University of Zurich^UZH

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

DDoSGrid-Mining: Analyzing and Classifying DDoS Attack Traffic

Luc Boillat
Zurich, Switzerland
Student ID: 14-715-577

Supervisors: Muriel Franco, Eder Scheid
Date of Submission: March 3, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Aufgrund der Nachfrage nach Systemen, welche Distributed Denial of Service (DDoS) Attacken erkennen, analysieren und mitigieren können, ist DDoSGrid entstanden. Eine Plattform welche Post-Mortem-Analysen von Netzwerkattacken ermöglicht. Der Fokus lag auf einer forschungsorientierten und erweiterbaren Plattform, welche Merkmale von Netzwerkaufnahmen extrahiert und durch Visualisierungen einem Benutzer hilft, Attacken zu erkennen und zu klassifizieren. Jedoch fand bisher diese Erkennung immer manuell statt, indem der Benutzer die Visualisierungen und Metriken anschaute und dann eine fundierte Entscheidung traf.

Diese Arbeit erweitert nun die DDoSGrid-Funktionalität, indem eine Machine-Learning basierte Angriffklassifizierung der Netzwerkaufnahmen konzeptualisiert und implementiert wurde. Zu den bestehenden Merkmalextraktoren wurde ein zusätzlicher entwickelt, welcher Zeitfenster-basierte Merkmale erstellt. Diese Zeitfenster von bekannten Attacken können während der Analyse manuell nach Angriffstyp klassifiziert werden, sodass ein Modell aus korrekt klassifizierten Attacken entsteht. Dieses Modell kann dann benutzt werden, um neue Aufnahmen mit verschiedenen Machine-Learning-Algorithmen automatisch zu klassifizieren. Die Applikation wurde mit forschungsüblichen Methoden evaluiert und stellt sich als sehr effektiv heraus, da die Performance und Genauigkeit mit vergleichbaren Beispielen aus der Literatur mithalten kann oder sogar besser abschneidet.

In order to provide a solution that aids in analyzing and detecting ever-increasing Distributed Denial of Service (DDoS) attacks in post, DDoSGrid was created, which offers a research-oriented and extensible platform that provides visualizations for analyzed traffic logs and lets a user configure dashboards using these visualizations. However, this analysis and attack-type detection was until now performed manually, by a user looking at visualizations and metrics and making an informed decision on what attack vectors could have been used in an attack.

This thesis expands on the base-idea of DDoSGrid and extends it by conceptualizing and implementing an extension that allows machine learning based attack-type classification of the uploaded data sets. In addition to the already existing feature extractors that DDoSGrid provides, a new extractor is created that creates time-window based features of the traffic log. These logs can be manually classified, and then added to a machine learning model, to create a true data set. This model is then used to automatically classify new data sets using different classification algorithms, in order to get an attack-type analysis of the traffic log. The solution was evaluated using well-established techniques and proved to be quite effective, both in terms of performance and accuracy, performing comparably or in some cases better than the existing literature.

Acknowledgments

I would like to thank my main supervisor, Muriel Franco for his excellent guidance and support during the writing of my Master Thesis. His knowledge on network security, visualizations and machine learning has been a great help and inspiration, from the inception of the DDoSGrid project, to the completion of this thesis derived from it.

Additionally, I'd like to thank Prof. Dr. Burkhard Stiller for letting me write this thesis at the Communication Systems Group, as well as for the CSG's lectures and seminars I was able to attend during my time as a student at the University of Zurich. It was truly a pleasure to write this and the previous theses at the CSG.

Contents

Chapter 1

Introduction

With an increasing number of network-connected devices, the potential to use these often unsecured devices in malicious ways grows in a similar fashion. Specifically, Distributed Denial-of-Service (DDoS) attacks where a variety of devices such as IoT hardware, regular modern computers or even legacy devices are used to drain resources of targets in an organized way, have become more frequent and severe in recent years [6]. Naturally, the consequence of this increase in attacks, is the rising cost of damages caused by them [24, 6]. In order to protect infrastructures from potentially malicious traffic and clients, many kinds of defense and analysis systems exist, that aid operators in detecting and countering these attacks. Since computer-processing power has become cheaper and more widely available with the introduction of cloud computing platforms, Machine Learning (ML) techniques have become an attractive option to offer assistance in data processing and pattern recognition pipelines, targeting unusual network traffic.

1.1 Motivation

Network-based attacks have become much more severe in both their volume and traffic [6], as well as costly for the targeted network or infrastructure, since service failures downtimes often lead to losses for the period of unavailability. It is estimated, that the number of DDoS attacks will double between the years 2018 and 2023, from 7.9 million to 15.4 million [6]. With the cost of a DDoS attack being approximated to be around \$20'000 to \$40'000 per hour [4], the need arises to quickly and efficiently detect and mitigate such attacks at both source and target levels. Current detection systems often analyze network traffic either on a per-packet basis, to gather information such as source location, connection duration and connection type, as well as by gathering metrics across all packets, such as number of connections over time or traffic features clustered by source location. In recent years, research was conducted that suggests using machine learning techniques could help automatically identifying the type of DDoS attack performed on a network [30]. For this, existing network capture data is used, certain features are extracted from them and fed to the machine learning engine to train the proposed models. New traffic can then be classified based on these trained models.

However, there are still challenges and opportunities on the research side to be addressed by both, academia and industry in order to have successful applications of machine learning in cyber security and on DDoS attacks specifically. Examples of such opportunities include *(a)* the integration of ML techniques with insightful visualizations to provide insights for network operators about possible cyber attacks and *(b)* the analysis of DDoS attacks to improve the understanding of DDoS attack characteristics in order to improve the efficiency of proactive and reactive approaches against such attacks. Besides that, ML techniques can be applied to automate different steps of a post-mortem DDoS analysis, which would be critical when analyzing a huge amount of data as those predicted to happen in DDoS attacks that incorporate 5G and IoT scenarios.

Network attacks also undergo constant changes in their behavior and the systems they target and use for their malicious intents. For example, memcached attacks are a new type of DDoS attack that have been become increasingly popular in recent times [12], due to the rather new mechanisms on the internet that allow them to occur, namely caching servers. In order to defend against such network attacks, Intrusion Detection Systems (IDS) are often used, which monitor traffic, system resources, as well as overall health of the infrastructure. DDoS detection and mitigation can also be outsourced to other infrastructures that offer such services, so called off-premise mitigation.

1.2 Description of Work

This thesis reviews the techniques proposed in the literature and develops a protoype machine learning based attack-type classifier integrated into DDoSGrid, with the goal of analyzing historic traffic data into different states of attack-types. DDoSGrid is a research-oriented DDoS attack visualization platform that is extensible and provides feature extraction and visualization of network capture files, primarily PCAP and TCPDUMP files. Based off of the existing research, a set of features are established that aid in recognizing different attack-types. A feature extractor is created as part of the DDoSGrid miner component, that extracts these features from packet captures. Additionally, a machine learning pipeline is created that uses these features sets for classification and model-training. A performance evaluation is then conducted to assess whether the prototype is a feasible tool to detect DDoS attack-types.

1.3 Thesis Outline

Chapter 2 documents the background information relevant to this thesis, such as DoS attacks, DDoS detection, as well as machine learning. In Chapter 3, an in-depth literature review is conducted, reviewing and summarizing the existing literature surrounding ML-supported DDoS attack detection and classification. Chapter 4 handles the selection of extracted data features and machine learning algorithms, as well as documents the preliminary test runs conducted for selecting the algorithms that will end up in the finished solution. Furthermore, architectural changes are discussed, detailing new and changed components of DDoSGrid. Chapter 5 documents the design and implementation-related steps taken, to include the ML-module into DDoSGrid. In Chapter 6, a performance evaluation is held, where factors such as speed and common ML metrics of the automatic classification functionality as part of DDoSGrid are assessed. Chapter 7 presents a summary and the author's conclusions of the thesis, as well as suggestions for future work to be done.

Chapter 2

Background

The following chapter discusses the background information regarding the topics touched upon in this thesis. First, an introduction to Denial-of-Service (DoS) attacks is presented, along with a discussion of different types and features of such attacks. Then, an overview of systems and tools that aid network operators in detecting and classifying such attacks is given. Lastly, machine learning and its most common types of algorithms are documented.

2.1 Denial of Service (DoS) Attacks

In a Denial of Service (DoS) attack, an attacker tries, as the name suggests, to stop a target from serving certain resources or to stop its operation completely, by overwhelming the target's infrastructure [23, 24]. This can be done in many ways, such as by sending large amounts of specific traffic to the target or by exploiting flaws in frequently used protocols that leave a node vulnerable. These attacks are often backed by malicious intent and may bring the attacker some sort of benefit, such as reduced competition. DoS attacks can be categorized into 2 distinct mechanisms, one being a *Buffer Overflow* attack where the target is forced to give up all its resources and capabilities because disk space, memory or CPU cycles are depleted in some way. The other category is comprised of *Flooding Attacks*, where the malicious actor uses up bandwidth of the target by sending an enormous mount of traffic and/or requests to the target, exhausting its networking capabilities for honest nodes to connect to the targeted node [7]. Examples for effects of such attacks include online stores being pulled offline, or other services such as financial transfers being halted, resulting in potentially large financial losses for the target device [23, 24].

Distributed Denial of Service Attacks (DDoS) work in a similar way with the key difference being that the malicious actor does not necessarily use his own machine but a myriad of controlled computers (so called bots) that execute the attack in parallel [22, 23]. This results in an orchestrated attack on the target, making it difficult to distinguish honest nodes from malicious ones, since an isolated source node may not look suspicious at all. Thus, research efforts in detection, defense and mitigation of such attacks have become common in recent times [30, 28, 32]. The two general attack categories for DoS attacks also apply to DDoS attacks, namely the overflowing of buffers (hardware) at the target, as well as the exhaustion of bandwidth, resulting in a loss of service.

The way an attacker gains control of such a large amount of nodes can vary [22]. Some botnets, such as the *Mirai* Botnet exploit vulnerabilities in hardware and software to push self-spreading malware onto target devices. The *Mirai* Botnet for example, targets internet-connected Linux devices such as IoT and other smart devices. By brute forcing factory passwords and usernames on the devices, which are often easily identified on a network, many such devices can be infected with the malware in a short amount of time [17, 18]. Botnets can also be formed voluntarily, as was the case with the Low Orbit Ion Cannon [16] used by 4Chan-affiliated hackers to target services such as Visa and Mastercard [8].

One can also differentiate between reflection and amplification attacks [21]. The former often uses packets that contain a spoofed IP address, namely the one of the target's exposed service, and sends them to available services that responds to such requests. Since the server does not realize that the IP address is spoofed, the response get returned to the target, resulting in large amounts of responses arriving at the same node, with the result of it being unable to process all of them. The latter uses specific services as intermediary nodes that reply with responses that are much larger than the original request, such as DNS resolvers or memcache servers. This can be done with a comparably small amount of bot computers that perform these kinds of attacks, since the original requests get amplified in packet size. Both attack methods can lead to a congestion of bandwidth or processing resources at the target's side. [21].

2.1.1 Types of Attacks

The following section describes some of the more frequently used attack types. For the purpose of clarification, specific attacks are divided into three distinct categories: *Volumetric Attacks*, *Protocol Attacks* and *Application Layer Attacks*. In Table 2.1, the most common types of attacks are presented, along with the categories they belong in and the most distinct features that can be found in traffic logs of such specific attacks.

Volumetric Attacks: Volumetric DDoS attacks are characterized by the overwhelming data and traffic that is sent to the target. This volume uses up all the available bandwidth and results in the offered service being unavailable. Sorted by frequency of occurrence, volumetric attacks are by far the most common type of attack with a share of about 65%, according to Arbor [5]. Furthermore, these large amounts of data do not necessarily have to be generated at the attackers side, since often reflection and amplification techniques are used, which do not require large botnets to be executed. Examples of such reflection and amplification attacks include NTP attacks and DNS amplification attacks. In these attacks, an intermediary service, NTP or DNS servers in the above mentioned, cases are flooded with many requests that contain spoofed source-IPs in their packets. This causes the servers to respond to all the requests at the same target location (reflection). In the DNS amplification case, this response-to-request data ratio is significantly higher (amplification), resulting in the target's servers bandwidth being used up, without using

Attack	**Type of Attack**	**Traffic Features**
SYN Flood	Protocol Attack	Many open connections on server side. High ratio of SYN and SYN-ACK to ACK packets.
ICMP Flood	Volumetric Attack	High amounts of Ping and Echo packets. High amounts of Ping packets per source IP.
UDP Flood	Volumetric Attack	High amounts of 'ICMP Destination Unreachable' packets from target.
Ping of Death	Protocol Attack	Large ping request packets
Teardrop Attack	Protocol Attack	Heavily fragmented TCP packets, that clog processing power of the target node.
SSDP Attack	Protocol Attack	UPnP responses
NTP Attack	Volumetric Attack	Large amounts of monlist requests to NTP servers and high ratio of response to request packet size.
Memcached Attack	Protocol Attack	Large UDP responses to spoofed IP addresses for small requests to Memcache servers.
DNS Floods	Application Layer Attack	Large ratio of response to request packet size in DNS traffic.
DNS Amplification	Volumetric Attack	Large amounts of traffic sent to infrastructure-related DNS servers.
HTTP Flood	Application Layer Attack	Many HTTP-GET or POST requests for the same resource in rapid succession.
Multivector Attack	Combination	Simultaneous or sequential combination of features mentioned above

Table 2.1: Overview of DDoS Attack Types and Their Features

as many malicous nodes as in the the reflection attack's case. Should the attacker have large amounts of bot nodes that they can use, a simple ICMP Flood or UDP flood where all nodes bombard the target with ping requests or UDP packets at various ports can be launched at a target. The target then has to echo the ping request, or in the UDP flood's case check, whether it is serving something on that port, and in most cases has to respond with an *ICMP Destination Unreachable* packet (often to a spoofed source IP), clogging its bandwidth as well. A recent trend in volumetric attacks has become using memcached attacks, where memcache servers are requested to respond with large payloads. However these payloads are reflected to the same victim. The DDoS attack on GitHub in 2018 was a memcached attack, resulting in the largest attack ever recorded for that time with a recorded bandwidth usage of 1.7 Terabytes per second [12].

Protocol Attacks: These kind of attacks center around exploiting a characteristic or flaw in the protocols on the OSI-Layers 3 and 4 (network and transport layers). In comparison to volumetric attacks, where bandwidth is used up, protocol attacks often cause the hardware in the target to reach its capacity by using up memory, clock cycles or causing errors in computation. The attacker uses a botnet to generate as much traffic as possible, targeting the victim. A simple example of a network protocol attack is the SYN-Flood attack, where the bots all send TCP-SYN requests to the target, which has to respond to all of them with a SYN-ACK packet. However, the bots ignore the response and continue to make SYN request. This causes the target node to be unable to respond to any new requests, since all possible open connections are used up by waiting for ACK requests from the bots. Honest nodes will now be unable to request service from the target, rendering it unreachable. In the case of teardrop attacks, the attacker sends fragmented TCP packets, which the target is unable to reassemble due to a bug in the TCP/IP protocol. This leads to a packet overlap which crashes the target. An example of a reflection attack would be an SSDP attack, which exploits devices that have Simple Server Discovery Protocol enabled and face the internet. Using the UPnP protocol, spoofed UDP packets are sent to various SSDP-enabled devices, all responding to the same target, eventually overwhelming it and causing it to stop serving its purpose.

Application Layer Attacks: The attacks described above all occur on layers 3 or 4 on the OSI reference layer, however it is also possible to achieve a denial of service on the application layer. The difference in effect these types of attacks have is that not just network resources are exhausted in the target, but the node's hardware and external resources such as databases may also be affected. Additionally, less network traffic may be required to achieve a denial of service, because of the additional system load the target generates by itself. A simple example of an application layer attack is a HTTP Flood attack, where a botnet requests the same resource of a node using either HTTP GET or HTTP POST requests, requiring all of the target's resources and making it unable to successfully handle all the incoming traffic. Similarly, DNS Flood attacks target a DNS server with requests, rendering it unable to process honest DNS requests. This is often used for DNS servers that belong to a certain infrastructure or company.

2.1.2 Multi-Vector Attacks

A large portion of DDoS attacks nowadays are so-called multi-vector attacks [4], meaning not just one type of attack is used at the same time. this can make detection, as well as mitigation harder for the defending target, since multiple facets of traffic need to be analyzed in order to form an opinion on irregular traffic [23]. Often, combinations of amplification and reflection vectors are used, since the two mechanisms combined can form massive amounts of data and traffic, which is sent to the target [19]. When defending against multi-vector attacks, one needs to keep multiple points in mind; All used vectors need to be identified to effectively combat the attack. The attack may hit on different layers of the OSI model. The attack vectors are not synchronized [20], meaning the timing of the vectors can be offset. Lastly, the combination of techniques makes defense in general more difficult due to resource exhaustion at multiple levels. These points lead to an ever changing mitigation procedure during the course of an attack [20].

2.2 DDoS Detection and Classification Methods

One major challenge in detecting DDoS attacks is the fact that, although the outcome of different types of attacks may be the same i.e. the blockage of a service on a network, the methods that allow this outcome do not necessarily share the same features. As mentioned in Table 2.1, different characteristics exist for all attack types. This, together with the fact that attackers use spoofing methods to hide their true identities, require effective DDoS detection and mitigation systems. Khalaf et al. [30] split DDoS defense and detection techniques into two distinct classes; Artificial Intelligence-based (AI) and Statistics-based. AI-based methods focus on building models on previous data that can predict future data and/or outcomes. The more traditional methods of DDoS defense are based on statistical models, that analyse current numbers based on previous averages and use thresholds and rules to decide whether abnormal traffic patterns are occurring on the server side.

Statistical methods can be further split into parametric and non-parametric methods. With the former being based on techniques that include threshold-based anomaly detection, spectral analysis of data and the usage of confidence interval anomalies based off past data. Disadvantages of these types of DDoS detection include limited support for a wide variety of possible attack vectors or being better suited for regular DoS attacks instead of distributed ones. Non-Parametric methods also base their decision on past data, but unlike parametric methods, they form the rules and patterns themselves in an iterative manner. Notable examples of non-parametric methods include the Markov Method and time series analysis. Markov-based systems form their decisions based on event patterns that occur on the network and evaluate new events on precalculated probabilities of these events actually occurring. The events are used to form state transition matrices that predict these probabilities. Time series analyses make use of models that are formed by the occurrence of events. Should low-probability events appear on the network, appropriate action is carried-out that filters such events. Drawbacks of non-parametric DDoS defense methods are similar to parametric methods, in that they are not suited for a wide variety of DDoS attack types.

2.3 Machine Learning (ML)

With a rise in malicious attacks on the internet, arose a need to successfully detect and mitigate such attacks. Not only DoS is a threat on the internet, but also fraud, intrusions and malware, to name a few. Since the early days of email, and still to this day, spam and the detection of it, has been an ongoing cat and mouse game between the attackers and email service providers. Naturally, the goal was to detect spam in the most efficient way, so automated classifiers and statistical models that collect data from previous spam emails were used to analyze new emails and classify them as either spam or honest mail [15]. While spam-detection programs were not the first to use machine-learning algorithms, the amount of available data to train the algorithms offered its application to be a successful undertaking.

In recent years, with the prevalence of mass data collecting and user tracking, more and more fields are suited to have use cases for machine learning. Be it for prediction of user needs based of past data, or the classification of network attacks, the spectrum of applications is an ever-growing market. Additionally, the companies that are frequently on government's radars for questionable practices regarding consumer protection and data privacy, are the same companies that offer large cloud computing infrastructures with built-in machine learning suites, so the incentive to use machine learning techniques has become higher in recent years.

The need to classify these machine learning techniques arose, since various approaches are available and used today. Not every approach works well for every use case, and some use cases might not even benefit at all from machine learning.

When working with data sets, we need to distinguish, whether the set is well labeled and categorized, or not. The type of data set determines what type of learning can be used in that specific use case. Is the data set labelled, supervised learning techniques are used, that use the known training data as a resource to classify new test data [15]. An example of supervised learning is as follows: A collection of different rocks is to be classified by type. In the training data, different rocks are categorized by the visual and haptic features they present; Some rocks have rough surfaces, while others are smooth. The rocks have different colors and homogeneity of materials. All these combinations of features can be used to determine the type of rock. Should a new rock be analyzed, the combination of its features are now used to determine its type using the data it collected from the existing collection of rocks. Supervised learning can further classified into two subcategories; Classification and Regression [15].

Should the input data be neither labelled nor classified in advance, we speak of unsupervised learning, since the algorithm has to find patterns, similarities and connections on its own. Unsupervised Learning can also be classified into two subcategories: Association and Clustering [15].

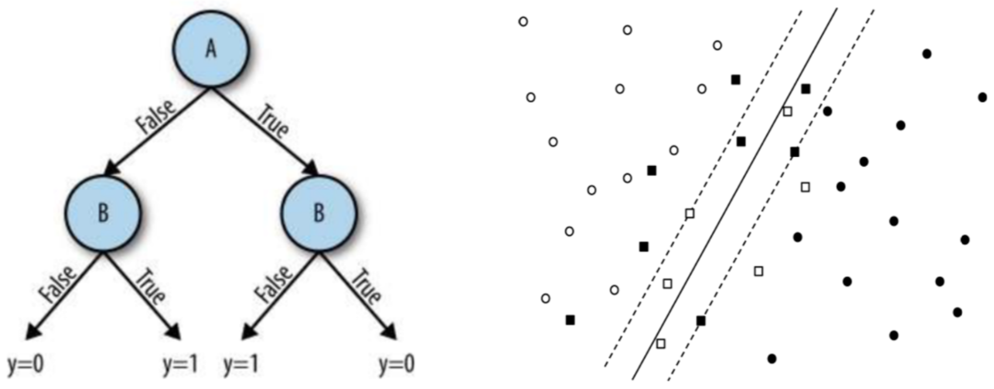


Figure 2.1: Two examples of classification algorithms. On the left a basic decision tree is seen, while on the right a linear SVM can be seen, dividing two types of data points. [15]

2.3.1 Classification and Regression

In the example above, the goal of the machine learning algorithm was to classify a new rock, based on labelled features of previously analyzed rocks. The output of such an analysis would be a class-based result, such as ”Granite” or ”Sand Stone”. These result classes can be either binary (True/False, Above/Below etc.) or multi-classed such as in the rock example. Taking classification-based machine learning into a context of computer networks, features of network traffic could be the identifying properties for the type of network attacks. In Figure 2.1, we can see two examples of classification algorithms. On the left there is a simple decision tree, that splits features on a boolean condition, with the result being a leaf node that represents a state of the item that is classified. On the right, an illustration of a Support Vector Machine is shown, where a linear vector divides two types of data points, here being black and white dots, so that both sides of the vector contain as many data points as possible.

However, what if the resulting value should not be a predefined category such as a string or a range of integers? In this case, Regression-based machine learning techniques are used, where the predicted value should result in a continuous number. Often, these techniques are using time-based features that can result in the prediction of numbers (Time Series Analysis) [15].

2.3.2 Association and Clustering

In association-based algorithms, the relations between the different samples are evaluated. The goal is to determine occurrences of values, when combinations of other values are also present. This way, combinations of e.g. items in a supermarket basket can predict additional items that will be put in the basket as well.

When using clustering algorithms such as k-Means on unlabelled data, the goal is to group the data into clusters that show similar properties. This can be visualized by plotting

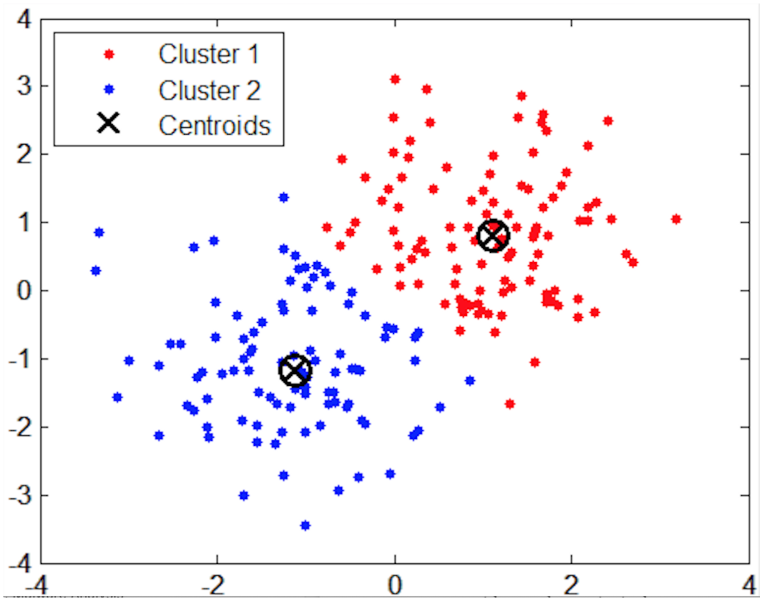


Figure 2.2: An example of k-Means clustering, where data points are clustered around 2 centroids that represent two different groups inside the data set. [9]

two-dimensional data onto a grid and setting central points of possible clusters, so called centroids. Depending on the location of these centroids, different clusters can be formed that rely on specific features and similarities of the data points [15]. In Figure 2.2, we can see that the centroids are placed in a way so that all data points have the minimum possible distance to their respective centroids.

Chapter 3

Related Work

This chapter focuses on the existing literature and technologies that set the foundation for this thesis. First, existing approaches in ML-supported DDoS attack recognition are reviewed and categorized by methods and extracted data set features they use as classification parameters. Secondly, an introduction to the existing DDoSGrid [1] platform, upon which our prototype is built is given, followed by tools and services that may serve as data set providers. From these data sets, the required features should be extracted to serve our prototype with attack-traffic data, such as DDoSDB [13], DDoS LogSim [14] and the DARPA intrusion detection tests, performed in 1998, 1999 and 2000 [42].

3.1 ML Applications in DDoS Attack Recognition

In recent years, the use of Machine Learning (ML) techniques has become a popular research field with many possible applications. With the large amounts of data available from network captures, the use of these data-focused algorithms may prove to be a viable technology in detecting network attacks, as opposed to the more traditional methods of detection. The traditional methods of attack-detection often use statistical models [28], such as the detection of significant deviations and anomalies in attack traffic [39]. Other research efforts include using time series analysis [40] and Exponentially Weighted Moving Average (EWMA) calculation [41]. A drawback of these traditional methods is the need for thresholds that act as a cut-off to determine, whether a DDoS attack is occurring or not. These thresholds have to be set manually and may be different for every scenario, depending on standard network load, network size and service-offered by the target nodes.

In the existing literature, the workflow of the authors evaluating ML algorithms for DDoS detection can be generalized in the procedure visualized in Figure 3.1. The approaches differ in the execution of the procedure's points due to the various data feature and algorithm combinations that are possible. The first step was the collection of network data sets. These usually consisted of raw network captures such as Packet Capture (PCAP) files, or statistical network information in NetFlow format. The origin of this data also differed, in that some approaches used generated traffic from laboratory network setups

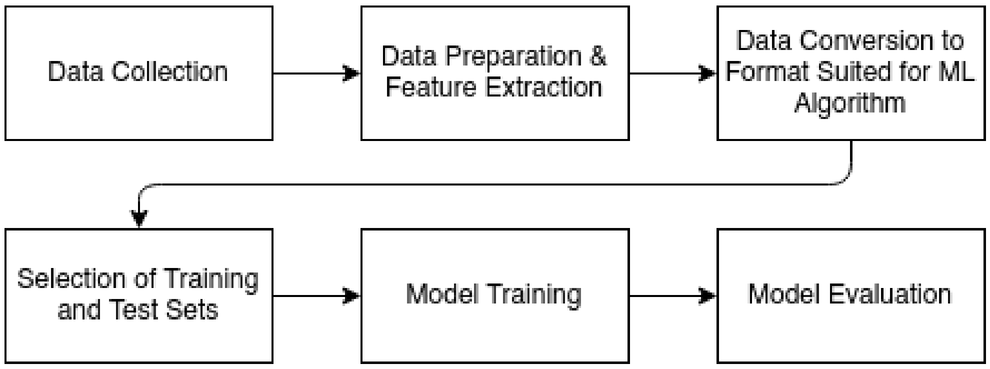

Data Collection
Data Preparation & Feature Extraction
Data Conversion to Format Suited for ML Algorithm
Selection of Training and Test Sets
Model Training
Model Evaluation

Figure 3.1: Generalized Workflow in ML-supported DDoS Attack Detection

that simulated DDoS attacks over time, and others used publicly available datasets, such as the DARPA [42] or CAIDA [44] sets which have been used in many DDoS related research efforts. The next step resolved around the preparation of the data and the extraction of features that are beneficial in determining the state of the network. Here again, various approaches were used, such as the distinction of feature sets that calculate statistical information over time and sets that are categorized on a per-packet basis. Due to the large amount of possible data in network captures, some used methods to shrink the data sets by sampling only a fracture of the data sets. Since different ML types and algorithms were used, the prepared data is converted to a format or data structure that can be fed into the algorithms. The next step is the division of the data set into training and test sets, where both sets contained regular and DDoS traffic, so that the model can be adequately trained. These training sets are then fed to the algorithm to train the model. Lastly, the test sets are used to evaluate the model using cross-validation techniques, which focus on metrics, such as accuracy, precision, recall and f1-score. In general, most approaches achieved very high accuracy values, often above 95%. Other factors that are considered during evaluation, are the speed of training and testing, as well as the possibility to use the approaches in real-time, as opposed to post-mortem analysis.

Table 3.1 provides an overview of notable examples of research efforts in detecting DDoS attacks. Note that some approaches only had the goal of binary DDoS attack detection, whereas others focused on detecting specific attack types. [27] evaluated the binary detection, focusing on two ML techniques: Support Vector Machines (SVM) and Deep Feed Forward (DFF). The two algorithms were compared by the accuracy and classification times they achieved in a cross-validated evaluation of the models. One distinction they made when it came to selecting features was, that they extracted two feature sets from the DARPA data set, namely one set based on time-window characteristics like number of IP packets, number of different source and destination IPs. The other set was based around packet-related information, such as packet type, time-to-live of the packet, etc. The total number of features amounted to 42, which in contrast with other papers is a very large amount. [37] and [36] only used four and five features, respectively. One possible reason for the usage of that many features was the fact that the DARPA data set contained many different DDoS attack types, so the authors wanted to cover all possible variations in traffic.

Another approach by Hou et al. [28], evaluated the Random Forest (RF) algorithm's accuracy by extracting features from captured network logs in a test network setup. This data was not in PCAP format like the DARPA set, but in NetFlow format, which focuses more on statistical measures of the traffic. Evaluation of the model was then performed using the CIC-IDS2017 datasets [45], which resulted in a very high accuracy of DDoS detection of 98.6%. Since the data they generated and collected were large amounts of data, they also experimented with data-sampling techniques to reduce the amount of input data, while keeping detection accuracy high. Extracted features of the data set included both pattern- and flow-based features. Their approach to feature extraction was done in real-time.

[29] proposed and evaluated a DDoS detection solution that aims at preventing attacks at the source side, in this case from attacks that originate from cloud-computing platforms. This way, the generated attack traffic should not even leave the cloud-platform's premises and be mitigated at the source. They focused on the more common attack types, such as SSH brute-force attacks, ICMP floods, DNS reflection attacks, and TCP-SYN floods, and modeled the extracted feature set after these attack types. The final architecture allows real-time detection of attack traffic. This is achieved via a continuously updating learning model that is backed up by a pre-trained model. As for ML techniques used, a variety of different algorithms were evaluated, with Random Forest and SVM achieving the highest accuracy scores. Evaluation and training was conducted using a test network setup in a cloud-environment.

One notable example of combining different ML techniques was performed in [35]. By using a so called ensemble-classifier, the results of a variety of ML techniques are compared and evaluated against each other, often offering a higher accuracy result than by using single classifiers [38]. The goal of the work was to create a network intrusion detection system that can detect DDoS attacks. The most accurate result of the different classifiers they tested proved to be the ensemble ML solution that combined the results of Neural Networks (NN), SVM, K-Nearest Neighbor and DT-C4.5 algorithms. The features that they chose, resulted from an evaluation process where features were grouped by the attack types they were most likely to detect. However, the final detection scheme was only on a binary basis.

Table 3.1: Classification of existing research in DDoS Detection

Work	**Application Field**	**Technique**	**Point-in-Time**	**Dataset**	**Result**
Khupiran et al. [27], 2018	Binary DDoS Attack detection	SVM and DFF	After Attack	DARPA (PCAP)	DFF achieved high accuracy, while SVM delivers faster classification times. Packet-based feature set achieved much higher accuracy than time-window-based.
Hou et al. [28], 2018	DDoS Attack detection. Specific attack types are detected	Random Forest	Real-time feature extraction	Generated Netflow data and ISP-supplied Netflow data	Very high accuracy. Experimented with sampling in order to process terabytes of network data.
He et al. [29], 2017	DDoS Attack detection from source side. Specific attack types are detected	Random Forest and SVM, among others	Real-time detection possible	Cloud-based VM traffic	High accuracy for RD and SVM models. Different feature sets for different attack types.
Li et al. [31], 2010	Binary DDoS attack detection	BP and LVQ Neural Network	After Attack	Collected Traffic data	Very high accuracy for LVQ method.
Suresh et al. [32], 2011	Binary DDoS attack detection	Evaluation of various techniques	After attack	CAIDA data set and collected traffic	Fuzzy C Means and Naive Bayesian returned best accuracy, but all tested techniques provided favourable results.
Das et al. [35], 2019	DDoS Intrusion Detection	Ensemble Machine Learning	After Attack	NSL-KDD data set	Distinction of features based on attack types, but detection result is binary.
Priya et al. [37], 2020	Binary DDoS attack detection	Naive Bayes, K-NN and Random Forest	After Attack	Collected data from generated traffic	High accuracy, despite having a small amount of features extracted. Ability to detect different types of attacks.
Doshi et al. [36], 2018	Binary DDoS attack detection for IoT devices	K-NN, SVM, Decision Tree, RF and NN	Real-time detection	Collected IoT traffic from test setup	High accuracy for all algorithms. Small feature sets due to limited processing power.

3.2 DDoSGrid

DDoSGrid [1] is a DDoS analysis and visualization tool developed at the University of Zurich. Its main goal is to be a research-oriented platform that allows different user groups to upload network capture data in PCAP format and study the analyzed data, using visualizations that are generated during analysis of these files. It is available as a self-hosted service and consists of a miner component that accepts the data sets, runs them through existing feature extractors and data miners and produces case-specific output files. Existing miners provide functionality such as the calculation of the most used ports at the target IP, or a list of the most common source IP countries. These miners are designed to be extensible and easy to include in a forked version of DDoSGrid, so that new use cases can easily be added to the platform. The second component of DDoSGrid is a web-based front end. It is on one hand, the entry point where users can upload and manage their data sets, while on the other hand being a configurable dashboard that lets users visualize their analyzed data sets on a grid. Such a dashboard is visible in Figure 3.2 where a SYN-Flood attack is inspected. Every item in the grid is either a visualization of a certain metric or a representation of the data set, along with general statistics that were calculated during data set analysis. The goal for users is to choose visualizations that are categorized by attack-type they are most likely to identify and guide them to decide what attack vectors were used in the attack. Setups on this dashboard can be saved and restored at a later date, as well as exported as PDF files, so the platform is well suited for applications that require live demonstrations of attack analyses, as well as documentation of an attack in post.

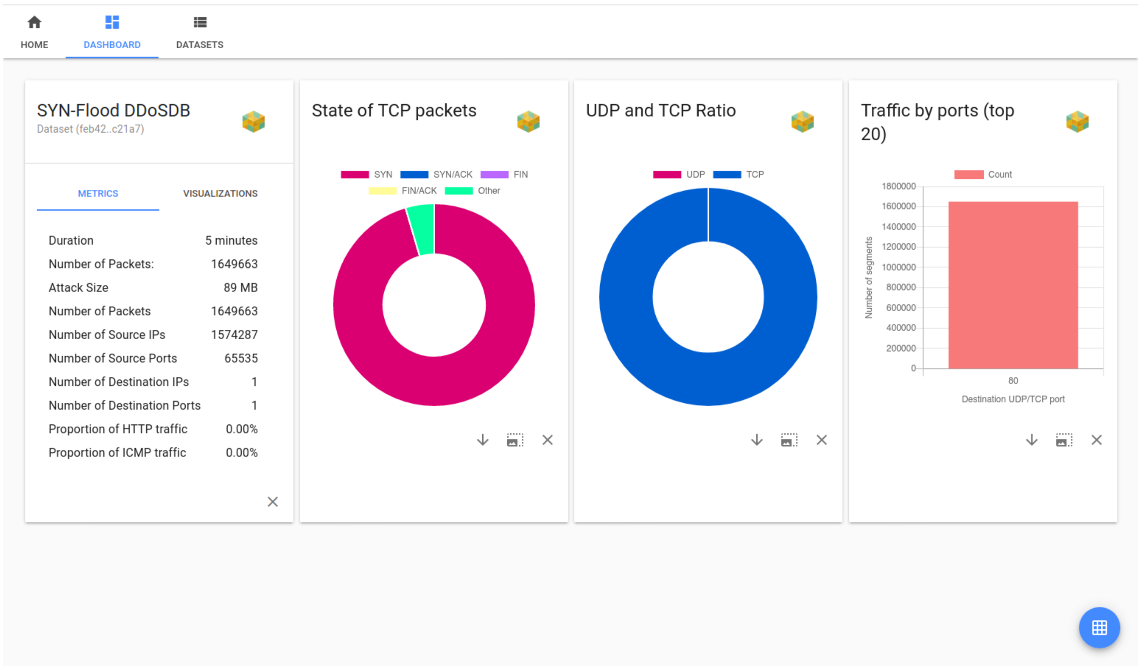
This thesis will extend the DDoSGrid platform with functionality that has will eliminate the need for a user to try out visualizations by hand to find out what attack-types were used and use machine learning approaches to determine the attack vectors. Both the feature extraction component, as well as the visualization dashboard will be used in this extension. Additional feature extractors will be conceptualized and implemented to pre-process the data for the different ML algorithms, while the dashboard will be used to present the findings of the analysis, in the form of fitting visualizations.

3.3 Data Providers

In order to train the chosen machine learning algorithms, sufficient data sets needs to be acquired. Different databases and data sets exist on the internet with the purpose of being studied and analyzed in a research context. Some of these stem from newer projects, while others have been staples in network intrusion related research.

3.3.1 DDoSDB

DDoSDB [13] is a project from the University of Twente, which has the goal of sharing network captures and fingerprints from actual network attacks. Security professionals and



Duration	5 minutes
Number of Packets:	1649663
Attack Size	89 MB
Number of Packets	1649663
Number of Source IPs	1574287
Number of Source Ports	65535
Number of Destination IPs	1
Number of Destination Ports	1
Proportion of HTTP traffic	0.00%
Proportion of ICMP traffic	0.00%

Figure 3.2: A DDoSGrid Dashboard, configured to show the characteristics of a SYN-Flood atttack.

researchers can download these captures to study and analyze past attacks. It consists of a web-frontend, that allows the querying of the datasets by certain parameters in the Lucene querying language. The second component is an ElasticSearch-based database that stores all the data sets and handles the search-operations. The data sets are stored in PCAP format, and contains fingerprints and full data sets of network attacks. The third component is the *ddos_dissector*, which is a CLI tools that generates fingerprints from network trace data.

Concurrently to writing this thesis, the integration of DDoSDB into DDoSGrid has happened [25]. This allows the direct import from DDoSDB data sets into DDoSGrid and makes separate downloading unnecessary. During the development of DDoSGrid, data sets from DDoSDB were frequently used, making these data sets especially suited for use in DDoS attack type recognition, since the DDoSGrid miners already understand the data structure of these data sets.

3.3.2 DARPA Intrusion Detection Evaluation

In 1998, 1998 and 2000, the MIT Lincoln Laboratory in conjunction with DARPA, have conducted a series of simulated network attacks on a test network infrastructure, with the aim of providing future research guidelines for detecting network intrusions and DoS attacks. The raw network traffic captures from these tests are publicly available and well documented, in terms of which attack types were launched at what point in time and also offer regular traffic captures of the system, to provide a baseline of what a healthy system should look like, from a network's perspective [42].

3.3.3 DDoS Logsim

Developed at the University of Zurich, DDoS Log Sim [14] is a tool that allows the generation of DDoS attack data in PCAP format. Based on a configuration file that contains the attack parameters, the tool generates large amounts of data that contain SYN Flood of DNS Amplification attacks. These log files were also used during the development of DDoSGrid, so these generated attacks might also be suitable to extract features from to be analyzed using machine learning algorithms. The tool was designed to be extendable, so additional generators could be written for it that suit the needs for the context of this thesis.

Chapter 4

Approach & Requirements

This chapter documents the process of selection for both, the features that will be extracted from uploaded data sets, as well as the machine learning algorithms that will be used in the prototype to-be-built. As mentioned in the literature review done in 3.1, various strategies were uses for achieving automatic DDoS attack detection, so we chose the procedure most fitting for our purpose which is a post-analysis, done in DDoSGrid, as an extension module.

4.1 Feature Selection

Since DDoSGrid has the goal of aiding in identifying different types of DDoS attacks, using its specific visualizations, the aim of a ML-supported detection extension should also to be an automatic attack-type detection, as opposed to just binary detection of an attack. This requires that the extracted feature set from an uploaded data set contains many features that indicate anomalies only for one or a few attack types. An example would be the percentage of TCP-SYN flags set in all TCP packets; A high percentage clearly indicates that a SYN Flood is occurring in the data set, whereas said feature does not necessarily help in detecting other attacks, such as UDP-Floods. This can result in a potentially larger set of features, depending on the number of attack types that should be identified. However, due to the environment the ML-module will be used (web-application located on a server), the resulting data amounts are of no concern, both from a memory and performance standpoint, as the web application is already able to handle large amounts of data in a short period of time, and the already existing visualization feature sets can be of similar size, depending on the visualization. Comparing the proposed feature set to the existing research, it falls in the middle, between small and larger feature-sets. Still, there are some features that can indicate the occurrence of different attack types, such as the average packet size and number of unique packet lengths. In regular traffic, packet sizes can vary quite a bit, resulting in these features being different for every time they are extracted, while during an attack, these features are recurring many times, indicating that *something* out of the ordinary is occurring.

Feature Name	**Attack-Types**
Number of Packets	General
Average Packet Size	SYN-Flood, Ping of Death
Number of Bytes	General
Number of Unique Packet Lengths	SYN-Flood
Number of Unique Source IPs	SYN-Flood, ICMP Flood
Number of Unique Destination IPs	IP-Sweep, UDP Flood
Number of Unique Source Ports	TCP-Flood
Number of Unique Destination Ports	UDP-Flood, SYN-Flood
Average Inter Packet Interval	General
Percentage of TCP Packets	SYN-Flood
Percentage of UDP Packets	UDP-Flood
Percentage of ICMP Packets	ICMP-Flood
Percentage of Other Packets	General
Percentage of TCP-SYN Flags Set	SYN-Flood
Percentage of TCP-FIN Flags Set	SYN-Flood
Percentage of TCP-ACK Flags Set	SYN-Flood
Percentage of ICMP Echo Replies	ICMP-Flood
Percentage of ICMP Destination Unreachable Messages	UDP-Flood
Average Inter-Packet Arrival Time	General
Attack Type	

Table 4.1: Overview of preliminary relevant extracted features on time-window basis from PCAP files.

We decided to extract a feature set that is time-window based, as opposed to a packet-based set, due to the much smaller resulting output file and the accuracy of window-based feature sets in the existing literature. Putting time-window and packet-based feature sets into perspective, a time-window feature set always has the same size per window for all data-sets, regardless of initial size. It only grows in size with each consecutive attack-window happening in the original packet-capture. This has the result of even very severe attacks with many packets per time-window, being reduced to a easily managed data set, that only scales linearly with time. On the other hand, a packet-based data set grows with each additional packet in the data-set. This can result in the extracted feature-set being larger in size than the original binary packet capture, depending on encoding and file format used, since the resulting feature set is just a decoded subset of the packet capture.

As a starting point, the first three attack-types to-be-detected, were decided to be SYN-Flood attacks, UDP-Flood attacks and ICMP-Flood attacks. Reasons for choosing these attack types include the relative simplicity of detection based on simple features, the availability of well-documented data sets containing these attack types, and the frequency of occurrence of these attacks in real-life. As different attacks are identifiable on different layers of the incoming packet, some of the extracted features, such as *Average Inter Packet Arrival Time* extract information on the frame-layer of the packet, whereas other features such as *Percentage of ICMP Destination Unreachable*, extract information on transport-layer level (ICMP-Type of a packet). In Table 4.1, an overview of all the features that

Feature	**Value SYN-Flood**	**Value ICMP-Flood**	**Value UDP-Flood**
num_packets	20409	1049	46609
avg_packet_size	54	1064	556
num_bytes	1102352	1116222	25932753
num_unique_packet_lengths	4	2	315
num_unique_source_ips	19905	783	1133
num_unique_dest_ips	1	2	1
num_unique_source_ports	17169	3	3
num_unique_dest_ports	1	3	2
avg_inter_packet_interval	27.4	951.4	20.5
tcp_perc	100	0.1	0
udp_perc	0	0	83.3
icmp_perc	0	99.8	16.7
other_perc	0	0	0
perc_tcp_syn	97.5	0.09	0
perc_tcp_fin	0	0	0
perc_tcp_ack	2.0	0.09	0
perc_icmp_echo_reply	0	99.8	0
perc_icmp_dest_unreachable	0	0	16.6
attack_type	1	2	3

Table 4.2: Extracted Features of one time-frame from different attacks.

are extracted is presented, along with the attack type, they are most likely to identify.

Important to note is the last feature, *Attack Type*. This feature is represented by an integer that defines the attack-state of the time-window. For example, a zero in this field means that there is no attack happening during the time-window, while a one means that a SYN-Flood is occurring, and so on. In Chapter 5, the setting of this variable is explained further in both manual and automatic classification modes of the final solution. In Table 4.2, a sample output of different time-windows are presented. All feature a different attack occurring at one point in time. In the second column, the fact that a SYN-Flood attack is occurring is identifiable by looking at the *perc_tcp_syn* values and comparing the *num_uniqu_packet_lengths* with the *num_packets* values. Only four different packet lengths were recorded in a total of over twenty thousand packets arriving. This leads to the assumption of these packets being sent by a botnet, which all send more or less the identical TCP-SYN enabled packets. Similarly, when looking at the ICMP-Flood window, a very high percentage of ICMP Echo Reply messages were sent in the network, leading to the assumption that and ICMP-Flooding attack was launched in the network. Lastly, looking at the UDP-Flood column, the packets are distributed with UDP and ICMP packets and the percentage of ICMP Destination Unreachable messages are unusually high; The usual results of a UDP attack. These time-windows were taken and extracted from different data sets from different environments, such as the DARPA attack tests from 1998 and DDoSDB. It is important the the data used for training the model comes from different scenarios experiencing the same attack circumstances, that way the model will be able to accurately classify attack types even under very different network and infrastructure conditions.

4.2 ML Algorithm Selection

In the existing literature, various machine learning classifier algorithms were used to automatically classify attacks and/or attack types. The following section describes the procedure that was undertaken to select algorithms to be used for the DDoSGrid attack detection module. Summarizing the literature, both simple and complex-to-use algorithms were used and both types achieved satisfying accuracies in their respective evaluations. This led us to select the following three algorithms for a preliminary test run, to find out whether they would be suited for a web-based application, where computing is handled server-side: Random Forest Classification, k-Nearest Neighbor Classification and Support Vector Machines.

4.2.1 Random Forest Classification

This algorithm can be thought of as an evolution of Decision Trees (DTs). DTs can be used for both classification and regression and, even though relying on relatively simple mechanics, can be used in many different environments. The algorithms revolve around a binary tree that classifies records according to their properties on a true or false basis, such as *Is value X larger than 10?*, which splits the data set into subsets with each further split. This classification starts from the root of the tree until either there are no further splits to be done, a leaf node contains less that minimum number of samples, or when the predefined maximum depth of the tree is reached. The final result is a tree-structure that represents the leaf nodes as a series of binary decisions, resulting in the classified value (In our case, the attack-type). Another reason that make them easy to use is the fact that they do not require normalization of the numerical values, although depending on the implementation, converting reoccurring strings to one-hot values or integers is often recommended [15].

Stemming from Decision Trees are Random Forest Trees and the classifiers that make use of them. So-called Ensemble Classifiers, incorporate many classifiers into one and are often better performing than their standalone origins. Random Forest Classifiers build many single Decision Trees independent from each other and build a final tree based off of the tree nodes by taking the statistical mode and mean of the independent trees and combining them. Since many of the independent trees would be nearly identical for the same data set, the trees will all be built using a random subset from the training data. This also mitigates over-fitting, a common occurrence in regular decision trees by incorporating many different results of the independent trees. Due to the independent nature of the individual trees, parallel processing can be used to efficiently build the model, leading to performance increases. An example of this classification can be seen in 4.1, where four trees are created and the final result will be decided with a majority voting. This algorithm was chosen for a preliminary test-run, due to the high accuracies achieved in the existing literature and the relative ease of configuration and use.

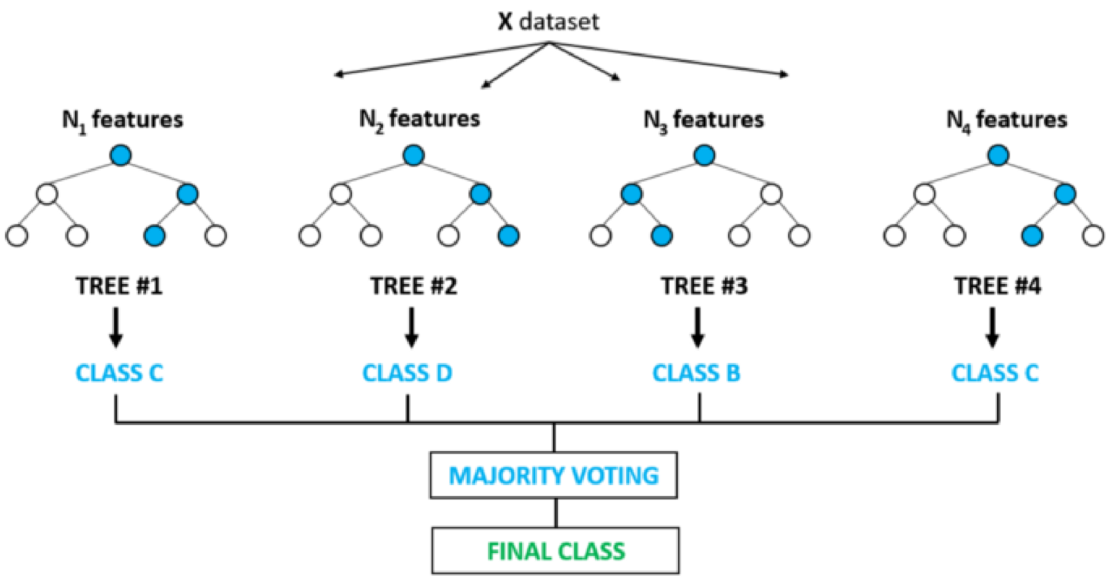


Figure 4.1: Example of a Random Forest Classification. Different Decision Trees are created and the final result will be voted on. [10]

4.2.2 K-Nearest Neighbor Classification

A so-called lazy-learning algorithm. As opposed to other algorithms, k-NN does not put the main computational effort in during model building, but during actual classification. Generalization of data points is not done in advance when training, but in-context when new data points are being classified. Thus, the model simply contains all the feature vectors and their complementary labels. During classification, the new record is plotted against the available feature vectors and classified according to the k-Nearest Neighbors' label, as can be seen in Figure 4.2 This method makes it very fast during training, due to the low processing time required. However classification has to make up for these circumstances and may take up longer times compared to other algorithms, such as Random Forest Classification. Another downside to this method includes large file-sizes for the lazily-trained modules, due to the low effort put into building them. Input and output file sizes are thus comparable in size. Despite these downsides, this algorithms was chosen for a preliminary test run due to its ease of use and high accuracy achieved in the existing literature.

4.2.3 Support Vector Machine

Taking its most plain form into view, Support Vector Machines or SVMs are essentially a linear classifier. This means that when plotting the data points in an X-dimensional room, a hyper plane tries to separate the data points into two distinct categories. The aim of the hyper plane is to find the position, where the maximum-margin of distance between the plane and the nearest data point on both sides of the plane is achieved. When

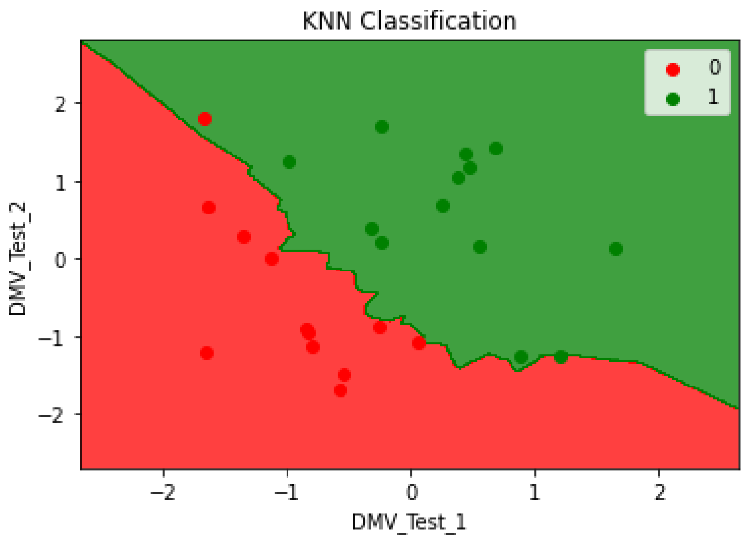


Figure 4.2: Example of a k-Nearest Neighbor Classification. Two features are plotted against each other, with the dependent variable being a boolean. Depending on which part of the separation, the new record falls, the classification will be made. [11]

predicting a new value, the feature-vector that should be predicted is plotted against this graph, falling on either side of the chosen hyper plane, predicting the category the fall into. Should the data not be cleanly separable, the data points that fall between the hyper plane and the maximum-margin are penalized against, the so-called Hinge Loss [15].

SVMs allow using different kernel tricks, that shape the form of the boundary lines according to the tricks used. Per default, many implementations use a linear kernel trick transformation, resulting in a straight line across the data points. However polynomial and radial transformations are also popular choices, as they allow the boundary lines to curve around data points, potentially resulting in a higher accuracy of results. Very well suited for high-dimensional data sets, such as the DDoSGrid use case, delivering very high accuracies. However, due to the highly complex procedure and resulting model, training can take up large amounts of time and get exponentially more difficult to the power of two, with each new record added to the data set. This algorithm was chosen for a preliminary test run, to have a complex algorithm running against the more simpler ones mentioned in the paragraphs above. As well as test the boundaries of the DDoSGrid use case for this algorithm, since many features and potentially records are present in the extracted feature set.

4.2.4 Evaluation of Algorithms

The three aforementioned algorithms were put through an evaluation procedure, to test out whether they would be well suited to be part of a web application. Factors that influence this decision include, the performance footprint on the host machine, disk requirements, as well as processing time to both train the model and predict future values, in this case DDoS attack types. With DDoSGrid potentially running on a lower-spec server, this test required that the algorithms should perform on a lower footprint when training the model and predicting values but still achieving satisfying accuracies during evaluation.

All three algorithms were tested using their implementation from the SciKit Learn Python package. Accordingly, Python was used for the code-base, since it allows quick implementation and test cycles, in addition to a full fledged feature set for Machine Learning applications. SkiKit Learn also allows creating confusion matrices and cross-validation metrics, that come in handy during evaluation of the trained model at the end of the procedure. The Python scripts were run inside JupyterLab notebooks, which allow sandboxed execution of the scripts, as well as easy handling of input and output files. These input and output files are both in CSV format for easy handling in conjunction with tools that produce and read these files, as can be seen in the implementation part of this thesis.

The evaluation procedure for all three algorithm was identical; A manually classified data set consisting of DDoS attack traffic windows and their features presented in the previous section, was read by the python script. After converting the CSV file into Pandas dataframes that allow easy handling of the data, the resulting dataframes were divided into a training and test set, with the training set consisting of 75% of randomly picked records. The data set was also split into data frames containing the features and a vector containing the dependant variable, which in our case was the attack type in integer format. The attack types that were part of the manually classified data set were *No Attack*, *SYN-Flood*, *ICMP-Flood* and *UDP-Flood*, represented by the integers 0, 1, 2 and 3 respectively. The DDoS attack traffic data sets were picked from both DDoSDB and DARPA data sets to provide different network environments but experiencing the same attack types. The features from these attack logs were extracted using a new miner as part of DDoSGrid that extracts all the features presented in the previous section and converts them to the aforementioned CSV file. Since the DDoSGrid miner component can be run as a standalone application and allows supplying arguments for the individual miners, the new miner was developed with the ability to manually classify the data sets into the correct attack types. More information about the miner development can be found in Section 5.3. These individually mined attack logs were then concatenated, resulting in one large CSV file, containing around 6000 records of all 4 different attack states.

After instancing the classifier and, if required, tweaking the options to fit our use case, the classifiers were fit with the training data to build the model that will be used for predicting new values. With the model built, the test data set will be supplied to the trained classifier, resulting in a vector containing the predicted attack types for all the records in the test data frame. Finally, the confusion matrix is calculated comparing the truth set with its predicted counterpart, presenting an accuracy score for the classifier.

This process was repeated for all algorithms, with every run having a different model at the end, due to the fact that the training and test split being done in a randomized fashion. We also ran tests with features we deemed to be omittable, such as the number of packets and the percentage of other packet types. Regarding completion times, both Random Forest and K-Nearest Neighbor classifiers finished building the model, predicting the values and evaluating the predicted values in the one to two second range, with computing performed on an average desktop computer. The accuracy for both algorithms resulted in the range between 95% and 100%, depending on configuration and algorithm, with RF resulting higher accuracies and lower completion times, compared to k-NN. The SVM algorithm had difficulty building the model, due to the complexity given by the number of features and the amount of records in the data set. This led to the SVM algorithm never finishing building the model, even after 6 hours had passed. This fact led to the decision of not including SVM-based classification in the end solution due to the infeasability of it running on a server machine. Regarding disk usage for the models, the two remaining algorithm both used very low disk space, even k-NN which is known for having large model-sizes due to it's lazy-learning architecture. The input file we supplied was around 470kB in size for around 6000 records. This file size could even be further shrunk, if we applied a rounding function to the features that are represented as floating point numbers. At the time of testing, the miner calculated floating point numbers to 17 decimals.

The result of this preliminary evaluation was the decision of including both Random Forest and k-Nearest Neighbor algorithms in the DDoSGrid module with the possibility for the user to choose a preferred algorithm during either upload or analysis time for each data set. For this test, the model was rebuilt for every test run. In the DDoSGrid module, the possibility to persist the model across file uploads is an option we kept in mind, but decided against in the prototype implementation, since the final evaluation in Section 6.3, proved that even for larger models, the processing time is still in a feasible range. Processing time on the deployed version of the classifier will also be shortened due to the fact that less steps will be performed compared to the evaluation. Splitting the data set into training and test set will not be required, as well as creating the confusion matrices at the end of processing. Due to the relative ease of swapping algorithms, the possibility to add further algorithms to the DDoSGrid module is there, but further evaluation efforts should be made before including potentially inefficient algorithms in the program.

Since Jupyter Notebooks consist of Python code for the most part, the relevant code sections can easily be extracted and placed inside their own Python files, making the machine learning part of the proposed module modular and separated from the main DDoSGrid application.

4.3 Extensions to the DDoSGrid Architecture

This sections documents the architectural changes necessary in DDoSGrid, in order to incorporate the desired ML-supported attack type classification functionality. A short overview of the existing architecture is given, followed by the definition and description of the additional modules required and how they interact with each other and the existing modules. Furthermore, needed changes to existing components are described.

4.3.1 Existing Architecture

As can be seen in Figure 4.3, the architecture consists of a User Layer and a Data Layer. As the name implies, the user layer is the web-facing interface, where all actions regarding the upload, visualization and dashboard arrangement can be carried out. The Data Layer is what makes data processing and persistence possible. The arrows in the diagram indicate relationships and data flows between the individual components of the layers.

From a user's perspective, the web-based interface of the User Layer, is the only point of access to the application. From here, a user can upload his or her data sets in PCAP or TCPDUMP format to the application and after processing has finished, analyze the output of the feature extractors by opening and arranging visualizations on its dashboard. The visualization module is responsible for that and consists of smaller modules that represent each individual type of visualization. All data transfers occur through a single RESTful API with the Data Layer. The requested data gets fed into the Visualization Module which then represents these data points on the individual Visualizations.

The Data Layer performs all data processing, management and storage-related functions for DDoSGrid. Through the RESTful API, network capture files are uploaded and first handled by the File Upload Module. Here, Metadata and the Raw Attack Traces are extracted and stored into their own databases. After the upload has finished, the Packet Decoder module retrieves the Raw Attack Trace and starts decoding the trace on a packet-stream basis. This method is used to prevent memory overflow issues, due to the fact that PCAP files can be very large in size. The individual packets are then parsed by type by the Protocol Parser. For all desired packet types, the Observer waits for a specific packet type and if present, calls the Feature Extractors that await and expect certain packet types to extract the desired properties. For example, the TCP-SYN miner, looks at each packet's TCP-Flag states and creates a statistic that represents the percentual representation of each state. This is helpful for identifying SYN-Flood attacks, since large percentage of SYN-Flags being set is a good indicator of a SYN-Flood attack occurring. The Feature extractors store their outputs in individual files that only contain the mined features and how they should be visualized, separately from the raw attack traces, though connected through IDs.

4.3.2 Additional Modules and Their Relationships

In Figure 4.3 the blue elements and arrows in the Data Layer represent the additional modules and relationships required for ML-supported attack classification. The addition of such functionality is supposed to be a modular extension to the existing DDoSGrid codebase, so that it can run with, and without the extension. As before, the user only interacts with the data layer through the RESTful API, so additional endpoints in the REST API are required to individually run the ML functionality. A new central node called the Data Handler is required, as well as a new storage node for ML Model Data and the ML Algorithm Modules. Additionally, one ore more new feature extractors and corresponding visualizations are required to extract and visualize the necessary data from PCAP files.

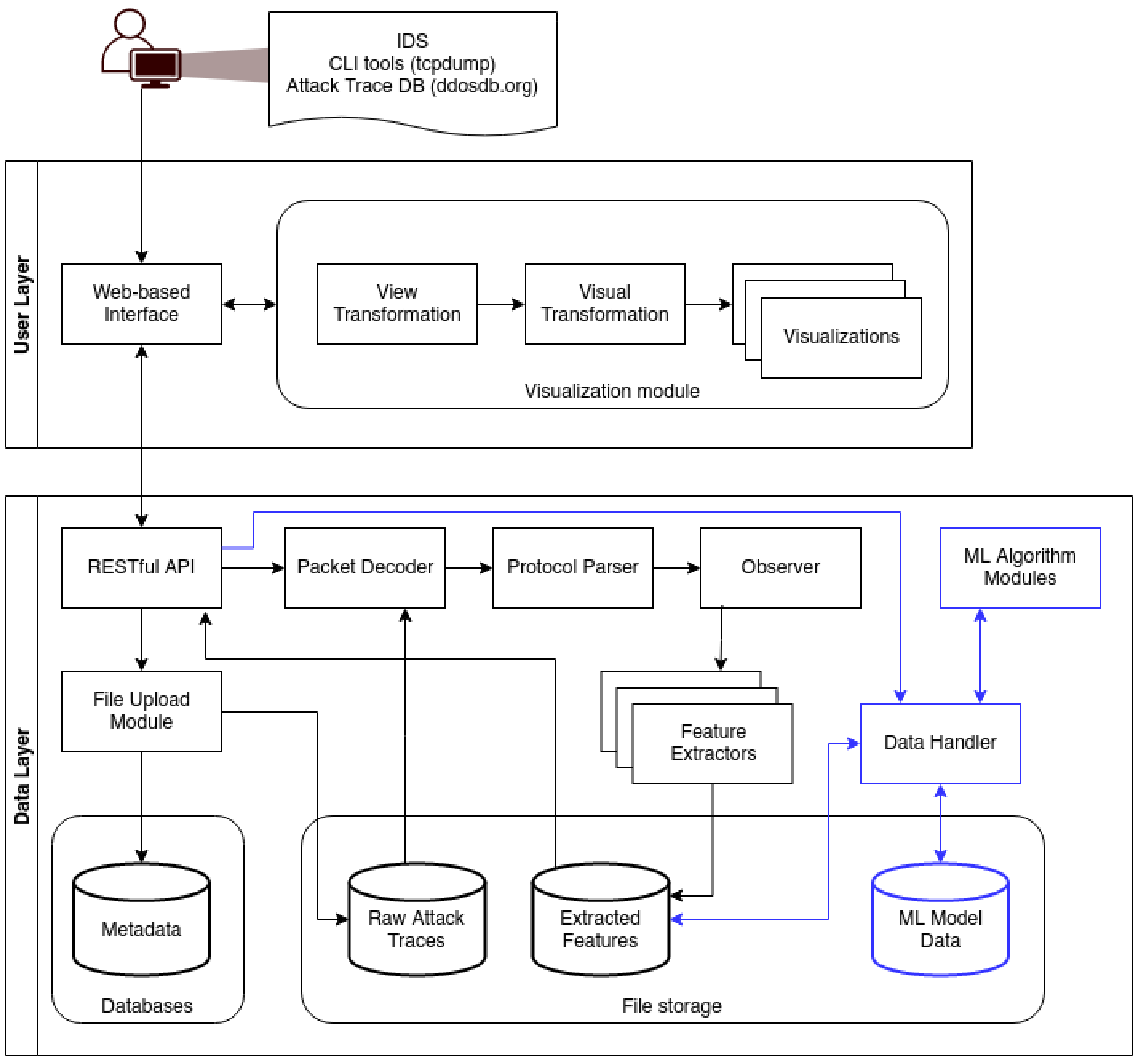


Figure 4.3: Existing Architecture, with the new modules drawn in blue.

Feature Extractors and Visualizations

The new feature extractors should have the ability to extract features over time-windows and not just over individual packets. This requires that the observer should be able to call the same miner, with different packet-types and data, in order to to fully inspect multiple packets inside a time-window of predefined length. The output of this miner will be stored in the same storage as the other extracted features, but will likely take up more space because of the number of features extracted. Since each feature extractor requires one or more visualizations to present the data to the user, new visualizations need to be created to present the new classification-related data in an understandable manner.

Data Handler

The Data Handler is responsible for retrieving, converting and returning all data, related to Machine Learning functionality. Since ML applications frequently use file formats different from web-applications, these need to be converted for ML-use and the result they return back for regular web-use. It gets called from the RESTful API to start ML-routines and handles all processes related to that. Since new feature miners are required to extract the proposed features in Table 4.2, the Data Handler mostly works with the results from those miners. This means it requires access to the Extracted Features Database.

ML Model Data

Since ML-Applications require a model of true data to perform their classifications, the DDoSGrid has to store this model data in its Data Layer. For every automatic classification, this model data is retrieved by the Data Handler and passed to the desired ML-Algorithm Module that returns a classified data set. However, this model data needs to be created by true data sets, in order to provide accurate predictions on future data sets. After a user upload a data set, he or she wishes to manually classify, since the user knows what kind of PCAP file it is, the ability to add this true data set to the model data is a requirement. Again this is handled by the Data Handler, since format conversions and file merging is required.

ML Algorithm Modules

As a starting point, following the preliminary evaluation of algorithms done in Section 4.2.4, the application should allow the selection of different ML-Algorithms. These individual algorithm modules should present a modular interface for easy interchangeability of algorithms. The Data Handler retrieves the model data, along with the data set to classify to the algorithm module. The automatic classification returns a vector of attack types, which needs to be merged by the Data Handler with the original data set. A component that lets a user see statistics about the model, such as distribution of model attack types and real-time evaluation results should also be provided to the user.

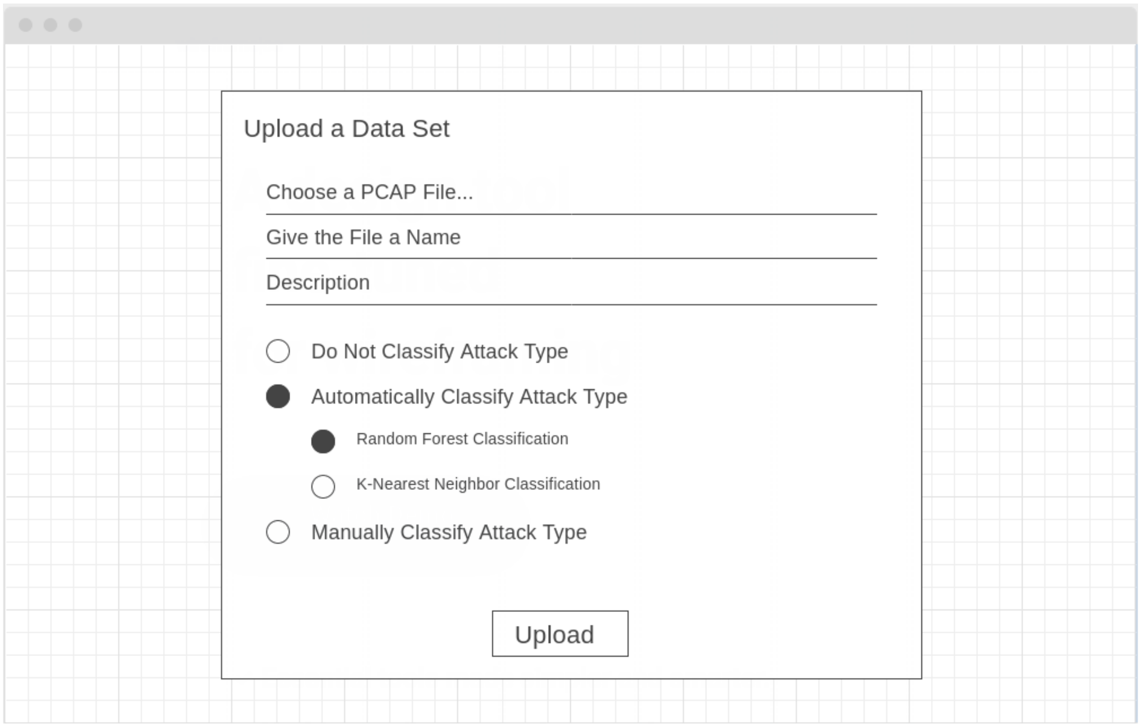

Upload a Data Set
Choose a PCAP File...
Give the File a Name
Description
Do Not Classify Attack Type
Automatically Classify Attack Type
Random Forest Classification
K-Nearest Neighbor Classification
Manually Classify Attack Type
Upload

Figure 4.4: Mockup for selecting the machine learning algorithm during upload.

4.4 Required Changes to the Implementation

The modules and relationships described in the previous sections, will be translated into real requirements in this section. First we begin with the changes necessary in the front end part of DDoSGrid. This includes mockups that illustrate the changes required to make ML-supported attack classification possible. Second, we document the required changes to the API, accepting calls from the front end. Since new data types are transferred between front and back end, these flows need to be defined. Third, the new miners and their integration with the existing parser-observer mechanisms need to be defined.

4.4.1 User Layer

In order to satisfy requirements UL-2, 3, 4 and 5, as seen in Table 4.3, a mockup was created to showcase the different customization possibilities during the upload process. These features will be integrated into the existing File Upload Form, to let a user define what type of classification and algorithm will be used on the file that is being uploaded. In Figure 4.4, the mockup for the algorithm selection is presented. Using a radio button, the user can select between all the available ML algorithms on the platform. Additionally, to satisfy requirement UL-1, a radio button also exists to not perform any ML-classification at all. In Figure 4.5, the process to manually classify a PCAP file is shown. A user can add or remove sections containing start and end times, as well as the attack type for each section. After these parameters have been set, the upload process continues as usual via the upload button.

ID	Description
UL-1	A user needs to be able to use DDoSGrid without any ML-supported classification enabled.
UL-2	A user needs to be able to manually classify a data set in order to create a truly correct dataset.
UL-3	Manually classified data sets can be classified on a file-basis into one attack type
UL-4	Manually classified data sets can be classified on a section-basis that allows setting multiple attack types for one file.
UL-5	A user can choose which machine learning algorithm will be applied to classify the file that was uploaded.
UL-6	Uploaded data sets should indicate the status of classification in the data set overview.
UL-7	Uploaded data sets should indicate the classification type in the data set overview.
UL-8	Automatic classification should be started after a file has been successfully analyzed by the miner.
UL-9	Data sets that have undergone classification, either manual or automatic can be added to and removed from the Model Data Set in the data set overview.
UL-10	Automatically classified data sets can be re-classified, for example to see possible changes to classification after the model has been extended.
UL-11	It should show in the data set overview, whether the classified data set has been added to the Model Data Set.
UL-12	There should be a visualization that shows the attack type over time to inspect the classified data sets.
UL-13	There should be a visualization that shows the total distribution of attack types to inspect the classified data sets.
UL-14	Data sets that make use of the classification functionality should have the new visualizations be displayed in a new category called *Attack Type Classification*.
UL-15	A new tab on the application should exist to inform the user about statistics and metrics about the machine learning model.
UL-16	A visualization that shows the distribution of attack types in the model should exist on the machine learning tab.
UL-17	The entire model should be removeable from the machine learning tab.

Table 4.3: Requirements for the User-Layer Part of DDoSGrid

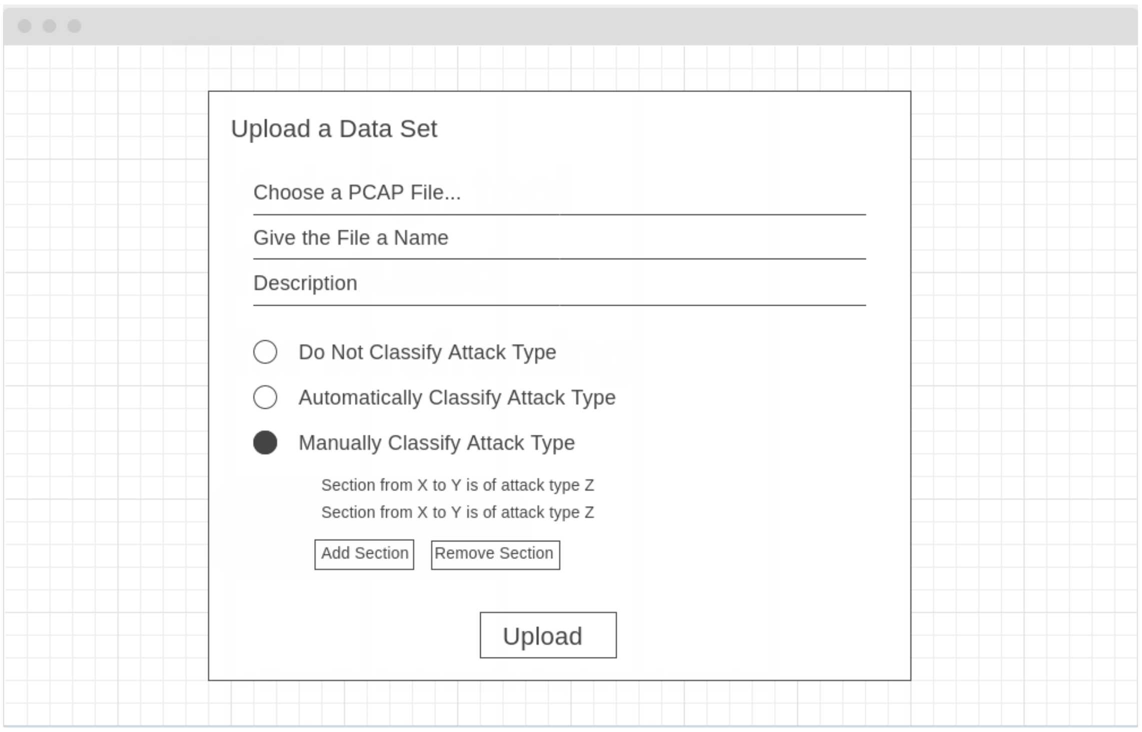

Upload a Data Set
Choose a PCAP File...
Give the File a Name
Description
Do Not Classify Attack Type
Automatically Classify Attack Type
Manually Classify Attack Type
Section from X to Y is of attack type Z
Section from X to Y is of attack type Z
Add Section
Remove Section
Upload

Figure 4.5: Mockup for manually classifying the attack type of a PCAP file.

Requirements UL-6, 7 and 11 can be easily integrated into the existing Data Set List Item, by adding additional rows, indicating classification type, status and whether the data set was added to the model.

4.4.2 Data Layer

Taking into account the existing architecture and the new proposed modules, changes to the existing components need to be done, in order to fit the new modules. The feature extraction should be able to output multiple files, meaning multiple visualizations for one miner should be possible, a change from the current implementation that only allowed outputting a single visualization per miner. Additionally the miner should be able to inspect multiple packets over a period of time, storing data temporarily from the current time window. This reduction of packet data into time windows allows for the output files to only increase linearly, regardless of the attack intensity and length. In order to allow the manual classification of PCAP files, the system should allow passing attack-type configuration data through the miner pipeline. This data is stored along the metadata in its respective database for easy retrieval. This configuration data should allow setting simple values to classify the entire data set, as well a section-based data structure to classify a data set in detail. In addition to the current JSON output files that the miners produce, the miner should also export the machine-learning related data into CSV format, to be used efficiently by the machine learning pipeline. The JSON outputs do not contain all the mined features, as they should only contain the necessary data to draw their respective visualizations.

For the User Layer and Data Layer to communicate correctly, the right connections need to be established that provide the new functionality. This includes changes to existing routes, as well as the addition of entirely new routes. Especially all machine learning related functionality should be separated from the existing analysis procedure, to provide a modular separation of functionality in the application.

Chapter 5

Design & Implementation

The following chapter details the implementation-related processes in a low-level environment, as opposed to the high-level and conceptual approach detailed in the last chapter. Originating from the modules in the new architecture, the actual software components and their relations are documented here, starting with the creation and integration of machine learning routines inside the existing web-application. Then, changes to the miner component are shown, that are required to process the new ML-related data and make it compatible with the Python-based ML-routines. As the new functionality should all be usable from the web-facing front end, the API also requires additional components. Lastly the components in the front-end require updates to make use of the new functionality.

5.1 Data Flow and Component Relationships

In Figure 5.1, the DDoSGrid components can be seen, as well as the relations they have with the produced files as inputs and outputs. The point-of-entry for the file processing is the Uploader module (1), that takes the original PCAP file and either attack-type information or classification from the web-frontend and stores this data in their respective databases. Whether attack-information of classification-information is stored, depends on whether manual classification of the PCAP file, or automatic classification should be done. The PCAP Analyzer (2) takes the uploaded PCAP file and, if available, attack-type information and runs the feature extraction pipeline, where all active miners are executed. In this example, we focus on the ML Feature Extractor, because its file output is different from the existing miners. The attack-type information gets added to every mined window, according to the information given during upload. The full mined feature set gets stored in a CSV file, while slimmed-down versions of the features get stored in multiple, visualization-compatible JSON files. If the user chose to automatically classify the data set, the ML Classifier module (3) uses the existing ML Model Data and the newly mined ML Features and feeds them through the ML pipeline, creating the attack-type variables for the new data set, based on the ML Model Data. These attack-types get updated in the ML Features file, as well as in the JSON Visualization Data, to keep the visualizations accurate and up-to-date. The Model Handler (4) performs addition and

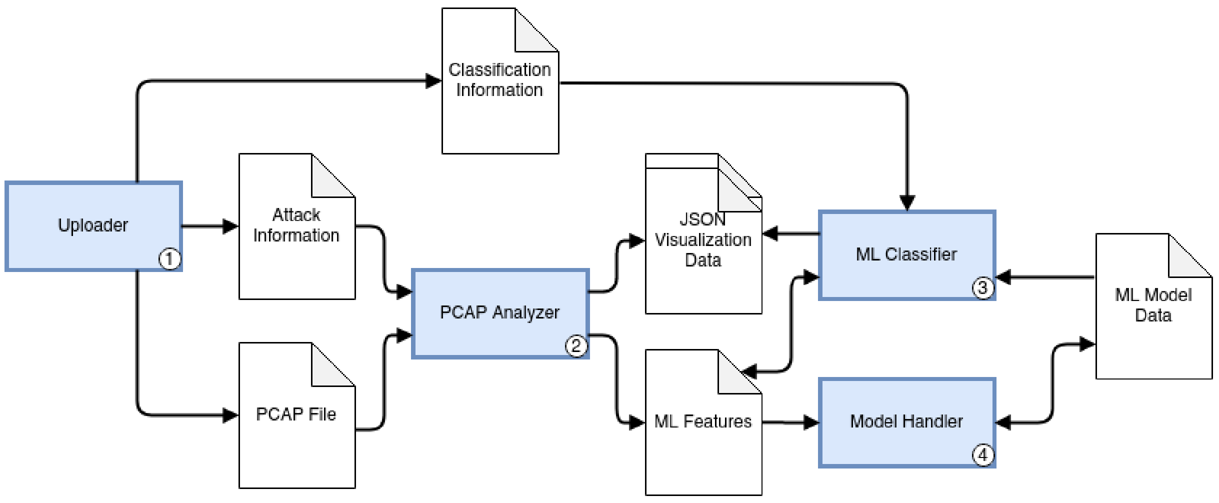


Figure 5.1: Data flows and relationships between files and components.

subtraction operations on the ML Model Data, and uses the ML Features when new data should be added to the model.

5.2 Machine Learning Components

This section details the structure and functionality of the machine learning components in a detailed manner. A set of algorithm components have been created, as well the infrastructure to create a model data set, which can be enriched with additional data sets, as well as have specific data sets removed.

5.2.1 Algorithms

As documented in 4.2, the classifiers consist of Python files that use the SciKit learn library's implementations of the respecive ML algorithms. It provides well-documented and feature-rich implementations of the most commonly used machine learning algorithms. The ease-of-use of the library and the sensible default options, make solving a machine learning supported task in SciKit Learn an attractive environment for beginners and experts alike. Due to the similar setup of both algorithms that we found suitable for inclusion in DDoSGrid, (Random Forest and k-NN) the procedures for all algorithms look very alike, making the integration of another algorithm a trivial task. The feasibility to have certain algorithms part of DDoSGrid is thus mainly the product of it's performance in both processing requirements and time constraints.

To illustrate an algorithm, as it's used inside DDoSGrid, we chose to present the Random Forest classifier module as code listing 5.1. Three libraries are imported right at the beginning of the program; *sys*, which handles the arguments for the program's input and output file paths, *numpy*, which is required to handle operations inside *sklearn* (SkiKit-Learn) and lastly *pandas*, which allows complex data structure operations, such as importing CSV files as DataFrames. On line 13, an additional library is loaded, namely the actual classifier, which is in this case a RF classifier from the *sklearn* library. The reason why this import is not grouped with the others, is to have an outer program shell that can be used interchangeably with different *sklearn* modules, or even different ML libraries. Inside the main function, lines 9 to 15 are used to import the model data in CSV format and convert it to more manageable DataFrame structures. The columns of the file which are the machine learning relevant values are extracted and split into the features and the dependent variable. In our case, the dependent variable is the *is_attack* column of the model data. After importing the classifier class, a new Random Forest classifier is instantiated and is given the features and the corresponding dependent variables in line 16 to create a trained model.

```
 1 #!/usr/bin/env python3
 2 # coding: utf-8
 3 
 4 import sys
 5 import numpy as np
 6 import pandas as pd
 7 
 8 def main():
 9     dataset = pd.read_csv(sys.argv[1])
10     X = dataset.iloc[:, 2:-2].values
11     y = dataset.iloc[:, -2].values
12 
13     from sklearn.ensemble import RandomForestClassifier
14     classifier = RandomForestClassifier(n_estimators = 10,
15                                         criterion = 'entropy')
16     classifier.fit(X, y)
17 
18     toPredict = pd.read_csv(sys.argv[2])
19     toPredictX = toPredict.iloc[:, 2:-1].values
20 
21     toPredicty = classifier.predict(toPredictX)
22 
23     for value in toPredicty:
24         print(value)
25 
26     toPredict['is_attack'] = toPredicty
27     pd.DataFrame(toPredict).to_csv(sys.argv[2], index=False)
28 
29 main()
```

Listing 5.1: Minimal Procedure to import a data set, the training data and apply Random Forest Classification on it.

In lines 18 and 19, the data set that should be classified is imported and has its features extracted. The difference to the import of the model-data is that the dependent variable is produced by the classifier that has been trained. This happens in line 21, where the features are fed into the classifier, which then returns a vector of dependent variables. In our case this would be the *is_attack* variable in integer format. The rest of the program handles the data export. First it is printed, so that the buffer can be read from the API part of the application which converts the resulting vector into JSON format which is used for visualizations. Lines 26 and 27 add the new vector to the existing data-set and export it as a CSV file, which can then be used further by the machine learning part of the application.

Configuration of Algorithms

The SciKit classifiers that we used, allowed for many customization options and parameter tweaks. For the Random Forest Classifier, we used 10 estimators, meaning 10 sets of random decision trees are created and compared to each other. We could have chosen to use more than 10 estimators, however due to the high accuracy and scores, that configuration achieved and the fact that more estimators means more time spent classifying records, it was deemed not necessary.

5.2.2 Model

In order for the classification to occur accurately, a set of various true data sets is required. This true set is compiled and used as the base data for the machine learning algorithms to build their models that will predict future values. In order for DDoSGrid to provide the functionality to maintain this model, the infrastructure for this maintenance needs to be created. When classifying a data set manually, i.e. tell the system which parts of the PCAP file correspond to what attack type occurring, the application stores these manually classified data sets as single files. A user then has the option to add these separate files to a central model, where a broad variety of both network scenarios and attack types should be represented, in order to provide a balanced and accurate prediction result. Adding a data set performs a simple concatenation operation between the two files (the data set and the existing model data). After adding a data set to the model data, a user should also be able to remove it, since an opinion about the truthfulness of an analysis might change or an error has slipped in during the configuration in the upload form. For this reason, all data sets added to the model are extended with the internal database id from DDoSGrid, to make removing entire data sets a simple operation of removing records by their dataset id. The question arose, whether the model should be persisted, or not. We decided against persistence, since the evaluation in 6.2 has shown, that even with larger model sizes, the model building procedure has negligible impact on overall performance. If the model was to be persisted, the model building step should be performed, everytime a data set is added to or removed from the model data, since it always has to be up to date with the internal state of which data sets should be included in the model.

5.3 Miner Component

The miners were extended by one feature extractor that has the task of extracting the values, presented in Table 4.1. The new ML-Feature miner is different to existing miners in a few ways. First, since the output information is not focused on a single packet type, OSI-Layer or connection status, this miner listens to all possible events emitted by the packet parser. This means that per PCAP packet, information is persisted until all OSI-Layers and packet events have been processed by the miner. Once all this information is processed and stored into variables, the next PCAP packet is processed. This goes on, until all packets in a time-window of a predefined amount of time in seconds have been analyzed. When a new time-window starts processing, the stored metrics and data of the packets are consolidated into a memory-efficient object containing all the relevant features for that window. In Figure 5.1 we can see the values of the window object before consolidation, along with the transformation that is applied to create the final window-based records. Many of the values are arrays, collecting the data of all packets in a window, which then get reduced to either extract the number of unique items or calculate averages or percentages. The result is a lightweight object that has the same size regardless of the amount of traffic in a time-window.

From the command line interface (described in 5.3.1), the parameters for manual classification are provided to the miners. This means that all miners can potentially work with these specified attack-types. For the prototypical implementation however, just the new ML feature extractor makes use of these parameters. In the case of a single attack-type specified, all windows in the resulting consolidated form are appended with an attack-type value. Should multiple types be specified, the miner checks the current window according to its UNIX timestamp and applies the specified attack-type in integer form. For windows that are in between sections, the '0' attack-type is given, meaning that no attack or an unknown attack is occurring at that point in time.

The result of the feature extraction is on one hand stored in JSON files, as are the results from all other miners. These JSON files are used for the visualization part of DDoSGrid. In the ML feature miner's case, a Line Chart was chosen as a visualization that shows the attack type over time. This requires an array of the attack types for the Y-axis and an array of time-window numbers for the X-axis. This way the change in attack-type over time is clearly visible. Another JSON file is created to present the distribution of all attack types of a file in a pie-chart, without any reference to time. For a miner to output multiple files of different file-types, meant that the export function of the miner component had to be revised in order to store the files at their correct locations, depending on format, as well as expect multiple files from the individual miners. On the other hand, all the time-window features need to be exported into a CSV file that is accepted by the machine-learning pipeline. Since from this point on, the file could be used both for prediction by choosing the *Automatic Classification* option during upload, as well as for building the model by choosing the *Manual Upload* option, the resulting CSV has the same structure in either case.

5.3.1 Command Line Interface

In order to use the miner as a standalone program, the CLI is responsible for accepting the required parameters such as the PCAP file path, parsing them and provide them to the feature extractor pipeline. In order to supply manual classification information to the ML feature extractor, this CLI tool had to be updated to accept a pre-formatted string that either represents one attack type that categorized the entire PCAP file as one

Value	**Type**	**Transformation**
arrival_time	Unix Time of the first packet in the window	Used for determining wether to skip a window that is empty
packet_sizes_bytes	Array containing the sizes of all packets	Summed up and used to calculate average packet size, number of bytes in window and the number unique packet lengths
num_packets	Integer counting the number of packets in the window	Used to calculate percentages of packet types
source_ips	Array containing all source IPs of packets	Reduced to get the number of unique source IPs
dest_ips	Array containing all destination IPs of packets	Reduced to get the number of unique destination IPs
source_ports	Array containing all source ports of packets	Reduced to get the number of unique source ports
dest_ports	Array containing all destination ports of packets	Reduced to get the number of unique destination ports
num_tcp	Integer counting the number of TCP packets	Used to determine TCP percentages
num_udp	Integer counting the number of UDP packets	Used to determine UDP percentage
num_icmp	Integer counting the number of ICMP packets	Used to determine ICMP percentages
num_other	Integer counting the number of other packets	Used to determine other packet percentages
of_tcp_syn	Integer counting the number SYN flags set	Used to determine SYN flag percentage
of_tcp_fin	Integer counting the number of FIN flags set	Used to determine FIN flag percentage
of_tcp_ack	Integer counting the number of ACK flags set	Used to determine ACK flag percentage
of_icmp_echo_reply	Integer counting the number of ICMP echo reply packets	Used to determine ICMP echo reply percentage
of_icmp_dest_unreachable	Integer counting the number of ICMP destination unreachable packets	Used to determine ICMP destination unreachable percentage
arrival_times	Array containing all arrival times of packets	Used to calculate the inter-packet interval

Table 5.1: Time-Window values before they are consolidated.

attack type or a more sophisticated string that allows setting start and end points for different sections in the PCAP file that all have distinct attack types. In Listing 5.2, we can see three different possible argument variations for the updated CLI module. In line 2, no attack-type parameters are given, resulting in no manual classification of the PCAP file to-be-analyzed. On line 5, a simple integer is given, resulting in the entire file being classified as a SYN-Flood attack. On line 9, different sections of the file are specified by their UNIX Epoch timestamps. Time-windows that fall in between these timestamps will be classified as the integer that follows the starting and end time. These sections are separated by commas, letting a user specify many different sections, all with the same syntax. The attack-type argument gets parsed and sent to the feature extractors through the protocol parser.

```
1 # Starts the feature extraction without any attack-type classification
2 node index.js pcap_path=pcapfile.pcap
3
4 # Classifies the entire file as attack-type 1 (SYN-Flood)
5 node index.js pcap_path=pcapfile.pcap attack_type=1
6
7 # Classifies different sections of the file as
8 # attack-types 2 and 3 (ICMP-Flood and UDP-Flood)
9 node index.js pcap_path=pcapfile.pcap attack_type
     =898084694:898084721:2,898084778:898084789:3
```

Listing 5.2: Possible argument variations for the standalone Miner module, integrating manual classification information.

5.4 API Component

The API part of DDoSGrid required several changes and extensions to accommodate for the new machine learning functionality. Existing API routes needed some changes to incorporate the new configuration data sent from the front end, as well as the persistence layer, storing data set related values. Furthermore a new router was added, handling all classification related endpoints that were added.

5.4.1 Existing Routes and Persistence

Because classification-related data is transmitted from the user-focused front end to the API, the upload route had to be extended to accept and process these values. During upload, a record for each data set uploaded is created that contains meta-data, such as a generated id, file size, metrics, etc. This record was extended to also include the classification status, the classification type, the chosen ML algorithm, the attack type and their section times if needed, as well as whether the data set was added to the model data set. This allows for the analysis and classification steps of the back end to get these values from the database, as opposed to temporarily persist them for further API calls. The endpoint that starts analyzing data sets can now retrieve the attack type(s) from its internal database.

5.4.2 New Endpoints

All ML-related functionality is triggered via the */ml* route of the API. In order to cleanly separate the automatic classification process from the analysis step done on the analysis route, a POST request specifying the data set id was created. Similarly to the analysis request, after authentication and authorization is done, after which the CSV file containing the extracted features is fed into the machine learning pipeline. This pipeline includes the prediction of attack-types, as well as the updating of the existing CSV and JSON files containing the same values, which are used for the visualization of the predicted values.

Various endpoints relating to information regarding the algorithms and model have been added as well. One one hand this information is used on the upload form, where a user can configure the classification. On the other hand it's used on the newly created *Machine Learning* tab in the front end, where these stats are displayed in a collected overview of the attack-type classification extension. This data includes the available algorithms for classification, the possible attack types the model is able to predict, and lastly statistics about the model itself, such as size, the number of data sets inside the model data, and the total number of records.

A third functionality bundle offer endpoints that allow managing the model data. Adding and removing specific data sets to and from the model, respectively. The removal of the entire model data set can also be done. All these model-related operations are done using Python processes, since it offers easy to use CSV manipulation functions, as described in the next subsection.

5.4.3 Spawning of Python Processes

In DDoSGrid, the feature miners are started as forked Node.js processes, allowing asynchronous operations to occur. This results in the miners being able to process the data freely, while not interrupting regular server availability. A similar feature was used to launch the Python-based machine learning and CSV processing functionality. Using the *spawn* functionality of Node.js, other programs can be launched from JavaScript code. These spawned programs can be supplied with arguments, just like a regular shell program. For the machine learning processes the chosen algorithm, data-set, as well as the model data set can be variable, leading to the API retrieving and passing the needed values to the routines. The spawned processes can emit regular stdout data, as well as error information which is listened to from the JavaScript code. This way, error handling is handled the same as the rest of the API is made possible.

5.5 Front End Components

As described in Section 4.4, the front end part of DDoSGrid needed to be updated to accommodate for the new functionality. Both on the *Dataset* and *Dashboard* tabs, changes were implemented in different components. First, a visualization that allows displaying the attack type over time was created on the Dashboard, followed by modifications on the upload form and data set tile components on the Dataset tab. Additionally, a separate new tab was created, along with the ability to toggle all machine-learning related functionality, to not confuse new or inexperienced users.

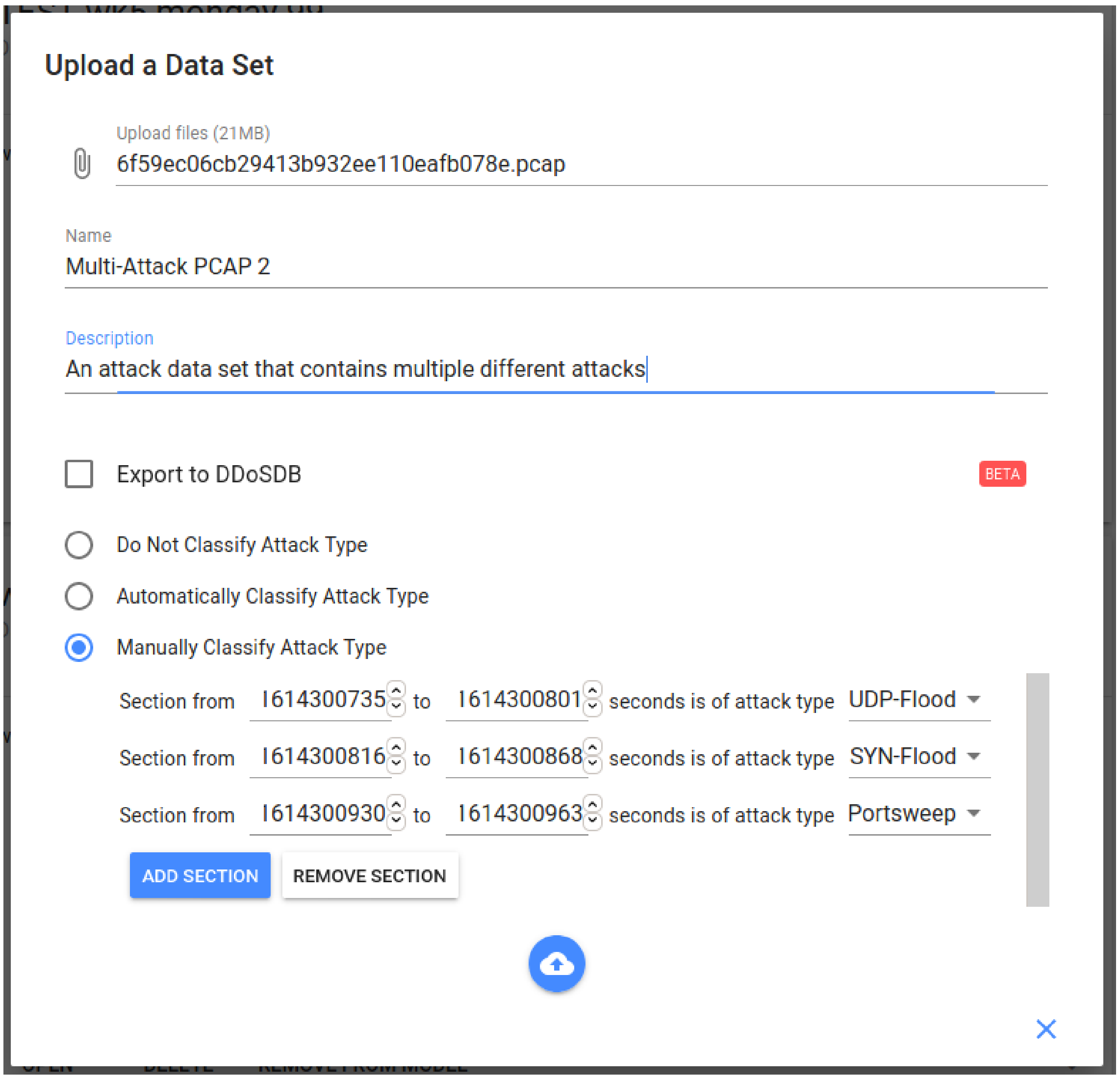
5.5.1 Visualizations

The new time-based visualization was created, similarly to the existing visualization components. A line chart was chosen to display the possible changing states of the *is_attack* value in the classified data set. To accurately display the legend, associating the attack-types with their integer ids, the new endpoint is used to retrieve all possible states, as opposed to hard-coding them into the component. The structure of the new line chart can be used to create other visualizations that display other time-based values of the data set. The values that it contains are arrays with the UNIX-Timestamps of the starting times of each window, and the attack-type that was either manually entered or automatically detected for every time-window.

Additionally, a pie chart containing the percentages of each individual attack states was created, to show, whether certain attack-types occurred at all and which ones occurred for longer periods of time. The *No Attack / Unknown* state is omitted in this visualization, because when uploading longer PCAP files, such as day-long network captures, the most traffic most likely is comprised of regular network traffic, skewing the visualization. Both visualizations can be seen in Figure 6.1, which displays an automatically classified data set that is one day long.

5.5.2 Upload Form

As specified in Section 4.4, the upload form needed to be extended to accommodate for the configuration of the classification step. If the *Extensions* toggle is active on the menu bar of DDoSGrid, the form has an additional three radio buttons the let a user select the type of classification. Either none, automatic, or manual classification. Should the automatic classification be selected, a sub menu appears, prompting the user to select which algorithm should be applied, with a default value already being set. Similarly, if the manual classification option should be selected, a sub menu appears that first lets a user specify the attack-type of the entire data set. From here, the user can click the *Add Section* button, which switches the look of the form to offer a start and an end time in UNIX epoch format. This way multiple sections of different attack-types can be specified for the same file, as can be seen in Figure 5.2. Removing sections is also possible, and if only one section is present again, the presentation switches to only offering to specifying the entire file. These form values are persisted in the state, so they can be converted to the format described in 5.3.1 before being sent to the back end.



Upload a Data Set

Upload files (21MB)
6f59ec06cb29413b932ee110eafb078e.pcap

Name
Multi-Attack PCAP 2

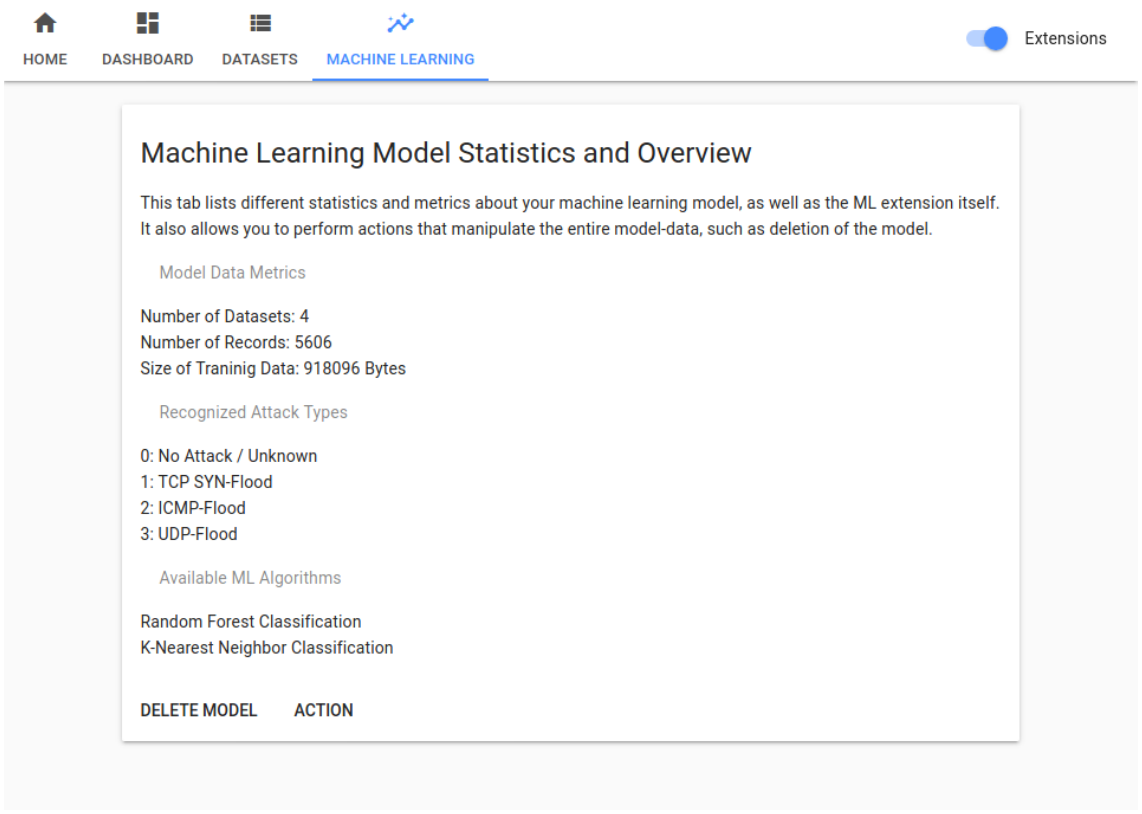
Description
An attack data set that contains multiple different attacks

☐ Export to DDoSDB BETA

○ Do Not Classify Attack Type
○ Automatically Classify Attack Type
● Manually Classify Attack Type

Section from 1614300735 to 1614300801 seconds is of attack type UDP-Flood
Section from 1614300816 to 1614300868 seconds is of attack type SYN-Flood
Section from 1614300930 to 1614300963 seconds is of attack type Portsweep

Figure 5.2: The final upload modal, displaying data set that is configured to be manually classified with different attacks.



Machine Learning Model Statistics and Overview

This tab lists different statistics and metrics about your machine learning model, as well as the ML extension itself. It also allows you to perform actions that manipulate the entire model-data, such as deletion of the model.

Model Data Metrics

Number of Datasets: 4
Number of Records: 5606
Size of Traninig Data: 918096 Bytes

Recognized Attack Types

0: No Attack / Unknown
1: TCP SYN-Flood
2: ICMP-Flood
3: UDP-Flood

Available ML Algorithms

Random Forest Classification
K-Nearest Neighbor Classification

Figure 5.3: Machine Learning Information tab. The extension toggle is active, making this tab visible.

5.5.3 Machine Learning Information Tab

In order to have a centralized location, where all the relevant information regarding the machine learning extension can be displayed, a new tab was created in the menu bar. This tab called *Machine Learning*, lists metrics regarding the model data, such as total number of data sets inside the model, the number of records inside the data set, as well as the file-size of it. All attack-types, recognized by the system are listed as well. This also includes the internal integer id for the attack types. Lastly, all available machine learning algorithms are listed that can make use of the model data and predict new attack-type values. To gather all this data in one place, the new /ml endpoints were used that provide the required data about the model. In order to give the user an impression of what parts of the model are made up of which attack types, a distribution graph is created by counting the content of the model data. This graph is in the form of a cake-diagram, along with a legend of all states.

The model can also be evaluated from here. When loading the page, an evaluation of the data set, using a X, Y train and test split is performed and its result are presented to the user, in form of a true/false matrix for all possible states, as well as a accuracy percentage. Chapter 6 documents this evaluation procedure in more detail, along with the system performance evaluation.

Chapter 6

Evaluation

The following chapter documents the evaluation process that was done, to assess the viability of the proposed and built solution, which was en extension to the existing DDoSGrid application. On one hand, the system performance in terms of speed and ability to run large data sets is evaluated. For this multiple systems will be tested, to locate possible bottlenecks. On the other hand, accepted machine learning metrics will be calculated to evaluate the overall accuracy of the combination of chosen features, as well as algorithms. Furthermore, these results will be compared to the existing literature, upon which the authors conclusions will be drawn. Additionally, a sample output of an automatically classified data set is visualized and analyzed in detail.

6.1 Data Sources and Model

For the evaluation, it was decided that the classifier should be able to identify 7 different network states. These were chosen due to their availability from the data sets, as well as their occurrence in actual attacks. Both protocol and volumetric attacks were chosen, to provide a balanced selection of attack types.

- No Attack / Regular Traffic
- TCP-SYN Flood
- ICMP-Flood
- UDP-Flood (Destination Unreachable)
- IP-Sweep
- Ping-of-Death
- Portsweep

To train the final model, mixed data sources were used. On one hand, DDoSDB was used for TCP-SYN Flood attacks and UDP-Flood attacks, since they provide clean and well documented attack traces. On the other hand, the DARPA Intrusion Detection Evaluation data sets, provided ample attack data across all attack states that we planned to classify, including regular traffic of the actual network. Attack logs from the 1998 evaluation that we used, included the entirety of weeks 5 and 6, as they provided well balanced data sets using all aforementioned attacks, except for UDP-Floods.

The data sets were cleaned up after downloading. That included removing interfering packages, such as Logical-Link Control and Configuration Test Protocol packages, since both hace the same source and destination addresses, meaning they only consisted of internal traffic. The data sets were then split up into their respective attacks, to make uploading them to DDoSGrid on a category-basis possible. They were uploaded using the *Manual Classification* functionality which provides a graphical user interface to select from a list of attack types. Once all data sets have been uploaded, they were added to the model, which gradually increased in size, due to the sometimes large nature of ICMP-Flood and UDP-Flood data sets. On the Machine Learning tab, a graph depicting the distribution of attack types in the model is calculated, as seen in Figure 5.3. One issue that we encountered was that due to the nature of some attacks being shorter than others, they will only occupy a fraction of the number of total records in the model, even when supplying more attacks compared to others. This is also due to the fact that we extracted the features on a time-window basis. An example would be to compare SYN-Flood attacks and Ping-of-Death attacks. The PoD attacks were only mere seconds in length, compared to SYN-Flood attacks which may last hours. In order to lessen that gap of distribution percentages, the idea of removing duplicates in the model arose. The results of this experimental change is detailed in Table 6.4.

Something we noticed was also was that certain attacks, which can be categorized as DDoS-supporting attacks, can be employed in a noticeable and in stealthy ways. For example IP-Sweeps and Portscans can either be done slowly, e.g. pinging an address or a port only every few seconds, or all at once, resulting in higher *Destination Ports* and *Destination IPs* features after being mined. We chose to only include the data sets containing the noticeable occurrences of these attacks, as the stealthy occurrences can be mistaken for either regular traffic or other attacks altogether.

The final model had a total length of 55'349 records and contained 18 datasets, with most of them containing two to four attacks. The model had a size of 8.9MB, which was only a fraction of the original datasets that were often gigabytes in size.

6.2 Performance Evaluation

To evaluate the performance of both the new miner, as well as the classification procedure, a time-based evaluation was performed. This process was conducted on two different systems with different specifications and architectures to gauge how much of a difference in performance these systems provide. Data sets of different lengths and traffic volumes will be fed into the two procedures to simulate real-case scenarios. For the classification

Computer	**CPU**	**RAM**	**OS**
Desktop	Intel 4770K @ 4x4.0GHz	16GB	Arch Linux
RaspberryPi 3 B	Broadcom @ 4x1.2GHz	1GB	Raspbian

Table 6.1: The two computers used to run the performance evaluation.

evaluation, the same model data set will be used as in section 6.3. The computers used for these tests can be seen in Table 6.1, with the first machine to run these performance tests is an Intel-based desktop computer running Arch Linux. The other is a RaspberryPi 3+ running Raspbian. The data sets that will be fed into the new feature extractor are as follows:

- One hour-long SYN-Flood Attack (230'630 packets of 60 Bytes)
- Half hour-long ICMP-Flood Attack (1'659'612 packets of 1066 Bytes)
- 45 second-long UDP-Flood Attack (1'410'972 packets of varying sized between 400 and 3000 Bytes)

The built-in performance test script of DDoSGrid will be used to single out the performance of the *Machine Learning Feature Extractor* miner, which allows setting multiple PCAP files and the number of iterations that should be made at launch. It will compute the average speed and peak memory used, the miner has taken to analyze the file and mine all required features. The results of the analysis evaluation can be seen in Table 6.2. It is quickly visible that the RaspberryPi performed significantly worse than the Desktop computer. This can be attributed to many factors aside from the obvious CPU and RAM differences. For one, the PCAP files were loaded from an SD Card that has significantly lower read and write speeds, compared to the Desktop's SSD storage, leading to less packets being transferred from storage to RAM. Second, the RAM of the RaspberryPi seemed to hit a cap at 10MB of heap memory used. This could either be bottle necked by the aforementioned storage speeds or CPU power. We believe that the Pi's analysis speeds could be improved by switching to a USB-mounted storage disk or using network storage, to not be limited in that factor. Looking at the Desktop's analysis speeds, we get a better picture of how resources are used in the miner. While being comparable in total packet numbers, the UDP-Flood and ICMP-Flood both had different analysis speeds and RAM usage. What leads to this difference is the intensity of the attack. Since the UDP attack only lasted for 45 seconds but has a similar amount of packets transferred, the time-window calculation is much more processing intensive when it comes to calculating metrics such as the total number of source IPs per time window. Comparing the *Machine Learning Feature Extractor* miner to existing ones running on the same machine, they only performed about 10-20% worse in terms of speed, depending on the use case, despite processing packets on multiple OSI layers.

The classification performance evaluation was set up in a similar way as the analysis evaluation. However, the three PCAP files were already analyzed in advance and fed into the Python routines that handled model building, classification and writing of the new data into the data set's analysis CSV file. Looking at the performance difference

File	Computer	Speed	Memory
SYN Flood	Desktop	2829ms	32MB
UDP Flood	Desktop	3822ms	84MB
ICMP Flood	Desktop	3033ms	46MB
SYN Flood	RaspberryPi	77'309ms	10MB
UDP Flood	RaspberryPi	69'256ms	10MB
ICMP Flood	RaspberryPi	116'513ms	10MB

Table 6.2: Elapsed time and memory consumption of the feature extractor, for both computers and all three data sets.

File	Algorithm	Computer	Speed Total	Speed Classification
SYN Flood	Random Forest	Desktop	651ms	27ms
SYN Flood	k-Nearest Neighbor	Desktop	1247ms	204ms
SYN Flood	Random Forest	RaspberryPi	8457ms	319ms
SYN Flood	k-Nearest Neighbor	RaspberryPi	-	-
UDP Flood	Random Forest	Desktop	638ms	5ms
UDP Flood	k-nearest Neighbor	Desktop	1051ms	7ms
UDP Flood	Random Forest	RaspberryPi	8121ms	44ms
UDP Flood	k-nearest Neighbor	RaspberryPi	4387ms	620ms
ICMP Flood	Random Forest	Desktop	642ms	17ms
ICMP Flood	k-Nearest Neighbor	Desktop	1115ms	61ms
ICMP Flood	Random Forest	RaspberryPi	8322ms	181ms
ICMP Flood	k-Nearest Neighbor	RaspberryPi	-	-

Table 6.3: Elapsed time of the classification process, detailed for every file, algorithm and computer combination.

between Desktop and RaspberryPi, the difference in file-loading times once again becomes visible, as well as the CPU and RAM throughput differences. In two cases using k-NN classification on the RaspberryPi, the process was killed by the operating system, either by using too much RAM or locking up the system, so no completion times are available for these tests. This makes sense, since k-NN is a memory-consuming process, especially with the large model and number of features in the data sets. Comparing the classification speeds under the same circumstances (Algorithm and Computer), it is apparent that the actual classification step only makes up for a fraction of the total processing time. Importing the and building the model of around 50'000 records takes up around 651ms in the Desktop's Random Forest classification evaluation, while actual classification only took 27 milliseconds. Still, the UDP Flood classification is the fastest, as it only contains 45 records for each second of traffic, while the SYN-Flood and ICMP-Flood files contain around 1500 and 3100 records, respectively.

6.3 ML Metrics Evaluation

In this section, a comprehensive evaluation is performed with the aim of reaching a conclusion about the system's viability as an automatic DDoS Attack Type Classifier. For this, a model is trained using data from different data sources that include a variety of DDoS attack types. This also includes regular traffic, so that the system does not accidentally classify regular network states as attacks. Two machine learning algorithms were evaluated; Random Forest Classification and k-Nearest Neighbor Classification, to show a possible difference in overall accuracy across different methods. The model was cross-validated using a 80% Train and 20% Test Split. This cross validation was performed ten times, with a randomized data set split. The reported results in form of *Precision*, *Recall*, *F1-Score*, and *Accuracy*, the evaluation procedure was run twice, once with removed duplicates in the data set and once with duplicates kept. The results of each evaluation run are displayed as macro averages and as weighted averages, due to the model data set not being balanced in terms of attack types represented. Another note, the accuracy was calculated by rounding a floating point value, while *Precision*, *Recall* and the *F1-Score* were calculated by averaging percentage integers.

Looking at the results in Table 6.4, the Precision, Recall and F1-Score in all unweighted tests reached 100% from which we can tell that the system performs very well with attack-types for which many records exist in the database. In this test the majority of records consisted of regular traffic, SYN Flood Attacks, ICMP Flood Attacks and UDP Flood attacks. Even though not all attack states were classified correctly, the large ratio of attack types that were classified 100% correctly to falsely classified types still averaged out to 100%. When comparing those values to their weighted counterparts it becomes clear that the system had trouble classifying attacks that were not well represented in the data set, such as IP Sweeps and Port Sweeps. We think that if the data set had more records of these underrepresented attack types, the metrics could be closer to their unweighted counterparts. The experimental test to see whether removing duplicates in the model data set proved to be fruitful, as all test that had duplicate records removed, performed better than their counterpart that kept the entire data set. Duplicate records shrunk the model data to 43'714 records which is a 21% reduction in length. Comparing Random Forest Classification with k-Nearest Neighbor Classification in our case, Random Forest performed significantly better in the weighted result scores. This is due to the fact that the Random Forest Classifier manages to better correctly classify low-occurrence attack types compared to k-Nearest Neighbor Classification. However both algorithms performed equally well in classifying the well-represented attack types in the model.

6.3.1 Findings and Discussion

In the existing literature [28] achieved an accuracy of 98.6% with Random Forest classification, however no other metrics were presented, except for the false positive rate of 2.4%. They used a combination of flow-based and pattern-based features for their classifiers. In [29], the authors achieved a Random Forest classification accuracy of 99.53%, precision of 100%, recall of 99.12% and and F1 score of 0.9956 in their joint detection solution, which

Method	**Accuracy**	**Precision**	**Recall**	**F1-Score**
RF with Duplicates	99.9%	100%	100%	100%
RF with Duplicates (Weighted)		95.1%	92.3%	92.4%
RF without Duplicates	99.9%	100%	100%	100%
RF without Duplicates (Weighted)		97.9%	95.9%	96.5%
k-NN with Duplicates	99.9%	100%	100%	100%
k-NN with Duplicates (Weighted)		83.7%	81.2%	82.3%
k-NN without Duplicates	99.9%	100%	100%	100%
k-NN without Duplicates (Weighted)		85.4%	83.5%	84.3%

Table 6.4: Results from the metrics evaluation, after cross-validating the built model.

involved multiple hosts analyzing traffic. Suresh [32] achieved an accuracy of 96.6% using k-Nearest Neighbor classification and an F-Score of 0.969. Comparing these values with the ones we achieved, we can place our performance, especially from the RF classifier among these previous research efforts.

Judging off of the performance difference between the two test devices, we believe that further optimizations to both the code base and test infrastructure could be done to tighten the gap between the two. Better RAM usage, improving read and write speeds using other storage media, as well as persisting the model could improve performance on lower-end hardware and virtual machines. Another change that may improve the metrics of the solution would be to use a more balanced model data set in terms of attack-types represented.

Still, the new feature extractor's performance is comparable and sometimes even faster than other existing miners, meaning on hardware where DDoSGrid already performed well, the new feature extractor is a non-conflicting addition to DDoSGrid's functionality. The same holds true for the classification pipeline, which produces results in a quick fashion, sometimes even without a user noticing an additional operation is performed.

6.4 Case Study: Visualization Analysis of Test Data Set Output

In order to evaluate the visual output of the classification pipeline, a data set from the DARPA intrusion tests was chosen. This data set was recorded on Friday of the seventh week of the 1998 tests. As can be seen in [43], in this attack, the test environment was scheduled to be the victim of two Neptune (TCP-SYN Flood) and one Smurf (ICMP Flood) attack. It also contained a Back attack, however the model was not trained to detect this attack. It was classified using the same model as was used to evaluate the performance and accuracy of the application, using Random Forest Classification. It is important to note that the time-zone difference from the authors and the documentation differs by four hours, meaning all time-comparisons will be formatted as *Author Time* (*Documentation Time*). We first take a look at the visualization on the left in Figure 6.1, which plots the DDoS-Attack type over time. Note that the X-axis is not necessarily a

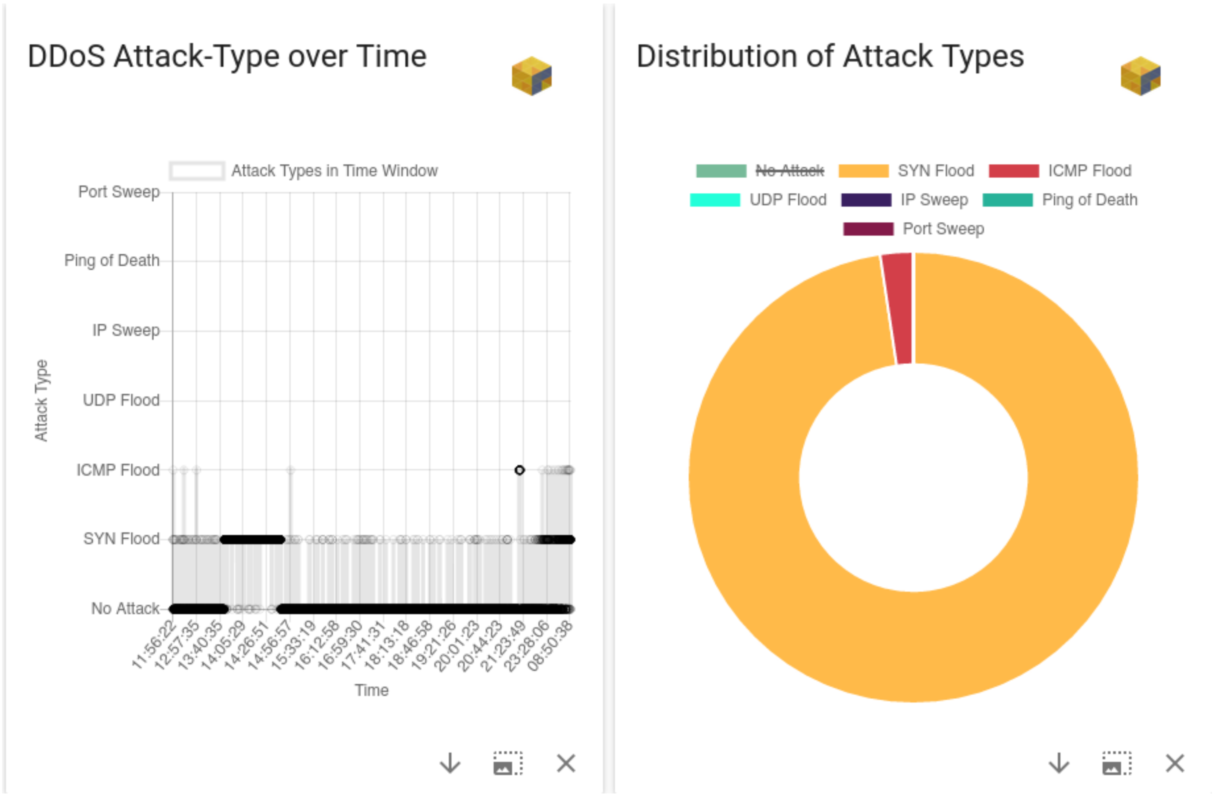


Figure 6.1: Visualizations, representing the test PCAP file, to show an over-time analysis and the total distribution of attack types.

linear time graph, as the analysis omits time-windows that do not contain any traffic. This results in more emphasis given to time-slots which are attack-heavy. The visualization is programmed to give the individual dots on the graph a very high opacity when the data set contains many data points, to better illustrate clusters of possible attack locations, as data points plotted close to each other will appear less opaque than single points.

Following the documentation of the tests, the first SYN Flood attack occurred at 13:48 (09:48) and lasted about an hour until 14:39 (10:39). During this attack, the classification clearly detected a long sequence of SYN-Flood attack states in the PCAP file. During this time, only a small amount of traffic was classified as Regular Traffic, leading us to believe that this attack was quite severe in its intensity, leaving little space for other traffic to reach the infrastructure. Throughout the entire file, a few individual data points falsely classify regular attack traffic as SYN-Flood traffic, due to the fact that HTTP traffic might share some characteristics with SYN-Flood traffic, such as a high TCP packet concentration. The second attack appeared much later in the day, at 21:16 (17:16) in form of a Smurf attack that lasted only for around 70 seconds. However it is visible on the visualization, because of its attack intensity. This means that the classifier classified most, if not all, the existing time windows as ICMP-Attacks, resulting in the darkened color of the data points in the visualization. The third attack occurred at 23:21 (19:21) again in the form of a TCP SYN-Flood attack. This time, the attack was not as severe as the first one, since it was launched in bursts of attack floods. This is also represented in the visualization, since both *No Attack* and *SYN Flood* Y-values contain many data points, indicating that regular traffic did occur in between bursts of attacks.

Looking at the right visualization in Figure 6.1, a total distribution of the classified attack states is visible. In the figure, the *No Attack* portion has been hidden, to show what parts of the analyzed and classified PCAP file contain which attacks. The majority of time windows was classified as SYN-Flood attacks, while a small portion of the time windows were classified as ICMP Floods This is accurate, comparing these results with the time-based visualization and the documentation of the DARPA intrusion tests.

Chapter 7

Summary, Conclusions and Future Work

This thesis assessed the existing literature, regarding machine learning supported DDoS attack detection and classification and put it into perspective regarding types of machine learning algorithms, traffic data and overall feasibility of pursuing certain methods that may be included in DDoSGrid as a modular extension. A set of features was decided on, that should aid in classifying multiple attack types, including regular traffic. While certain features are only used for one attack type, others allow identifying multiple different attack types. The combination of those two feature types proved to be fruitful when testing three machine learning algorithms in a preliminary test-run with data sets from different sources and the aforementioned features, upon which models were built. Out of the three algorithms, two were decided to be suitable to be included in DDoSGrid, namely Random Forest Classification and k-Nearest Neighbor classification; Two algorithms that provided low classification times and high accuracies, even with a limited model data set. In order to build a classification module into DDoSGrid, it's existing architecture had to be revised, which entailed changing the purpose and functionality of some existing models and the addition of new ML-related modules, in order to keep the proposed functionality modular from the main application.

Implementation-wise, a new miner was created that extracted the required features from data sets on a time-window basis and on multiple OSI layers, something the existing application did not yet do. For the machine learning pipeline, python processes were spawned in a similar fashion to the asynchronous miners, letting the system continue its intended purpose, even while analyzing and classifying data sets. In order to populate the model data set from the web-application, manual classification had to be possible, since the model data set should consist of true data. This can be done from the front end, along with the functionality for automatic classification. A new tab was also created that allowed displaying metrics and statistics of the model to be displayed, along with functions that relate to the model as a whole, such as deleting it. For the new miner outputs, new visualizations were created that aid in displaying the new extracted features and metrics, such as over-time visualizations of the attack type, or the distribution of attack types inside the entire data set.

Lastly the classification processes were evaluated in both performance (i.e.) time dimensions, as well as calculating common machine learning related metrics, such as accuracy, precision, recall and f1-score. Different data sets, algorithms and devices were used for this evaluation in every possible combination, to show how the program performs under different circumstances. As a whole, the evaluation results proved to be quite favorable, in line with the existing research or performing even better, in some cases.

7.1 Conclusions

Comparing the results of the evaluation part of this thesis with the existing research, we proved that machine learning based DDoS and network attack detection is a viable option when sufficient data of past attacks is provided. The results proved that very high accuracies, as well as other machine learning metrics can be achieved, even when classifying multiple different attack types. One issue with the current implementation that we found was classifying attacks that are very short time-wise. Since these short attacks only provide a limited number of time-frames or records in the model, less information about them is known, making their classification less accurate than with longer attacks. Additionally, visualizing these short attacks can also be a challenge, especially when analyzing a data set that was recorded over the period of a day, or more. Even when correctly classified, the data points only show up as very small specks on the over-time classification and might be ignored by a user or accepted as falsely classified records. The solution to this problem and visualizing day-long data sets in general could be solved by implementing clustering methods, that either use the most-occurred attack-type over a period of time or by using anomaly detection that analyze the time frames before and after for each record.

The extension to DDoSGrid works well with the existing functionality it already provides, making it a well-integrated module. This thesis also showed the extensibility of DDoSGrid, a goal that was set during conceptualization and development of the base DDoSGrid application [1]. Minor tweaks were necessary to the existing code base, however most of the changes that were made, could be also used by the base functionality or future extensions, such as the ability to export multiple files per miner. The tab-based interface also makes adding new functionality as a separate module easy.

7.2 Future Work

Since DDoSGrid and its development is an ongoing project, certain features would be well suited to be included or tested in future installments of the application, especially for the machine learning components. Since ML algorithms offer a plethora of configuration options, configuring these from the front end would be a useful feature, especially when implementing new algorithms. Tweaking their parameters from a GUI lets a user play around with different configurations and evaluating their results at the same time from the *Machine Learning* tab. This would have to be implemented specifically for every algorithm and imply changes from the front-end all the way to the ML pipeline. Naturally the inclusion of more recognized attack-types is a desired feature, due to the ever-changing landscape of DDoS attacks. Depending on the characteristics of attacks, these new attacks may or may not need new features extracted from the traffic, so a case-by-case analysis and accuracy evaluation might be needed to know if new extracted features are required. Another feature that could be required, once significant data volumes are uploaded to an active DDoSGrid instance would be model persistence. For this prototype the need was not seen to include this functionality due to the processing times being of reasonable amounts. One thing we encountered once we uploaded many datasets for the evaluation was that the list of data sets could grow rather long and unintuitive. A way to combat this would be to include filtering and sorting options to the *Data Sets* tab.

Since there are efforts to take the DDoSGrid platform and use it as a general cyber security analysis tool [25, 26], the possibility of using its machine learning functionalities becomes an attractive option. Depending on the use case though, the extracted feature set and analysis pipelines would have to be changed drastically. An investigation into trustworthiness levels of ML approaches can also be made in the context for DDoSGrid since there are demands in literature, such as the research and implementation of features that helps to investigate key aspects such as fairness, robustness, explainability, and accountability of an AI system for cyber security.

Bibliography

[1] L. Boillat, J. von der Assen: ”A Tool for Visualization and Analysis of Distributed Denial-of-Service (DDoS) Attacks”, University of Zurich, 2020

[2] S. T. Zargar, J. Joshi and D. Tipper: ”A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks,” in IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 2046-2069

[3] Detection techniques of DDoS attacks: A survey, P. Kamboj, M. C. Trivedi, V. K. Yadav and V. K. Singh, 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON), DOI:10.1109/UPCON.2017.8251130

[4] 39 Jaw-Dropping DDoS Statistics to Keep in Mind for 2020, Accessed: 11. September 2020 https://hostingtribunal.com/blog/ddos-statistics

[5] DDoS Attacks 101: Types, targets, and motivations, Accessed: 11. September 2020 https://www.calyptix.com/top-threats/ddos-attacks-101-types-targets-motivations/

[6] Cisco Annual Internet Report (2018-2023) White Paper, Accessed: 11. September 2020 https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html/

[7] What is a Denial-of-Service (DoS) Attack?, Accessed: 18. September 2020 https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/

[8] What Is LOIC?, Accessed 19. February 2021 https://gizmodo.com/what-is-loic-5709630

[9] Mining for Groups Using Clustering Algorithms, Accessed 3. March 2021 http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/mvoget/cluster/cluster.html

[10] APPLYING RANDOM FOREST (CLASSIFICATION) — MACHINE LEARNING ALGORITHM FROM SCRATCH WITH REAL DATASETS, Accessed 3. March 2021 https://medium.com/@ar.ingenious/applying-random-forest-classification-machine-learning-algorithm-from-scratch-w

[11] Machine Learning Basics: K-Nearest Neighbors Classification, Accessed 3. March 2021 https://towardsdatascience.com/machine-learning-basics-k-nearest-neighbors-classification-6c1e0b209542

[12] Detecting Memcached DDoS Attacks Targeting GitHub, Accessed 26. February 2021 https://logrhythm.com/blog/detecting-memcached-ddos-attacks-targeting-github/

[13] DDoSDB, https://ddosdb.org/help, last accessed 25. September 2020

[14] R. Odoni: "Design and Implementation of a Distributed Denial-of-Service Data Simulator", University of Zurich, 2019

[15] C. Chio, D. Freeman: "Machine Learning & Security; Protecting Systems with Data and Algorithms", O'Reilly, 2018

[16] Understanding DoS attacks and the best free DoS attacking tools, Accessed: 1. October 2020 https://resources.infosecinstitute.com/dos-attacks-free-dos-attacking-tools/

[17] Heightened DDoS Threat Posed by Mirai and Other Botnets, Accessed 1. October 2020 https://us-cert.cisa.gov/ncas/alerts/TA16-288A

[18] What do Mirai & IoT botnets mean to the public sector?, Accessed 1. October 2020 https://blogs.cisco.com/security/what-does-mirai-iot-botnets-mean-to-the-public-sector

[19] Multi-Vector DOS Attacks Turning into the New Normal, Robert Lemos, Accessed: 1. October 2020 https://symantec-blogs.broadcom.com/blogs/expert-perspectives/multi-vector-dos-attacks-turning-new-normal

[20] Understanding and Stopping Multi-Vector DDoS Attacks, Accessed: 1. October 2020 https://www.corero.com/blog/understanding-and-stopping-multi-vector-ddos-attacks/

[21] Reflection Attacks and Amplification Attacks, Accessed: 1. October 2020 https://www.cloudbric.com/blog/2015/03/reflection-attacks-and-amplification-atttacks/

[22] How To Build A Botnet In 15 Minutes, Brian Proffitt, Accessed: 24. March 2020 https://readwrite.com/2013/07/31/how-to-build-a-botnet-in-15-minutes/

[23] J. "Nazario, DDoS Attack Evolution", Network Security, Volume 2008, Issue 7, 2008, Pages 7-10 https://doi.org/10.1016/S1353-4858(08)70086-2

[24] Nationales Zentrum für Cybersicherheit NCSC, DDoS Attacken, Accessed: 1. October 2020 https://www.melani.admin.ch/melani/de/home/themen/DDoSAttacken.html

[25] J. von der Assen, "DDoSGrid 2.0: Integrating and Providing Visualizations for the European DDoS Clearing House", University of Zurich, February 2021

[26] M. Franco, J. von der Assen, L. Boillat, C. Killer, B. Rodrigues, E. Scheid, L. Granville, B. Stiller, ”SecGrid: an Extensible Platform for the Analysis and Visualization of Cyberattacks Traffic”, In Preparation, 2021

[27] P. Khuphiran, P. Leelaprute, P. Uthayopas, K. Ichikawa, W. Watanakeesuntorn: ”Performance Comparison of Machine Learning Models for DDoS Attacks Detection”, ICSEC 2018, 2018

[28] J. Hou, P. Fu, Z. Cao, A. Xu: ”Machine Learning based DDos Detection Through NetFlow Analysis”, MILCOM 2018 Track 3 - Cyber Security and Trusted Computing, 2018

[29] Z. He, T. Zhang, R. Lee: ”Machine Learning Based DDoS Attack Detection From Source Side in Cloud”, 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing, 2017

[30] B. Khalaf, S. Mostafa, A. Mustapha, M. Mohammed, W. Abdullah: ”Comprehensive Review of Artificial Intelligence and Statistical Approaches in Distributed Denial of Service Attack and Defense Methods”, IEEE Access PP(99):2169-3536, 2019

[31] J. Li, Y. Liu, L. Gu: ”DDoS Attack Detection Based On Neural Network”, 2010 2nd International Symposium on Aware Computing, 2010

[32] M. Suresh, R. Anitha: ”Evaluating Machine Learning Algorithms for Detecting DDoS Attacks”, Communications in Computer and Information Science 196:441-452, 2011

[33] B. Zhang, T. Zhang: ”DDoS Detection and Prevention Based on Artificial Intelligence Techniques”, 2017 3rd IEEE International Conference on Computer and Communications, 2017

[34] K. Wehbi, L. Hong, T. Al-salah, A. Bhutta: ”A Survey on Machine Learning Based Detection on DDoS Attacks for IoT Systems”, 2019 SoutheastCon, 2019

[35] S. Das, A. Mahfouz, D. Venugopal, S. Shiva: ”DDoS Intrusion Detection through Machine Learning Ensemble”, 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2019

[36] R. Doshi, N. Apthorpe, N. Feamster: ”Machine Learning DDoS Detection for Consumer Internet of Things Devices”, 2018 IEEE Symposium on Security and Privacy Workshops, 2018

[37] S. Priya, M. Sivaram, D. Yuvaraj, A. Jayanthiladevi: ”Machine Learning based DDOS Detection”, 2020 International Conference on Emerging Smart Computing and Informatics (ESCI), 2020

[38] T. Dietterich: ”Ensemble Methods in Machine Learning”, MCS 2000: Multiple Classifier Systems, 2000

[39] M. Ahmed, A. N. Mahmood, and J. Hu, ”A survey of network anomaly detection techniques”, J. Netw. Comput. Appl., vol. 60, pp. 19-31, 2016

[40] M. Gyanchandani, J. L. Rana, and R. N. Yadav, ”Taxonomy of anomaly based intrusion detection system: A review”, Int. J. Sci. Res., vol. 2, no. 12, pp. 1-13, Dec. 2012

[41] P. Machaka, A. Bagula and F. Nelwamondo, ”Using exponentially weighted moving average algorithm to defend against DDoS attacks”, 2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), Stellenbosch, South Africa, 2016

[42] DARPA Intrusion Detection Evaluation, Accessed: 26. February 2021, `https://archive.ll.mit.edu/ideval/index.html`

[43] DARPA Intrusion Detection Evaluation: 1998 Trainin Data Attack Schedule, Accessed 26. February 2021, `https://archive.ll.mit.edu/ideval/docs/attacks.html`

[44] CAIDA Data - Overview of Datasets, Monitors and Reports, Accessed: 26. February 2021, `https://www.caida.org/data/overview/`

[45] Intrusion Detection Evaluation Dataset (CIC-IDS2017), Accessed: 26. February 2021, `https://www.unb.ca/cic/datasets/ids-2017.html`

[46] DDoSGrid Github Repository, Accessed 3. March 2021, `https://github.com/ddosgrid/ddosgrid-v2`

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CLI	Command Line Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
DARPA	Defense Advanced Research Projects Agency
DB	Database
DDoS	Distributed Denial of Service
DFF	Deep Feed Forward
DNS	Domain Name System
DT	Decision Tree
DoS	Denial of Service
EWMA	undefined
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
KB	Kilobyte
MB	Megabyte
MIT	Massachusetts Institute of Technology
ML	Machine Learning
NN	Neural Network
NTP	Network Time Protocol
OS	Operating System
OSI	Open Systems Interconnection
PDF	Portable Document Format
RAM	Random Access Memory
REST	Representational State Transfer
RF	Random Forest
SSD	Solid-State Drive
SSDP	Simple Service Discovery Protocol
SVM	Support Vector Machine
TCP	Transmission Control Protocol

UDP	User Datagram Protocol
USB	Universal Serial Bus
k-NN	k-Nearest Neighbor

List of Figures

List of Tables

Appendix A

Installation Guidelines

This chapter documents how to locally install a DDoSGrid instance with the added machine learning capabilities. Note that the implementation may change in the future, so please check with the most up-to-date instructions on the GitHub repository [46]

A.1 DDoSGrid Installation

The repository should be cloned first, which contains all DDoSGrid-related components. Either clone the most recent version from GitHub, or use the provided code from the CD.

```
git clone git@github.com:ddosgrid/ddosgrid-v2.git
```

The following programs should also be installed on the target machine. For installation guidelines, refer to the official documentation of the respective programs.

- Node.js
- npm
- git
- Python, Version >= 3.7
- libpcap
- SSH Client

A.1.1 Machine Learning

First, all machine learning related functionality should be installed. For this, change into the */ml* directory in the repository and install the Python packages using

```
cd ddosgrid-v2/ml
pip3 install -r requirements.txt
```

A.1.2 Miner

The miner should be installed before using the API module and can be done by via:

```
cd ddosgrid-v2/miner
npm i
```

A.1.3 API

Change into the API module's directory, and install it the required components.

```
cd ddosgrid-v2/api
npm i
```

Then, in order to locally run the API and it's connection to a running OAuth instance, use the provided script to connect to the CSG's service. Note that you may require a working account and your IP or domain might need to be added to the allowed hosts by a CSG member.

```
./scripts/start_dev_server.sh
```

A.1.4 Front End

The last step is installing and running the front end part of DDoSGrid. First, install all required dependencied with:

```
cd ddosgrid-v2/frontend
npm i
```

Then, run the program, by executing:

```
npm run serve
```

Which runs the application locally and listens to port 8081.

A.2 Evaluation

In order to reproduce the evaluation of the feature extractor, as well as the classifier, scripts are provided that aid in evaluating these components. Make sure that the components have been installed, as instructed in Section A.1

A.2.1 Feature Extractor Performance

Inside the */miner* directory, a file called performance-test.js is located. Inside, the feature extractors that should not be evaluated can be commented out, or removed from the file in the require statement, starting at line 2. In order to only perform the evaluation for the new miner, only leave the *MachineLearningFeatureExtraction* miner in the require statement. To start the evaluation, use the following statement:

```
node performance-test.js pcap_path="path/to/pcap/file.pcap"
```

The results will be printed on the command line.

A.2.2 Classifier Performance

For the classifier performance to be evaluated, a well-built model is required. We supplied the model in the CD content, to be used for classification and testing. We also included Python scripts that accept this model and a PCAP file that should be classified and output the average completion times of the classification.

To use these scripts, inside the */ml* directory you will find a folder called *evaluation/performance* that contains one file for every machine learning algorithm. Use these files in the following way:

```
python3 algorithm_eval.py /path/to/model/data/training.csv /path/to/pcap
    /file.pcap
```

This will run the execution 10 times and print the average results on the command line.

A.2.3 Classifier Metrics

Similarly to the classifier performance, the classifier metrics evaluation can also be reproduced. In the */ml* directory, there is a folder called *evaluation/metrics*. Inside, evaluation scripts are located for each algorithm. Inside the scripts, a comment should point to the location where the duplicates inside the model can be determined to be omitted or kept in. Run the evaluation as follows:

```
python3 algorithm_eval.py /path/to/model/data/training.csv
```

This prints the metrics in the terminal, in both weighted and unweighted forms.

Appendix B

Contents of the CD

The CD contains the source code for DDoSGrid and its performance evaluation scripts, A sample model to be used for the evaluation, as well as the original data sets that make up the model