University of Zurich[UZH]

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

DDoSGrid 3.0: Enabling the Real-time Processing and Analysis of Cyber Attacks Traffic

Sandro Padovan, Michael Nadig, Christian Birchler
Zurich, Switzerland
Student ID: 17-721-291, 16-918-690, 15-924-160

Supervisor: Dr. Bruno Rodrigues, Muriel Franco, Jan von der Assen, Prof. Dr. Burkhard Stiller
Date of Submission: February 18, 2022

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

ifi

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

Abstract

Die Technologie als Ganzes hat sich rasant entwickelt und die Welt, in der wir leben, verändert. Alltägliche Gegenstände werden mit dem Internet verbunden und ermöglichen es uns, neue Dinge zu tun. Dieser Anstieg der Nutzerzahlen hat auch zu einem starken Anstieg von DDoS-Angriffen geführt im Laufe der Jahre, was wohl auch in der Zukunft weiterhin der Fall sein wird. Um DDoS-Angriffe besser zu verstehen und zu analysieren, wurde DDoSGrid entwickelt.

Dieses Projekt enthält die Arbeit, die geleistet wurde, um eine Live-Analyse für DDoSGrid zu ermöglichen, was diesen Bericht und auch den im Programm geschriebenen Code umfasst. DDoSGrid, wie es vor unserer Beteiligung war, ermöglichte die Analyse von sogenannten PCAP-Dateien. Attribute dieser PCAP-Dateien können extrahiert und visualisiert werden, um besser zu verstehen, was zum Beispiel während eines DDoS-Angriffs passiert ist. Mit unserer Arbeit erweitern wir diese Funktionalitäten, um auch eine Live-Analyse des laufenden Datenverkehrs zu ermöglichen. Dies ist auch aus der DDoS-Perspektive wichtig, da Live-Angriffe erkannt werden können. Das Tool kann aber auch zur allgemeinen Datenverkehrsanalyse verwendet werden. Ein weiteres Augenmerk wurde auf die zukünftige Erweiterbarkeit unseres Tools gelegt, um weitere Anwendungsmöglichkeiten sowie sonstige Erweiterungen zu ermöglichen. Die Anwendung wird auch anhand ihrer Fähigkeiten bewertet, da bei solchen groß angelegten Angriffen einige Engpässe auftreten könnten. Die Bewertung umfasst auch eine Fallstudie für einen Netzwerkadministrator, um zu verstehen und zu bewerten, wie wertvoll das Tool für einen solchen Benutzer sein könnte.

Technology as a whole has rapidly developed and changed the world we live in. Everyday items are becoming connected to the internet and enable us to do new things. This rise in users overall also gave a large rise to DDoS attacks which have strengthened over the years and will likely continue to do so. To understand DDoS attacks better and analyze them, DDoSGrid was developed.

This project contains the work which was done to enable live analysis for DDoSGrid, which includes this report and also the code written in the program itself. DDoSGrid, as it was before our involvement, enabled the in-depth analysis of so-called PCAP files. Attributes of those PCAP files can be extracted and visualized to better understand what has happened during a DDoS attack for example. With our work, we extend these functionalities to also enable a live analysis of ongoing traffic. This is also important from the DDoS perspective, as live attacks could be detected, but the tool can also be used for general traffic analysis. A focus was laid on the future extendibility our tool provides as the project continues to grow in the future. The application is then evaluated based on its capabilities as some bottlenecks could occur in such large-scale DDoS attacks. The evaluation also includes a case study for a network administrator, to understand and evaluate how valuable this tool could be to such a user.

Acknowledgments

This project required the help of multiple people to whom this chapter is dedicated. They helped improve the quality of this paper massively or even made its completion possible.

Firstly we would like to thank Prof. Dr. Burkhard Stiller who gave us the possibility to write our masters project in his department.

Further we have to thank Dr. Bruno Rodrigues, Muriel Franco and Jan von der Assen for their time in our biweekly meetings. They also spent their time outside of those meetings answering our questions, perfecting our report or helping with the coding part of the project.

Contents

Abstract **i**

Acknowledgments **iii**

1 Introduction **1**

1.1 Motivation 1

1.2 Thesis Goals 2

1.3 Methodology 3

1.4 Thesis Outline 3

2 Background **5**

2.1 DDoS Attacks 5

2.2 Network Monitoring 7

2.2.1 NetFlow 8

2.2.2 SFlow 9

2.2.3 IPFIX 9

3 Related Work **11**

3.1 Network Traffic Visualization for DDoS Detection 11

3.1.1 DDoSGrid 12

3.2 Machine Learning for DDoS Detection 12

3.3 Commercial Applications 13

4 Design **15**

4.1 UI Layer . . . 15

4.1.1 Web-based Interface . . . 18

4.2 Logical Layer . . . 18

4.2.1 REST API / WebSocket . . . 18

4.2.2 Live Data Manager . . . 18

4.2.3 Collector Connector . . . 19

4.2.4 Live Miners . . . 19

4.3 Data Layer . . . 20

4.3.1 Live Traffic Data . . . 20

5 Implementation **21**

5.1 Components . . . 21

5.1.1 Front-end . . . 21

5.1.2 API . . . 22

5.1.3 Data Manager . . . 23

5.1.4 Collector Connector . . . 24

5.1.5 Live Miners . . . 25

6 Evaluation **27**

6.1 Performance Evaluation . . . 27

6.1.1 Back-end . . . 27

6.1.2 Front-end . . . 29

6.2 Case Studies . . . 31

6.2.1 Network Administrator . . . 31

6.2.2 SYN-Flood Detection . . . 31

6.3 Limitations . . . 31

7 Final Considerations **33**

7.1 Summary 33

7.2 Conclusions 33

7.3 Future Work 35

Bibliography **35**

Abbreviations **41**

List of Figures **41**

List of Tables **43**

List of Listings **45**

A Installation Guidelines **49**

A.1 API 49

A.2 Front-end 49

A.3 Traffic simulation 49

A.3.1 PCAP player 50

A.3.2 NetFlow Exporter 50

B Source code **51**

Chapter 1

Introduction

Distributed Denial-of-Service (DDoS) attacks are a major threat to internet availability that remains unmitigated despite many commercial and research efforts. DDoS attacks happen every single day and, as society follows an increasing digitization trend, such attacks can pose many threats to businesses and individuals. A common reason is the large number of unsecured devices connected to the Internet and their growing processing capacity, which allows attackers to take control of a vast amount of unsecured devices that range from connected cameras to smart fridges to launch malicious attacks [28]. Many of these devices are insecure by design and not often they are impossible to be secured due to their hardware and software constraints.

The result of such a DDoS attack might be that the company website is overloaded with calls and shuts down. As the website is unavailable due to the excessive number of requests, customers cannot purchase products, and revenue might be lost. To prevent or reduce damages caused by these DDoS attacks, different detection and mitigation methods are available, whereas the detection of such a live attack is crucial to provide situational awareness to network security experts, and could help prevent further damage which might occur otherwise. Detection methods are mainly implemented based on dedicated ASIC-based (Application Specific Integrated Circuit) appliances to analyze flow records exported from software switches or edge routers, to enable further filtering or load balancing traffic.

1.1 Motivation

In previous work, the *DDoSGrid 1.0* platform was proposed by L. Boillat and J. von der Assen [7] to help the network operator detect DDoS attacks by providing visualized data representation of the network. The detection of DDoS attacks still requires manual investigation of network data. *DDoSGrid 1.0* takes as input the captured network data that has many dimensions so that it can visualize them to the network operator for analysis. This tool also allows the users to implement their own visualization methods that can easily be integrated into the system to fulfill the specific needs of users.

The implementation of *DDoSGrid 2.0* [4] combined the components of two different tools to visualize and share data of DDoS attacks. The visualization component of the application comes from the *DDoSGrid 1.0*[7], whereas the component to share and distribute the data of DDoS attacks comes from the DDoS Clearing House [11] project. This leads to the capability of sharing the data of DDoS attacks and providing different abstraction levels with visualizations for the users. These abstraction levels are important for interpretation since this part is still done manually by a human operator. The last updates moved *DDoSGrid* to become SecGrid [13] since new miners and features were implemented to support not only DDoS attacks analysis but also traffic from different cyberattacks (*e.g.,* Ransomware, Phishing, and other threats).

The previous implementations of *DDoSGrid* rely on post-mortem data, which means there is no real-time analysis capability integrated. *DDoSGrid 3.0* will add a further increment of the preliminary work with the goal of analyzing real-time data to determine whether a DDoS attack is happening. Data will be fetched live from the networking equipment and servers with the help of the third-party protocol NetFlow which collects data of packets (traffic of the IP-layer) and evaluates them. In this sense, it is essential to visualize and categorize the current, live traffic organized in rolling time windows, so that the operator has situational awareness and can make decisions regarding the mitigation of attacks that are duly substantiated. By analyzing the live traffic, conclusions can be drawn as to whether a DDoS attack is happening or not. This data will also be visualized live as it is already done currently for the past data for *DDoSGrid*.

1.2 Thesis Goals

Driven by the description of the work description, the following master project goals are stated:

- Development of miners for the processing and extraction of NetFlow traffic
- Selection and implementation of visualizations for real-time traffic, e.g., using the *DDoSGrid* templates or integrating visualizations provided by Grafana
- Full integration of real-time support with the *DDoSGrid 2.0*
- Evaluation of the performance and scalability of the new features, e.g., maximum throughput supported and possible bottlenecks during the real-time analysis

Furthermore, these goals may result in slightly different requirements, which may influence the activities scheduled. In this sense, premises could be considered in an agreement with the supervisor in order to restrict, change, or adapt activities to the scope of the project, and consequently, its objectives. Moreover, a scientific publication containing the results of the project might be produced, which should be accompanied by a project's website hosted within the CSG Web, such that ongoing activities and intermediate results are openly accessible. Thus, the development of the publication must be aligned with the project development.

1.3 Methodology

The first step of our project was to gain an insight into the entire subject which was done by in-depth literature research. After the research, we had to run the already existing *DDoSGrid* on our computers. The code also had to be analyzed thoroughly so that we were able to make the first design decisions and also a prototype sketch of what it will look like. After the design decisions were completed the important libraries were selected and our design was implemented as our first prototype. After some final iterations of our prototype, the evaluation of our prototype was the next step. In this part, we made some performance testing and also checked in a case study how well an actor could use the system. Lastly, we drew some final conclusions and finalized this report. During all the steps above, we were also writing this report in parallel.

1.4 Thesis Outline

This current chapter introduced the user to why the application was developed and also what is to be achieved by it. Following the introduction chapter, some important information is given in the background chapter to make the rest of the work easier to understand. For example, DDoS attacks are explained in detail or different kinds of network monitoring protocols. Chapter three then goes over the related work which has already been done in this field. Chapter four is dedicated to the design of the project which was implemented. Important design decisions are highlighted and also explained as to why they were made. In chapter five, the implementation, detailed insight is given into its implementation, and also some code examples are highlighted. In chapter six we make an evaluation, where the performance is evaluated, a case study is made, and also the project's limitations are discussed. In the last main chapter, the final considerations are made to round up this master project.

Chapter 2

Background

This chapter gives an introduction to the fundamental concepts on which this thesis is based. The section 2.1 covers DoS and DDoS attacks and their countermeasures, while the section 2.2 introduces network monitoring with available protocols being compared.

2.1 DDoS Attacks

Denial-of-Service (DoS) attacks are a broad class that disrupts service and prevents legitimate users from accessing an online service. The fact that resources such as network bandwidth, memory, or processing power are typically available according to the number of users that use the services daily, attackers aim at overloading the server's capacity to fulfill the goal of a DoS attack [12]. Distributed Denial-of-Service (DDoS) attacks are based on the same concept as DoS attacks but distributed. The attack is often carried out by a network of remotely controlled computers forming a botnet to scale up the attack and hide the attacker's identity [28]. These agent machines are recruited by the attacker and infected with the attack code. While in early DDoS attacks, the agent machines were recruited manually, today's recruitment process usually happens automatically by means of malware [21].

In the last decades, many different types of DDoS attacks have emerged. The attacks differ in the degree of automation, the communication mechanisms used, the weaknesses exploited, the protocol level, among others [21, 28]. In the following, some examples of types of DDoS attacks will be given and explained. Starting with application layer attacks, *HTTP-based attacks* exploit the HTTP protocol by sending large amounts of HTTP requests. On the transport layer, TCP and UDP are used for DDoS attacks. With TCP, *SYN Flooding* is an attack in which the victim is flooded with SYN requests to use up its resources. Another attack using TCP is *TCP Null Flooding*, in which packets do not have a TCP segment flag set, making the packet invalid. In *UDP Flooding*, the victim is flooded with usually large UDP packets, making the system unresponsive. On the internet layer, *ICMP Flooding* or *Ping Flooding* is an attack in which the attacker sends Internet Control Message Protocol (ICMP) Echo Request packets (ping). With the

Table 2.1: Overview of different DDoS attacks

Attack Name	Layer	Protocol	Description
HTTP-based	Application layer	HTTP	The victim is overwhelmed by a large amount of HTTP requests.
SYN Flooding	Transport layer	TCP	The victim is flooded with SYN requests.
TCP Null Flooding	Transport layer	TCP	TCP segment flag is missing, resulting in invalid packets.
UDP Flooding	Transport layer	UDP	The victim is flooded with large UDP packets.
Fraggle Attack	Transport layer	UDP	The victim is flooded with replies to spoofed UDP traffic from the attacker broadcasted to a network. Very similar to a Smurf Attack.
ICMP Flooding / Ping Flooding	Internet layer	ICMP	The victim is flooded with ICMP echo requests (pings).
Smurf Attack	Internet layer	ICMP	The victim is flooded with replies to ICMP Pings coming from the attacker. Very similar to a Fraggle Attack.
TearDrop Attack	Internet layer	TCP/IP	The attacker sends overlapping fragmented packets, resulting in the victim's device crashing.

victim sending reply packets to the pings, its resources are depleted. Similarly, in a *Smurf Attack*, the attacker broadcasts ICMP Echo Requests with the victim's spoofed IP source address to a network, resulting in the victim being overwhelmed with ICMP replies. The same concept applies in a *Fraggle attack*. However, instead of ICMP, UDP is used. A *TearDrop Attack* exploits a bug in TCP/IP fragmentation reassembly. The attacker sends overlapping fragmented packets, which result in the victim device crashing [1]. Table 2.1 provides an overview of the attacks mentioned above.

To take effective countermeasures against DDoS attacks, it is important to understand the details and nature of the network traffic. Network monitoring and visualization of traffic characteristics can support network administrators to this end. Generally, defense mechanisms against DDoS attacks can be classified by their deployment location or by the point in time when the countermeasure takes place. For network and transport level attacks, there are four different deployment locations: *Source-based*, *destination-based*, *network-based*, and *hybrid / distributed*. For application level attacks, only *destination-based* and *hybrid* apply. For classification by point in time, *before*, *during*, or *after the attack* can be distinguished.

Source-based defenses try to detect and limit IP spoofing from outgoing traffic (ingress/egress filtering), monitor traffic and compare it to a predefined normal state or with heuristics, or deploy a reverse firewall that limits the rate of forwarded packets that

do not reply. Nevertheless, source-based countermeasures alone are not efficient, mainly because attacks usually are distributed. Thus, they are difficult to detect, and legitimate traffic is difficult to distinguish from an attack. Additionally, the motivation is low since there is no financial incentive to invest in source-based defense mechanisms.

Destination-based mechanisms include IP traceback mechanisms to find the true sources rather than the spoofed address, analyzing Management Information Base (MIB) data which is comprised of statistical data on packets and routing to detect an attack. Further, packet marking and filtering is used to drop the undesirable packets as well as packet dropping to drop traffic based on congestion levels.

Network-based mechanisms try to detect a DDoS attack before it reaches the victim. Here, route-based packet filtering is used to find unexpected source addresses, or malicious routers are detected and filtered. In comparison to the above-mentioned defense mechanisms, hybrid mechanisms operate at multiple locations. For instance, an attack could be detected at the victim and a response initiated at routers in the network [28].

2.2 Network Monitoring

With computer networks increasing in complexity, the task of network administrators becomes more complicated as well. To provide a reliable service, the different aspects of the network (e.g., performance, reliability, availability) must be monitored to detect and correct malfunctions or security breaches as fast as possible. In this Section, several different approaches to network monitoring are explained.

In general, monitoring can be performed either actively or passively. The difference is that active monitoring modifies the traffic, whereas, in passive monitoring, the traffic can only be read. To monitor a network, traffic has to be duplicated such that the duplicated traffic can be analyzed without interfering with the regular traffic. There are different ways of mirroring traffic, such as port mirroring, Test Access Port (TAP), or a TAP-like setup that uses a Network Interface Card (NIC). To analyze the traffic, there are three different approaches:

1. **Packet capture**: the capturing of full network data, either stored to a file (usually a PCAP file) or analyzed directly.
2. **Deep Packet Inspection (DPI)**: refers to the automated analysis of collected packets' contents.
3. **Flow Observation**: packet headers are analyzed and aggregated into flows [26].

The advantage of packet capture is the unrestricted access to all network data; however, it does not scale well to speed and size. In DPI, however, the inspection of the packets can be based on a pattern matching method using regular expressions or an event-based analysis. Lastly, in flow observation, a flow can be defined as a set of IP packets passing an observation point during a certain interval, with packets having common properties such

as IP source and destination address, source and destination port, and transport protocol [2]. The payload of packets is not analyzed, resulting in less data to be processed, increased scalability and privacy [26].

2.2.1 NetFlow

NetFlow is a protocol for gathering network traffic data. The following explanation follows the protocol definition of RFC 3954 [8].

Cisco developed the NetFlow protocol. It is a service that provides network administrators access to IP flow information of a network. Network hardware like switches and routers gather data of the traffic and export it to collectors. The gathered data provides fine-grained metering for flexible and detailed resource usage accounting.

A unidirectional sequence of packets that share common properties is defined as a flow. The network traffic that passes through a network device is gathered and bundled to flows. These flows will then be exported to an external device, the NetFlow collector. Typically a flow can have IP addresses, packet and byte counts, timestamps, Type of Service (ToS), application ports, input and output interfaces, etc.

The applications of such NetFlow data are diverse. NetFlow can be used for enterprise accounting and departmental chargebacks, ISP billing, data warehousing, network monitoring, capacity planning, application monitoring and profiling, user monitoring and profiling, security analysis, and data mining for marketing purposes.

Templates determine the exported data of NetFlow. Templates provide access to observations of IP packet flows in a flexible and extensible manner. A template in general defines a collection of fields, with corresponding descriptions of structure and semantics.

There are multiple advantages a template-based approach provides. Firstly new fields can be added to NetFlow flow records without having to change the structure of the export record format. In previous NetFlow versions, the addition of a new field in the flow record also implied a new version of the export protocol format and a new version of the NetFlow collector that supported the parsing of the new export protocol format. This approach means it heavily simplifies adding new fields.

A further advantage the template-based approach provides is that templates that are sent to the NetFlow collector contain the structural information about the exported flow record fields; therefore, if the NetFlow collector does not understand the semantics of the new fields, the flow record can still be interpreted.

A last advantage highlighted is that because the template mechanism is flexible; it allows the export of only the required fields from the flows to the NetFlow collector. This then helps to reduce the exported flow data volume and provides possible memory savings for the exporter and NetFlow collector. Only sending the required information can also reduce the network load.

2.2.2 SFlow

sFlow is another technology used for monitoring traffic in data networks with switches and routers. sFlow stands for sampled flow. The following text follows the protocol definition of RFC 3176 [23].

sFlow defines in particular the sampling mechanisms which are implemented in an sFlow agent for monitoring the traffic, an sFlow MIB for controlling the agent, and the format of sample data that is to be used by the agent when the data is forwarded to a central data collector.

The sFlow monitoring system consists of the aforementioned sFlow agent which is embedded in a switch, router, or stand-alone probe and also a central data collector or sFlow analyzer. The sFlow agent uses sampling technology to capture traffic statistics of the device it is monitoring. sFlow datagrams are then used to forward this sampled traffic statistics to a sFlow analyzer for analysis.

The overall architecture and sampling techniques that are used in the sFlow monitoring system are made to provide continuous site-wide and also network-wide traffic monitoring destined for high-speed switched and routed networks.

The design is specifically addressing issues associated with:

- Accurate monitoring of network traffic at Gigabit speeds and higher.
- Scaling to manage tens of thousands of agents from a single point.
- Extremely low-cost agent implementation.

One can further read the specifications of sFlow in the memo [23] which describes the sFlow system. The previously shortly described sFlow agent, the sFlow MIB, sFlow analyzer, and the sFlow format are described there in greater detail. For this paper, the previously provided information is sufficient to draw some connections between the different network protocols.

2.2.3 IPFIX

IPFIX stands for IP flow information export. The following text follows the protocol definition of RFC 7011 [9] for IPFIX.

The IPFIX protocol serves as one means for transmitting traffic flow information over the network. Network administrators are provided with access to IP flow information. For traffic flow information to be transmitted from an exporting process to a collecting process, a common representation of flow data and a standard means of communicating them are required, which is the main goal of IPFIX.

IPFIX shares many similarities with NetFlow and is also sometimes called NetFlow v10. The main difference between NetFlow and IPFIX is the way the messages are structured.

With IPFIX users are able to heavily customize templates for IPFIX messages and also their length. Those messages are then pushed to receivers as part of data collection.

A template is an ordered sequence that is used to completely specify the structure and semantics of a particular set of information that needs to be communicated from an IPFIX device to a collector.

A collector is a device that is hosting the collection process. The IPFIX message is the message that originates from the exporting process and carries the IPFIX records of this process to the collecting process. The IPFIX messages are encapsulated at the transport layer.

We won't dive deeper into the large specifications of IPFIX [9] as more than a basic understanding is not required for this paper.

Chapter 3

Related Work

This chapter aims to relate this work to already carried out literary research by other authors, as well as the already existing *DDoSGrid*. As DDoS attacks carry heavy implications for companies involved there also exists a wide range of non-open sourced projects which are sold on the market. As these commercial products have to be purchased and are not publicly available only a quick overview is given.

3.1 Network Traffic Visualization for DDoS Detection

One of the first studies on visualization of network traffic was done by *Pearlman and Rheingans* [22]. The authors developed a visualization tool by approaching the problem from a service-oriented perspective. The visualization was a node graph with glyphs, which visually details the specific nodes in the network. Based on this approach, they could detect security (i.e., availability) issues in the network, such as DDoS attacks.

Over the past few decades, the networks grew rapidly in size and complexity, meaning that network infrastructures were increased in capacity to support higher bandwidths. Since the network traffic grew exponentially, a network operator cannot efficiently identify potential threats by observing static log files. A tool called *DDoSViewer* developed by *Zhang et al.* [29] was designed to help network operators identify DDoS attacks by displaying a visual representation of network traffic. They used the statistical model of the time-stamps rather than the packet analysis model.

One of the first real-time visualization approaches for DDoS attacks was studied by *Jin et al.* [17]. Their approach and framework are based on spark-streaming, which requires a spark cluster for fast network traffic analysis. They used Wireshark as a packet capture software and temporarily stored it as JSON files. This approach is different from the *DDoSGrid 3.0* since it is not using a cluster nor Wireshark for capturing network data.

There is also work that focuses on visualizing DDoS attacks based on IoT (Internet of Things) devices. For example, *Kaneko et al.* [18] used mixed reality technology for network packets visualization. One of their main focuses is identifying the specific devices

that behave like a bot for a DDoS attack, whereas our work is not limited to networks with IoT devices.

3.1.1 DDoSGrid

Identifying DDoS attacks based on network data is not a straightforward task. To help a network administrator identify possible DDoS attacks, a visualization of the captured data is essential to provide situational awareness. In this regard, the first open-source implementation called *DDoSGrid* was implemented by *Boillat and von der Assen* [7]. This tool uses captured data as PCAP files to extract features and visualize them on several grids in a browser-based application. Further, this version of *DDoSGrid* allows the user to write their own feature extractors and visualizations to fulfill the specific user's needs.

The second version, *DDoSGrid 2.0*, developed by *von der Assen* [4, 5], focused on integrating *the DDoS Clearing House* [11] project into the existing *DDoSGrid* implementation. *DDoS Clearing House* is a platform for sharing information and data about known DDoS attacks. With this data collection, researchers can investigate the various cyber-attacks over the network.

A further extension of *DDoSGrid* was done by *Boillat* [6]. This work was done in order to try and not only analyze but also classify DDoS attack traffic with the help of machine learning. A model can be made by manually classifying attack types that were previously recorded. With the help of this model new attacks can be automatically classified by the machine learning algorithm.

This master project describes the design and implementation of the third version of *DDoSGrid*, namely *DDoSGrid 3.0*. The third version herein proposed includes the real-time visualization feature of network traffic. With a real-time visualization approach, a network operator can detect potential DDoS attacks earlier than with the previous implementations of *DDoSGrid*. Early detection might reduce the damage of such an attack.

3.2 Machine Learning for DDoS Detection

Many works automatically detect DDoS attacks by profiling DDoS attack traces to differentiate them from regular traffic. These works differ from *DDoSGrid 3.0* concerning the provision of visualization and decision-making. In this regard, *DDoSGrid 3.0* aims at providing situational awareness to a network operator as a support to his decision-making instead of automating the decision on whether there is a DDoS attack in place. Thus, based on situational awareness, a network operator must decide if a scenario is a potential DDoS attack. Nonetheless, these related works might be interesting for future work to integrate into *DDoSGrid* in future versions to provided further insights or recommendations that could be investigated and confirmed by the network operator.

There are also techniques for detecting potential DDoS attacks automatically by using machine learning models. The study of *He et al.* [15] investigated the collinearity of the extracted features of the captured network traffic used in machine learning models. Based on the evaluation of the KDD CUP 99 dataset, they concluded that correlativity between the features of the captured data could accurately distinguish DDoS attacks from normal traffic.

Another approach to detect DDoS attacks is to use recurrence plots as proposed by *Kirichenko et al.* [19]. The authors transformed the captured network data into black-and-white images to classify them with a residual neural network model. Their results showed sufficiently high accuracy for detecting DDoS attacks in intrusion detection systems.

3.3 Commercial Applications

As the tools listed in this section are commercial applications and have to be purchased, no detailed insight can be given. Yet, as the prevention of DDoS attacks are a necessity for many large companies, it can also be lucrative to provide this service commercially as this section discusses.

Solarwinds [16] provides a wide array of products. One tool is the security event manager [24] which comes with an extensive UI to view the incoming traffic. Logs can also be gathered with hundreds of pre-built connectors. The logs can be parsed into readable data in order to investigate potential threats. A further offer is the possibility to mitigate DDoS attacks. A notification is sent to the responsible user when a DDoS attack is occurring. Automated responses to certain types of attacks can also be set. Possible responses are for example to change privileges, block IPs, disable accounts, block USB devices or kill applications. The tool has the goal of offering all features to non-programmers and seems to be one of the most widely used commercial applications for the purposes listed above.

A further tool is Arbor DDoS by Netscout [3]. The tool offers DDoS and cyber threat protection, network visibility and they claim it is backed by industry-leading threat intelligence. It has been under operation since 1999 and its aim is to help customers understand not only their own environment, but threat actors, their tools, behaviors, and campaigns on a global basis. They also offer a wide array of visualization tools to help understand the network traffic better and make it easier to understand.

Another fairly known tool is Cloudflare [10]. Cloudflare aims to protect and accelerate external, public-facing web properties while securing the internal operations on a single global network. Cloudflare's application is a lot wider than the previously mentioned solution and its application is not limited to DDoS attack mitigation. There are many more applications that offer DDoS protection services, however, Cloudflare is one of the most famous ones. As Cloudflare offers mainly protection from those attacks, its main focus does not lie on the visualization of attacks.

The focus of many of these commercial solutions like [3], [16] and [10] is mainly on threat detection and mitigation as opposed to our tool, where the visualization is the main point.

These commercial solutions are made to directly offer a solution to DDoS attacks in order to mitigate them while they are happening.

Chapter 4

Design

This chapter provides an overview of the architectural design and functioning of *DDoSGrid 3.0*, including the real-time capture, analysis, and visualization of traffic. Figure 4.1 shows the overall architecture of *DDoSGrid 3.0* with the extended components highlighted in the picture. Figure 4.2 displays a sequence diagram of the data analysis flow, and Figure 4.3 shows a mock-up of the proposed user interface. The following sections explain the different components of the system in detail and also some of the reasoning behind those design decisions. The UI layer is comprised of our front-end and the logical layer consists of multiple components which are part of our back-end.

4.1 UI Layer

The already existing User Interface (UI) is extended to include an additional tab called live analysis, as seen in our mock-up in Figure 4.3. This additional tab makes it easier to distinguish between live analysis visualizations and the visualizations from PCAP files in the tab dashboard. Another option would have been to include the live analysis visualizations in the already existing dashboard tab. We decided against this, as segmentation of live and non-live analysis is easier to understand for the user. If there were live and non-live visualizations combined, it would be confusing for the user, as one is updating in real-time and no updates are happening to the other visualization. It also makes sense from a logical point-of-view as in the PCAP analysis past files are analyzed and in a live visualization, the data is updated in real-time.

In the live analysis tab, each live visualization displays another data point. For each analysis, there exists a separate miner in the back-end. The design is made such that new visualizations can be easily added depending on the need. As a prototype, we include three visualizations that can be used to evaluate the system. The type of visualization needed depends on the need and the data, as some data might be more useful to be displayed in a pie chart, while others might require a line graph to be useful. The specific miners we implemented are further described in 4.2.4.

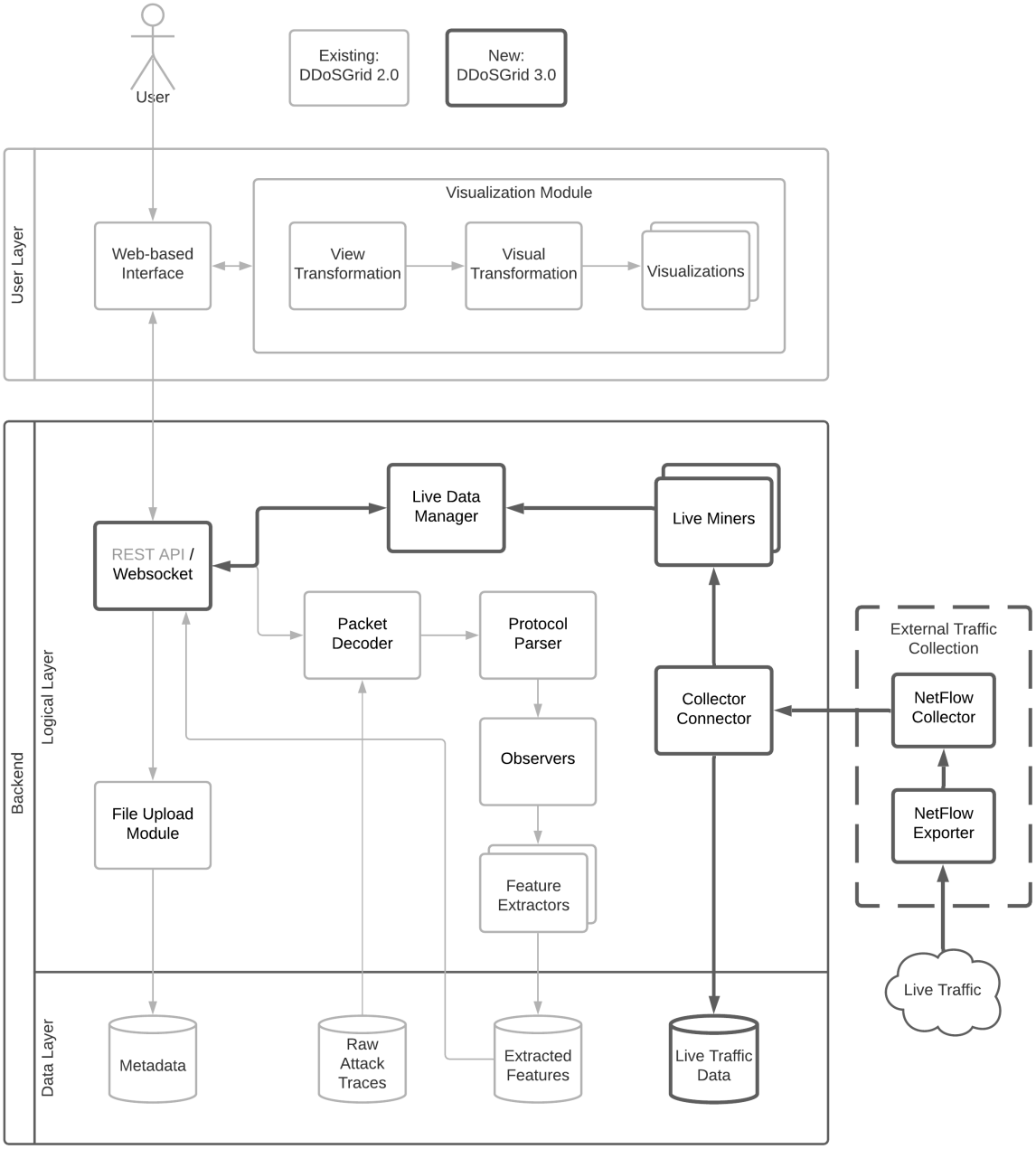


Figure 4.1: Architecture with added components highlighted

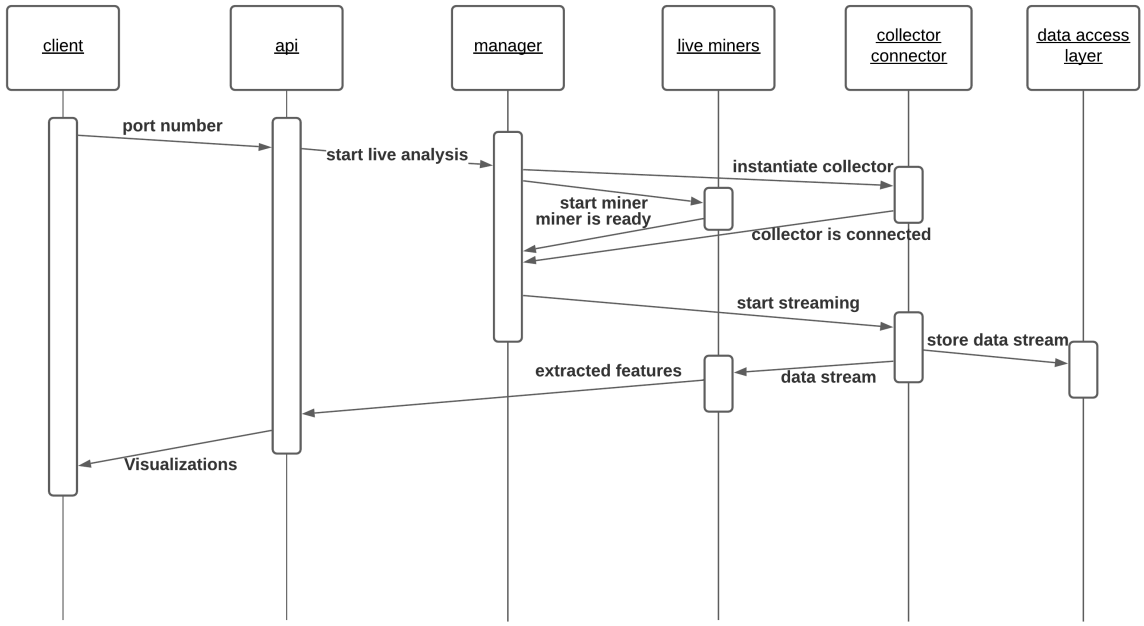


Figure 4.2: Sequence Diagram

Figure 4.3: Live analysis mock-up

A further visual addition is made to the data sources tab. There is now the option to plug in a current live connection. A port has to be specified in order to parametrize the collector in our application. This adds the possibility to then add the live visualizations to the live analysis tab. Thus, a user is able to select any port in order to monitor the port.

4.1.1 Web-based Interface

A *WebSocket* endpoint is implemented to continuously receive data each second to update the live diagrams. The received data is stateless and adds additional data points to the graphs while keeping the old data points. A *WebSocket* endpoint was added as opposed to a regular REST endpoint for various reasons which are discussed in more detail in 4.2.1. The updates are received from the back-end through the *WebSocket* endpoint and continuously added to the graphs.

4.2 Logical Layer

4.2.1 REST API / WebSocket

The existing RESTful API is extended with a *WebSocket* connection to allow for real-time data transfer from the server to the client. *WebSocket* is a protocol enabling two-way communication between a server and a client using a single TCP connection. In comparison, with a traditional REST endpoint, a connection is always requested by the client.

WebSockets have several benefits in use cases where two-way communication or real-time services are needed [20]. In the case of *DDoSGrid 3.0*, a *WebSocket* connection is useful as live traffic data is collected, analyzed, and sent to a client as a continuous data stream. The server pushes are needed in the two-way communication in our case, as our back-end has to continuously send data without the front-end having to request it all the time. Without a *WebSocket* endpoint and with a traditional REST endpoint, the front-end would have to continuously request updates each second from the back-end. With the enabled two-way communication, this is not needed and the back-end can push the updates without requests to the front-end.

4.2.2 Live Data Manager

To factorize the logic out of the API component, we introduce a *Live Data Manager*. This manager is in charge of instantiating the live miners and also the connector component that connects to the NetFlow collector. Our reasoning for adding an additional component to do that, is because of the single responsibility principle, which we adhere to. Giving the responsibility of setting up both the live miners and the connector, in our opinion,

should not be done by other components, as we want to limit the responsibility of each component to its purpose. Furthermore, the addition of more components is simplified as it just has to be now initiated and managed in the *Live Data Manager*. It is a requirement of our project to make the application as easily extensible as possible.

4.2.3 Collector Connector

As we need a connection to a NetFlow exporter, it makes sense to integrate an interface to handle the connection. For this purpose, we introduce the *Collector Connector* component. This component deals with different kinds of connections and protocols while giving a simple interface to it. Furthermore, the connector also forwards traffic into the data layer by saving the files.

In the future, it is fairly straightforward to include further collectors if one wants to use a different exporter instead of the NetFlow exporter in use. A future collector which isn't NetFlow based can be implemented in a new class and then just has to be added in the *Live Data Manager* to be instantiated. We also limit the responsibility of the connector component just to its simple purpose of enabling the collection of flows.

4.2.4 Live Miners

The *Live Miners* are a set of components analyzing and extracting features from the collected live traffic data. Further, the miner components process the data for visualization. An example for simple data processing is the extraction of the size of each packet and adding additional information like the visualization type needed to display this data for the front-end. Once the data has been processed, it is forwarded to the *Live Traffic Manager*. The set of miners is designed for an easy extension such that new kinds of features can be extracted with the addition of new miners. The easy extension is possible because of our *Live Data Manager* design where a new miner can just be added without any further additions. We are only constrained in what type of data we are using in the miners by what the collector is collecting from the NetFlow exporter.

The first three miners designed are for the byte count, the packet count, and the SYN flag count. The byte count miner is responsible for extracting the number of bytes that are sent. Each miner aggregates the data they collect for a second before they are forwarded. The same goes for the packet count miner, which adds up all the packets which are received and then forwards them. The SYN flag count miner is intended to to count the flows that are TCP SYN traffic. A SYN traffic has a predefined flag enabled that can be counted to identify a possible SYN flood attack.

4.3 Data Layer

4.3.1 Live Traffic Data

Since we collect live data, it makes sense to store it for additional post-mortem analysis. Thus, the data collected by the collector is saved in log files and acts for the sake of simplicity as a *Live Traffic Data* recorder in the data layer. This way the data is available for future analysis and evaluation of our real-time visualizations.

A further possibility is that the user records the data himself by using *tcpreplay* [27] to store the data as a PCAP file directly. Doing it this way, makes it possible to replay the traffic, while also making changes to the traffic.

Chapter 5

Implementation

The previous chapter highlighted the design decisions made before any implementation was done. This chapter focuses on implementing the prototype, highlighting its challenges, choices made during this process, and changes that had to be made in contrast to the previous chapter.

5.1 Components

The focus is on the components to which we have made meaningful changes; the front-end, the API, the data manager, the collector connector, and the live miners, as highlighted in Figure 4.1 of the design chapter. In the implementation of our components we aim to consistently adhere to the SOLID design pattern principles.

5.1.1 Front-end

Many capabilities from the front-end were implemented in the previous versions of *DDoSGrid*, *i.e.,* 2.0. We also continued to use Vue.js in the front-end to be consistent with the previous version of *DDoSGrid*. The front-end implementation consists of the inclusion of specific graphs as the one seen 5.1, which better support real-time and over-time data visualizations. A bar graph or pie chart is not helpful for most of our over-time visualizations, including a timeline. Thus, we are extending different visualizations to support the presentation of data over time better.

Initially, Grafana [14] was the tool proposed to do additional graphs over time. However, Grafana proved to be not a straightforward solution for our project, as it included many capabilities which were of no use for us. Thus, creating additional overhead. For example, we do not need to implement an entirely new dashboard with fairly fancy visualizations. Therefore, we settled with the Vue-chart-js library as it was already used in previous versions, which simplified our task of extending them. This makes extending our current prototype easy and uses graphs implemented from *DDoSGrid 2.0* like simple pie charts.

Figure 5.1: Front-end view with little traffic

As our data is real-time, we must keep the graphs updated and cache the previous data. Due to the choice to make the graphs with Vue-chart-js, we can easily enable caching in the front-end as it offers simple updates with the ability to add data over time and adapt the charts. Data in JSON format is sent from our *WebSocket* every second from the back-end to the front-end, which directly has the visualization type and the data for the graph. The graph will be updated as more data from the *WebSocket* is continuously sent in real-time. No processing is done to the data in the front-end to limit its responsibility.

5.1.2 API

To enable a real-time data connection, we added a *WebSocket* endpoint to handle the live traffic. We defined a *WebSocket* endpoint, because the current REST endpoints did not offer sufficient real-time support, which is needed for us to enable real-time data flow. Whenever a flow or data point is reported from the collector, the *WebSocket* initially pushed it to the front-end. In the final stages of development, we chose to not push every single data point directly to the front-end. Especially in case of a DDoS attack, the amount of data points sent to the front-end was enormously large and introduced large delays in the visualizations. Further, the whole web application was rather unresponsive. To resolve this, the data is now aggregated in the miners and pushed only every second to the front-end for better performance. This made the front-end independent of the incoming network traffic, whether there is no traffic at all, normal traffic, or a large scale DDoS attack. In any case, the user experience is not impaired. Although with the aggregation the data is not visualized immediately, the application feels more real-time to an end user, as there are no bottlenecks on the client side. With the initial REST endpoints from *DDoSGrid 2.0*, this is was not possible, as the front-end is the one sending the request to the back-end for an answer. Whereas we want a continuous connection enabling the back-end to

send data to the front-end whenever needed.

```
socketBroadcaster.on('newData', (data) => {
  sockets.forEach((socket) => {
    socket.emit('newData', data)
  })
})

class SocketHandler {

  onNewSocketConnection(socket) {
    sockets.add(socket)

    socket.on('disconnect', () => {
      sockets.delete(socket)
    })
  }
}
```

Listing 5.1: WebSocket class

Every time the connector gets a new flow, the live miners extract data and aggregate the data for one second. Then, the *WebSocket* endpoint sends the aggregated data from the back-end to the front-end in order to be displayed in a graph. The exact order that the data goes through with every component is highlighted in the design section.

5.1.3 Data Manager

The data manager, as the name suggests, is there to manage the miners as well as the collector. It acts as a manager by initializing both the connector and the miner. A connection is established between those to enable direct data transfer from the collector to the miner. The two functions shown below are part of the data manager and for used for this exact purpose.

```
 establishConnection(port) {
    this.setupCollector(port)
    this.setupMiners()
    this.startStreaming()
  }

  setupMiners() {
    const byteMiner = new ByteCountLiveMiner(this.dataBroadcaster, '
       ByteCount')
    const packetMiner = new PacketCountLiveMiner(this.dataBroadcaster, '
       PacketCount')
    const synCountLiveMiner = new SYNCountLiveMiner(this.dataBroadcaster
       , 'SYNCount')
    this.miners = [byteMiner, packetMiner, synCountLiveMiner]
  }
```

Listing 5.2: Data manager methods

We found that it would be easiest, when the collector sends the data directly to the miner after the initial setup, in order to limit the data flow to the essential components. After

consideration we found this to be the smoothest and also best architecture design to have the collector send the new data always to the miner directly after the initialization has been made of both by the manager. This decision was made to reduce the responsibility and data flow to only those two components. The data is, therefore, not processed in any way by the manager. We limit its responsibility and by that follow the single responsibility principle.

5.1.4 Collector Connector

There are multiple different protocols to export network traffic as it enters a network interface. In this regard, we considered as possible solutions either NetFlow or sFlow, which were already discussed in Chapter 2 of this paper. We settled for NetFlow, as described further in the design section, as it suited our needs best and has the largest libraries, which were being updated and used. Community support and available documentation were important aspects considered in such a decision and sFlow did not fit that criteria and also almost no libraries existed that suited our needs. As future scalability was a requirement for our work, we had to insist on good documentation for our tools used. Without good documentation, the future work on our prototype would be made significantly more difficult.

We implemented the collector with a specific NetFlow collector class shown in Listing 5.3. If one was to add additional collectors with different libraries or tools, it can easily be done by simply creating another class and using the data flow collected from the new collector class. Every time a new data point is received by the collector, a new event is emitted to notify the miner and the data is directly sent.

```
1  class NetflowCollector {
2      constructor(port_, dataBroadcaster) {
3          this.port = port_
4          this.dataBroadcaster = dataBroadcaster
5          this.collector = netflowCollector({ port: this.port })
6      }
7      start() {
8          console.log('netflowcollector: start')
9          this.collector.on("data", (data) => {
10             console.log('collector emits data')
11             this.dataBroadcaster.emit('data', data)
12         })
13     }
14
15     stop() {
16         console.log('closing collector connection.')
17         this.collector.server.close()
18     }
19 }
```

Listing 5.3: Netflow collector class

A challenge in this sense was that it was desired to test our project by replaying PCAP files. This was important also for the evaluation to consistently be able to replay the same

data stream. *DDoSGrid 2.0* is mainly built around PCAP files and there are already a lot of PCAP files of DDoS attacks which were collected from the previous DDoSGrid version, which we wanted to utilize. The tool we used in order to replay the files themselves was *tcpreplay* [27]. There are many adjustments one can make, like adjusting the speed with which the files are replayed and the packets are sent. The traffic replayed from the PCAP files then has to be streamed to a NetFlow exporter. The tool we used for this is *softflowd* [25]. This tool would also allow to directly replay PCAP files, however there are fairly limited replay abilities, which is why *tcpreplay* is better suited if one wants to simply replay the files. Due to the rather poor documentation of the *softflowd* tool, it took us some time to figure out the correct configuration to be able to use the tool as a NetFlow exporter, despite it not being part of our main development, as it is outside of our scope. The tool is also not very intuitive to use with many flags one needs to add in order to make it work as intended.

5.1.5 Live Miners

The live miners are an extension of the current miners of *DDoSGrid 2.0*. An adaption of the current miners to enable live data broadcasting would not be straightforward and would not have been a good choice from an architectural point of view. Almost every single line of code would have had to be adapted and would overcomplicate the already existing component. Also, the responsibility of both live miners and non-live miners is limited with this implementation, and makes them non-dependent on each other. The task they do is similar to the old miners, though. They get the raw data from the collectors and then, depending on the type of miner, provide a specific visualization based on the data they receive. The miners also aggregate the data each second before they are sent to the front-end. This aggregation had to be done in order to relieve the front-end and produce more consistent results in the visualization. This means each type of data that wants to be analyzed has a specific miner which then produces specific aggregated JSON data for the front-end to be displayed in a visualization.

The miners of *DDoSGrid 2.0* communicate with the rest of the components via Inter-Process Communication (IPC) as they are designed to be able to be run as a standalone application. The live miners that we implemented do not need to fulfill the same requirements and, therefore, cannot be run as a standalone application and the data manager directly initializes them. As the old miners only analyzed past data from one PCAP file, the data sent through IPC was fairly limited and they were also shut down once the miners were done. *DDoSGrid 3.0* live miners have to be continuously running after they were started as the traffic is coming live, and they cannot be shut down when they are done analyzing a single data file. The live miners are only shut down when the connection to the port is ended. In our system, it was also not a requirement to run the miners as standalone applications.

```
1 class PacketCountLiveMiner extends AbstractLiveMiner {
2   constructor (dataBroadcaster, type) {
3     super(dataBroadcaster, type)
4     this.logFileBaseName = 'PktCountMiner'
5   }
6
```

```
7   miningNetFlowPacket (data) {
8     console.log('mining total number packets')
9     const timeStamp = Date.now()
10    const flows = data.flows
11    let totInPackets = 0
12    for (const i in flows) {
13      totInPackets += flows[i].in_pkts
14    }
15    const res = { miner: 'PacketCount', aggData: totInPackets,
          timestampBeforeMiningFirstFlowPacket: timeStamp }
16    this.aggregateMinedData(res)
17  }
18 }
```

Listing 5.4: Example of live miner PacketCountLiveMiner

As a simple example, we see the *PacketCountLiveMiner*. The incoming data is processed according to the specific miner upon arrival and an event is emitted each second with the processed data.

Chapter 6

Evaluation

6.1 Performance Evaluation

The main performance analysis was assessed for the back-end with the miners. Since we aggregate data of several flows together for the visualizations already in the back-end, there is no need to do time benchmarking in the front-end, because in the front-end no data is processed, but only visualized. The back-end pushes the data that needs to be visualized in a constant interval of one second and then the front-end visualizes the data in this interval of one second.

The performance analysis of *DDoSGrid 3.0* in the back-end was assessed by evaluating timestamps of the processed data at different stages. Performance is measured considering the time difference between the timestamps. The smaller the differences between the timestamps the better is the performance. On the other hand, the performance of the front-end is evaluated in terms of looking at specific use cases. The visualizations should demonstrate how an attack can be identified in real-time from a different perspective.

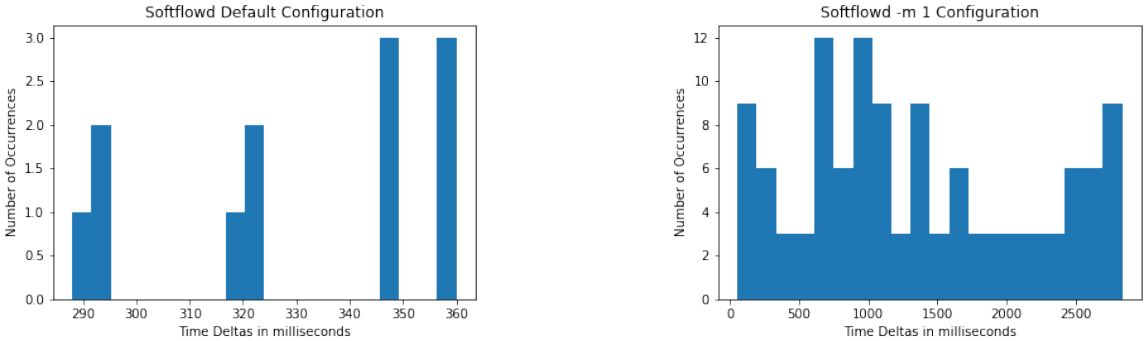
Performance analysis is organized into two sections. In Section 6.1.2, we analyze the real-time visualization performance of the front-end's perspective. Whereas in Section 6.1.1, we assess the performance of the back-end, which includes mining flow data, aggregation, and forwarding to the front-end over the *WebSocket* protocol.

6.1.1 Back-end

The performance analysis of the back-end is assessed by evaluating timestamps at different stages where a flow packet is processed. At the first stage, we log the timestamp is when a flow packet arrives at a miner. The timestamp is accessed with the *JavaScript* function `Date.now()`, which returns the number of milliseconds since 01.01.1970 00:00:00 UTC. The second stage where the timestamp is logged is before emitting the aggregated data to the socket broadcaster that sends the data to the front-end with the *WebSocket* protocol.

The evaluation was done with two different configurations of the *NetFlow* exporter *softflowd* regarding the number of maximum flows that need to be processed per packet. A flow packet is sent by the *NetFlow* exporter (*e.g., softflowd*), which contains several data fields. One data field is an array of flows that is collected in the network. Depending on the configuration of *softflowd*, we can set an upper bound for the number of flows for this data field.

Based on the default configuration of *softflowd*, that set the upper bound for flows to be collected for a single flow packet to 8,192 flows. We observed, as shown in Figure 6.1a, that the time deltas vary between 250 milliseconds to 360 milliseconds.



(a) Softflowd default configuration

(b) Softflowd -m 1 Configuration

Figure 6.1: Time deltas of processed flow packets in DDoSGrid 3.0

In case the upper bound of flows in a flow packet is restricted to one, then we see a broader distribution of the time deltas as shown in Figure 6.1b. We also see that the distribution of the time deltas ranges from 300 milliseconds up to 3,000 milliseconds.

The Table 6.1 shows the summary statistics of the collected time deltas. Based on the default configuration, which means that the maximum number of flows for a flow packet is set to 8,192, we have only 12 collected time deltas, whereas with the `-m 1` option for *softflowd* we have 117 flow packets. We observe that the mean, median, and standard deviation for the default configuration are lower than with the `-m 1` configuration of *softflowd*. In summary, considering the customized configuration of *softflowd*, we have a broader distribution of the time deltas, which indicates a varying processing time of flow packets by the miners.

	Configuration	
	Default	*-m 1*
Sample size	12	117
Standard Deviation	25.696	826.134
Mean	329.833	1342.504
Median	334.0	1159.0

Table 6.1: Summary statistics of the time deltas

While trying to find bottlenecks in our system by attempting to overload it we experienced a bottleneck, however not in our implementation, but in the tool we used to replay the recorded PCAP files. The CPU in our testing machine was used to full capacity by the `tcpreplay` processes when replaying multiple PCAP files simultaneously. This made it hard to draw meaningful conclusions about the maximum throughput. During our stress testing, our application never seemed to show any signs of a bottleneck or reach a limit of supported throughput.

6.1.2 Front-end

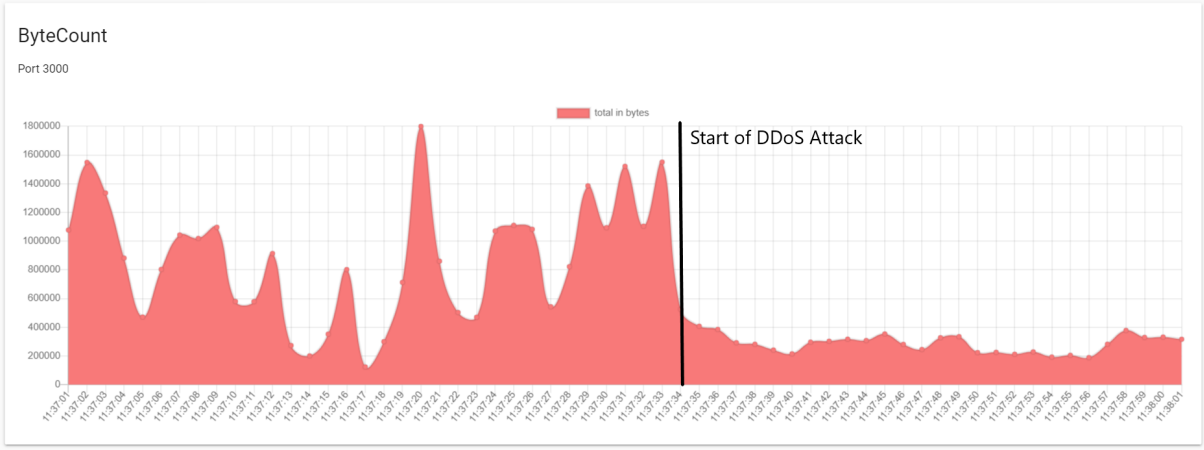
The performance of the front-end is evaluated by assessing three real-time visualizations of three miners concurrently. There is no actual need for assessing timestamps in this case since the data that need to be visualized is aggregated in the back-end and rendered in the front-end for every second. The visualization with this approach did not present performance variations nor delays, which leads to a good perception of the graphs for the end user. The computational load of the front-end is completely independent of the network traffic, since the data is aggregated on the server side and sent to the client every second. Thus, there are no bottlenecks in the front-end in case of a DDoS attack.

We replayed several captured network traffic recordings (stored as PCAP files) with `tcpreplay`[1]. The replayed network traffic is on one hand a "normal" base traffic that should serve as a baseline. On the other hand we have a collection of several PCAP files that captured real attacks from the past (*e.g.,* SYN-flooding attacks).

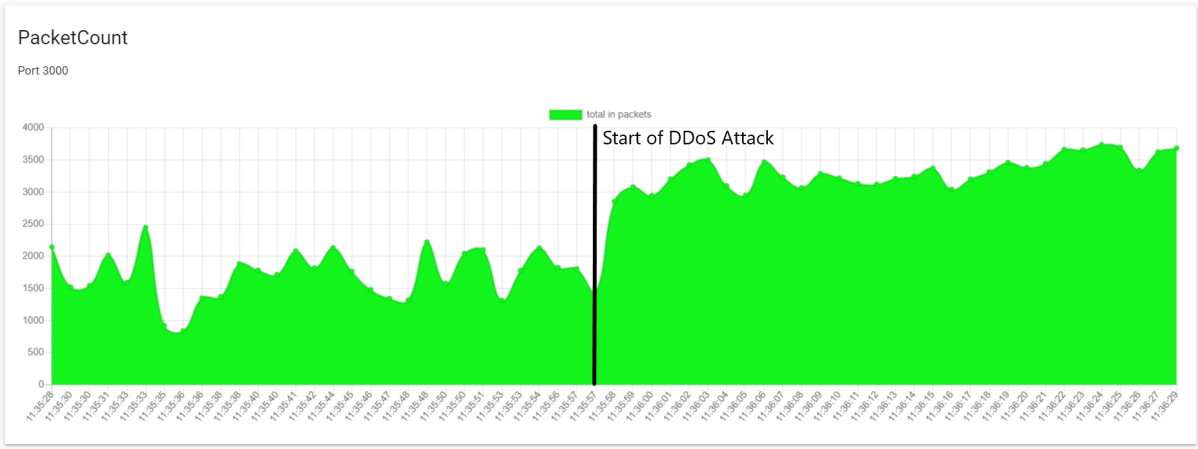
The Figure 6.2 shows how the start of replaying a SYN-flooding attack influence the real-time visualizations. For the *ByteCount* miner in Figure 6.2a, we note that at the start of the attack, a significantly lower number of total bytes are processed by the miners, which also indicates that there are not many more bytes transmitted in the network. However, a different picture shows the Figure 6.2b, where the number of packets significantly increased by starting the attack. This makes sense in terms of that flooding the network with SYNCs means also that there must be more packets transmitted but not necessary more bytes as shown in Figure 6.2a. The number of TCP SYN flags seems also to increase, as shown in Figure 6.2c, if the network is flooded with many similar packets, namely SYNC packets. From the visualization, we clearly see that mining the TCP SYN flags is useful to detect an SYN flood attack by *DDoSGrid 3.0*. Before the attack, we have a "normal" network traffic with a broader distribution of diverse packets, which we also could observe in the *SYNCount* visualization.

In summary, we identify visually a change in the network such as a sudden increase of packets within a time frame when a SYN-flooding attack happens. This visualization will make it more intuitive to identify a potential threat of DDoS attacks.

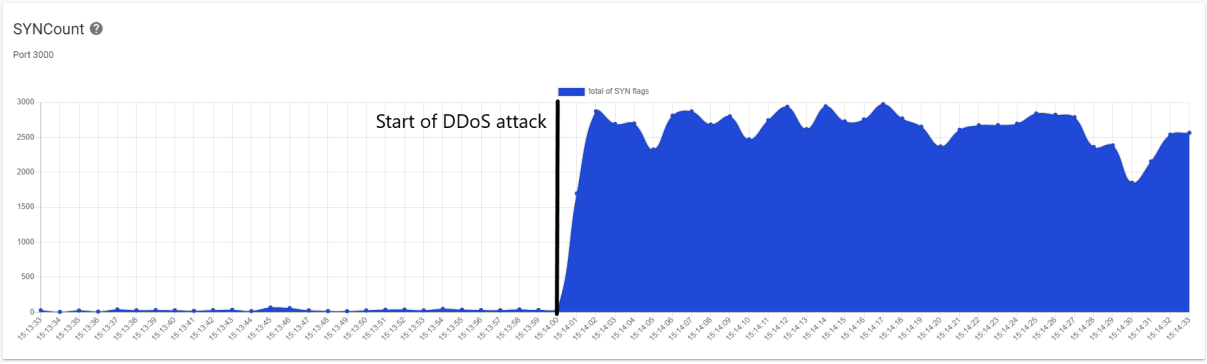
[1] `https://tcpreplay.appneta.com/`



(a) Total number of bytes processed by the miners



(b) Total number of packets processed by the miners



(c) Total number of SYN TCP flags

Figure 6.2: Real-time visualizations during a SYN-flooding attack

6.2 Case Studies

The capability of *DDoSGrid 3.0* to visualize network data specified by the *NetFlow* protocol enables a new way to detect attacks instantly due to the real-time visualizations of certain traffic properties, *i.e.,* miners. These properties can be used in different use cases, such as in the role of a network administrator who needs to protect the network of his organization instantly when an attack occurs. Another concrete use case is to identify a certain attack, such as an SYN-flood attack. There is the need to identify also this very specific kind of attack. The Section 6.2.1 shows how *DDoSGrid 3.0* can be used from the perspective of a network administrator, whereas in Section 6.2.2 we elaborate a more concrete use case of *DDoSGrid 3.0* by identifying SYN-flood attacks in real-time.

6.2.1 Network Administrator

From the viewpoint of a network administrator in a large organization, analyzing the network traffic enhances the attack identification process. Since the visualizations are in real-time, the network administrator can immediately take countermeasures to prevent further damage caused by the attack.

Of course, a large organization might be also interested in the details of the attacks. For this purpose *DDoSGrid 3.0* can also log the data, *i.e.,* mined data and flows. This stored data can be analyzed afterward in case there is not yet a concrete miner implemented for a specific visualization of a property. Another way to analyze an attack post-mortem is to use the capabilities and features of *DDoSGrid 2.0*. That version of *DDoSGrid* is developed to analyze captured network traffic, *e.g.,* as PCAP files.

6.2.2 SYN-Flood Detection

Also, concrete types of attacks want to be identified by a real-time visualization. For example, an Internet Service Provider (ISP) might be interested to detect SYN-flooding attacks in their backbones. For this purpose a miner that exclusively looks at the TCP flags of the flows provided by the *NetFlow* exporter might give a clear visualization to detect this very specific attack.

Like in Figure 6.2c, we observe how the graph changes when an SYN-flooding attack appears. Depending on the *NetFlow* version, different types of miners could be implemented in a modular, and extensible manner. This allows having more diverse concrete miners for different use cases.

6.3 Limitations

The main limitations of *DDoSGrid 3.0* is the fact that the implementation is based on *NetFlow* version 5. This specific version of *NetFlow* is still *de facto* standard, but the

disadvantage comes with its strict definition of data that can be extracted from a flow packet. For the next version of *DDoSGrid*, it might be worth to consider *NetFlow* version 9. This newer allows the user to define so-called "templates" for defining the data to be collected by the *NetFlow* collector or *NetFlow* exporter. This would give more flexibility and cover more possible use cases for *DDoSGrid*.

A possible bottleneck could be the update frequency of the visualizations. *DDoSGrid 3.0* is configured to visualize data in intervals of one second each. This relativizes the term "real-time" then it is actually a "near real-time" visualization. Depending on the needs of the user, the sampling rate for the visualizations might be too low but for the general use case, a sampling rate of one second should be sufficient.

We evaluated *DDoSGrid 3.0* with *softflowd* as a *NetFlow* exporter. *softflowd* is an open-source implementation but probably not commonly used in an industrial setting. Usually, routers or switches of a network are already shipped with the capability to export flows according to a *NetFlow* protocol. The evaluation on a real-life industrial setting would be worth considering in a follow-up work of *DDoSGrid 3.0*.

Chapter 7

Final Considerations

7.1 Summary

The main goal of this work was to enable real-time support for the project *DDoSGrid* of the University of Zurich. As the previous version already allowed for the visualization of PCAP files, the live analysis was the next logical step we had to tackle. To attain this goal, the creation of this report was a requirement on top of the implementation itself, as well as the design.

The first chapter of this report gives the reader an introduction to the subject. It highlights what we are trying to achieve and how. The requirements are also listed as well as the methodology. The thesis outline in the introductory chapter gives an overview of the structure of this written report. The introduction chapter is followed by the background chapter, where the necessary background knowledge is given to the reader. DDoS attacks overall are discussed and some basic knowledge is given, which is required to understand this project. We also dive into the different network monitoring protocols from which we had to make a choice at a later stage of our work. In the related work chapter, projects, which are closely related to what we did, are discussed. As the first chapter for our programming work itself, the design chapter was a key milestone for us. The design part lies the groundwork for the following implementation, where all the choices are further discussed as to how and why we chose to implement it. As the final chapter before this conclusion, we evaluated our solution in order to test our implementation. A small case study is also conducted and the project's limitations are discussed.

7.2 Conclusions

Overall we managed to fulfill all the requirements which were set. Here we provide a quick list of the fulfillment of those requirements as stated in Section 1.2.

- Multiple miners for processing and extracting NetFlow traffic are available. The implementation of these miners follows an extensible design so that new miners can

be added easily. In a similar way, the NetFlow collector provides a clear interface to allow for the integration of protocols other than NetFlow in the future.

- The real-time implementation was done by adding live miners. The visualizations now also better support real-time data by displaying the data in line graphs over time. The previous visualizations did not support a good display of data over time with bar charts. This implementation was also done in such a way to enable future extensibility for more visualizations.

- Full integration of real-time support with the *DDoSGrid 2.0* was done successfully to allow traffic to be seen in real-time by the user in visualizations. This was done in one second intervals.

- The evaluation was done with PCAP files of DDoS attacks to test our application. A bottleneck in the front-end was fixed in the early part of our evaluation to be able to process the attacks and not overload the browser in the front-end. A case study with a network administrator was also done and it shows that it could prove useful for such a user. One more miner was added towards the end of the timeline as well as the accompanying visualization which proved the easy extensibility of the project.

All of this however did not go seamlessly as there were multiple hick-ups along the way. An important part that we underestimated for example was the NetFlow exporter itself, which is not part of our implementation scope. The documentation was fairly limited and there is no feedback when something is done incorrectly. As there was simply no traffic shown in the collector it was hard to spot where the error was made.

As we specifically tried to replay PCAP files of DDoS attacks we also found a large bottleneck at first in our front-end. Those DDoS attacks are designed to overload the system, however, we quickly found a solution for that. We concluded that not each individual packet should be sent to the front-end by the back-end. Instead, the server now aggregates the data for one second before sending the data to the front-end. However, this meant that it would slightly impair the real-time aspect. Having said that, the maximum delay in such an implementation was still one second because of the aggregation, which is very low.

An additional challenge was working in a group and especially resolving scheduling issues. Finding a common date can be hard in a group of three and also the coordination can take up a big part. After all of this, we certainly learned a lot, whether that is coordination in the group, more knowledge in programming languages and frameworks used, or the specific network knowledge which was needed.

In summary, we can conclude that we made some important contributions to *DDoSGrid* by providing live analysis while each team member also learned a lot. We consider our project a success, because live DDoS attacks can now be detected by our tool in real-time.

7.3 Future Work

The future work can go into multiple directions. For example, many more miners could be added to provide multiple more visualizations and also different types of features to be tracked. Another collector could also be added in order to provide not only NetFlow support, but also support for other protocols for other use cases.

A further way the application could be extended is that new data sources could be integrated. For example, to be able to directly replay saved PCAP files from the *DDoSGrid* application to a port in the current live visualization. Currently, this has to be done externally through a PCAP file player and a NetFlow exporter.

Bibliography

[1] Saket Acharya and Namita Tiwari. "Survey Of DDoS Attacks Based On TCP / IP Protocol Vulnerabilities". In: *ISOR Journal of Computer Engineering* 18.3 (June 2016), pp. 68–76.

[2] Paul Aitken, Benoît Claise, and Brian Trammell. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011. Sept. 2013. URL: `https://rfc-editor.org/rfc/rfc7011.txt`.

[3] *Arbor DDoS Protection Solutions ¦ NETSCOUT*. https://www.netscout.com/arbor-ddos#.

[4] Jan von der Assen. "DDoSGrid 2.0: Integrating and Providing Visualizations for the European DDoS Clearing House". MA thesis. University of Zurich, Feb. 2021.

[5] Jan von der Assen et al. "Analysis and Classification of Cyberattack Traffic using the SecGrid Platform". In: *LCN 2021*, pp. 1–3.

[6] Luc Boillat. *DDoSGrid-Mining: Analyzing and Classifying DDoS Attack Traffic*. Feb. 2021.

[7] Luc Boillat and Jan von der Assen. *A Tool for Visualization and Analysis of Distributed Denial-of-Service (DDoS) Attacks*. Communication Systems Group, Department of Informatics, Sept. 2020. URL: `https://files.ifi.uzh.ch/CSG/staff/franco/extern/theses/MAP-Jan-Luc.pdf` (visited on July 1, 2021).

[8] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. RFC Editor, 2004.

[9] B. Claise, B. Trammell, and P. Aitken. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. STD 77. RFC Editor, 2013.

[10] *Cloudflare - The Web Performance & Security Company ¦ Cloudflare*. https://www.cloudflare.com/.

[11] *DDoS Clearing House*. `https://ddosdb.org/`.

[12] CERT Division. *Denial of Service Attacks*. Software Engineering Institute, Carnegie Mellon University, 2017. URL: `https://resources.sei.cmu.edu/asset_files/WhitePaper/1997_019_001_496601.pdf` (visited on Aug. 12, 2021).

[13] Muriel Franco et al. "SecGrid: A Visual System for the Analysis and ML-Based Classification of Cyberattack Traffic". In: *LCN 2021*. 2021, pp. 1–8.

[14] *Grafana: The open observability platform*. `https://grafana.com/`.

[15] Xiaowei He et al. "DDoS Detection Based on Second-Order Features and Machine Learning". In: *Trustworthy Computing and Services - International Conference, ISCTCS 2014, Beijing, China, November 28-29, 2014, Revised Selected Papers*. Ed.

by Yueming Lu, Xu Wu, and Xi Zhang. Vol. 520. Communications in Computer and Information Science. 2014, pp. 197–205.

[16] *IT Management Software & Remote Monitoring Tools ¦ SolarWinds*. https://www.solarwinds.com/.

[17] Yiqiao Jin et al. "A Real-Time Visualization Defense Framework for DDoS Attack". In: *Data Science - Third International Conference of Pioneering Computer Scientists, Engineers and Educators, ICPCSEE 2017, Changsha, China, September 22-24, 2017, Proceedings, Part I*. Ed. by Beiji Zou et al. Vol. 727. Communications in Computer and Information Science. 2017, pp. 341–351.

[18] Kosuke Kaneko et al. "PACKUARIUM: Network Packet Visualization Using Mixed Reality for Detecting Bot IoT Device of DDoS Attack". In: *Advances in Internet, Data and Web Technologies, The 8th International Conference on Emerging Internet, Data and Web Technologies, EIDWT 2020, Kitakyushu, Japan. 24-26 February 2020*. Ed. by Leonard Barolli, Yoshihiro Okada, and Flora Amato. Vol. 47. Lecture Notes on Data Engineering and Communications Technologies. 2020, pp. 361–372.

[19] Lyudmyla Kirichenko et al. "Machine Learning Detection of DDoS Attacks Based on Visualization of Recurrence Plots". In: *Proceedings of the International Workshop on Conflict Management in Global Information Networks (CMiGIN 2019) co-located with 1st International Conference on Cyber Hygiene and Conflict Management in Global Information Networks (CyberConf 2019), Lviv, Ukraine, November 29, 2019*. Ed. by Solomiia Fedushko et al. Vol. 2588. CEUR Workshop Proceedings. 2019, pp. 23–34.

[20] Alexey Melnikov and Ian Fette. *The WebSocket Protocol*. RFC 6455. Dec. 2011. URL: https://rfc-editor.org/rfc/rfc6455.txt.

[21] Jelena Mirkovic and Peter Reiher. "A taxonomy of DDoS attack and DDoS Defense mechanisms". In: *ACM SIGCOMM Computer Communication Review* 34 (May 2004).

[22] Jason Pearlman and Penny Rheingans. "Visualizing Network Security Events Using Compound Glyphs from a Service-Oriented Perspective". In: *4th International Workshop on Visualization for Computer Security, VizSEC 2007, Sacramento, CA, USA, October 29, 2007*. Ed. by John R. Goodall, Gregory J. Conti, and Kwan-Liu Ma. Mathematics and Visualization. 2007, pp. 131–146.

[23] P. Phaal, S. Panchen, and N. McKee. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. RFC 3176. RFC Editor, 2001.

[24] *Security Event Manager - View Event Logs Remotely ¦ SolarWinds*. https://www.solarwinds.com/security-event-manager.

[25] *Softflowd - A flow-based network traffic analyser capable of Cisco Netflow data export software*. https://github.com/irino/softflowd.

[26] Jakub Svoboda, Ibrahim Ghafir, and Vaclav Prenosil. "Network Monitoring Approaches: An Overview". In: *International Journal of Advances in Computer Networks and Its Security - IJCNS* 5 (Oct. 2015), pp. 88–93.

[27] *Tcpreplay - Pcap editing and replaying utilities*. https://tcpreplay.appneta.com/.

[28] Saman Taghavi Zargar, James Joshi, and David Tipper. "A Survey of Defense Mechanisms against Distributed Denial of Service (DDoS) flooding attacks". In: *IEEE communications surveys & tutorials* 15.4 (2013), pp. 2046–2069.

[29] Jiawan Zhang et al. “A Novel Visualization Method for Detecting DDoS Network Attacks”. In: *Visual Information Communication, selected papers from the Visual Information Communications International Conference, VINCI 2009, Sydney, NSW, Australia, September 2009*. Ed. by Mao Lin Huang, Quang Vinh Nguyen, and Kang Zhang. 2009, pp. 185–194.

Abbreviations

ABI	Application Binary Interface
AITF	Active Internet Traffic Filtering
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CIA	Confidentiality, Integrity and Availability
CSIRT	Computer Security Incident Response Team
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
DOTS	Distributed-Denial-of-Service Open Threat Signaling
DPI	Deep Packet Inspection
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process Communication
ISP	Internet Service Provider
JSON	JavaScript Object Notation
MIB	Management Information Base
NFV	Network Function Virtualization
P2P	Peer to Peer
REST	Representational State Transfer
RTT	Round Trip Time
SDN	Software-Defined Networking
SLA	Service Level Agreement
TCP	Transmission Control Protocol
ToS	Type of Service
UDP	User Datagram Protocol
UI	User Interface
VNF	Virtualized Network Function

List of Figures

4.1 Architecture with added components highlighted 16
4.2 Sequence Diagram 17
4.3 Live analysis mock-up 17

5.1 Front-end view with little traffic 22

6.1 Time deltas of processed flow packets in DDoSGrid 3.0 28
6.2 Real-time visualizations during a SYN-flooding attack 30

List of Tables

2.1 Overview of different DDoS attacks 6

6.1 Summary statistics of the time deltas 28

Listings

5.1 WebSocket class 23
5.2 Data manager methods 23
5.3 Netflow collector class 24
5.4 Example of live miner PacketCountLiveMiner 25

Appendix A

Installation Guidelines

This chapter is written to guide through the process of replaying PCAP files to simulate live traffic and then use DDoSGrid to analyze this live traffic.

In order to to get the entire project firstly, clone the project from the github repository:

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

A.1 API

It is necessary to install the dependencies of the miner subprocess as well, as the project is also able to use the functions of DDoSGrid 2.0. A requirement is to have Node.JS version 10 running and libpcap installed.

```
1 cd miner; npm i; cd ..;
2 cd api; npm i
```

To the run the api enter the following command:

```
1 -npm run dev
```

A.2 Front-end

The front-end installation is fairly easy aswell. The dependencies have to be fetched first, before it can be run:

```
1 npm i; npm run serve
```

A.3 Traffic simulation

In order to replay PCAP files and be able to pick up the packets with our NetFlow collector we need both a PCAP replayer as well as a NetFlow Exporter.

A.3.1 PCAP player

For the PCAP file replay we use tcpreplay [27]. Download the software at https://tcpreplay.appneta.com/wiki/installation.html and follow the installation guidelines depending on your operating system.

Running a PCAP file with the name file.pcap on interface wlo1 looks as follows:

```
1 sudo tcpreplay -i wlo1 file.pcap
```

The interface will depend on your operating system and how you are connected.

A.3.2 NetFlow Exporter

Now you still need a NetFlow exporter so that the netflow collector can pick up the packets. For that we use softflowd [25]. Head over to https://github.com/irino/softflowd to clone the project and follow its installation guidelines as well.

In order to now use the live NetFlow exporter on port 3000 with our wlo1 interface the following command is required:

```
1 sudo softflowd -i wlo1 -d -m 1 -n 0.0.0.0:3000
```

To enable real time exporting, the -d and -m 1 flags are required. The interface and also the port can of course be adapted according to preference.

Appendix B

Source code

All the source code can be found on the github repository https://github.com/ddosgrid/ddosgrid-v2/tree/flow-ddosgrid-v3.