to be introduced to the data using contextual features [11]. These features need to be designed in a way that algorithms can determine if a sample is anomalous based on the contextual features given.

While point anomalies can be in any dataset, a relationship between samples needs to be present for the occurrence of collective anomalies, and context needs to be set to detect contextual anomalies. Point and collective anomalies can be transformed into contextual anomalies if the context is given [11].

2.2 ANOMALY DETECTION

Anomaly detection is the process of identifying anomalies within the data. This process has been studied intensively for decades, and many different reviews, surveys, and books have been published. Diverse research fields have been tackling the anomaly detection problems like statistics, machine learning, information theory, and more. Hodge and Austin provide an extensive survey focusing on statistical and machine learning models [27], another structured overview of the different approaches was published by Chandola, Banerjee, and Kumar [11]

2.2.1 *Challenges*

We defined an anomaly as a pattern which does not conform to normal behavior. This notion can be applied to single data points and groups where each member is not an anomaly, but their joint appearance. Nevertheless, how do we know what anomalous behavior is? By intuition, one could argue that we first need to know what normal behavior constitutes to differentiate. While this seems logically sound, it proves to be quite tricky in practice. There are many challenges that anomaly detection faces, with different severity depending on the application task. We will relate those challenges to the network traffic environment. Some of these challenges are:

1. **Encompassing every form of normal behavior**. While this task might be easier in static systems where the boundaries are clearly defined, it

is especially complicated in a computer network environment. This task quickly becomes overwhelming as networks deal with a great amount of variability. Diverse network technologies and different operating systems are in use while communicating through numerous applications and protocols. Paxson and Floyd formulated this problem concisely while discussing the difficulties in simulating the internet: "IP buys uniform connectivity in the face of diversity, not uniform behavior" [49]

2. **The notion of normal.** Even a single network can show high variability in bandwidth, duration of the connections, applications, and protocol use [49]. Networks can show even greater variability when compared to each other, as not only will the usage profile differ but also the network topology, load, and vulnerabilities. An anomaly detection system that works great in one network might not show the same performance in another. For example, the network of a military base will show vastly different characteristics compared to a single-family home network, and the analyst of one might be interested in different kinds of anomalies than the other. However, not only do the anomalies of interest change, an anomaly detection system primed on one network might perform worse when used in another as the notion of normal does not align.

3. **The application domain is not static.** This problem arises when the notion of normal changes over time, which is especially prevalent in the domain of computer networks. Even properties which one might assume to be statistically unchanging can change in a brief amount of time [49]. This effect becomes obvious considering the emerging technologies and the different ways we use the internet now compared to just a few years ago. The internet has shown faster growth than any other comparable technology platform [28]. However, even viewed on a scale of seconds to hours, the composition and quantity of network traffic can change. This variability not only affects packet-level features but extends to application-level features as well [63].

4. **Unavailability of labeled data.** To validate and measure the performance of an anomaly detection approach, correctly labeled data is necessary. This data should exhibit real-life properties and be up to date to the current standards. The difficulty to reliable capture or generate realistic network traffic, which is correctly labeled, still proves to be a difficult task, and although many new datasets were published in the last years, much research still relies on outdated data [26]. The network security research community has been facing a lack of publicly available data sets [2]. We discuss this issue further in Section 3.1.

5. **High cost of error.** A more detailed explanation of the different kinds of errors such a system can produce is discussed in Section 2.3. To shortly summarize, a basic anomaly detection system can make two

kinds of errors. Incorrectly label an anomalous data point as normal and labeling a normal data point as an anomaly. To apply this concept to a network example, an anomalous data point that passed the detection system undiscovered could indicate an ongoing attack that breaches network security. On the other hand, a normal data point marked as an anomaly calls for further investigation, for example, by a network administrator. The huge amount of traffic packets which pass through even a small network can quickly render an anomaly detection approach useless if too many normal packets get flagged [63].

These are some of the challenges which anomaly detection tasks face. They are especially prevalent in the network environment as the ambiguity and changing nature of network traffic combined with the difficulty to produce correctly labeled and representative datasets makes this task very difficult in practice.

2.2.2 *Approaches*

To tackle an anomaly detection problem, we can choose from various techniques. Most of these techniques have been designed to solve a specific problem formulation. The problem formulation is the nature of the data, the type of anomalies we are interested in, the availability of data, and the output we desire [11]. The application domain and problem formulation, therefore, narrow down the suitable approaches. Multiple surveys on anomaly detection techniques have been conducted. Hodge and Austin provided an extensive survey, comparing motivations and advantages of statistical, neural network, and machine learning based methodologies [27]. An in-depth analysis of anomaly detection algorithms based on a categorization of the techniques' underlying approach has been published by Chandola, Banerjee, and Kumar [11]. Markou and Singh present an overview of techniques based on statistical methods and neural networks with a focus on novelty detection [37]. While a categorization of approaches should not be the deciding factor in choosing a technique, understanding the underlying assumptions is necessary to know which technique can be applied to the problem formulation. In the further discussion of the problem formulation, we will reference example techniques that are affected or the underlying assumption made by the approach.

2.2.2.1 *Nature of the Data*

Most anomaly detection algorithms rely on the data being mapped on to vectors, where each vector represents one sample in the multidimensional feature space, consisting of the values of the observed attributes. These values can be numeric (discrete, continuous) or categorical (nominal, ordinal). Samples can consist of only one attribute (univariate) or multiple attributes (multivariate). In the multivariate case, each attribute can be of the same type or a mixture of multiple types. In the context of network traffic, an ex-

ample of a continuous numerical attribute could be the connection duration, while the protocol being used is a nominal categorical attribute. The nature of the attributes is essential for the applicability of an approach as most techniques require the attributes to be of a specific type or make assumptions about their distributions. For example, in most statistical approaches, different models have to be used for numerical or categorical attributes [11].

In general, most implemented techniques require the attributes to be numeric. The conversion of categorical attributes into a numeric representation might require additional preprocessing steps and needs to be done mindfully. For example, proximity-based techniques (techniques that rely on calculating a distance measure between data points) need to establish a meaningful distance measurement between data points to detect anomalies. The euclidean distance or L2 norm ($d(p,q) = \sqrt{(p-q)^2}$), for example, can be suitable for continuous attributes but lacks interpretability and meaning for categorical attributes. To further clarify this with an example, assume the protocol attribute from the example above, utilizing the numeric representation as defined by the Internet Assigned Number Authority (IANA) [31]. Assuming we are inspecting three data points with the attribute protocol being 6 (TCP), 17 (UDP), and 1 (ICMP), respectively. Calculating the distances between those data points using the equation above would not yield a meaningful result, and can also convey a false sense of relationship between the points. Distance-based techniques rely on the closeness of data points to distinguish between normal instances and anomalies. The data point with 6 (TCP) and 1 (ICMP), in the above example, would be considered to be closer together than 6 (TCP) and 17 (UDP) even though there is no real-world basis for that assumption.

Issues like these can be overcome by using a suitable distance measurement, meaningful conversion into numeric values, or choosing an algorithm that employs a scale-invariant distance measure to the attributes, such as Isolation Forest [12]. However, each data transformation comes at the cost of processing time. The applicability of algorithms which need complex data transformations is thus limited for time-sensitive detection tasks [27].
Another factor that can significantly affect the processing time and make it more challenging to distinguish anomalies from normal instances is the number of features. Many algorithms like k-NN, neural networks, minimum volume ellipsoid, and many more are susceptible to the so called 'Curse of Dimensionality'. With increasing dimensionality, the data points are spread through a larger volume and are less dense, making it more challenging to establish a convex hull [27]. Another issue which can arise with increasing dimensionality is that the proportional distance between the closest point and the one farthest away vanishes [8]. This concentration effect limits the applicability of approaches that employ a distance measurement to distinguish anomalies from normal instances, as this hinders the discrimination between near and far points [79].

2.2.2.2 *Data Availability*

Obtaining accurate, realistic, and ideally noise-free data to fit an anomaly detection approach is a complicated and expensive task in many application domains. This difficulty is especially prevalent in network anomaly detection as further discussed in Section 3.1 and gets more severe if labels are necessary. Labels are used to denote if each sample belongs to the normal instances (usually represented as 0) or is anomalous (1). If the anomalies are divided into different classes, those labels can be extended to be unique for each class. Those two scenarios can be seen, in machine learning terms, as a binary classification task and a multi-class classification task, respectively. Getting accurately labeled data which encompasses every possible anomalous behavior is often not feasible and becomes prohibitively expensive as labeling often requires human experts and thus substantial effort and time investment [11].

Besides the more difficult acquisition of data with accurate labels it also extends our possible techniques which we can employ. We can categorize the different techniques into three distinct strategies for which we will use the standard machine learning terminology in the following explanations.

SUPERVISED ANOMALY DETECTION - This approach is analogous to the above-described classification task and requires labeled training and test data. Having data labels allows for many different classification algorithms to be used like decision trees (C4.5), support vector machines (SVM), or artificial neural networks (ANN). It is important to note that anomaly detection tasks are usually highly imbalanced with a majority of normal instances and rare anomalies, unlike most classification tasks. Decision trees are not suitable for unbalanced data while support vector machines should perform better [23]. Classifiers are usually best fit for static data as they need to be retrained when the data distribution shifts and should cover the entire distribution of the data to allow for good generalization by the algorithm [27]. However, knowing every possible anomaly, a priori is rarely the case and limits the usefulness of this approach [23].

SEMI-SUPERVISED ANOMALY DETECTION - also requires labeled data as only normal instances are used for training. This approach's intuition is that we build a normality model with the training data and classify data points that deviate from this model as anomalies. The training data would still ideally encompass every form of normal behavior to permit generalization but does not need prior knowledge about the occurring anomalies [27]. One-class classification algorithms can be applied to this strategy, such as one-class SVMs or Autoencoders, or any density estimation method that can model the probability density function of normal instances like Gaussian Mixture Models [23]. It is important to note that although the required prior knowledge is less than in the supervised case, acquiring data without any anomalies can be

considered nearly as difficult. Anomaly free data requires an enclosed environment as internet traffic contains many packets that should be regarded as an anomaly by a detection system [34]. However, designing an enclosed environment that accurately reflects network traffic is a complicated task in itself.

UNSUPERVISED ANOMALY DETECTION - The unsupervised approach is the most flexible as no prior knowledge about the data is necessary, which eliminates the necessity for labels and training data. The idea is that the distinction between normal and anomaly can be made solely based on the intrinsic properties of the data [22]. These techniques make the implicit assumption that anomalies are a rare occurrence in the data. They can suffer from a high false alarm rate and thus hinder their applicability in real-world scenarios [11]. In theory, unsupervised anomaly detection methods would be the best fit for the network anomaly detection task as no training data is necessary, which allows the application in many network environments as discussed in Section 2.2.1. Furthermore, not needing prior knowledge about the anomalies allows for novel anomalies to be detected. It is important to note that although no labeled data is needed for the algorithms' training, it is still needed for the validation of the approaches.

When considering the availability of data, it is important to not only view it from a model fitting or training point of view but also take the application domain into account. Many application domains are time-sensitive, such as fault detection of critical systems or network intrusion detection. The time between anomaly occurrence and detection/alert needs to be minimized.
In scenarios like these, the data arrives sequentially, and the algorithm needs to detect anomalies in an online matter. Preprocessing steps and the algorithm need to be selected carefully to accommodate that demand. Proximity-based methods, for example, suffer from exponential computational growth as the computational complexity is directly proportional to the dimension of the data and the number of data points [27].
Without modifications, these methods are not suitable for network anomaly detection as gigabit networks can pose an immense computational workload which needs to be processed as quickly as possible. The biggest issue with a high data volume is that a low false alarm rate can amount to many false alarms, rendering such a detection useless in a real-world application [63].
Parametric methods scale better with big data sets as the model complexity does not grow with data size. Parametric approaches assume that the data comes from a family of known distributions. Parameters are calculated to fit the distributions to the data. However, the underlying distributions are rarely known; therefore, those methods are only applicable where we have prior knowledge about the distributions [37].

2.2.2.3 *Output*

An essential part of anomaly detection algorithms is their output. After all, this output should indicate if a data point is to be considered an anomaly. Usually, the outputs of the algorithms can be classified into two categories, labels, and scores. Algorithms that return a label return the corresponding predefined class label, for example, 0 for normal instances and 1 for anomalies. Many of the supervised techniques output a class label as they use existing classification algorithms [23]. Deriving a class label is often internally based on scores or probability estimates. Methods that return scores are more informative as they transport a degree of abnormality [23]. Scores allow for the incoming data to be ranked to either choose an appropriate threshold to turn the output into alerts or, for example, utilize the ranking to only consider the top ranked data points as anomalies.

To cover all pitfalls of selecting an appropriate algorithm would be too much for this work's scope; however, those are some main points that need to be considered. While some are set in stone, many can be overcome by either fitting the model to the data or fitting the data to the model and the task. For example, Ramaswamy, Rastogi, and Shim proposed a modification to the k-NN algorithm where the data is divided into cells to reduce the number of distances that need to be calculated [53]. The data can also be transformed to fit the model by converting categorical variables into numeric representation, transforming the data, or reducing the dimension with feature selection or reduction.

2.3 METRICS

A suitable metric is needed to quantify, evaluate, and compare an anomaly detection algorithm's performance. If a metric is suitable depends on multiple factors such as class distribution, number of different classes, and desired properties of the metric.
We can simplify the decision of an anomaly detection task into a binary classification problem. At one point in the pipeline, a decision is made if the analyzed packet/flow is to be considered an anomaly or not. As previously discussed in Section 2.2.2.3, we differentiate between a score output and a class output of an anomaly detection algorithm. However, even if the algorithm outputs a real number score, we need to decide at one point if the packet/flow in question should be further investigated or not. This can be done by a different model, thresholds, or security analyst.

This notion of a binary problem is also analogous to how the benchmark data is structured as described in Section 3.1. A packet/flow is either an anomaly or a normal instance. For now, we will disregard the possibility of a graded anomaly classification. To choose an appropriate metric, it is essential to discuss the four different outcomes an anomaly detection system can produce. This information is often presented as a confusion matrix, as

seen in Table 2.1. This matrix allows us to conceptualize the different cases quickly and is often used in binary classification tasks.

		Actual	
		P	**N**
Predicted	**P′**	True Positive (TP)	False Positive (FP)
	N′	False Negative (FN)	True Negative (TN)

Table 2.1: Confusion Matrix

This matrix's rows represent the algorithm's predicted values, while the columns represent the actual classes. We denote anomalies as *Positive* cases (P) and normal instances as *Negative* (N), later a numeric representation is chosen with 1 and 0 respectively.
This representation shows all possible interactions between the actual instance classes and the output of the algorithm's prediction. The different cases are:

- **True Positives TP** - Anomalies that are labeled as anomalies.
- **False Positives FP** - Normal instances that are labeled as anomalies (Type I error).
- **True Negatives TN** - Normal instances that are labeled as normal instances.
- **False Negatives FN** - Anomalies that are labeled as normal instances (Type II error).

It is worth discussing the two types of error such a system can make in further detail as the consequences and costs can be severe in a network anomaly detection setting. *False Positives*, in statistical hypotheses testing referred to as *Type I error* [70], also referred to as false alarms occur when a normal instance is falsely classified as an anomaly. The consequence of such an error is that further investigating of the issue is initiated without merit. False Positives or False Alarms will use up human resources as the flagged traffic needs to be verified. This becomes especially troublesome as huge volumes of traffic are transferred through a network.

False Negatives, in statistical hypotheses testing referred to as *Type II error* [70], or miss, occurs when an anomaly remains undetected. This can

have many outcomes, but the more severe cases are network attacks being undetected, harming the network's integrity, and possibly revealing private data to attackers.

$$True\ Positive\ Rate\ (TPR) = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNR \tag{2.1}$$

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - TNR \tag{2.2}$$

$$True\ Negative\ Rate\ (TNR) = \frac{TN}{N} = \frac{TN}{TN + FP} = 1 - FPR \tag{2.3}$$

$$False\ Negative\ Rate\ (FNR) = \frac{FN}{P} = \frac{FN}{FN + TP} = 1 - TPR \tag{2.4}$$

From the information presented in the confusion matrix, it is possible to conduct the ratios shown in Equation 2.1 - 2.4. These ratios can also be interpreted as probability estimates. For example, the *TPR* (Equation 2.1), also referred to as sensitivity, recall or hit rate, can be interpreted as the probability that the model will classify a positive instance as positive. The ratio of the falsely classified normal instances to all normal instances is referred to as the false positive rate (*FPR*) or the probability of false alarm. Equation 2.3 and Equation 2.4 have equivalent interpretations for the *Negative* cases. The *True Negative Rate* is also called specificity.

An anomaly detection system set to achieve high sensitivity (TPR) will offer the best protection. If the priority is to maximize specificity (TNR), the system will be more efficient [72]. Therefore, an anomaly detection system aims to maximize the correctly labeled instances (TP, TN) while minimizing the errors (FP, FN).

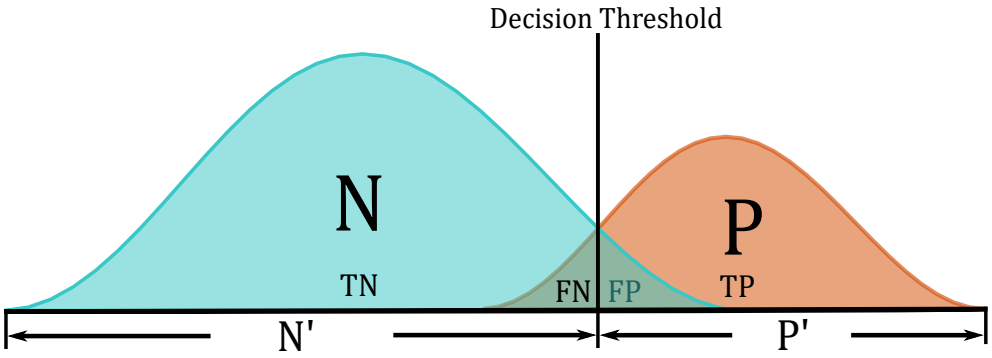

Figure 2.2: Decision Threshold

Although we need to decide at one point in the detection process if a sample is an anomaly or not, we can utilize a score output (Section 2.2.2.3) to get a sense of the overall discrimination capabilities of an approach. Figure 2.2 illustrates the interaction between the prediction scores distribution and the chosen decision threshold. Assume an anomaly detection algorithm

that maps samples to an anomaly score. Smaller values correspond to normal instances, while higher values indicate anomalous behavior. The decision threshold is the point along this axis, which will classify every sample below it, as normal and every sample above it, as an anomaly. Unless the model can entirely separate the instances, as shown in Figure 2.3, there will be a trade-off between the true positives and false positives and true and false negatives.

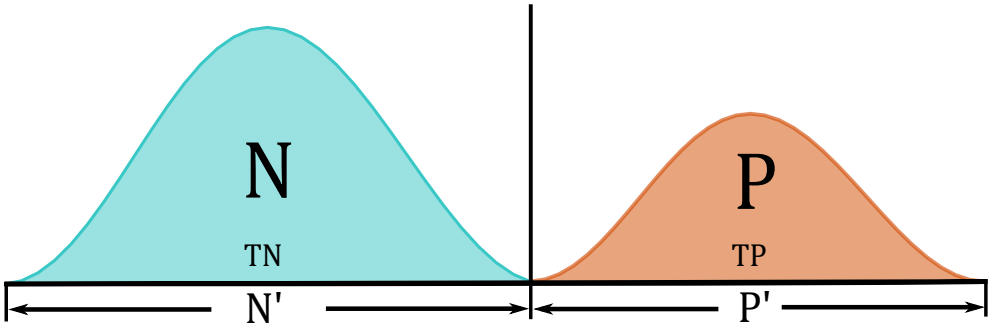

Figure 2.3: Perfect Discrimination

This dilemma is inevitable and should be thoroughly considered as the choice of the decision threshold is not only a matter of detection performance but also dictated by the application domain, the severity of missed anomalies and how false alarms are handled.
Figure 2.4 shows the impact of a threshold set to maximize the True Positives. While this choice uncovers all anomalies, it will cause the most false positives. As stated in Section 2.2.1, this is a big concern of network anomaly detection as handling false positives is expensive and time-consuming.

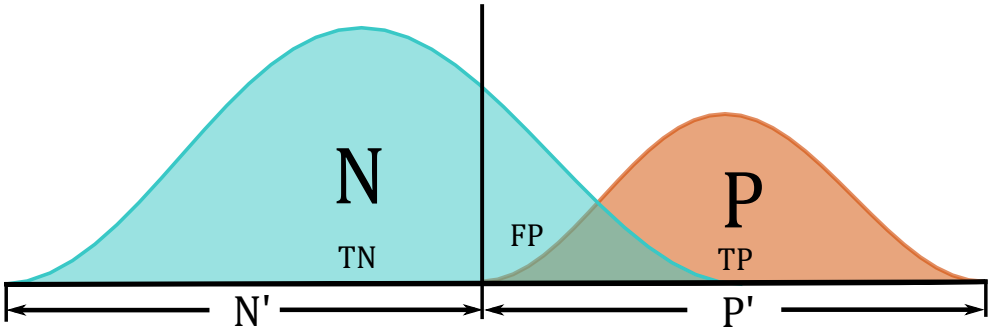

Figure 2.4: Threshold maximizing True Positives

Another important point to consider when choosing a performance metric is the class distribution. In anomaly detection and especially unsupervised anomaly detection, one of the prior assumptions is that the anomalous instances a rare compared to the rest of the data (Section 2.2.2). This causes a class imbalance. An often-used performance metric in classification tasks is the *Accuracy* which is defined as seen in Equation 2.5. It is the ratio of overall correctly classified instances over all instances. However, this heavily skews in favor of the majority class. For example, if 100 samples are to be analyzed for anomalies, including 99 normal instances and one anomaly -

an algorithm predicting normal instances for every sample would achieve an accuracy of 99%. It is not providing a meaningful way to measure the performance of an anomaly detection system.

$$Accuracy\,(ACC) = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.5}$$

2.3.1 *Receiver Operating Characteristics (ROC)*

The Receiver Operating Characteristics (ROC) analysis originates from signal processing theory but is applicable in many different domains such as medical diagnosis, radiology, data mining, or machine learning [9]. In the anomaly detection context, it is a concise way to visualize the relationship between the true positive rate and false positive rate across different threshold values.

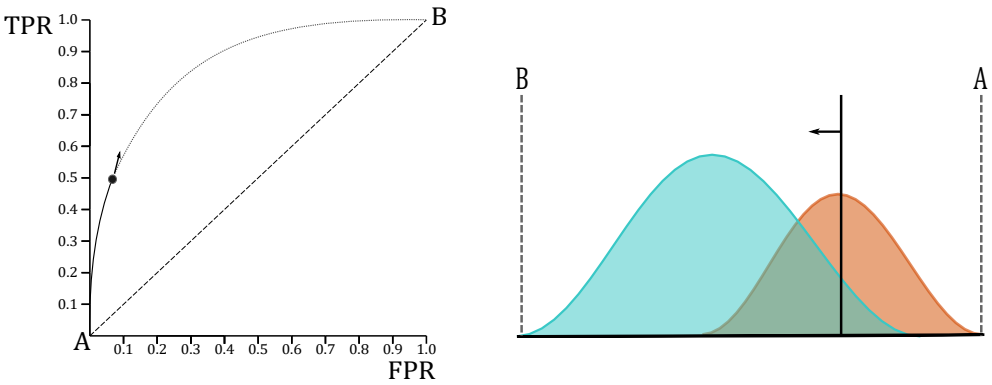

Figure 2.5: Receiver Operating Characteristics (ROC)

The ROC curve is constructed by sliding the decision threshold over the anomaly scores (from maximum to minimum) and evaluating the TPR and FPR at each point. Figure 2.5 shows an example of this process. When the threshold is set to the maximum output of the algorithm, it will classify everything as negative, resulting in zero TP and FP. This is indicated by the decision threshold A on the right side of figure and point A in the coordinate system. The scores are then evaluated in descending order in the same fashion until the minimum is reached, where consequently every sample is classified as an anomaly resulting in a true positive rate and a false positive rate of 1.0, indicated by B. The threshold value itself is not present in the ROC plot. The diagonal line shows a random classification of anomalies and normal instances. An algorithm that would produce a ROC curve along this line would not separate the data. Therefore the ROC curve is always above this line.
The ideal ROC curve would start vertical till 1.0 TPR while still being at 0.0 FPR. Only after reaching 1.0 TPR, the FPR would increase, the scores shown in Figure 2.3 would result in such a ROC curve. The ROC curve provides a visual way to assess how well an algorithm can separate anomalies from normal instances and can help evaluate an anomaly detection algorithm's

performance without the need to specify a specific threshold while showing the relationship between two key metrics for an anomaly detection system.

2.3.1.1 *AUC*

The *Area Under Curve (AUC)* is a way to compress the ROC curve into a single number between 0.5 and 1. The maximum value is achieved when the model perfectly separates anomalies from normal instances as in the example above (Figure 2.3). Quantifying the ROC curve in a single value makes it possible to compare different models and use this metric for optimization purposes. It is important to note that it is not possible to reconstruct a ROC curve from the ROC AUC value. When comparing two models with the ROC AUC value, the higher ROC AUC value model is better at separating anomalies from normal instances when all thresholds are considered. However, under some circumstances, a model with a smaller AUC may perform better than a larger AUC model.

2.3.2 *Precision Recall Curve (PRC)*

Precision and Recall are two metrics often used in information retrieval. An information retrieval system's task is to return relevant documents to a query cast by a user. Recall is the proportion of found relevant documents over all relevant documents in the search space. Recall is indifferent from the already defined TPR/Sensitivity in Equation 2.1.
Precision is also called Positive Predictive Value (PPV) and is, to stay in terms of information retrieval, the ratio of the correctly labeled relevant documents, over all as relevant labeled documents. This concept is easily transferred to the context of anomaly detection or binary classification as Precision gives an estimate of how accurate the predictions of the detection system are. Precision is defined as shown in Equation 2.6.

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

As Recall and Precision need a threshold to be set in order to be evaluated, a similar curve to the ROC curve can be constructed by varying the threshold. In the Precision-Recall curve, the Recall value is on the x-axis while the Precision value is on the y-axis as seen in Figure 2.6. With Recall being the same as the TPR in the ROC curve, the key difference is between FPR and Precision. The denominator of False Positive Rate (Equation 2.2) are the negative cases and is consequently affected by a class imbalance.

As stated in Section 2.2.2, unsupervised anomaly detection requires the anomalies to be rare within the data causing the FPR to be damped by the high number of negative cases. Precision's calculation is more concerned with Positive cases, as the negative cases are only present as the ones that are falsely classified (FP) and are therefore less influenced by the class imbalance making the Precision-Recall Curve the better fitting visual tool for imbalanced data [55].

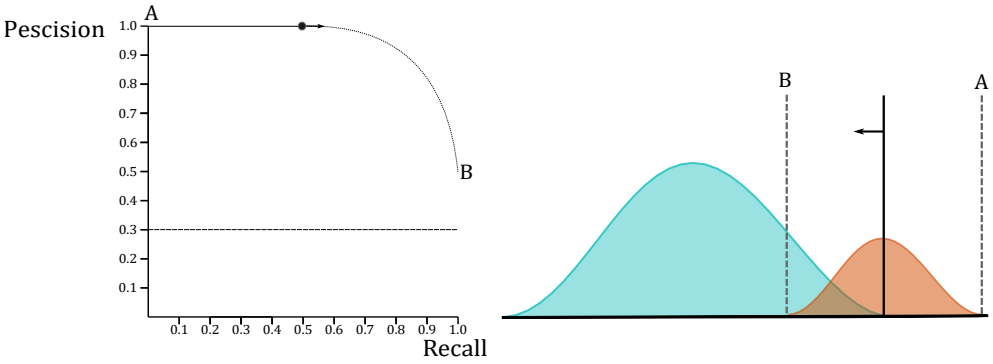

Figure 2.6: Precision-Recall Curve

Unlike the ROC curve, the baseline is not fixed. It is dependent on the proportion of positives within the data ($\frac{P}{P+N}$). To illustrate this with an example, assume a balanced data set with an equal amount of positive and negative cases and an algorithm that cannot discriminate between the classes, consequently labeling every instance as an anomaly. In the balanced case, the algorithm's predictions would be correct 50% of the time, resulting in a precision of 0.5 overall decision thresholds. Similarly, with a 70% negative cases, the output would only be accurate for 30% of the instances resulting in a precision of 0.3, as shown in the example Figure 2.6. The graph is constructed similarly to the ROC graph. The decision threshold is being moved across the output scores from A to B, evaluating Precision and Recall at every value. Point A on the left-hand graph is (0,1) by definition as the actual value is undefined. Unlike the ROC curve, the PRC does not encapsulate the information over all threshold values as the focus lies primarily on the positive class. An algorithm with perfect discrimination (see Figure 2.3) of the classes would result in a PRC where the curve is a straight line from (1,0) to (1,1). Similarly, the area under the PRC can be used as a condensed metric to compare different algorithms, analogous to the ROC AUC.

2.4 NETWORK FUNDAMENTALS

A network is a set of connected devices that communicate among themselves. These devices can be computers, workstations, servers, smartphones, or any digital device with the necessary capabilities to connect to a network. The devices need to follow the rules to enable successful communication; these rules are called protocols. The following will cover the fundamentals of networking necessary for the performed experiments.

2.4.1 *Network Topology*

A network topology is the arrangement of the devices within a network. A network topology can be either physical or logical. Physical represents the network devices' arrangement, while a logical topology represents how the data flows [9]. There are many different ways network devices can be ar-

ranged, the most simple being two connected devices. Basic topology setups include bus, ring, star, mesh, and tree. However, we will only be focusing on a star topology as this represents the setup of the experiment data as well as being one of the most used topologies today.

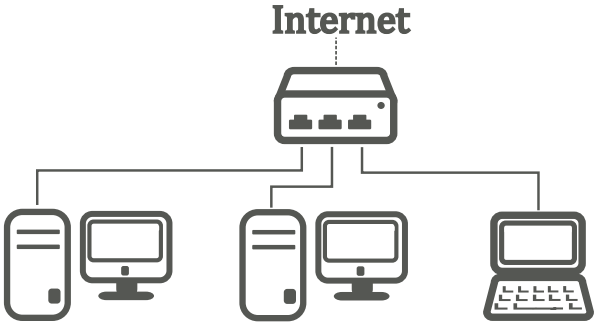

Figure 2.7: Star Topology

A central controller defines a star topology, such as a switch, hub, or router, where all communication between devices and external sources is routed through. All outgoing or incoming traffic from the internet is also routed through this component, functioning as a single gateway. In terms of network anomaly detection, this topology enables to gather and analyze the traffic at a single access point, the gateway.

2.4.2 *Network Traffic*

Network traffic is the communication present in a network at a given point in time. A single element of this communication is generally referred to as a packet or frame, depending on the viewpoint. It is crucial to understand the core concepts on how packets are formed as this data provides the basis for all following analysis. This work is primarily concerned with the traffic to and from the internet. Therefore the basic concepts used to construct network traffic will be explained using the internet protocol suite, the TCP/IP model.

The TCP/IP model consists of four layers, with each of the layers providing a set of services to the layer above with the application-layer providing services to the end-user as seen in Figure 2.8. The corresponding protocols implement these services. A protocol defines the meaning and the format of messages, packets, and frames. The following description illustrates this interplay between network layers using services and protocols.

- The **Application-layer** is the interface to the end-users. For example, the most prominent internet application, the WWW, can be accessed by the hyper transfer protocol (HTTP).
- The **Transport-layer**, also known as a host-to-host layer, allows for logical connections to be made between devices and to exchange messages. These messages can, for example, be exchanged reliably via the

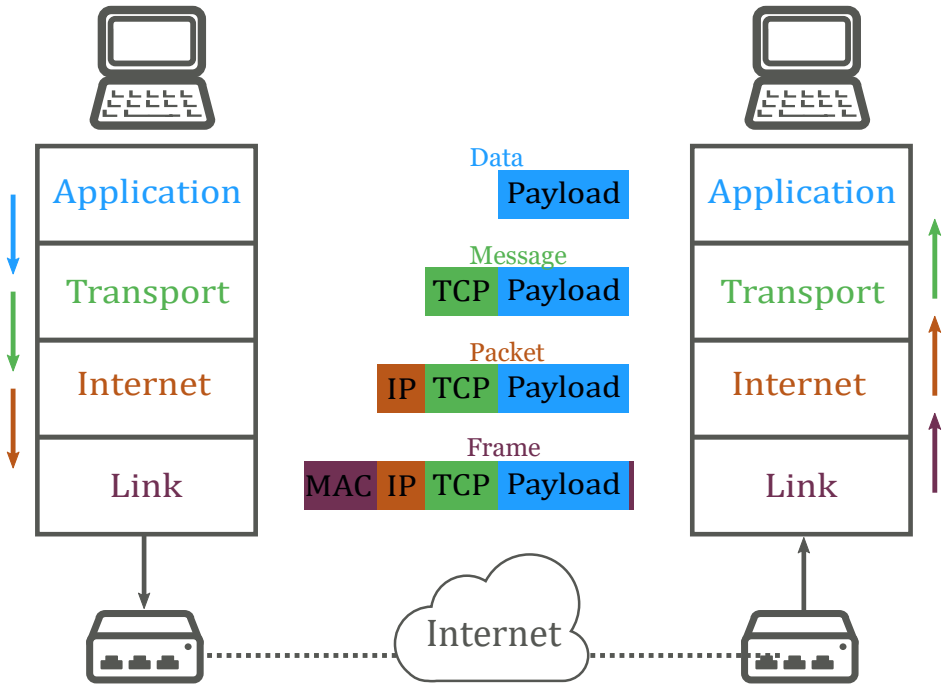

Figure 2.8: TCP/IP model

Transfer Control Protocol (TCP) or unreliable using the User Datagram Protocol (UDP).

- The **Internet-layer (IP-layer)** handles addressing of source and destination and routing across the internet. The Internet Protocol (IPv4 and IPv6) fulfills this by assigning internet devices with a unique IP-address.
- The **Link-layer**'s protocol stack consists of protocols that are only focused on the communication between adjacent network devices. For example, the communication between the clients and the router.

Next to the above-mentioned protocols, many more protocols are included in the different layers with a selection provided in Table 2.2.

Layer	**Protocol**
Application	HTTP, HTTPS, SMTP, POP3, SSH, ...
Transport	TCP, UDP, SSL, ...
Internet	IPv4, IPv6, ICMP, ...
Link	Ethernet, 802.11, IEEE 802.16, ...

Table 2.2: Protocol Examples

For network traffic analysis, the protocols' and services' technicalities are not as important as the structure of the data and the layer of interest. As the

Link-layer protocols only provide data of the low-level communication between two linked devices, the information with the most value can be found on the IP-layer. This layer provides all information about the connection to outside sources and contains all the information added to the packet by the layers above.

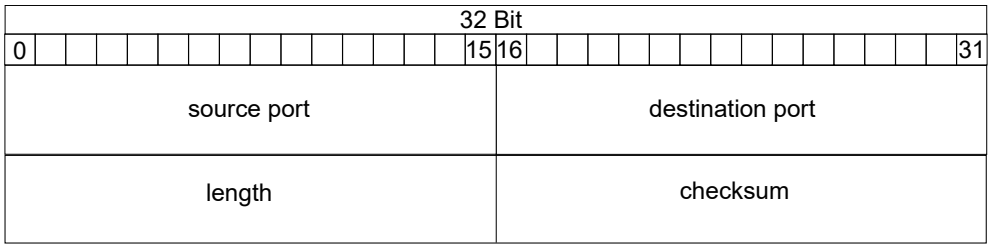

Figure 2.9: UDP-Header

Each layer adds a header to the data provided by the layer above, as seen in Figure 2.8. Different protocol headers provide different information. Figure 2.10 shows the TCP-header. The header contains the source and destination port numbers along with necessary information to enable a reliable connection. The UDP protocol, which can be used for unreliable connections, provides less information as seen by the corresponding header in Figure 2.9.

Although the concrete values of each protocols' header are secondary, the fact that each packet can contain a different set of information is crucial when anticipating the application of machine learning models. Basically, each packet can consist of a different set of features that needs to be accounted for when used for machine learning.

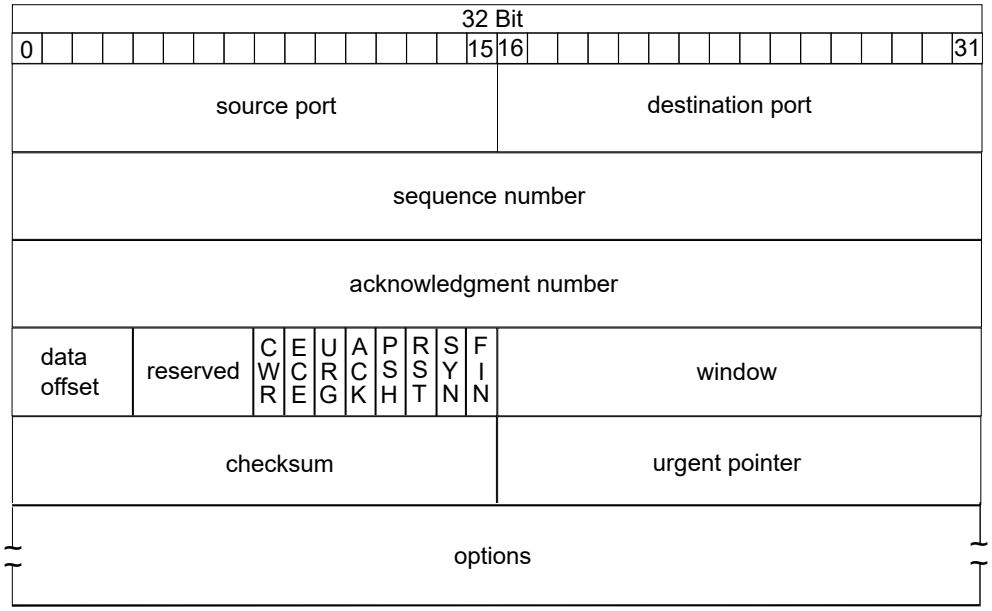

Figure 2.10: TCP-Header [16]

2.5 DATA

There are two main ways network traffic data is captured: either in a packet-based or a flow-level format. Capturing the traffic within a network is usually done by port mirroring. This technique is implemented on network devices like switches or routers. The essential concept behind this technology is that the network traffic is copied at the networking device. The original traffic is forwarded to its destination, while the copy is delegated to a device, where the data is stored and processed.

2.5.1 *Packet-level*

The main way to achieve packet-level capture is by utilizing the pcap (packet capture) API. Unix based operating systems implement pcap in the libcap [64] library. For Windows two options are WinPcap [74] (no longer maintained) or Npcap [46], both being based on the libcap library. Monitoring software can be used in conjunction with these libraries to capture network traffic packets on the IP-layer as described in Section 2.4.2. One of the most prominent capture tools is tcpdump [64], it is a command-line packet analyzer which, for example, has been utilized in the generation of the UNSW-NB15 dataset [42]. The produced files are currently the de facto standard in public network traffic archives [9]. Next to tcpdump there are countless other software products which make use of this API, Nmap [45] a port scanning utility, Snort [62] an intrusion detection system or Wireshark [75] which is similar to tcpdump but includes a graphical front end.
As previously stated, the basic principle behind packet capture is to make an exact copy of the entire network traffic, including payload. While this allows for the most exhaustive investigation as no information is lost, it also presents some difficulties. The composition of a network packet is highly dependent on the protocols being used, as they define the structure of the packets and their content. In machine learning terms, the features of the data vary with the used protocol stack [10]. This variability in each packet's provided features needs to be treated with either additional preprocessing steps or model adaptation. Raw packet capture does not contain contextual information about the network status or the relationship between packets. It is also worth noting that storing and analyzing packet-based network traffic can become quite difficult. Growing network sizes, speeds, and access points increase the computational effort necessary to process the data. Additionally, payload encryption often limits the usefulness of investigating the payload [41].

2.5.2 *Flow-level*

Flow-level capture was first introduced and implemented by Cisco as a router feature called NetFlow [17]. Since then, many other manufactures have enabled their devices with similar flow monitoring technology, such

as sFlow and IPFIX [82][52]. In the case of NetFlow a device needs to be specified via IP address and UDP port to function as a NetFlow collector. On the collector, a tool like nfdump [21] can be used to store the flow data. An alternative option to flow-level capture is to convert packet-capture into flow format using tools like Argus or CICFlowmeter which were used in the generation of the benchmark datasets described in Section 3.1. However, it is essential to note that a conversion from flow-level to packet-level is not possible.

Flow data is more encapsulated, as it does not contain any raw payload data and focuses more on meta-information about network connections [54]. The basic concept behind flow capture is that the corresponding router checks each incoming package for a series of flow defining attributes. A standard set of attributes are the source/destination IP addresses, the source/destination ports, and the transport protocol [54]. If the incoming package's attributes match a flow created before (within a time window), the flow will be updated with the newly gathered information. Otherwise, a new flow is created. Next to the flow defining attributes, the flows contain statistical features such as number of transmitted bytes, number of transmitted packages, duration [54]. Network flows can be classed into two types, uni-directional flow, and bidirectional flow. Assuming client A communicates with server B, in the uni-direction case, packets sent from A to B are aggregated into a flow, and packets from B to A are aggregated into a different flow. In the bidirectional case, the whole communication is aggregated into a single flow.

In many publicly available data sets, the flow data is extracted from packet-level data or enriched with additional features that the flow capture technologies do not provide. However, as this work's focus is not on the gathering process, we do not differentiate the pure flow-capture from the extracted or enriched flow-level data.

2.5.3 *Packet-level vs. Flow-level*

An excellent metaphor for the differences between packet-level and flow-level data is given by Wheelus et al.: *"A good example of the difference between captured packet inspection and NetFlow would be viewing a forest by hiking through the forest as opposed to flying over the forest in a hot air balloon."*[73]. With this analogy in mind, it is easy to see that both ways offer a different perspective, and some information provided by one technique might not be provided by the other. From the literature reviewed, it was not apparent if there is a proven optimal way to present the data for network anomaly detection. However, one could argue that some kind of aggregation is necessary to combat the vast volumes of data and the computational effort necessary for analysis in an online matter [41]. One of the big advantages of packet-level data is the presence of payload data; however, this is negated when end-to-end encrypted protocols are used. Flow-level data reduces the com-

putational requirement and also offers some distinct advantages over packet-level capture. Aggregating packets into flows with additional statistical features provide metadata. This metadata is more descriptive about the network characteristics than the raw packet capture and can be further enriched by deriving features that capture inter-flow behavior to extract valuable context features (Section 2.2). Because of the smaller computational footprint and the provided statistical features, this work will focus solely on using flow-level data.

2.6 INTRUSION DETECTION SYSTEMS

The majority of network anomaly detection research is done in the context of intrusion detection systems (IDS). An IDS is an additional network security layer with the objective to identify, classify, and alert malicious activity. Intrusion detection systems have been a focus of the network research community for over two decades as one of the first landmark papers was published in 1987 by D. Denning [14]. The author presented a ruled-based pattern matching system, based on statistical analysis and user profiles, to detect a wide range of intrusions. One key finding of this work and one of the reasons why intrusion detection remains an integral part of network security today is that these systems can help identify malicious activity without knowing the network/system's underlying vulnerabilities. IDS can be classified by the system they are intended to protect or by the taken detection approach.

2.6.1 *Overview*

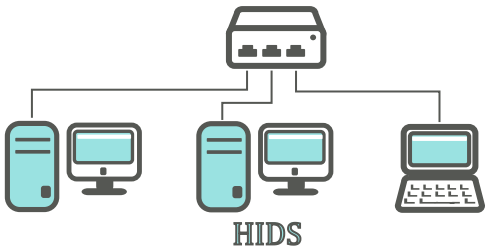

Figure 2.11: Host Intrusion Detection System

Host Intrusion Detection Systems (HIDS) run on the system they are intended to protect. Running on the system has the advantages of many different information sources being available to monitor for suspicious activity, such as log files, system objects, processes, and regions of memory. The closeness to the system often allows the HIDS to identify processes or users involved in the suspicious activity such as changes to the system registry or installation of a backdoor [51]. Many security violations are caused by malicious code and unauthorized events. Having a HIDS closely monitor all the activities on a system can help avoid unwanted access and secure a system [56]. However, HIDS will use the system's resources, possibly limiting performance. Another issue is that attacks cannot be identified until the attack

reaches the target system, and thus too late.

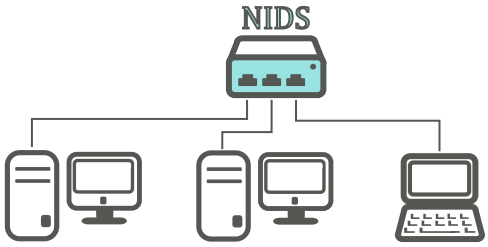

Figure 2.12: Network Intrusion Detection System

Network Intrusion Detection Systems (NIDS) are used to monitor a whole network or network segment. NIDS only access the network data, e.g., packet capture or flow data to analyze for suspicious activity. The usual process is that all the network traffic at a capable device (switch, router, gateway), which connects the network to the internet, is copied and mirrored to a port where additional hardware is connected that runs the NIDS software. A NIDS has less information at its disposal to identify intrusions than a HIDS. However, a NIDS can detect intrusions before they reach the target system. It can also identify network-wide attacks; for example, a large number of TCP requests to many different ports and devices in a short amount of time could indicate the occurrence of a port scan attack [9]. As the NIDS monitors all the connected devices' traffic data, it can also help uncover attacks from within the network.

Misuse Detection, also called signature-based detection, identifies intrusions based on known signatures. A database is populated with the signatures of known intrusion, which the traffic is compared against. An alert is cast if known signatures are detected. The attack signatures can be gathered in different ways, such as machine learning or expert knowledge. Misuse detection systems generally have a low false-positive rate and a high accuracy in identifying known attacks. However, they can not reliably detect unknown attacks and need to be updated regularly in order to maintain an up-to-date database of known attacks [39].

Anomaly Detection based IDS follow the principles of anomaly detection described in Section 2.2.2. It is implicitly assumed that intrusive behavior is different from normal behavior. A profile of normal behavior is constructed, and deviations from this profile are identified as anomalies. Anomaly detection mechanisms go along with the described approaches in Section 2.2.2. These systems can detect known and novel attacks. However, they often suffer from a high false alarm rate [36].

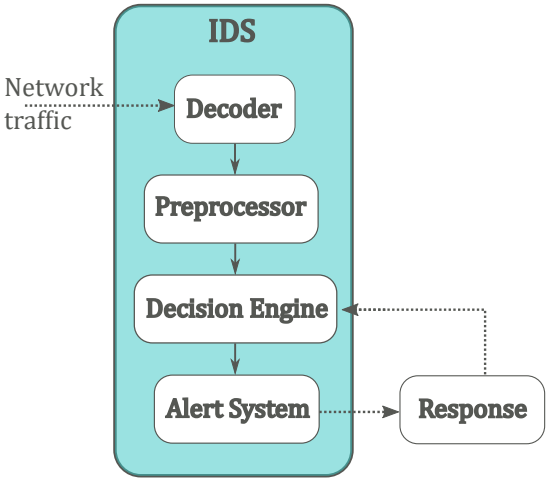

Figure 2.13: IDS components

2.6.2 *Components*

Intrusion detection systems are designed and implemented in many different ways; however, they generally use the four main components shown in Figure 2.13. In the following, the concepts of the four components are explained on the example of an anomaly-based NIDS [36].

- The **Decoder** gathers the network data making use of, for example, libcap in the case of packet capture or NetFlow when using flow-level data. A more in-depth description is provided in Section 2.5.
- The **Preprocessor** prepares the data for the decision engine. This can include many different data processing steps such as aggregation, feature selection, feature generation, or transformations. The preprocessing steps are chosen to fit the decision engine's required input and achieve the best detection accuracy.
- The **Decision Engine** analyzes the data from the Preprocessor and scores or labels the data points based on the employed techniques. These techniques can stem from different fields, such as statistics, machine learning, or expert knowledge. Further explanation of different approaches regarding anomaly detection is discussed in Section 2.2.2.
- The **Alert System** receives the labels/scores of the decision engine and raises an alert if an intrusion or suspicious activity is detected. The decision engine can generate many alerts, including true alerts and false alerts - the alert systems' role is to analyze these alerts, pinpoint the location and supply all the necessary information to the security administrator. The alert system can include steps such as alert merging, intention recognition, or alert correlation [9].

Classification of intrusion detection systems can help to summarize the operating principles and the different roles they fill, yet it is easy to see that

best protection can be achieved if strategies from all categories are employed. However, this work is primarily focused on the decision system regarding anomalies. Most research regarding network traffic anomalies is focused on intrusion detection systems which is reflected in the benchmark data. The anomalies present in those datasets are attacks and exclude other possible network anomalies such as misconfiguration or device failure. For the purpose of this work, intrusion detection data sets will be used as a proxy for anomaly detection.

2.7 RELATED WORK

Network anomaly detection has been a hot topic for the network research community for over two decades. The majority of research is done in the area of Intrusion Detection Systems (IDS) where it was kicked off in 1987 by Dorothy E. Denning with the landmark paper "An Intrusion-Detection Model" [14].
The author presented a comprehensive model combining the behavior of subjects and objects, along with the usage profiles and statistical models to identify anomalous behavior of a computer system. The model is based on the hypothesis that the exploitation of a system's vulnerabilities involves abnormal usage of said system. Decades later, this assumption is still a primary basis for IDS systems. Denning also provided a key finding, which is one of the reasons why Intrusion Detection Systems are still the main focus of network security research. These systems can identify anomalies without knowing the specific vulnerabilities of the underlying system.
Today, intrusion detection systems can be classified into misuse and anomaly detection systems, with the latter being used to identify novel anomalies. This part of network security research overlaps greatly with the research of outlier detection as many outlier detection algorithms have been applied to network data, with the techniques originating from fields such as machine learning, statistics, or information theory [11][27].
Tree-based techniques have been applied in a number of different ways. Zhang, Zulkernine, and Haque presented a complete IDS solution, including misuse and anomaly detection based on the Random Forest algorithm. The authors applied the supervised learning capabilities of the random forest algorithm to identify known attacks while also building a model on the prediction of the used service to establish a proximity measure in order to identify novel anomalies [78]. Marteau, Soheily-Khah, and Béchet proposed an adaptation of the Isolation Forest for intrusion detection by adding a centroid based distance measure as well as the ability to use the distance to known anomalies to further improve the detection [38]. Kim, Lee, and Kim presented a hierarchical intrusion detection system utilizing decision trees and SVMs. The decision trees were used for misuse detection of known attacks and decomposing the normal traffic into smaller subsets. These subsets are then used to train support vector machines in order to derive the specific normal profiles and thus classifying the points which deviate from those pro-

files as anomalies [33]. Countless other different approaches have been published utilizing unsupervised and supervised machine learning techniques such as k-NN, SVM, or ANNs [26] with no particular strategy being established to be superior for network anomaly detection [10].

2.7.1 *Shortcomings*

A survey conducted in 2018 by Hindy et al. found that 97.25% of the surveyed research utilize machine learning to identify intrusions [26]. However, these methods are rarely adopted by the industry. This discrepancy of real-world application and research was reasoned by multiple authors. Tavallaee, Stakhanova, and Ghorbani surveyed 276 studies and concluded that the majority could not hold up to scientific standards, partly because of the overall lack of scientific rigor in the field of experimental computer science research. In particular, the study found one main contributor to this issue being the lack of appropriate benchmark data. The public datasets often do not represent normal traffic as well as being too large to be evaluated in total. This leads to researchers sampling the data, which not only distorts the data but also hinders the comparability [66]. Gates and Taylor are challenging the validity of the anomaly detection approach by questioning the underlying assumptions. The authors state that there is neither a guarantee for anomalous traffic to be different from normal traffic in regards to the feature, nor can it be confidently assumed that anomalous traffic is rare. The authors also noted that the research rarely specifies how updating of the detection model should be done, leading to the implicit assumption that network traffic is static, disregarding the changes in network traffic composition and concept drifts [18].

3

EXPERIMENT

3.1 BENCHMARK DATA

Researchers rely on realistic and correctly labeled data to evaluate, validate, and compare anomaly detection solutions. The critical nature of many anomaly detection tasks requires a rigorous evaluation and approval of the approach as a failure when applied can cause substantial damage. In the scope of network anomaly detection, researchers use intrusion detection (Section 2.6) datasets consisting of normal traffic and network attack traffic representing the anomalies. For this evaluation and comparison, benchmark data is needed. The data should ideally be non-anonymized and publicly available. However, one of the network research community's most significant problems has been a lack of up-to-date benchmark data [63][2].

One hurdle of publishing benchmark data is the sensitive nature of network traffic. Network traffic can reveal sensitive information, like private communications, confidential business information, or user access patterns - the data can reveal more than expected. For example, Yen et al. were able to identify the browser a client uses from flow data [77]. Wright et al. succeeded in identifying the spoken language from encrypted VoIP data [76]. The immense risk and legal barriers of publishing such data have been a significant hindrance for researchers to share their data [63].

Another issue with benchmark data for network anomaly detection is that unlike other application domains where the possible normal behavior is better understood and defined, one benchmark dataset cannot encompass every possible variance of normal behavior. The ever-changing composition of network traffic, network topology, and the emergence of new intrusions and security vulnerabilities also causes benchmark data to have a relatively short period in which they are up-to-date [43].

With the last point in mind, it is quite interesting that the go-to benchmark dataset for network anomaly detection research was published over 20 years ago. The DARPA dataset was generated at the MIT Lincoln Laboratory (1998) by simulating a military LAN network with normal and anomalous traffic [1]. The dataset includes four gigabytes of raw tcpdump that spans over seven weeks and includes around five million connection records [32]. The KDD CUP 99 dataset was used for "The Third International Knowledge Discovery and Data Mining Tools Competition" and is a modification of the DARPA dataset. The data was prepared in tabular form for a more straightforward application of detection algorithms, it includes basic

attributes about the TCP connections and high-level attributes like failed login attempts [54].The majority of the data, however, are redundant records. This issue was discussed and resolved by Tavallaee et al. with the introduction of the NSL-KDD dataset [65]. Although these iterations solved some of the issues since the first introduction of the dataset, the main problem still stands as the data does not represent real-world traffic and is outdated in both network structure and attacks [58]. A big part of the network security research still uses one of the derivates of this data to evaluate their approaches. A survey from 2018 showed that of the 64 research papers (2008-2018) surveyed, over 85% still use some variation of the DARPA data for their evaluation [26].

However, the lack of suitable benchmark data has been noticed, and many new datasets have been published over the last years. For example, the Canadian Institute for Cybersecurity (CIC) located at the University of New Brunswick published the datasets ISCXIDS2012 [59], CIC-IDS2017 [57], and in collaboration with the Communication Security Establishment (CSE), CSE-CIC-2018 [58] dataset. The Australian Defence Force Academy at the University of New South Wales published the UNSW-NB15 dataset [40]. A comprehensive survey of Network-based Intrusion Detection data sets is provided by Ring et al. [54]. As previously stated, the variety in network topology, network traffic, and user behavior prohibits a one-for-all type data set for evaluating network anomaly detection approaches. There is no guarantee that a previously on benchmark data trained and evaluated algorithm will perform well in the environment it is intended for [40]. However, as this data is needed for evaluation and proof of concept, multiple data sets should be used to ensure generalizability [15].
While it is not feasible for a network traffic data set to contain the full spectrum of all possible usage patterns, Sharafaldin et al. and Viegas, Santin, and Oliveira proposed a similar set of rules which the dataset should follow to be considered as valid [57] [71]. The most important features a data set should exhibit are:

- **Realism** - The data should contain realistic network traffic as expected in a production environment, as well as realistic attack scenarios.
- **Validity** - The data should contain well-formed packets/flows and include the complete interaction.
- **Labels** - Complete and especially correct Labels are required.
- **Variability** - The data should exhibit diversity in protocols, services, client behavior, and anomalies.
- **Sensitive Data** - The data should not reveal private data in order to be shareable.

3.1.1 *UNSW-NB15*

The UNSW-NB15 data set was generated in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) [5]. The IXIA's Perfect Storm ONE was utilized, as it can simulate real-world normal and malicious traffic and can handle 1 to 10 Gbps (Gigabytes per second) of network traffic [41]. Perfect Storm ONE can generate traffic for more than 150 web applications such as Skype, Google, Facebook services, etc. It can also simulate more than 28000 live malware attacks [41].

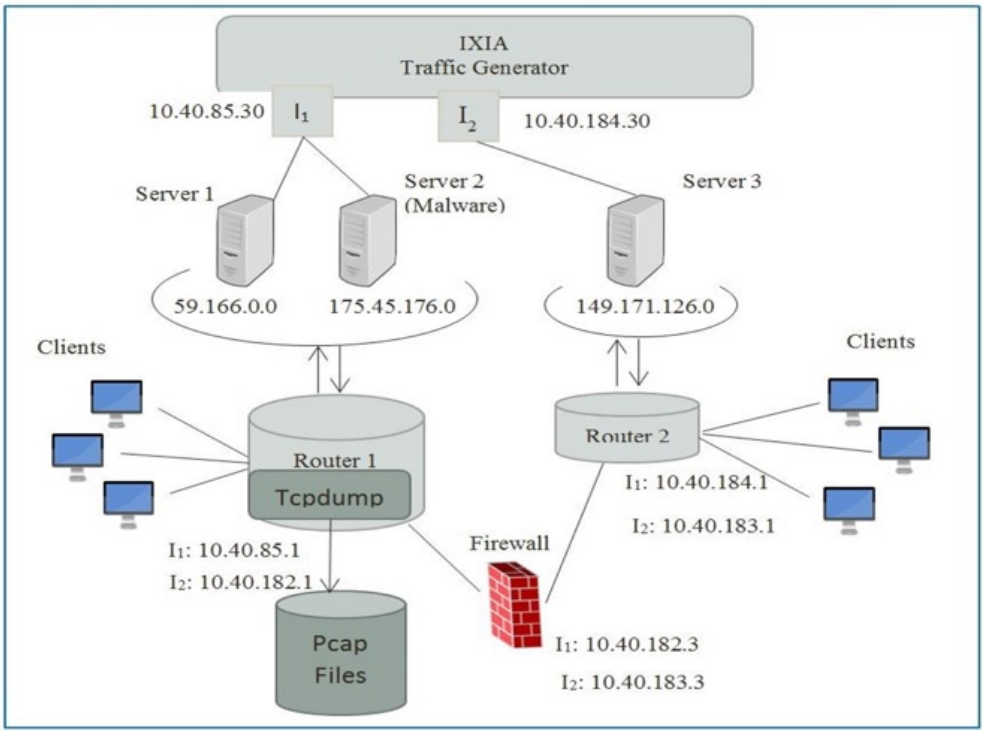

Figure 3.1: UNSW-NB15 Testbed [42]

The IXIA tool was used to generate normal and abnormal traffic and to disperse it through network nodes (Clients, Routers, Servers). An overview of the testbed is shown in Figure 3.1. All traffic was also passed through a firewall, showing that the included security events could not be detected by a traditional signature-based tool [42]. The data includes nine different attack scenarios over the two day time frame.

Two simulations were conducted, generating 50 GB of pcap files each. The first simulation had a runtime of 16 hours, while the second simulation ran for 15 hours. Argus, Bro-IDS and 12 algorithms for feature extraction were used to transform the pcap data, which was gathered with tcpcump, into flow data resulting in close to two million unique flows with 49 features [42]. Argus is an open-source auditing tool. It consists of an Argus-Server and Client, where the Server converts raw network packets into a binary format. The client reads the binary, extracts features from the data, and writes the

flows to a database. Bro-IDS is an open-source security monitoring tool that is used to analyze the network and creates log files for all connections in the pcap files (CONN), all HTTP requests and replies (HTTP), and all FTP events (FTP). The data of both are joined on the flow identifying features (Source IP, Source Port, Destination IP, Destination Port, Protocol). From those features, twelve more are created to encapsulate relationships among flows that could identify stealth attacks [41]. Next to the flow features, as described above, the authors further categorize the features into Basic, Content, Time, and Additional. A full overview of the features is provided in the Appendix A and in the works of the authors [42][41].

The data set includes nine different attack scenarios, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnissance, Schellcode, and Worms. As this work is mainly concerned with anomaly detection and not focused on the attacks themselves, many different attacks are still valuable as the different strategies behind the attacks will cause various anomalies.
The UNSW-NB15 data set also includes a predefined train-test split with 175341 flows in the training data and 82332 in the test set. However, the pre-split data consists of around 68% and 55% anomalies, respectively [41]. The high percentage of anomalies limits the uses of the pre-split data in unsupervised anomaly detection as the assumption that anomalies are a rare occurrence in the data is not fulfilled.

3.1.2 *CIC-IDS2017 / CSE-CIC-IDS2018*

The CIC-IDS2017 [29] and CSE-CIC-IDS2018 [30] both originate from the Canadian Institute for Cybersecurity (CIC). They both follow the same methodology in that so-called B-Profile and M-Profile are used to generate benign traffic and malicious traffic, respectively. B-Profiles are derived by analyzing user behavior via different statistical and machine learning models [57]. Deriving B-Profiles is a two-step process, as first, individual profiles are created from observing user behavior in terms of distribution of packet size of a protocol, the number of packets per flow, specific patterns in the payload, size of the payload, and request time distributions of protocols for the protocols HTTP, HTTPS, FTP, SSH, and ICMP. The user behavior is recorded daily and aggregated into 30 minutes intervals. The individual profiles are then clustered into groups of similar user behavior, with the cluster centroid of those groups being the B-Profiles.

The authors claim that by randomly selecting from those B-Profiles, realistic benign background traffic can be generated to mimic user behavior without the need for anonymization [57]. Parallel to the execution of autonomous agents that generate benign traffic via B-Profiles, M-Profiles are triggered to introduce malicious activities. M-Profile are specific attack scenarios designed to execute six complex attack scenarios on the network, like infiltration from the inside or brute force attacks. To convert the pcap into flows

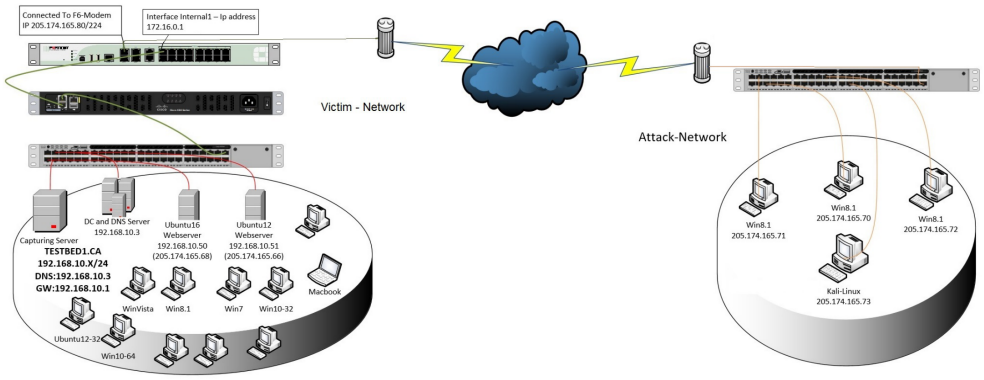

Figure 3.2: CIC-IDS2017 Testbed [29]

and to extract more than 80 features, the network analyzer CICFlowmeter was used [4].
As far as the documentation suggests, the main difference between the CIC-IDS2017 and CSE-CIC-IDS2018 is the network topology of the testbed and the amount of traffic gathered. CIC-IDS2017 (Figure 3.2) was recorded for a duration of five days, one day including only benign traffic while the other four include different attack scenarios depending on the day. The topology is split into two separate networks. An attacker network with four workstations to trigger the M-Profiles, and a victim network with 12 devices, including servers and workstations running different operating systems to execute the B-Profiles. A total of around 2.8 million flows were recorded for this dataset. The network topology used in the 2018 version was extended to replicate a larger network topology (Figure 3.3). Implemented on AWS, the test structure still contained a victim and an attacker network, but the attacker network consisted of 50 machines, while the victim network was further divided into subnets, replicating a business environment with 420 workstations and 50 servers. The workstations were configured in a plausible way as, for example, the management department was running Windows while the IT department was running Ubuntu. The dataset consists of 10 days of network traffic amounting to 16 million flows and 450 GB of pcap files.

3.1.3 *Summary*

Researchers have answered the lack of public available realistic network anomaly data sets by publishing more and more data in the last few years. The utilization of those data sets in research still has some catching up to do, as the vast majority of researchers still rely on the outdated DARPA data and its variants.
The slow adaptation of those data sets can be partially explained because there is no one clear winner in the benchmark category, as many data sets cover different scenarios. For example, while the CIC-IDS2017 covers many different attacks, the schedule in which they occur (split on different days)

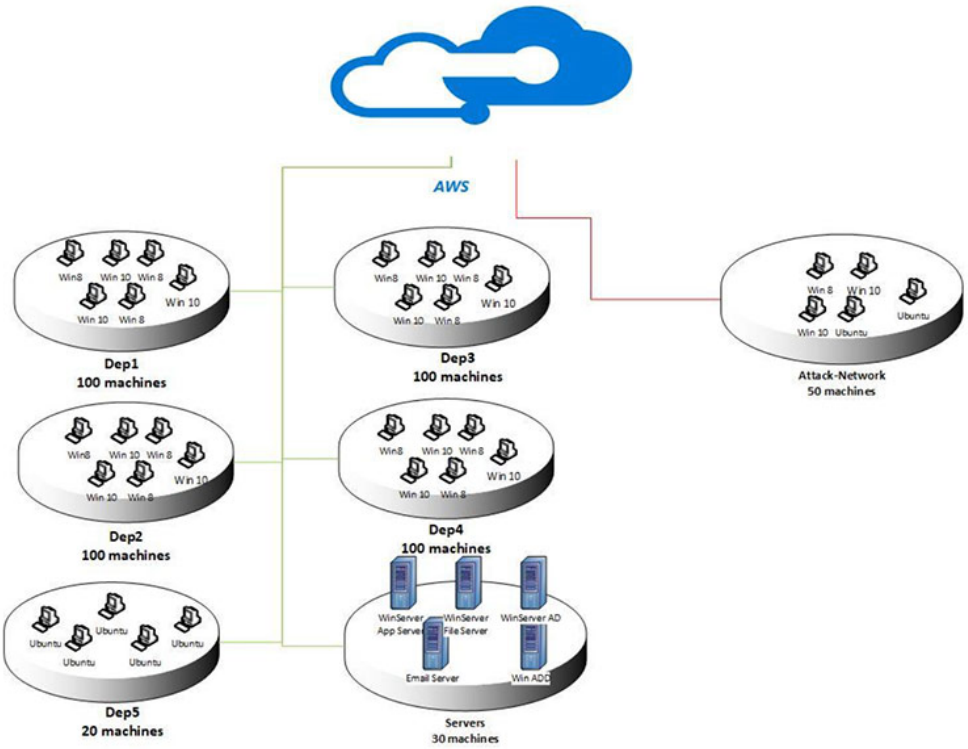

Figure 3.3: CSE-CIC-IDS2018 Testbed [30]

makes it challenging to construct a test set for an overall performance evaluation.
The huge data volume and the different objectives of researchers lead to alterations of the data. For example, only parts of the data are used to validate an approach, or different data preprocessing steps are used. This preprocessing of the data makes it difficult to get an exact replicate of the data, making it harder to validate scientific claims [43].

Both of the reviewed datasets contain irregularities. The CIC-IDS2017 contains around 288 thousand samples with missing labels and 203 instances where values are missing [48]. The UNSW-NB15 contains around 500 thousand duplicate records; these duplicates might be explained by the discrepancies between the accompanying paper and the web description of the data set [42] [67]. Although there are irregularities in both data sets, both are still valid candidates as a starting point to evaluate network anomaly detection approaches [54]. We decided to use the UNSW-NB15 dataset for the first evaluations, as the clear documentation provides in-depth descriptions of the generated features and the generation process. Additionally, the variety in anomalies provided in a short timespan and lower volume made it the obvious choice in terms of efficiency.

3.2 ALGORITHM

When choosing an appropriate algorithm to detect network anomalies, a few fundamental properties were essential to us. These properties result from inherent difficulties when analyzing network traffic.

- Network traffic is present as a stream of data.
- Network traffic is not static.
- Transferability of detection models is not given.

Network data needs to be analyzed in an online fashion - the detection of anomalies needs to be timely. However, the stream of network data is not static; concept drifts can happen in many timescales (minutes, days, weeks). Ideally, a detection model would adapt to these concept drifts to fit the current network traffic profile. As network traffic differs from network to network, fitting a model on a network and applying it to another might not provide the desired results. Obtaining labeled network data of the network which ought to be analyzed is a complicated and expensive task. Therefore, an unsupervised methodology would be ideal if a high detection rate and low false alarm rate can be achieved. More info and explanation regarding these issues can be seen in Section 2.2.1.

3.2.1 *Detection by Isolation*

As stated in Section 2.2.2, most anomaly detection approaches construct a profile of normal behavior and detect anomalies as data points that do not conform to this profile. However, the algorithms were rarely designed to detect anomalies but for clustering or classification tasks. Furthermore, many algorithms rely on density or distance measures to detect anomalies, which puts a high computational requirement on the system, especially in high dimensional settings [35].
To combat these shortcomings, Liu, Ting, and Zhou proposed the concept of Isolation-based anomaly detection [35]. The basic idea is that if the requirements are met (anomalies rare within the data and anomalies are different concerning their features), anomalies are easier to isolate from the rest of the data than normal instances. Isolation in this context means to separate an instance from the rest of the instances.
How this isolation process is done is secondary, but a data structure suitable to this task was used by the authors Liu, Ting, and Zhou [35], and Guha et al. [24], for the later discussed RRCF, is that of binary trees.

3.2.2 *Isolation by Binary Trees*

To better illustrate the concept of anomaly detection by isolation, binary trees are an effective tool to explain this concept while still being useful

in practice. Different algorithms utilize this data structure in order to detect anomalies, such as Isolation Trees [35], Extended Isolation Trees [25], and Robust Random Cut Forest [24]. They all share the underlying binary tree structure while showing differences in creation and scoring.

Definition 1 (Isolation Tree (IT)) Let T be an Isolation Tree. Let N be a node within the tree, N can either be an external node with no children (leaf) or an internal node with a test and exactly two child nodes (N_l,N_r). A test consist of an attribute q and a split value p where the condition $q < p$ determines the traversal of a sample to either N_l or N_r [35].

Definition 1 shows the basic construct of an isolation tree. Given a set of data points $X = \{x_1, x_2, \ldots, x_n\}$ with m attributes, a tree is constructed by choosing an attribute $q \in \{q_1, q_2, \ldots, q_m\}$ at random and a random splitting values p between q_{min} and q_{max}. An internal node with the test $q < p$ is formed, splitting the data points according to the test. This process is recursively reapplied on the subsets $X_l = \{x | x_q < p\}$ and $X_r = X \setminus X_l$ until (i) the node contains only one instance or (ii) the instances consist of the same attribute values. An upper bound on the nodes and thus the memory requirement of a tree can be calculated from the number of instances (n) included in the tree. When every instance is isolated, a tree consists of n leaves and $n - 1$ internal nodes resulting in a total number of $2n - 1$ nodes [35]. With the tree structure in place, anomalies should be located closer to a tree's root than normal instances. A primary measure to rank data points for their degree of abnormality can be the path length.

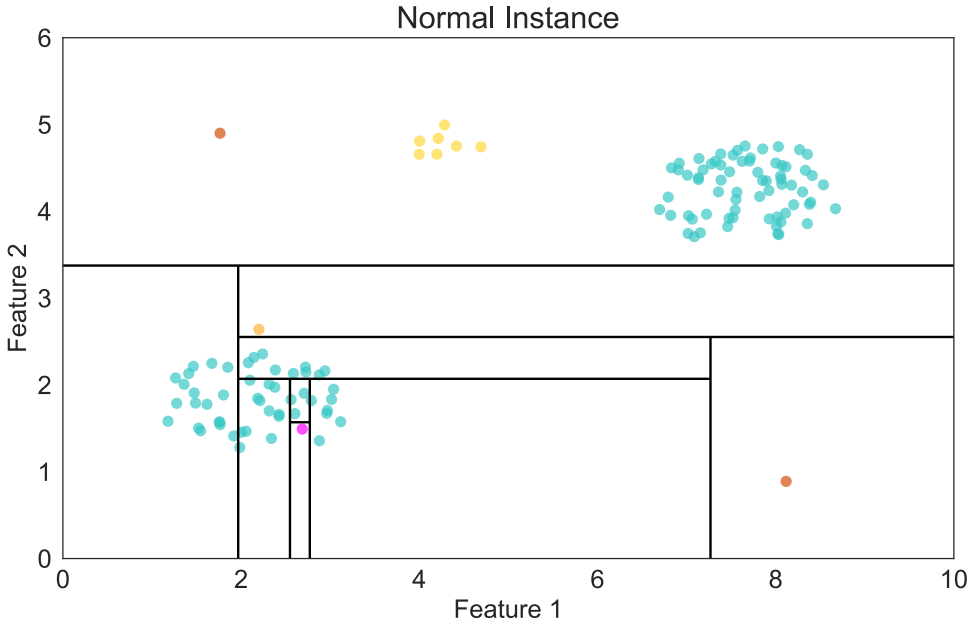

Figure 3.4: Binary Tree example Isolation of a Normal Instance

Definition 2 (Path Length) The path length of a point x is the number of edges x must traverse until an external node is reached. [35].

In other words, - the path length is how many random cuts are needed to isolate the instance. As this random process produces different cuts for different trees generated on the same data, we can derive the expected path length by averaging several trees' path lengths. Therefore, the average path lengths of anomalies should be shorter than those of normal instances. A more in-depth probabilistic explanation was presented by Liu, Ting, and Zhou [35].

Figure 3.4 shows an example of the isolation process using a binary tree. To illustrate the functionality, we focus on two data points, the anomaly in the top left corner and the highlighted normal instance in the bottom left normal region. In the example tree, eight random separations were needed to isolate the highlighted normal instance. On the other hand, the anomaly was separated with a path length of two, as seen in Figure 3.5.

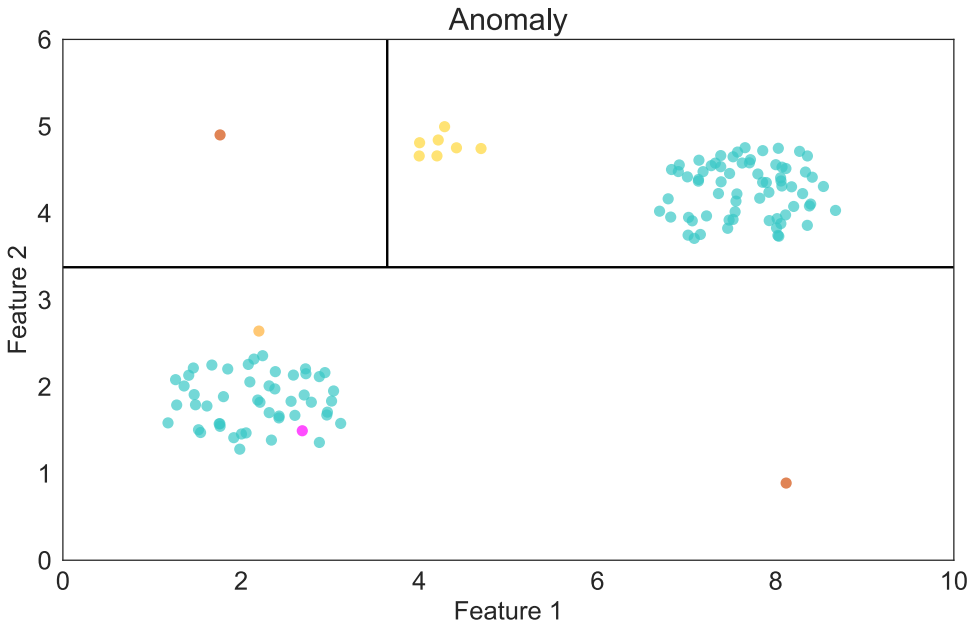

Figure 3.5: Binary Tree example Isolation of an Anomaly

The raw path length is a simplified way to rank anomalies. The Isolation Forest [35] employs a more sophisticated score, including the path length, while the Robust Random Cut Forest [24] employs a different measure discussed in Section 3.2.3.3.

Establishing an ensemble of multiple trees - a forest - allows for a path length estimation and helps with swamping and masking when subsampling is employed. The convergence of average path lengths of the two data points in the above example can be seen in Figure 3.6. Swamping and masking are two phenomena that result from too much data. Swamping refers to situations where normal instances are wrongly identified as anomalies because the number of normal instances increases or become more scattered. On the other hand, masking is when too many anomalies form big enough clusters to be regarded as normal instances. Both of these problems can at

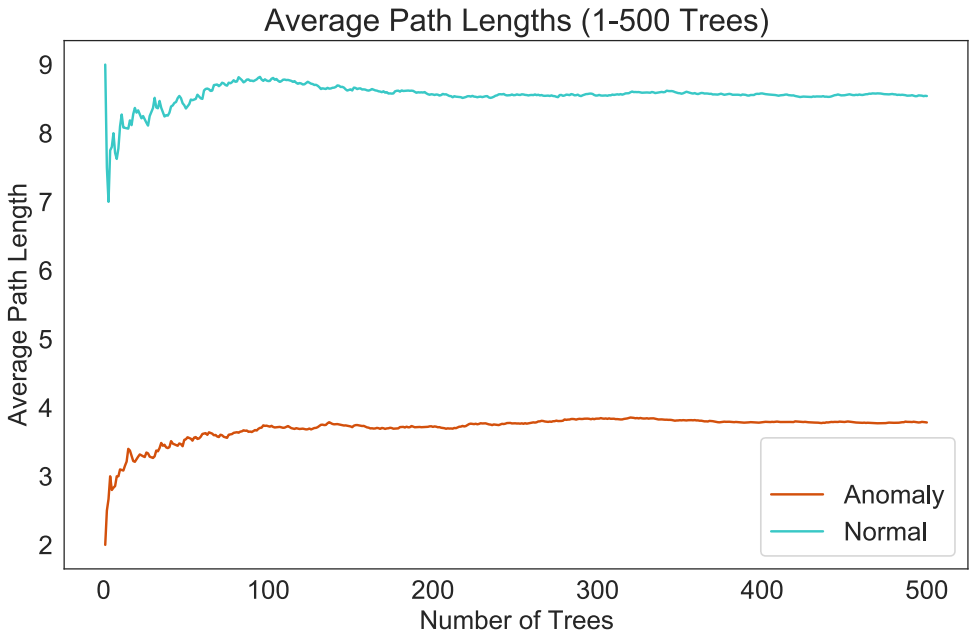

Figure 3.6: Average Path Lengths over 1 to 500 trees

least partially be alleviated by subsampling the data and building smaller trees [35].

3.2.3 *Robust Random Cut Forest*

The Robust Random Cut Forest algorithm is an unsupervised algorithm designed to detect anomalies. It is based on the same core principles as described in Section 3.2.1. The anomaly scoring is done utilizing a binary tree structure. However, the differences allow the algorithm to be applied to data streams. Unlike Isolation Forests or SVMs, no retraining is needed to incorporate new data into the model. As described in Section 2.2.1, one big problem of network anomaly detection is that network traffic is not static. Continuously updating the model during runtime could mitigate those problems. The following is based on the proposed Robust Random Cut Forest sketch algorithm by Guha et al. [24], this section aims to present the key properties of the algorithm and why they are essential, the proofs of the presented concepts are provided by the author of the algorithm.

Definition 3 (Robust Random Cut Tree (RRCT)) Generation of a RRCT on a data set S:

1. Choose a random dimension proportional to $\frac{l_i}{\sum_j l_j}$
   where $l_i = max_{x \in S} x_i - min_{x \in S} x_i$
2. Choose $X_i \sim Uniform[min_{x \in S} x_i, max_{x \in S} x_i]$
3. Let $S_l = \{x | x \in S, x_i < X_i\}$ and $S_r = S \setminus S_l$ and recurse on S_l and S_r

The first key property of the RRCT algorithm is that pairwise distances are preserved. The weight of a node is defined as the sum of dimensions $\sum_i l_i$. Consider two points $u, v \in S$. The tree distance is defined as the weight of the least common ancestor of u and v. Then the tree distance is at minimum the Manhattan distance $L_1(u, v)$ and in expectation at most $O(d \log \frac{|S|}{L_1(u,v)}) L_1(u, v)$.
This property is crucial as points which are far away from each other remain this way in an RRCT. As a reminder, we assume that anomalies are different from normal instances regarding their attributes. Therefore this allows for distance-based anomaly detection without focusing on a particular distance function.

As mentioned in Section 3.2, one desired property is to be able to adapt for occurring concept drifts in the network traffic. In other words, we are interested in adding and removing data points from the tree structure to sketch the current network behavior. Unlike the Isolation Tree, where the cut dimension is chosen uniformly at random, the cut dimension in an RRCT is chosen proportional to each attribute's range ($max - min$). This small but significant difference allows for an RRCT to be altered dynamically while preserving the distributions. Consider a $T(S)$ and a point $x \in S$, which is to be removed. The resulting tree T' has the same probability as if it was drawn from $T(S - \{x\})$.
To clarify this property and compare it to the behavior of the IT we want to look at a basic example. Assume a two-dimensional data set with three data points $p_1 = (0, 1), p_2 = (\epsilon, \epsilon), p_3 = (1, 0)$ where $0 < \epsilon << 1$. Two cuts are needed two isolate every point which can be any combination of horizontal (H) and vertical (V) cuts with the root of the tree being a cut between p_1 and p_2 or p_2 and p_3. Consider the deletion of p_3; this is done by removing the leaf containing p_3 and replacing the parent with its sibling.
If we were to draw a tree from $T(\{p_1, p_2\})$ the probability for the two points to be separated by a vertical cut to would be $\frac{1}{2}$ using the IT algorithm as the dimension is chosen uniformly. Using the RRCF algorithm the probability for a vertical cut would be $\frac{l_i}{\sum_j l_j} = \frac{1-e}{e+(1-e)} = 1 - e$.
However, arriving at a tree over the two points by deleting p_3 from the tree yields a different result when using the Isolation Tree algorithm. Figure 3.7 shows the four, of the eight trees over three points, which satisfy the condition of a vertical cut between p_1 and p_2. The probability for a vertical cut $P(V)$ can be calculated by summation of the probabilities of the four trees shown in Figure 3.7.

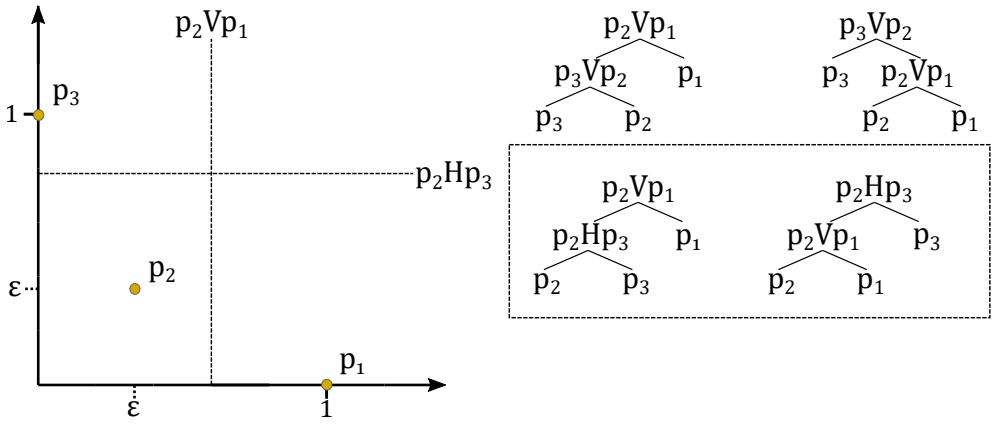

Figure 3.7: Tree Probability Example

$$
\begin{aligned}
P(V_{IT}) &= P(V \wedge V) + P(V \wedge H) \\
&= (\frac{1}{2}(1-\epsilon)\frac{1}{2} + \frac{1}{2}\epsilon\frac{1}{2}) + 2(\frac{1}{2}(1-\epsilon)\frac{1}{2}) \\
&= \frac{1-\epsilon}{4} + \frac{\epsilon}{4} + 2(\frac{1-\epsilon}{4}) \\
&= \frac{3}{4} - \frac{\epsilon}{2}
\end{aligned}
\tag{3.1}
$$

Equation 3.1 shows the calculation for the probability of a vertical cut after deletion of p_3, using the Isolation Tree algorithm. The resulting probability is not $\frac{1}{2}$ as when a tree is built over just the remaining points. However, the calculation in Equation 3.2 shows then when using the RRCT algorithm. The probability is as p_3 was never part of the point set, thus preserving the distributions. The same principles can be extended to the insertion of points, allowing for dynamic updating of the trees to maintain an up-to-date window of the network traffic by inserting the new incoming points and deleting the oldest points within the trees.

$$
\begin{aligned}
P(V_{RRCT}) &= P(V \wedge V) + P(V \wedge H) \\
&= 2(\frac{1}{2}(1-\epsilon)\epsilon) + 2(\frac{1}{2}(1-\epsilon)(1-\epsilon)) \\
&= (\epsilon - \epsilon^2) + (1 - 2\epsilon + \epsilon^2) \\
&= 1 - \epsilon
\end{aligned}
\tag{3.2}
$$

3.2.3.1 *Deletion*

As previously stated, the deletion of point p from tree T is done by deleting leaf v, which isolates p and replacing the parent with its sibling u. Note that during this process, the path lengths of all points which reside in the subtree

originating at u will be shortened by one. Algorithm 1 shows the algorithm as presented by Guha et al. [24].

Algorithm 1 : Point Deletion [24]

Result : $T'(S - \{p\})$

1 Find the node v in the tree where p is isolated in T.
2 Let u be the sibling of v. Delete the parent of v (and of u) and replace that parent with u.
3 Update all bounding boxes starting from u's (new) par- ent upwards – this state is not necessary for deletions, but is useful for insertions.
4 Return the modified tree T'.

3.2.3.2 *Insertion*

The insertion of a point into an RRCT is more complicated than the deletion. Unlike the deletion, the resulting tree T' is not uniquely determined. The bounding box, introduced in Algorithm 2, describes each attribute's outer bounds used in the tree construction. When traversing down a tree, the bounding boxes of the left and right subtrees are updated to represent the points that traverse the corresponding paths. During the insertion, a point can only be isolated if its attribute values exceed the bounding box.
The insertion of a point p starts at the root of the tree. First, a random cut is generated. If this cut successfully isolates p from the rest of the data, then a new parent is formed with the child nodes containing p and the original tree, depending on the split value. If the cut does not isolate p, we discard the split and repeat the same process on the subtree where p is delegated to depending on the original split. This process is repeated until the point is

isolated as described in Algorithm 2.

Algorithm 2 : Point Insertion [24]

Result : $T'(S' \cup \{p\})$

1 We have a set of points S' and a tree $T(S')$. We want to insert p and produce tree $T'(S' \cup \{p\})$.
2 If $S' = \emptyset$ then we return a node containing the single node p.
3 Otherwise S' has a bounding box $B(S') = [x_1^l, x_1^h] \times [x_2^l, x_2^h] \times \ldots \times [x_d^l, x_d^h]$. Let $x_i^l \leq x_i^h$ for all i.
4 For all i let $\hat{x}_i^l = min\{p_i, x_i\}$ and $\hat{x}_i^h = max\{x_i^h, p_i\}$.
5 Choose a random number $r \in [0, \sum_i (\hat{x}_i^h - \hat{x}_i^l)]$.
6 This r corresponds to a specific choice of a cut in the construction of $RRCF(S' \cup \{p\})$. For instance we can compute $argmin\{j | \sum_{i=1}^{j} (\hat{x}_i^h - \hat{x}_i^l) \geq r\}$ and the cut corresponds to choosing $\hat{x}_j^l + \sum_{i=1}^{j} (\hat{x}_i^h - \hat{x}_i^l) - r$ in dimension j.
7 If this cut separates S' and p (i.e., is not in the interval $[x_j^l, x_j^h]$) then and we can use this as the first cut for $T'(S' \cup \{p\})$. We create a node – one side of the cut is p and the other side of the node is the tree $T(S')$.
8 If this cut does not separate S' and p then we throw away the cut! We choose the exact same dimension as $T(S')$ in $T'(S' \cup \{p\})$ and the exact same value of the cut chosen by $T(S')$ and perform the split. The point p goes to one of the sides, say with subset S''. We repeat this procedure with a smaller bounding box $B(S'')$ of S''. For the other side we use the same subtree as in $T(S')$.
9 In either case we update the bounding box of T'.

3.2.3.3 *Anomaly Score*

To rank anomalies, the RRCF utilizes the difference in model complexity when inserting or deleting a point. The path to each point in an RRCT can be expressed as a bit sequence where taking the left path is denoted as 0 and the right path as 1. The number of bits required to describe every point in the tree is the model complexity. In other words, the model complexity is the sum of the depth of every point in the tree. The change in the model complexity can be used to rank data points for their degree of abnormality. Suppose a point set Z and a point $y \in Z$ then $f(y, Z, T)$ is the depth of y in $T(Z)$. The bit displacement, the change in model complexity, can be calculated as seen in Equation 3.3.

$$\textsc{Disp}(x, Z) = \sum_{T, y \in Z - \{x\}} P[T]\big(f(y, Z, T) - f(y, Z - \{x\}, T')\big) \tag{3.3}$$

In simpler terms, the displacement of a point x is the expected number of points belonging to the leaf's sibling containing x. However, problems arise

when multiple similar anomalies occur. This phenomenon is called outlier masking; an example illustration can be seen in Figure 3.8.

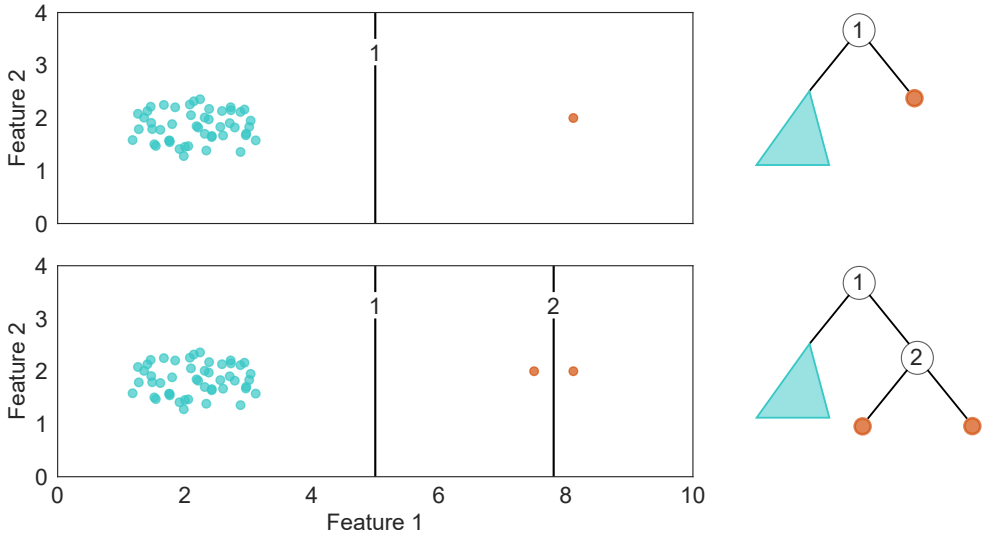

Figure 3.8: Outlier Masking

The top graph shows one anomaly. The number of points belonging to the sibling will indicate that this point is an anomaly. However, the bottom shows how the addition of a similar anomaly mask the presence of any anomaly as the number of points belonging to the sibling gets drastically reduced. To combat this, the collusive displacement was introduced, as seen in Equation 3.4.

$$\text{CoDisp}(x, Z, |S|) = \mathop{\mathbb{E}}_{S \subseteq Z, T} \left[\max_{x \in C \subseteq S} \frac{1}{|C|} \sum_{y \in S - C} \left(f(y, S, T) - f(y, S - C, T'') \right) \right] \tag{3.4}$$

The collusive displacement can be estimated efficiently and can be reasoned quite intuitively. The idea is that we do not just observe the model change caused by one point; we also observe the model change caused by points close to it. This is done by traversing the tree upwards from the leaf of the point of interest, dividing the number of instances in the sibling of the current node with the points attached to the current node. The maximum along the path to the root of the tree is chosen as the collusive displacement. A large collusive displacement indicates anomalies.

3.3 CONCEPT

After verifying the algorithm's functionality in combination with samples of the chosen dataset USNWB-15 (Section 3.1.1), an experiment was designed to investigate the applicability in a real-world scenario. This experiment's priority was not necessarily to find an optimal model but to get an estimate

of the online detection performance.

As mentioned in Section 2.6 the data source and our point of view is a single gateway where all incoming and outgoing traffic is routed through (consistent with the generation of the UNSW-NB15 dataset). The traffic at this point in chronological order of occurrence. The data set was sorted by the timestamp of the flows to preserve this order. The second important characteristic in selecting the data to test against was that the timeframe needs to be represented without any gaps, as we wanted to observe the traffic in the same way it passed through the gateway. With these two requirements in place, we can replicate the online performance of the RRCF by inserting the flows sequentially. As the structure of the RRCT shifts with insertion and deletion operations, it is also important to score a flow at the time it occurred. Therefore, the CoDisp (Section 3.2.3.3) of each flow was recorded during insertion.

To form a baseline, we chose to use the Isolation Forest algorithm. The two algorithms are closely related as they both require the same implicit assumptions and utilize binary trees to represent the data. The two key differences are the probability of the chosen cut dimension as well as the anomaly score (Section 3.2). As a reminder, the difference in the way the cut dimension is chosen allows the RRCF to be updated dynamically. By comparing the results, we looked for a performance baseline and observed if dynamic updating improves performance over a static model in the given timeframe. The IF was trained on the same day one data as the RRCF is populated with, predicting on the same timeframe on day two. Isolation Forests are not updated dynamically. Therefore this procedure was akin to a train/test split validation.

A timeframe of 60 consecutive minutes was chosen from day two of the UNSW-NB15 dataset as a test set. The RRCF was populated with data points from day one according to the algorithm's main hyper-parameters, tree size, and the number of trees. The hyper-parameters are model parameters that are used to control the learning process. These parameters can be optimized in order to find a hyper-parameter set which maximizes model performance. The parameters discussed in the rest of this work will be referring to these hyper-parameters.

As the model itself does not provide many parameters, we introduced additional parameters concerned with data preprocessing. The algorithm chooses the cut dimension proportional to the range of the features (Section 3.2). Therefore, the probability of a feature being chosen can be manipulated by scaling the features, transforming the feature space, or reducing the number of features. These scalings and transformations have been fitted to the training data and later applied to day two's test data. The model parameters and additional parameters as described in the following were used

along with a parameter optimization strategy to inspect the capability of the RRCF to detect anomalies online in high-dimensional network traffic.

3.4 DATA PREPERATION

The RRCF algorithm (Section. 3.2.3) can only handle numeric attributes. Other symbolic or categorical attributes need to be transformed in a meaningful way to a numeric representation. The following will describe preparing the data and giving insight into the irregularities found within the UNSW-NB15 dataset (Section 3.1.1).

3.4.1 *Missing Values and Irregularities*

The data of the UNSW-NB15 dataset is provided in a flow-format contained in five CSV-files. The data contains a total of 2.540.047 flows. This amount is consistent with the online provided description of the dataset [67]. However, the analysis revealed 480.630 duplicate records bringing the unique records count down to 2.059.417, which is close but not consistent with the description provided in the associated publication (1.964.509) [42]. Of the duplicate records, only the first occurrences were retained. A full feature list is provided in the Appendix A. Only two attributes were missing values. The first attribute 'is_ftp_logi', is a binary attribute indicating if the FTP session is accessed by a user and a password (1 else 0). As most missing values are associated with flows not utilizing the FTP-protocol, we replaced the missing values with 0. However, some of the records also contained integer values greater than one. These records were removed as these values indicated measurement errors.
Additional missing values were found in the attribute 'ct_flw_http_mthd'. This integer attribute counts the flows which contain HTTP methods like Get or Post. Logically, every flow not utilizing HTTP services should have a value of 0 for this attribute. However, almost half of all records showed a missing value for this attribute, indicating other reasons than measurement error. To separate records with null values but keep them spatially close to the presumed true value of 0, all null values were replaced with -1.

3.4.2 *Categorical Features*

The data contains five categorical features as seen in Table 3.1. The features proto, state, and service were encoded using one-hot encoding. One-hot encoding is done by adding a feature for each expression of the attribute. These new feature vectors are binary and will take the value of 1 if the record's corresponding attribute is of that expression and 0 if not. This encoding technique was chosen as it is a way to represent categorical attributes without implying an ordinal relationship numerically. Additionally, it also fits intuitively considering a binary tree structure. For example, consider an encoded

features vector for the HTTP service. A generated split of a random binary tree will divide the samples in all flows the include HTTP services (1) and all that do not (0).

Name	Unique Values	Description
srcip	43	Source IP adress
dstip	47	Destination IP address
proto	135	Transaction protocol
state	16	The state, dependent protocol
service	13	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service.

Table 3.1: Categorical Features

The IP-addresses are a complicated case, as they can be considered a categorical feature, with each IP address representing a unique entity. The 43 and 47 unique source and destination IP-addresses of the dataset can be encoded using one-hot encoding. However, outside of this data set's limited scope, new unseen IP-addresses can not be processed. To cover the whole range of IP-addresses, an encoding would need to encapsulate 2^{32} elements using IPv4 and 2^{128} in the case of IPv6. Additionally, we can not count on IP-addresses to be static. The public IP-addresses can get reassigned by the internet provider or spoofed by an attacker, and the internal network IP-addresses can also be dynamically assigned by the gateway if not specified to be static. While identifying the elements which communicate via network traffic is needed to identify specific network attacks [3], associating them to concrete IP-addresses or their encoded representation might not yield the desired result as anomalous behavior should be detected regardless of where it occurs. Other possible measures to convey the mapping of source and destination are to derive contextual features, for example, for DDoS prevention features such as traffic volume per IP address [19] or the proportion of new unseen IP-addresses [50] can be used. For this work, we decided to discard the IP-addresses as a feature and rely on the data set's contextual attributes to detect anomalies. However, additional investigation on how to handle IP-addresses needs to be conducted if the taken approach provides promising results in detecting network anomalies.

3.4.3 *Data Overview*

As stated in Section 3.3, the UNSW-NB15 data was split into two, according to the day the flows were recorded. Data of day one was used to train the Isolation Forest and populate the RRCF. One hour of day two was chosen as a first test set. The hour was chosen to approximately reflect the expected load and to contain every category of the provided anomalies.

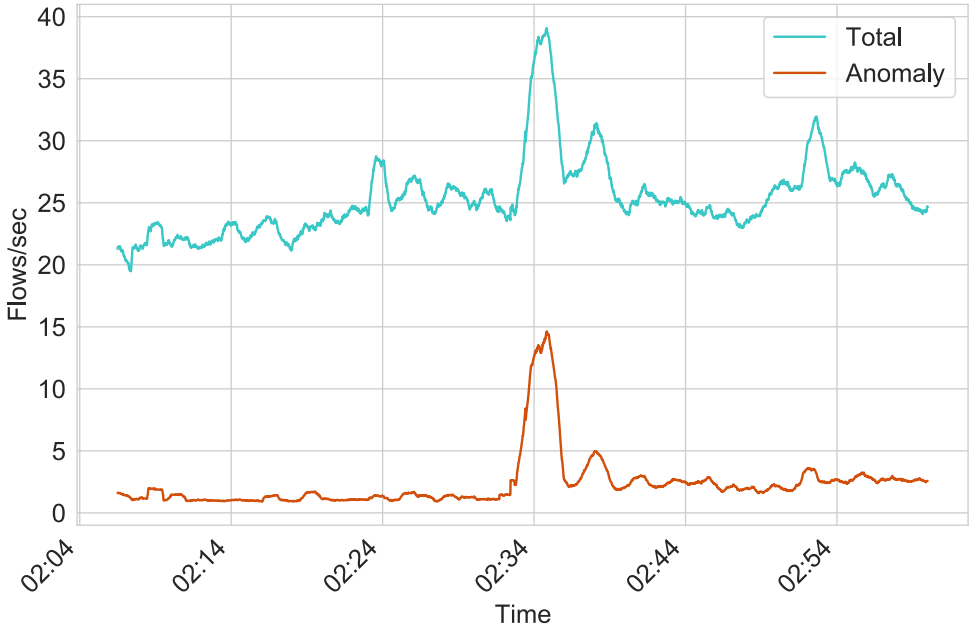

Figure 3.9: Moving Average of Flows per Second (Window=60 sec)

Figure 3.9 shows the moving average of flows per second over a one minute window for the total workload and the anomalies occurring during the timeframe. The throughput of a network is usually measured in bits per second; however, using flows per second provides a better indication of the actual workload expected at this point in the detection process. The one-hour test set consists of 81547 unique flows with 10.2% anomalous traffic.

The distributions of the transaction protocols (Figure 3.10) closely resemble the distributions observed in common internet traffic [69]. The type of attacks present and the corresponding number of flows in the test data is shown in Table 3.2. Although the focus is not the cause of the anomalies, viewing the detection performance in relation to the attack can help identify the possible scope of the detection as well as indicate which attacks meet the requirements of being different to normal instances regarding their attributes.

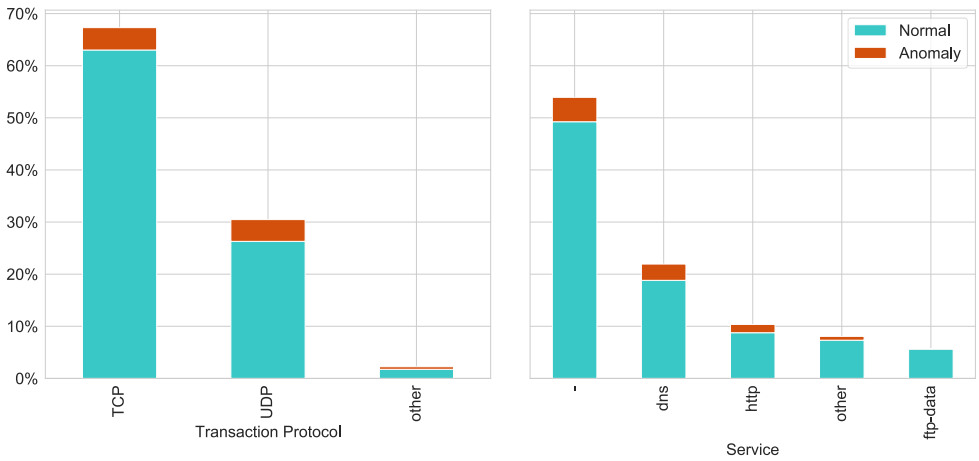

Figure 3.10: Distribution of Transaction Protocols and Services

Attack Category	Count
Generic	2962
Exploits	1893
Fuzzers	1441
Reconnaissance	990
DoS	443
Analysis	253
Backdoor	231
Shellcode	102
Worms	10

Table 3.2: Attacks during Test Timeframe

3.5 HYPERPARAMETERS

3.5.1 *Model Hyperparameters*

The main parameters of the RRCF are the number of trees and the size of the tree (number of leaves). These parameters affect the size of the window of the network traffic included in the model at any point during the execution, and form the basis for the CoDisp calculation. Another parameter provided by the used implementation is precision. This value is the floating-point precision used to distinguish between duplicate points. The parameters and their chosen value ranges can be seen in Table 3.3.

Name	Values	Type	Description
n_trees	10-100	Integer	Number of trees
tree_size	100-2000	Integer	Number of leafs
precision	0-9	Integer	Number of decimal places

Table 3.3: Model Parameters

3.5.2 *Scaling Hyperparameters*

The model parameter itself limits the possibilities to tweak the performance of the model. However, data manipulation can directly impact the model performance and the underlying tree structure as the splitting dimensions are chosen proportional to their value range. For example, without manipulation, the source or destination ports would have a much higher probability of being chosen as one of the one-hot encoded attributes as the range is much larger (0-65535 compared to 0-1). The Tab. 3.4 shows the different scalers used, these scalers were implemented as a categorical parameter $scalers \in \{None, MinMax, MaxAbs, Standard\}$.

3.5.3 *Dimension Reduction Hyperparameters*

The encoding of the categorical attributes has introduced 164 additional attributes. Additional dimensionality reduction techniques were included in the data preprocessing to counteract the curse of dimensionality (Section 2.2.1). For this experiment, we chose to keep the notion of unsupervised learning consistent. Therefore only unsupervised dimensionality reduction techniques are used. Additionally, a binary parameter is used to control the features affected by the dimensionality reduction. As previously mentioned, the one-hot encoding of categorical attributes added a multitude of sparse

Name	Formula	Description
None	-	No scaling applied.
Min Max	$\frac{x-x_{min}}{x_{max}-x_{min}}$	Scales features to the range [0,1].
Max Abs	$\frac{x}{\lvert x_{max}\rvert}$	Scales each feature according to its maximum absolute value.
Standard	$\frac{x-mean(x)}{std(x)}$	Removes mean and scales data to unit variance.

Table 3.4: Scaling Parameters

feature vectors. This binary parameter controls if the techniques are applied to all attributes or to each of the one-hot encoded attribute groups.

Name	Values	Description
None	-	No dimension reduction applied.
All	0,1	Apply dimension reduction to the added features or all.
PCA	0.75, 0.85, 0.95	Apply PCA, keep the minimum pcs with the cummulative explained variance $\geq$ value.
Variance Threshold	0-0.5 (step 0.1)	Drop low variance features below value.

Table 3.5: Dimension Reduction Parameters

- **Principal Component Analysis (PCA)** is a dimensionality reduction technique that projects the data into a lower-dimensional space while retaining the maximum variance and ideally filtering noise. PCA assumes that large variances represent the interesting structures and low variances represent noise [60]. This re-basis of the data is done by computing the principal components; these are the vectors that explain the variance in descending order (variance explained by $PC_1 > PC_2 > \ldots > PC_n$). These vectors can be used to linear transform the data into a lower-dimensional space by using the components that explain the largest portion of variance and discarding the rest. The parameter of the PCA is the explained variance. Suppose we desire to retain 75% of the variance in the data; to achieve this, we select the first n principal components until their cumulated sum of explained variance meets this

threshold. These n components are then used to linearly transform the data to n dimensional space.

- **Variance Threshold (VT)** is also an unsupervised method to reduce dimensionality. Unlike PCA, no linear transformation of the data is applied. The features with low variance are simply discarded. This approach assumes that a feature with very low or zero variance (thus being approximately constant) will not improve the model performance and can therefore be discarded. By default of the used variance threshold implementation (Section **??** only the features with a single value over all samples will be discarded. A threshold value can be set to discard all features with a variance below this value.

3.6 HYPERPARAMETER OPTIMIZATION

In the context of this experiment, we are interested in investigating two questions.

- How well does the RRCF algorithm distinguish anomalous traffic from normal traffic during online detection?
- Are there data preprocessing steps that improve detection?

Those two question can be viewed as a hyper-parameter optimization task. Hyperparamter optimization is the process of finding a set of parameters which maximize model performance. Basically, the goal is to solve Equation 3.5, with X being the paramteter space defined by the paramaters as described above.

$$x^* = arg\max_{x \in X} f(x) \tag{3.5}$$

The function $f(x)$ is called the objective and can be any function that takes parameters as an input and outputs a value we are trying to maximize/minimize. For the parameter optimization itself, the objective is a black box - in our case, it is applying the RRCF on the test set and evaluating the performance with one of the metrics provided in Section 2.3, primarily PR AUC.
Basic strategies include techniques such as grid search or random search [7]. Grid search evaluates each parameter combination of a pre-defined grid (cartesian product of the parameters) while random search evaluates random combinations of the hyperparameters. Both of these techniques are not affected by previous evaluations. However, these techniques can spend a lot of time exploring parameter spaces where improvement is unlikely. If the objective function is time-intensive, which is the case using the RRCF, it can be an advantage to focus on exploiting parameter regions where an improvement is likely.
This can be achieved by utilizing Bayesian optimization techniques. The following section will cover the basic concepts and motivation behind Bayesian optimization; a more in-depth explanation is provided by Bergstra et al. [6].

Bayesian optimization strategies belong to the sequential model-based optimization (SMBO) algorithms. These algorithms generally introduce two new functions to aid the hyperparameter optimization process. This process is sequential as all prior evaluations are used to decide which parameter region to explore next. The two functions are:

- The **Surrogate** function is used to approximate the objective. Using Bayesian techniques, it is a probability model that maps the parameters x to the probability of the output of the objective y. Different techniques use different surrogate functions, for example Gaussian Processes (GP) model $P(y|x)$ whereas tree-structured Parzen Estimator (TPE) use p(x | y) and p(y) [6]. Regardless of the implementation, the basic concept is to update this probabilistic model after each optimization, iteratively achieving a closer and closer approximation of the objective.
- The **Acquisition** function is used to chose where to evaluate the objective function next. In parameter optimization, there is usually a trade-off between exploration and exploitation. Assuming similar sets of parameters will yield similar results for the objective function, exploration investigates parameter regions that are far away from previous evaluated ones, whereas exploitation focuses on regions that previously yielded good results. The goal of the acquisition function is to strike a balance between exploration and exploitation to find the best parameter combination to evaluate the objective on, such as Expected Improvement (EI) or Probability of Improvement [6].

With these two functions, the optimization flow can be summarized as follows.

1. Initialize Surrogate
2. Use Acquisition and Surrogate to identify best parameter combination x
3. Evaluate objective $y = f(x)$
4. Update Surrogate using x and y
5. Repeat steps 2-4 until the desired number of iterations is completed

We can explore the capabilities of the RRCF and the effect of different data preprocessing steps without prior assumptions about their effect on the detection, utilizing this approach. For the experiment, we chose the TPE technique, as it can handle categorical parameters and continuous parameters alike as well as providing the possibility to define the parameter search space during runtime [6].

3.7 TESTBED

3.7.1 *Hardware*

All tests were performed on a server provided by the Institute for Computer Science of the University of Applied Sciences Darmstadt within the research project ANEMO. The hardware configuration is shown in Tab 3.6.

Component	**Description**
Processor	(2x) Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz (32 cores - 64 threads)
Memory	DIMM DDR4 2933 MHz (192 GB)
Storage	5 TB
GPU	Nvidia GM107GL [Tesla M10]

Table 3.6: Hardware Configuration

3.7.2 *Software*

The server is configured running Ubunutu (Linux 4.15.0-88-generic). For easy remote access, a pre-configured machine learning docker container is used [20]. The container is pre-installed with Python 3.7+, Jupyter, the python libraries sklearn, pandas, and many more additional tools. For the data preparation steps and the baseline algorithm (IF), the implementations provided by sklearn were used. Additionally, the following third party libraries were used in the implementation.

- The **Algorithm** is provided by [81] (2019). The authors published an open-source python implementation of the RRCF algorithm akin to the algorithm by Guha et al. [24]. The source code has been thoroughly checked against the algorithm description with no deviations found. The implementation provides a batch and streaming detection, whereas the batch detection is done through recursive tree construction, and the streaming detection is done by the provided insertion and deletion methods (Section 3.2.3).
- The **Parameter Optimization** is done with the help of the python library Optuna [47]. Optuna is an automatic hyperparameter optimization framework designed for machine learning. The accessible API uses studies and trials to define the process; a trial is one evaluation of the objective, whereas a study is the optimization loop consisting of multiple trials. Optuna uses the trials object to sample the parameters, which

allows for adapting the parameter search space during runtime. Additionally, Optuna allows to either maximize or minimize the objective. The default optimization strategy utilizes a tree-structured Parzen Estimator (TPE),

- The **Experiment Tracking** was done using Neptune. Neptune provides a python framework as well as a web client to track metrics, hyperparameters, data versions, and more [44].

3.8 EXPERIMENT EXECUTION

During this work, more than 200 runs of the experiment were conducted on the described testbed. Multiple different parameter ranges, as well as parameter combinations, were tested. In general, one execution of the experiment included 25 hyperparameter optimization cycles. We noticed surprisingly long execution times during the experiments, with some taking multiple days to compute - much longer than the actual test timeframe. Inspecting the code did not reveal any obvious error or optimization potential in the tree computations.
Interference could identify two main contributors to the long runtimes. For one, the scaling of the features allowed for the biggest runtime savings (as seen in Section 3.9). The second issue with the used RRCF implementation is that it does not provide any capability for parallel processing. The insertion and deletion of data points need to happen sequentially for each tree. However, this process can be executed parallel on multiple trees as the trees are independent. The algorithm was extended with basic parallel processing capabilities in order to combat the long runtimes.
Another shortcoming of the used implementation is that the algorithm does not support subsampling during streaming detection. The authors of the Isolation Forest found that subsampling during the tree generation can help with swamping and masking [35].

PR AUC	Number of Trees	Tree Size	Scaling	Dimension Reduction	Subsampling
0.677	40	100	MaxAbs	PCA(All)	0.5
0.677	70	100	MaxAbs	PCA(All)	0.7
0.673	40	100	MaxAbs	PCA(All)	0.6
0.633	40	450	MaxAbs	PCA(All)	-
0.617	40	600	MaxAbs	PCA(All)	0.6

Table 3.7: Optimization Studies Sample (PR AUC)

However, during streaming detection with the RRCF implementation, each new point is inserted into each tree, leading to a forest where each tree consists of the same data points. The random structure of the trees still allows the trees to form an ensemble. However, to further increase variation between the trees, we implemented a subsampling feature via an insertion probability. By not inserting every sample into every tree, we can diversify the trees and widen the window of included network traffic, retaining the same model size. Table 3.7 and Table 3.8 shows a sample of the performed studies. The tables are split for the metric used to maximize. The following columns are the parameters that were used to achieve the best results for each study.

ROC AUC	Number of Trees	Tree Size	Scaling	Dimension Reduction	Subsampling
0.973	30	1800	MaxAbs	PCA(All)	-
0.972	80	1400	MaxAbs	PCA(All)	0.7
0.965	60	1900	MaxAbs	PCA(All)	1
0.945	60	340	MinMax	PCA(All)	-
0.942	140	1000	MaxAbs	PCA(All)	-

Table 3.8: Optimization Studies Sample (ROC AUC)

3.9 RUNTIME PERFORMANCE BENCHMARK

A series of runtime benchmarks was conducted in order to investigate the long runtimes which occured during the experiment trials. To better understand the impact of the hyperparameters (tree size, scaling, number of columns) on the runtimes of the algorithm we observed the runtimes according to those hyperparameters. As a reminder, updating the RRCF consist of three distinct steps, inserting the datapoint into the tree structure, calculating the CoDisp of the data point and deleting the oldest data point from the tree (Section 3.2.3). The insertion and deletion, along with the updating of multiple data points, need to be executed sequentially.

3.9.1 *Single-Tree Benchmark*

The anomaly detection itself is done with an ensamble of trees. however for the investigation of the runtime issues the first benchmark was conducted with a single tree, to closer inspect the behaviour of the core algorithm. Trees were generated with sizes ranging from 100 to 10000, inserting 1000 data points into each of them, measuring the average time per data point. A visu-

alization of the recorded times during this benchmark using the unprocessed data can be seen in Figure 3.11.

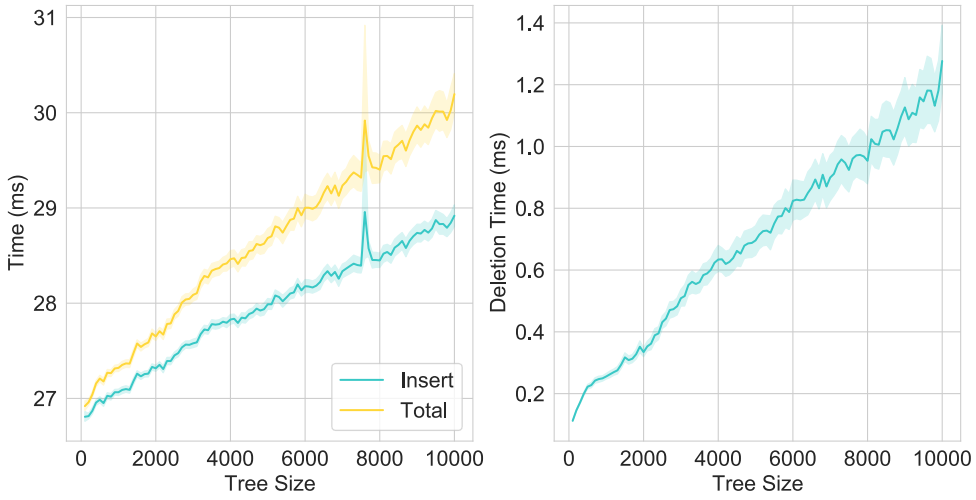

Figure 3.11: Avg. Processing Times per Data Point (Raw Data)

An approximate linear increase in total, insertion and deletion time in relation to the tree size can be observed. The long processing times restrict the maximum achievable online detection from 37 to 33 flows per second for a single tree.

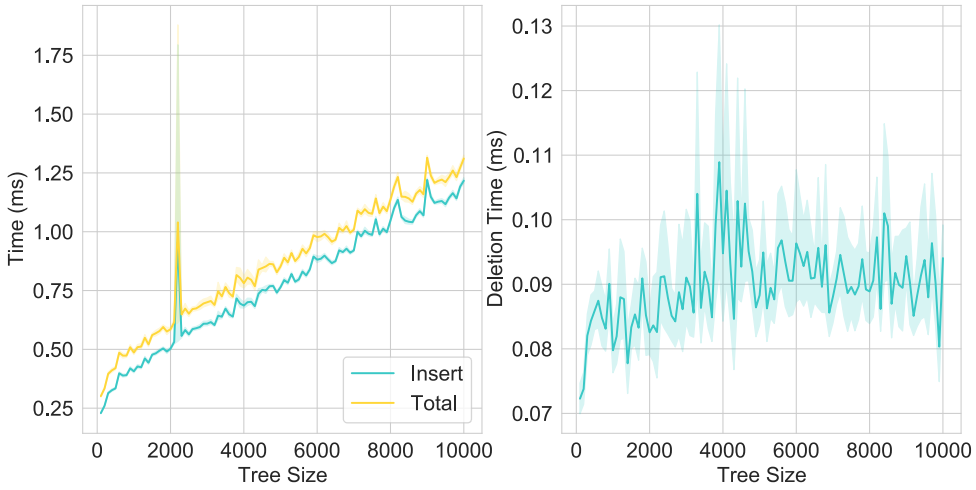

Figure 3.12: Avg. Processing Times (Preprocessed Data)

Data preprocessing according to the best achieved models, PCA and MaxAbs scaling combined, caused a substantial gain in runtime performance. The biggest contributor is the scaling of the attributes, reducing the number of features only had a small impact on the runtime. Additionaly, the deletion time stayed approximately constant in contrast to the measurments done with the raw data. The average cuts needed for every point was recorded as well, however no significant difference was found between the trees based on the original and the preprocessed data. Therefore, the gain in runtime per-

formance can be attributed to numeric operations done during the updating process. This performance gain is especially valuable as we suspect there to not be any optimization oportunity as these operations are impelmented with already optimized libraries.

3.9.2 *Forest Benchmark*

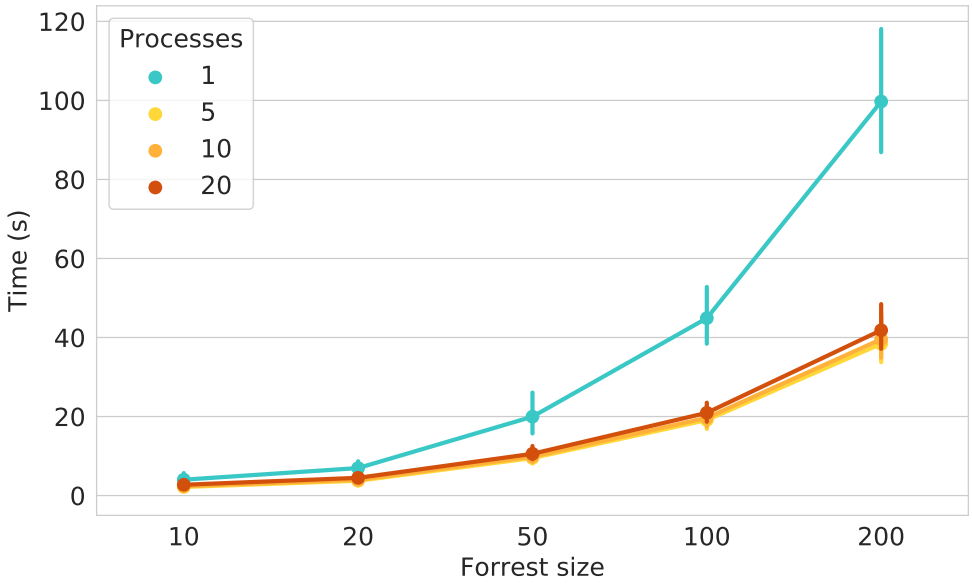

Figure 3.13: Total times (1000 insertions) for different Forrest Sizes.

As previously stated, the used RRCF implementation does not include parrallel processing. The insertion and deletion of data points on a single tree need to be executed sequentially, however the trees of the forest can be handled in parallel as the trees are independent from each other. As stated in Section 3.8, the implementation was extended with the capability to process trees in parallel. To investigate the runtimes of the adapted implementation, we have prepared a forest with each tree containing 2000 data points. Different forrest sizes (10-200) were used to review the performance impact of splitting the workload on different processes. The parallel computing implementation contained substantial overhead - in order to achieve comparable results the original implementation was used for the trials using a single process. The parallel execution caused a speedup of the processing time between 1.8 and 2.6, compared to the seriell processing (Figure 3.13). The best runtime performance was achieved by utilizing the above mentioned data preprocessing steps in conjunction with the parallel handling of insertions.

3.10 EVALUATION

Before we begin investigating the results, a short disclaimer, the algorithm's long runtimes prevented extensive testing on longer timespans and different data sets to verify the approach's generalizability. The scalability issue needs to be resolved before applying the RRCF to network traffic can even be considered. We provide a few possible solution ideas in Chapter 4 to reduce the workload and increase performance. With these limitations in mind, we want to investigate the overall discrimination capabilities to see if future research is warranted.

3.10.1 *Hyperparameters*

One of the experiment's objectives was to understand the impact of the chosen hyperparameters on the model performance. We have gathered over 250 objective evaluations of the best optimization cycles for this analysis. While the focus is primarily on the achieved metric, we will also utilize the optimization process's properties to identify good hyperparameter settings. The optimization process chooses the hyperparameter combination in order to maximize expected improvement. Observing the relative number of evaluations done with each concrete hyperparameter value can also indicate parameter regions that should be considered.

Data Preprocessing

While scaling of the features turned out to be a necessity in terms of runtime performance, it was also needed to achieve the best detection performance. All implemented scaling techniques offered a performance increase compared to the original scaling. However, scaling according to the maximum absolute value (MaxAbs) was the overall best method.

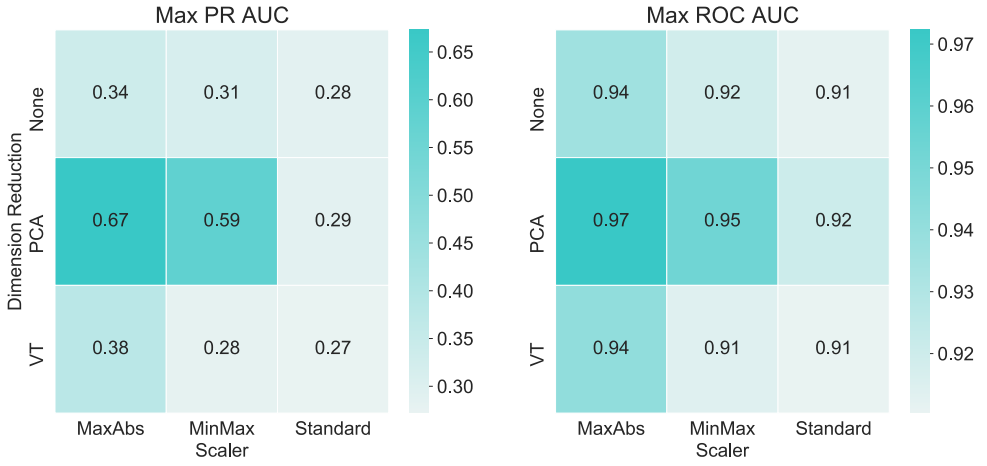

Figure 3.14: Max PR/ROC AUC over 250 optimization Itereations

Additional improvement in ROC AUC and PR AUC was achieved by applying PCA to all features with an explained variance of 75%. A combination of PCA and MaxAbs resulted in the best models, as seen in Fig.3.14. This is also clearly indicated by the evaluations done with each method, 50% of the objective evaluations were conducted with PCA and 64% utilizing MaxAbs. We found the data preprocessing steps to have the most impact on detection performance.

Model Hyperparameters

Focusing on the evaluations with the data preprocessing hyperparameters set to the above revealed that the algorithm is robust to the remaining model hyperparameters' choice. The deviations in scores were negligible for the most part, so that other factors should be considered when choosing the hyperparameters. The following will cover the suitable hyperparameter regions along with necessary considerations.

- **Number of Trees [10-100]**. The best models were achieved using 40 trees in the forest. However, the deviation was overall minimal, with promising results also produced with 90 trees. Considering the computational limitations, it is preferable to choose a smaller number of trees.
- **Tree Size [100-2000]**. Surprisingly, the best models were achieved with a small tree size of 100 to 300, although the deviations were small and bigger trees (e.g., 1900), achieved only slightly worse results. While preferable in the context of runtime, a small tree size sacrifices detection capabilities by the possibility of being altered by too many anomalies in a short period.
- **Precision**. Choosing a different floating-point precision for the features could not enhance model performance.
- **Subsampling**. The later introduced hyperparameter subsampling rate overall increased detection performance. No certain value could be identified, but a good range of values is 0.5 to 0.9. The rate indicates the proportion of trees that keep the new datapoint after calculating the impact on model complexity. Choosing a subsampling rate also comes with the additional benefit of extending the observed timeframe as some trees retain older data points.

The algorithm has shown to be robust to the other parameters' choice as the achieved performance was comparable throughout the parameter range, and no significant difference could be concluded.

3.10.2 *Performance*

Baseline

Finding the best hyperparameter combination for the baseline algorithm was achieved via the same optimization process. The data preprocessing parameters were the same while the model parameters were adapted to fit the provided parameters of the sklearn implementation [83]. The best detection was achieved by scaling the attributes (MaxAbs) with no additional dimensionality reduction.

Alogrihtm	**ROC AUC**	**PR AUC**
RRCF	**0.976**	**0.700**
Isolation Forest	0.951	0.544

Table 3.9: Baseline Comparison

Tab.3.9 shows the performance metrics of the best Isolation Forest compared to the best RRCF. The RRCF was able to outperform the Isolation Forest both in regards to the ROC Auc as well as the PR Auc. While the table only shows the best-achieved models, this was a trend throughout the testing as multiple optimization cycles of the RRCF outperformed the best IF. The IF's performance was substantially improved when using attack-free data for training; however, as discussed in Section 3.1, obtaining anomaly free data of a target network is not easily achievable.

Discrimination Capabilities

The experiment's main focus was to get an overall indication of the algorithm's discrimination capabilities, while actual detection would need some threshold or other means to translate anomaly score into an alert. The optimization process was conducted with a smaller subset of the data due to the runtime performance limitations. Therefore, we have chosen the best hyperparameter combinations achieved to be tested against the test set, as described in Section 3.4. The best performing models were chosen for the following evaluation (PCA, MaxAbs, 40 trees, 1900 tree size, and 0.9 subsampling).

Fig.3.15 shows the corresponding ROC and PR curves. The RRCF was able to achieve a high ROC AUC value of 0.976. However, as stated in Setion 2.3, it can be misleading to evaluate model performance with the ROC AUC when the data is imbalanced. A high value of correctly identified normal instances will shrink the FPR, resulting in an overall good AUC score. In the context of network anomaly detection, we are primarily interested in the performance with low false-positive rates as the high volume of traffic combined with the high cost of false alarms limits the false-positive rates,

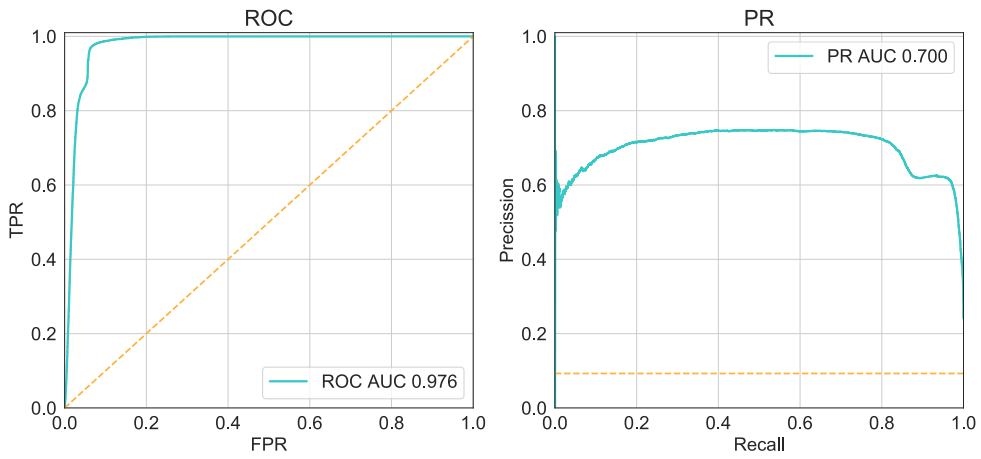

Figure 3.15: ROC and PR Curve

which can be tolerated. When controlled for a FPR of 1%, the RRCF was able to correctly identify 25% of the anomalies, a FPR of 2% resulted in a correct identification of 58%. Inspecting the precision-recall curve shows an average precision of 0.7, being almost constant overall threshold values. Indicating that, while the algorithm tends to score anomalies overall higher than normal instances, there is still significant overlap in the scores.

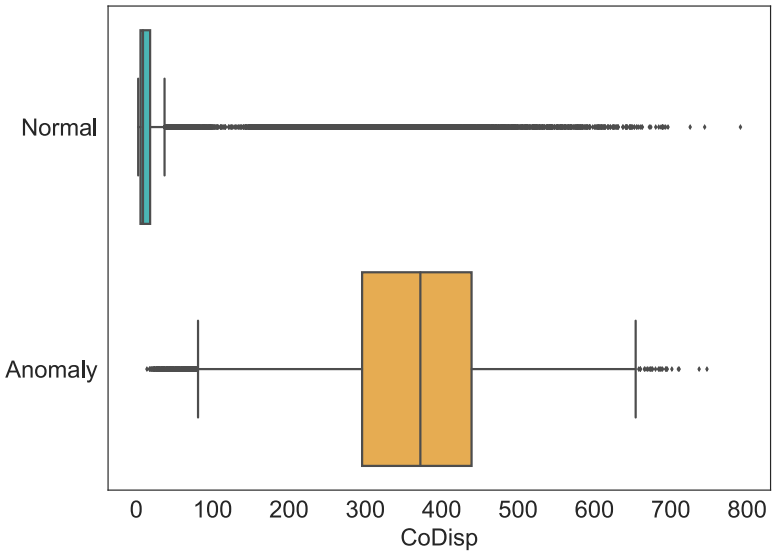

Figure 3.16: Normal and Anomaly CoDisp D istribution

This assumption is substantiated by the CoDisp distributions, Fig.3.16 shows the corresponding boxplots. It is clear that the RRCF scores the majority of anomalies and normal instances vastly different. However, swamping and masking (Setion 3.2), resulting in too many false positives caused by the overlapping of the distributions and the high volume of normal instances. Please note that raw CoDisp is dependent on the size of the used trees. The

values need to be scaled accordingly before comparing them to different tree sizes.

The used data set includes attacks from nine different categories. Using these categories, we can explore if the algorithm's possible expertise is in detecting certain kinds of attacks. Fig.3.17 shows the distributions of the attack categories. Disregarding the many false positives' problem shows that the algorithms' ability to score Worms and Shellcode attacks, especially well as a threshold of 200, would suffice for perfect detection. Worms and Shellcode were the attack with the least amount of flows within the test set (Table 3.2). The low volumne of these attacks, compared to the volume of the other attacks further indicate a susceptibility to outlier masking.

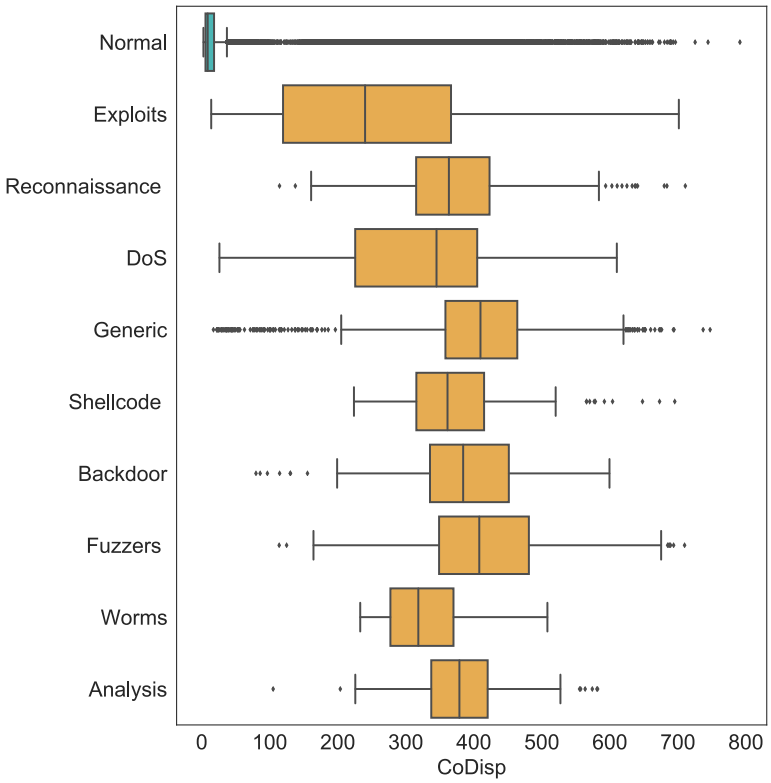

Figure 3.17: Attack Distributions

4

CONCLUSION AND FUTURE WORK

We improved the algorithm's detection capabilities by the implemented subsampling extension and the used data preprocessing steps. The algorithm showed promising discrimination capabilities but suffered, like many other anomaly detection techniques, from too many false alarms. However, the fact that the algorithm tends to score anomalies higher than normal instances solely based on the limited window of the current network traffic is auspicious in the context of network anomaly detection. One of the main issues of applying the RRCF algorithm to network traffic remains the difficulty of scaling the algorithm to accommodate large traffic volumes. We were able to substantially reduce the algorithm's runtime by extending it with parallel processing capabilities and the utilization of appropriate data preprocessing steps. However, the hardware used for testing can not be considered representative of a real-world use-case. The high computational requirements and the trend of network traffic volume overall increasing limit the possible application of the algorithm to detect network anomalies in the explored setting. However, expecting one detection mechanism to be the be-all and end-all solution is simply not realistic, as the research shows. Different mechanisms need to be considered, including host-based techniques, misuse detection as well as possible ensemble strategies for anomaly detection. We believe that the tree structure's intrinsic properties can be valuable for anomaly detection in the ever-changing network environment. Therefore, we want to propose possible solutions to overcome the limitations and guide future research efforts.

Scope

If no possibilities are found to improve the efficiency of the algorithm itself, other means are necessary. One proposed solution is to utilize the algorithm hierarchically. Suppose a two-layer detection system with layer one being a misuse approach. The misuse approach will filter out all normal traffic and anomalies, which it can identify with high certainty. Only flows where the misuse system is unsure would be delegated to the second layer, the RRCF. Utilizing this approach could substantially reduce the workload and, consequently, the number of expected false alarms. Furthermore, this system can be configured only to use flows classified as normal by layer one to update the trees used for anomaly detection. By intuition, this could result in higher detection rates as trees only reflect normal behavior while also limiting the danger of too many anomalies in a short period to alter the tree structure substantially. To clarify, suppose a traffic window with a majority of anomalous traffic. Using this sequence to update the trees could lead to the opposite of the desired detection, where the normal traffic is regarded

as anomalous. In theory, this approach could also help to support the basic assumptions of anomaly detection, namely that anomalies are rare and that anomalies are different in regards to their features.

Another option that should be considered is to subdivide the input data. One of the main issues when utilizing unsupervised anomaly detection on network traffic, regardless of approach, is the vast space of possible normal behavior. This problem could be mitigated if network traffic regions are identified where normal behavior is more precisely defined. Ad-hoc categorization could be done regarding the transport protocols or services. Other ways to categorize the data, such as machine learning approaches, should also be considered. While the pre-categorization of network traffic does not reduce the workload as such, it opens up the possibility of further parallelization. Furthermore, restricting the algorithm's scope to regions where normal behavior is better understood could increase the detection performance.

Stepping away from anomaly detection using flow data could also open up possible applications of the RRCF algorithm. The algorithm has shown to have adequate detection capabilities analyzing univariate time series data and concept drifts within the data [24]. Aggregating the flow data further and extracting features that describe the overall network characteristics enables the algorithm to be utilized to monitor the overall health of a network. For example, features such as bytes/s, open connections, or more sophisticated measures such as the number of unique IP pairings could be gathered in intervals and analyzed by the algorithm. We would obviously lose the ability to pinpoint anomalies in the same granularity as flow level analyzes can. However, we think that utilizing the RRCF as a network monitoring tool is an option worth exploring.

Decoupling Insertion and Scoring

Another option that can potentially increase the scalability of the algorithm is to decouple insertion and scoring. In the original publication of the algorithm, insertion is needed to score the data points. The measure is based on the impact to the model complexity caused by insertion or deletion, which are computationally expensive operations. Using different means to score new incoming traffic, which is not tied to the insertion, can allow for fast on-line detection while retaining the algorithm's adaptability. Fig.4.1 shows a basic example architecture to illustrate the proposed concept better. We retain two versions of the forests, one that scores the incoming traffic and a second one used to insert the new data points into the tree structures. The insertion module is organized in a queue like fashion, replacing the scoring module's corresponding trees when the insertion operation is completed. An apparent alternative score is the measure based on the path length as used by the Isolation Forest [35]. As the RRCF retains pairwise distances between data points, other measures such as density-based measures could also be utilized. Although the viability of such an approach could not be verified

during this work, other score measures should be explored to enable faster detection.

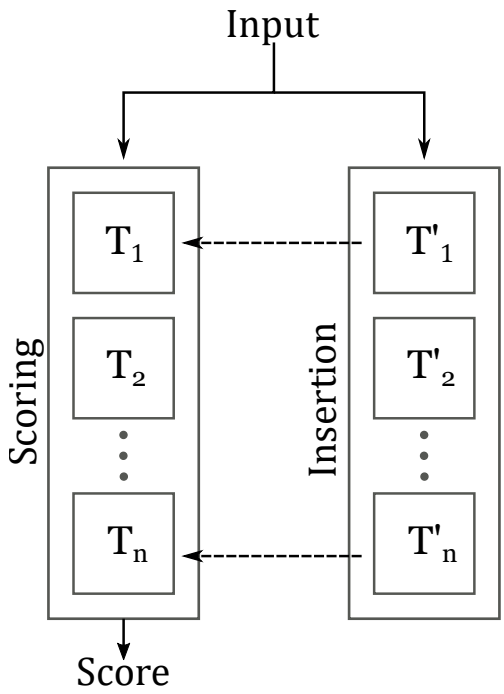

Figure 4.1: Example Architecture

Many questions about the suitability of unsupervised anomaly detection in the realm of computer networks remain open. Although public datasets' availability has improved over the last years, there is still a lack of true benchmark data to verify approaches and compare research. Two current datasets were examined for this work, both showing irregularities in the data compared to the accompanying publications. Additionally, these datasets contain huge volumes of network traffic leading to further segmentation during testing and thus further decreasing comparability.

We believe that the most important open question remains how much can we trust the prior assumptions to be true? We can not guarantee that anomalous traffic is rare in a dynamic computer network. While this can be better controlled when utilizing static models, it is especially a concern when using a dynamic model. This leaves us with, in our opinion, the most challenging problem left to solve: anomalies and normal instances need to differ in regards to their feature values. The whole endeavor stands and falls with this requirement, consequently emphasizing feature design rather than model selection. Intelligent feature design, especially regarding contextual features, can help to reveal anomalies within the data. However, this can only be verified against known anomalies, conflicting with anomaly detection's primary goal - to identify novel anomalies. While we do believe that unsupervised anomaly detection has its place in network anomaly detection, we want to conclude with this paradox to provide food for thought. Considering the

undefinability of normal network traffic boundaries, how can we make sure that anomalies manifest themselves in the data without prior knowledge?

Part II

APPENDIX

A

UNSW-NB15 FEATURES

	Name	Type	Description
1	srcip	nominal	Source IP address
2	sport	integer	Source port number
3	dstip	nominal	Destination IP address
4	dsport	integer	Destination port number
5	proto	nominal	Transaction protocol
6	state	nominal	"Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)"
7	dur	Float	Record total duration
8	sbytes	Integer	Source to destination transaction bytes
9	dbytes	Integer	Destination to source transaction bytes
10	sttl	Integer	Source to destination time to live value
11	dttl	Integer	Destination to source time to live value
12	sloss	Integer	Source packets retransmitted or dropped
13	dloss	Integer	Destination packets retransmitted or dropped
14	service	nominal	"http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service"
15	Sload	Float	Source bits per second
16	Dload	Float	Destination bits per second
17	Spkts	integer	Source to destination packet count
18	Dpkts	integer	Destination to source packet count

19	swin	integer	Source TCP window advertisement value
20	dwin	integer	Destination TCP window advertisement value
21	stcpb	integer	Source TCP base sequence number
22	dtcpb	integer	Destination TCP base sequence number
23	smeansz	integer	Mean of the flow packet size transmitted by the src
24	dmeansz	integer	Mean of the flow packet size transmitted by the dst
25	trans depth	integer	Represents the pipelined depth into the connection of http request/response transaction
26	res bdy len	integer	Actual uncompressed content size of the data transferred from the server's http service.
27	Sjit	Float	Source jitter (mSec)
28	Djit	Float	Destination jitter (mSec)
29	Stime	Timestamp	record start time
30	Ltime	Timestamp	record last time
31	Sintpkt	Float	Source interpacket arrival time (mSec)
32	Dintpkt	Float	Destination interpacket arrival time (mSec)
33	tcprtt	Float	"TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'."
34	synack	Float	"TCP connection setup time, the time between the SYN and the SYN ACK packets."
35	ackdat	Float	"TCP connection setup time, the time between the SYN ACK and the ACK packets."
36	is sm ips ports	Binary	"If source (1) and destination (3)IP addresses equal and port numbers (2)(4) equal then, this variable takes value 1 else 0"
37	ct state ttl	Integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).

38	ct flw http mthd	Integer	No. of flows that has methods such as Get and Post in http service.
39	is ftp login	Binary	If the ftp session is accessed by user and password then 1 else 0.
40	ct ftp cmd	integer	No of flows that has a command in ftp session.
41	ct srv src	integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
42	ct srv dst	integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
43	ct dst ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
44	ct src ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
45	ct src dport ltm	integer	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
46	ct dst sport ltm	integer	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
47	ct dst src ltm	integer	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
48	attack cat	nominal	"The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms"
49	Label	binary	0 for normal and 1 for attack records

BIBLIOGRAPHY

[1] *1998 DARPA Intrusion Detection Evaluation Dataset | MIT Lincoln Laboratory*. https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset. (Accessed on 11/29/2020).

[2] Sebastian Abt and Harald Baier. "Are We Missing Labels? A Study of the Availability of Ground-Truth in Network Security Research." In: *Proceedings - 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2014*. Institute of Electrical and Electronics Engineers Inc., 2016, pp. 40–55. ISBN: 9781479983087. DOI: 10.1109/BADGERS.2014.11.

[3] Ejaz Ahmed et al. "Use of IP Addresses for High Rate Flooding Attack Detection To cite this version : Use of IP Addresses for High Rate Flooding Attack." In: (2014).

[4] *Applications | Research | Canadian Institute for Cybersecurity | UNB*. https://www.unb.ca/cic/research/applications.html. (Accessed on 09/18/2020).

[5] *Australian Centre for Cyber Security | UNSW Canberra*. https://www.unsw.adfa.edu.au/tags/australian-centre-cyber-security. (Accessed on 11/29/2020).

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization." In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011* (2011), pp. 1–9.

[7] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305. ISSN: 15324435.

[8] Kevin Beyer, Jonathan Goldstein, and Uri Shaft. "When Is \ Nearest Neighbor " Meaningful ? Raghu Ramakrishnan 1 Introduction 2 On the Signi cance of \ Nearest Neighbor " 3 NN in High-Dimensional Spaces." In: *Young* ().

[9] Monowar H. Bhuyan, Dhruba K. Bhattacharyya, and Jugal K. Kalita. *Network Traffic Anomaly Detection Techniques and Systems*. 2017, pp. 115–169. ISBN: 9783319651866. DOI: 10.1007/978-3-319-65188-0_4.

[10] Anna L. Buczak and Erhan Guven. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection." In: *IEEE Communications Surveys and Tutorials* 18.2 (2016), pp. 1153–1176. ISSN: 1553877X. DOI: 10.1109/COMST.2015.2494502.

[11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. *Anomaly detection: A survey*. 2009. DOI: 10.1145/1541880.1541882.

[12] David Cortes. "Distance approximation using Isolation Forests." In: *ArXiv* abs/1910.12362 (2019).

[13] *DICOS Kommunikationssysteme*. https://www.dicos.de/. (Accessed on 12/02/2020).

[14] Dorothy E Denning. "Intrusion-Detection Model." In: 2 (1987), pp. 222–232.

[15] Andrew F. Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng Keen Wong. "Systematic construction of anomaly detection benchmarks from real data." In: *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description, ODD 2013* (2013), pp. 16–21. DOI: 10.1145/2500853.2500858.

[16] *File:TCP Header.svg - Wikimedia Commons*. https://commons.wikimedia.org/wiki/File:TCP_Header.svg. (Accessed on 11/30/2020).

[17] *Flexible Netflow Cisco :: NetFlow*. https://netflow.de/konfiguration/flexible-netflow-cisco/. (Accessed on 09/24/2020).

[18] Carrie Gates and Carol Taylor. "Challenging the anomaly detection paradigm a provocative discussion." In: *Proceedings New Security Paradigms Workshop* (2007), pp. 21–29. DOI: 10.1145/1278940.1278945.

[19] Thomer M Gil and Massimiliano Poletto. "Proceedings of the 10 th USENIX Security Symposium MULTOPS : a data-structure for bandwidth attack detection." In: *Statistics* (2001).

[20] *GitHub - ml-tooling/ml-workspace: All-in-one web-based IDE specialized for machine learning and data science*. https://github.com/ml-tooling/ml-workspace. (Accessed on 11/15/2020).

[21] *GitHub - phaag/nfdump: Netflow processing tools*. https://github.com/phaag/nfdump. (Accessed on 09/25/2020).

[22] Markus Goldstein and Seiichi Uchida. "Behavior Analysis Using Unsupervised Anomaly Detection." In: *The 10th Joint Workshop on Machine Perception and Robotics* October (2014). DOI: 10.13140/2.1.3349.7605.

[23] Markus Goldstein and Seiichi Uchida. "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data." In: *PLoS ONE* 11.4 (2016), pp. 1–31. ISSN: 19326203. DOI: 10.1371/journal.pone.0152173.

[24] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. *Robust Random Cut Forest Based Anomaly Detection On Streams*. Tech. rep. 2016.

[25] Sahand Hariri, Matias Carrasco Kind, and Robert J. Brunner. "Extended Isolation Forest." In: *IEEE Transactions on Knowledge and Data Engineering* (2019), pp. 1–1. ISSN: 1041-4347. DOI: 10.1109/tkde.2019.2947676. arXiv: 1811.02141.

[26] Hanan Hindy, David Brosset, Ethan Bayne, Amar Seeam, Christos Tachtatzis, Robert Atkinson, and Xavier Bellekens. "A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets." In: 1.1 (2018). arXiv: 1806.03517. URL: http://arxiv.org/abs/1806.03517.

[27] Victoria J Hodge and J I M Austin. "<2004. Hodge and Austin (2004). A Survey of Outlier Detection Methodologies.pdf>." In: 1969 (2004), pp. 85–126.

[28] Joseph D Houle, K K Ramakrishnan, Rita Sadhvani, Murat Yuksel, and Shiv Kalyanaraman. "The Evolving Internet - Traffic , Engineering , and Roles." In: (), pp. 1–23.

[29] *IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*. https://www.unb.ca/cic/datasets/ids-2017.html. (Accessed on 09/18/2020).

[30] *IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB*. https://www.unb.ca/cic/datasets/ids-2018.html. (Accessed on 09/18/2020).

[31] *Internet Assigned Numbers Authority*. https://www.iana.org/. (Accessed on 11/29/2020).

[32] S. Ashwini V. Jatti and V. J.K. Kishor Sontif. "Intrusion detection systems." In: *International Journal of Recent Technology and Engineering* 8.2 Special Issue 11 (2019), pp. 3976–3983. ISSN: 22773878. DOI: 10.35940/ijrte.B1540.0982S1119.

[33] Gisung Kim, Seungmin Lee, and Sehun Kim. "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection." In: *Expert Systems with Applications* 41.4 PART 2 (2014), pp. 1690–1700. ISSN: 09574174. DOI: 10.1016/j.eswa.2013.08.066. URL: http://dx.doi.org/10.1016/j.eswa.2013.08.066.

[34] Stephen Lau. "The spinning cube of potential doom." In: *Communications of the ACM* 47.6 (2004), pp. 25–26. ISSN: 00010782. DOI: 10.1145/990680.990699.

[35] Fei Tony Liu, Kai Ming Ting, and Zhi Hua Zhou. "Isolation-based anomaly detection." In: *ACM Transactions on Knowledge Discovery from Data* 6.1 (2012), pp. 1–44. ISSN: 15564681. DOI: 10.1145/2133360.2133363.

[36] Hongyu Liu and Bo Lang. "Machine learning and deep learning methods for intrusion detection systems: A survey." In: *Applied Sciences (Switzerland)* 9.20 (2019). ISSN: 20763417. DOI: 10.3390/app9204396.

[37] Markos Markou and Sameer Singh. "Novelty detection: A review - Part 1: Statistical approaches." In: *Signal Processing* 83.12 (2003), pp. 2481–2497. ISSN: 01651684. DOI: 10.1016/j.sigpro.2003.07.018.

[38] Pierre-François Marteau, Saeid Soheily-Khah, and Nicolas Béchet. "Hybrid Isolation Forest - Application to Intrusion Detection." In: (2017). arXiv: 1705.03800. URL: http://arxiv.org/abs/1705.03800.

[39] Preeti Mishra, Vijay Varadharajan, Uday Tupakula, and Emmanuel S. Pilli. "A detailed investigation and analysis of using machine learning techniques for intrusion detection." In: *IEEE Communications Surveys and Tutorials* 21.1 (2019). ISSN: 1553877X. DOI: 10.1109/COMST.2018.2847722.

[40] N. Moustafa and J. Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." In: *2015 Military Communications and Information Systems Conference (MilCIS)*. 2015, pp. 1–6.

[41] Nour Moustafa. "Designing an online and reliable statistical anomaly detection framework for dealing with large high-speed network traffic." In: 2017.

[42] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." In: *2015 Military Communications and Information Systems Conference (MilCIS)*. Vol. 8. 2 Special Issue 11. IEEE, 2015, pp. 1–6. ISBN: 978-1-4673-7007-3. DOI: 10.1109/MilCIS.2015.7348942. URL: http://ieeexplore.ieee.org/document/7348942/.

[43] Joshua Ojo Nehinbe. "A critical evaluation of datasets for investigating IDSs and IPSs researches." In: *Proceedings of 2011, 10th IEEE International Conference on Cybernetic Intelligent Systems, CIS 2011* (2011), pp. 92–97. DOI: 10.1109/CIS.2011.6169141.

[44] *Neptune.ai | Experiment tracking tool for you and your team*. https://neptune.ai/. (Accessed on 11/15/2020).

[45] *Nmap: the Network Mapper - Free Security Scanner*. https://nmap.org/. (Accessed on 09/22/2020).

[46] *Npcap: Windows Packet Capture Library & Driver*. https://nmap.org/npcap/. (Accessed on 09/22/2020).

[47] *Optuna: A hyperparameter optimization framework — Optuna 2.3.0 documentation*. https://optuna.readthedocs.io/en/stable/. (Accessed on 11/15/2020).

[48] R. Panigrahi and S. Borah. "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems." In: *International Journal of Engineering and Technology(UAE)* 7.3.24 Special Issue 24 (2018). ISSN: 2227524X.

[49] Vern Paxson and Sally Floyd. "Why We Don ' t Know How To Simulate The Internet 2 An Immense Moving Target 3 Heterogeneity Any Which Way You." In: *Simulation* May (1997). DOI: 10.1109/WSC.1997.640988.

[50] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. "Proactively detecting distributed denial of service attacks using source IP address monitoring." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3042 (2004), pp. 771–782. ISSN: 16113349. DOI: 10.1007/978-3-540-24693-0_63.

[51] Thomas Porter and Michael Gough. "Host-Based Intrusion Detection Sys- tems." In: (2007).

[52] *RFC 7011 - Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. https://tools.ietf.org/html/rfc7011. (Accessed on 11/29/2020).

[53] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. "Efficient algorithms for mining outliers from large data sets." In: *SIGMOD Record (ACM Special Interest Group on Management of Data)* 29.2 (2000), pp. 427–438. ISSN: 01635808. DOI: 10.1145/335191.335437.

[54] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, Andreas Hotho, and C R Mar. "A Survey of Network-based Intrusion Detection Data Sets." In: (), pp. 1–15. arXiv: arXiv:1903.02460v1.

[55] Takaya Saito and Marc Rehmsmeier. "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets." In: *PLoS ONE* 10.3 (2015), pp. 1–21. ISSN: 19326203. DOI: 10.1371/journal.pone.0118432.

[56] Rafath Samrin and D. Vasumathi. "Review on anomaly based network intrusion detection system." In: *International Conference on Electrical, Electronics, Communication Computer Technologies and Optimization Techniques, ICEECCOT 2017* 2018-Janua (2018), pp. 141–147. DOI: 10.1109/ICEECCOT.2017.8284655.

[57] Iman Sharafaldin, Amirhossein Gharib, Arash Habibi Lashkari, and Ali A Ghorbani. "Towards a Reliable Intrusion Detection Benchmark Dataset." In: Cic (2017), pp. 177–200. DOI: 10.13052/jsn2445-9739.2017.009.

[58] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." In: Cic (2018), pp. 108–116. DOI: 10.5220/0006639801080116.

[59] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection." In: *Computers and Security* 31.3 (2012), pp. 357–374. ISSN: 01674048. DOI: 10.1016/j.cose.2011.12.012. URL: http://dx.doi.org/10.1016/j.cose.2011.12.012.

[60] Jonathon Shlens. "A Tutorial on Principal Component Analysis." In: (2014). arXiv: 1404.1100. URL: http://arxiv.org/abs/1404.1100.

[61] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. "A Deep Learning Approach to Network Intrusion Detection." In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.1 (2018), pp. 41–50. ISSN: 2471-285X. DOI: 10.1109/tetci.2017.2772792.

[62] *Snort - Network Intrusion Detection & Prevention System*. https://www.snort.org/. (Accessed on 09/22/2020).

[63] Robin Sommer and Vern Paxson. "Outside the closed world: On using machine learning for network intrusion detection." In: *Proceedings - IEEE Symposium on Security and Privacy* (2010), pp. 305–316. ISSN: 10816011. DOI: 10.1109/SP.2010.25.

[64] *TCPDUMP/LIBPCAP public repository*. https://www.tcpdump.org/. (Accessed on 09/21/2020).

[65] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set." In: *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009* Cisda (2009), pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

[66] Mahbod Tavallaee, Natalia Stakhanova, and Ali Akbar Ghorbani. "Toward credible evaluation of anomaly-based intrusion-detection methods." In: *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 40.5 (2010), pp. 516–524. ISSN: 10946977. DOI: 10.1109/TSMCC.2010.2048428.

[67] *The UNSW-NB15 data set description*. https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA%NB15-Datasets/. (Accessed on 11/08/2020).

[68] *The Virus Changed the Way We Internet - The New York Times*. https://www.nytimes.com/interactive/2020/04/07/technology/coronavirus-internet-use.html. (Accessed on 12/01/2020).

[69] Kevin Thompson, Gregory J. Miller, and Rick Wilder. "Wide-area internet traffic patterns and characteristics." In: *IEEE Network* 11.6 (1997), pp. 10–23. ISSN: 08908044. DOI: 10.1109/65.642356.

[70] *Type I Error and Type II Error - Experimental Errors in Research*. https://explorable.com/type-i-error. (Accessed on 10/06/2020).

[71] Eduardo K. Viegas, Altair O. Santin, and Luiz S. Oliveira. "Toward a reliable anomaly-based intrusion detection in real-world environments." In: *Computer Networks* 127.August (2017), pp. 200–216. ISSN: 13891286. DOI: 10.1016/j.comnet.2017.08.013.

[72] Yun Wang. "Statistical Techniques for Network Security: Modern Statistically-Based Intrusion Detection and Protection." In: 2008.

[73] Charles Wheelus, Taghi M. Khoshgoftaar, Richard Zuech, and Maryam M. Najafabadi. "A session based approach for aggregating network traffic data - The SANTA dataset." In: *Proceedings - IEEE 14th International Conference on Bioinformatics and Bioengineering, BIBE 2014* (2014), pp. 369–378. DOI: 10.1109/BIBE.2014.72.

[74] *WinPcap - Home*. https://www.winpcap.org/. (Accessed on 09/21/2020).

[75] *Wireshark · Go Deep*. https://www.wireshark.org/. (Accessed on 09/22/2020).

[76] Charles V. Wright, Lucas Ballard, Fabian Monrose, and Gerald M. Masson. "Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob?" In: *16th USENIX Security Symposium* (2007), pp. 43–54.

[77] Ting Fang Yen, Xin Huang, Fabian Monrose, and Michael K. Reiter. "Browser fingerprinting from coarse traffic summaries: Techniques and implications." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by Ulrich Flegel and Danilo Bruschi. Vol. 5587 LNCS. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 157–175. ISBN: 3642029175. DOI: 10.1007/978-3-642-02918-9_10.

[78] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque. "Random-forests-based network intrusion detection systems." In: *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* (2008). ISSN: 10946977. DOI: 10.1109/TSMCC.2008.923876.

[79] Arthur Zimek, Erich Schubert, and Hans Peter Kriegel. "A survey on unsupervised outlier detection in high-dimensional numerical data." In: *Statistical Analysis and Data Mining* 5.5 (2012), pp. 363–387. ISSN: 19321872. DOI: 10.1002/sam.11161.

[80] *anomaly | Origin and meaning of anomaly by Online Etymology Dictionary*. https://www.etymonline.com/word/anomaly. (Accessed on 11/29/2020).

[81] "rrcf: Implementation of the Robust Random Cut Forest algorithm for anomaly detection on streams." In: *Journal of Open Source Software, The Open Journal* 4.35 (2019), p. 1336.

[82] *sFlow Version 5*. https://sflow.org/sflow_version_5.txt. (Accessed on 11/29/2020).

[83] *sklearn.ensemble.IsolationForest — scikit-learn 0.23.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html. (Accessed on 11/23/2020).