University of Zurich^UZH

Emulator for Distributed DDoS Datasets (EDDD)

Calvin Falter
Zurich, Switzerland
Student ID: 17-708-934

Supervisor: Dr. Bruno Rodrigues, Chao Feng, Prof. Dr. Burkhard Stiller
Date of Submission: June 16, 2023

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

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
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
URL: https://www.csg.uzh.ch/

Zusammenfassung

Mit der rasanten Zunahme der Häufigkeit und Schwere von Distributed Denial of Service (DDoS)-Angriffen ist der Bedarf an robusten und effektiven Gegenmassnahmen von grösster Bedeutung. In dieser Arbeit wird ein neuartiger Ansatz zur Bewältigung dieses Problems durch die Entwicklung eines Emulator-Tools vorgestellt, das verteilte DDoS-Datensätze generiert. Dieses Tool geht auf die Einschränkungen bestehender, vorwiegend zentralisierter DDoS-Datensätze ein und bietet eine verteilte Perspektive, die entscheidende Einblicke in die Dynamik dieser Angriffe ermöglicht.

Der Emulator basiert auf der Open-Source-Flexibilität des Network Simulator 3 (NS3) und ist in der Lage, SYN-Flood-Verkehr, ICMP-Flood-Verkehr und legitimen Verkehr zu modellieren, jeweils auf der Grundlage bereits vorhandener Datensätze, wodurch die Vielfalt und der Realismus der simulierten DDoS-Szenarien erhöht werden. Das architektonische Design des Tools ermöglicht eine umfassende Konfiguration von Netzwerkstrukturen, die sich realistisch über mehrere Länder erstrecken können, was die Bandbreite der zu untersuchenden Angriffsszenarien deutlich erhöht. Das Tool liefert Ergebnisse im weit verbreiteten PCAP-Format und verfügt über eine unkomplizierte Befehlszeilenschnittstelle, so dass es sowohl für Forschungs- als auch für operative Anwendungen leicht zugänglich ist.

Im Wesentlichen stellt dieses Tool einen bedeutenden Fortschritt in der DDoS-Forschung dar und bildet eine solide Grundlage für künftige Erweiterungen. Es ist ein Zeichen für das Potenzial zur Verbesserung unseres Verständnisses und unserer Abwehrstrategien angesichts zunehmend komplexer und zerstörerischer DDoS-Angriffe. Die Einblicke in die Angriffsdynamik sind eine wertvolle Ergänzung zu den laufenden Bemühungen im Bereich der Netzwerksicherheit.

Abstract

With the rapid escalation in prevalence and severity of Distributed Denial of Service (DDoS) attacks, the need for robust and effective countermeasures has become paramount. This thesis presents a unique approach to tackling this issue through the development of an emulator tool that generates distributed DDoS datasets. Addressing the limitations of existing, predominantly centralized DDoS datasets, this tool provides a distributed perspective, offering critical insights into the dynamics of these attacks.

Built upon the open-source flexibility of Network Simulator 3 (NS3), the emulator is capable of modeling SYN flood traffic, ICMP flood traffic, and legitimate traffic, each one based on pre-existing datasets, thereby increasing the richness and realism of simulated DDoS scenarios. The tool's architectural design allows for comprehensive configuration of network structures that can realistically span multiple countries, significantly enhancing the range of attack scenarios that can be explored. Providing outputs in the widely used PCAP format and featuring a straightforward command-line interface, the tool is designed to be highly accessible for both research and deployed applications.

In essence, this tool constitutes a significant step forward in DDoS research, laying a solid foundation for future enhancements. It stands as a testament to the potential for improving our understanding and mitigation strategies in the face of increasingly complex and destructive DDoS attacks. The insights it offers into attack dynamics mark a valuable addition to the ongoing efforts in network security.

Acknowledgments

I would like to extend my deepest gratitude to my supervisor, Dr. Bruno Rodrigues, as he has been a great support to me throughout the duration of this thesis. His enthusiasm, has been instrumental in keeping me motivated and committed to the task at hand. His expert guidance and talent for providing me with insightful perspectives were of great help in bringing this project to life. This journey has been both challenging and rewarding, and I am very grateful for his supervision.

Additionally, my sincere thanks go out to Prof. Dr. Burkhard Stiller. By providing me with the opportunity to be a part of an intriguing project of the Communication Systems Group at the University of Zurich, he has enabled my exposure to a novel and enriching academic environment.

Contents

Abstract **i**

Acknowledgments **v**

1 Introduction **1**

1.1 Motivation 1

1.2 Thesis Goals 2

1.3 Methodology 3

1.4 Thesis Outline 4

2 Fundamentals **5**

2.1 Background 5

2.1.1 DDoS Attacks 5

2.1.2 Network Traffic Simulation 8

2.1.3 Network Topologies 8

2.2 Related Work 9

2.2.1 Attack Tools and Traffic Generators 10

2.2.2 Existing Datasets 11

2.2.3 Dataset Generation 12

3 Design **17**

3.1 Generation Approach 17

3.2 Architecture 18

3.3 Topology 19

3.4 Legitimate Traffic 22

3.5 Attack Traffic 27

3.5.1 ICMP Flood Attack 27

3.5.2 Syn Flood Attack 29

4 Implementation **33**

4.1 Configuration 34

4.2 Shell Script Controller 35

4.3 Simulator 38

4.4 Topology 42

4.4.1 Routers and Service Area 42

4.4.2 Placing Nodes 43

4.4.3 Connecting Nodes 44

4.4.4 Connecting to Sub-Networks 46

4.4.5 Connecting Countries 48

4.4.6 PCAP Nodes 50

4.5 Legitimate Traffic 52

4.6 Attack Traffic 55

4.6.1 ICMP Flood Traffic 55

4.6.2 Syn Flood Traffic 58

5 Evaluation **61**

5.1 Topology 64

5.2 Legitimate Traffic 67

5.3 Attack Traffic 69

5.3.1 ICMP Flood Attack 69

5.3.2 Syn Flood Attack 72

5.4 Distributed Attack View 74

5.5 Performance 78

5.5.1 Parallelism 79

5.5.2 Scalability 80

5.6 Discussion 82

6 Final Considerations **85**

6.1 Summary 85

6.2 Conclusions 86

6.3 Future Work 87

Bibliography **87**

Abbreviations **95**

List of Figures **95**

List of Tables **98**

List of Listings **99**

A Contents of the CD **103**

B Installation Guidelines **105**

Chapter 1

Introduction

Distributed Denial-of-Service (DDoS) attacks, while a longstanding threat in the cyber domain, have seen an alarming escalation in frequency and magnitude in recent years, thus highlighting their increased destructive potential. With shrinking intervals between new record-breaking attack intensities, DDoS threats demonstrate a swift and alarming evolution. For instance, the largest attack ever reported, as of February 2023, involved an astounding 71 million requests per second, surpassing the previous record set just eight months prior by a staggering 35% [20, 53]. This worrying trend not only indicates the persisting threat of DDoS attacks but also highlights their escalating danger to daily network operations.

The proliferation of botnets - responsible for a significant share of DDoS attacks - has been facilitated by the rise in poorly secured IoT devices. While the processing power of individual devices may be limited, their collective strength, exemplified by the 1.35 million bots detected from malware families like Mirai, Meris, and Dvinis in 2022, can inflict substantial damage [61].

DDoS attacks pose not just a technical but also a significant financial burden to businesses. The associated financial loss is influenced by numerous factors, including the size of the business, the industry in which it operates, and the specific services targeted by the attack. A study by Kaspersky Lab and B2B International indicated that the average cost of a DDoS attack in 2016 was approximately $106,000 for small businesses and over $1.6 million for larger enterprises. This figure spiked to $123,000 and $2.3 million respectively by 2017, revealing another distressing escalation [44]. Notably, there have been instances where a single DDoS attack has inflicted financial losses of up to $160 million on a business, demonstrating the catastrophic potential of these attacks [45].

1.1 Motivation

The prevalence and escalating severity of DDoS attacks necessitate the development of robust and adaptable countermeasures. Intrusion Detection Systems (IDS) constitute an essential barrier, intercepting and scrutinizing traffic before it reaches an Intrusion Prevention System (IPS). Given its sentinel role in identifying and alerting about suspicious or malignant traffic, the IDS is indispensable to the secure operation of any network [27].

To retain the effectiveness of an IDS, generating network traces that enable the assessment of IDS signatures and refining of the system in light of previously undetected cases is crucial [55]. IDS calibration typically relies on datasets that represent attack patterns [6]. However, the current datasets adhere to a victim-centric perspective, thus overlooking the broader landscape of the attack [1, 3]. This approach may suffice for profiling straightforward attack frameworks like volumetric attacks, but falls short in deciphering the sources of more complex orchestrations like botnets [5, 72].

The importance of a distributed perspective on an attack cannot be overstated: it illuminates the full scope of an attack, facilitates the discernment of traffic patterns, and provides a rough identification of the participating attackers. Moreover, observing the attack from various network vantage points allows one to trace its evolution, potentially enabling its interception and mitigation earlier, while the malicious packets are still scattered across the network and not yet concentrated on the target.

Despite the evident advantages, the distributed perspective has not been adopted yet. One primary challenge stems from the prevailing target-centric approach of today's defense strategies. This approach focuses on recording attacks at their target, resulting in a lack of real datasets providing a distributed view of an attack. Consequently, understanding how such attacks evolve across the network becomes challenging, hindering the creation of additional distributed datasets and the refinement of collaborative DDoS defense strategies.

Therefore, this thesis paves the path toward new approaches for evaluating DDoS defenses by enabling the creation of complex attack scenarios under a distributed viewpoint. This enables, for example, to analyze the influence of intermediate nodes within the attack, the geographical distribution of attack sources, and observe other traffic patterns for improving IDS systems.

1.2 Thesis Goals

This thesis aims to expand current DDoS datasets' limitations by establishing an emulator that proficiently creates distributed DDoS datasets. A concerted effort is made to architect and actualize a system that offers a distributed lens into DDoS attacks. This approach paves the way for a comprehensive understanding of attack dynamics and encourages the exploration of collaborative detection and mitigation strategies.

The main goal is the design and implementation of a DDoS dataset emulator. The envisioned system should be capable of generating DDoS datasets with a distributed perspective. It would allow the configuration of a multitude of attack parameters, network topologies, and attack scenarios to create diverse and realistic datasets. Subsequently, the thesis aims to lay the groundwork for better insight into DDoS attacks. This is achieved by capturing and analyzing attack data from various distributed perspectives. Considering the collaborative behavior of different entities in the network, the emulator aims to uncover valuable insights that might be overlooked in centralized datasets.

A vital aspect of the work involves comparing the emulator-generated datasets with existing centralized datasets. This comparative study helps evaluate and validate the emulator's output to assess the fidelity of the generated datasets, verifying the accuracy of the distributed perspective and ensuring the emulator captures key characteristics of real-world DDoS attacks.

Finally, a significant ambition of this master's thesis is to contribute to the field of DDoS research. By providing an emulator that enables the generation of distributed DDoS datasets, the work aims to enhance the understanding of DDoS attacks, facilitate the development and evaluation of collaborative detection and mitigation techniques, and ultimately contribute to network infrastructures' overall resilience and security.

Developing a robust and configurable emulator for generating distributed DDoS datasets is envisaged in achieving these goals. By enabling the exploration of DDoS attacks from a distributed viewpoint, the emulator can advance the understanding of attack dynamics and contribute to the evolution of effective defense mechanisms.

1.3 Methodology

This thesis adopts a systematic methodology to achieve the stated goals. The initial stage of this process involves a comprehensive examination of existing literature on the subject. This includes understanding the construction and operation of networks, familiarizing with the variety of DDoS attacks to be emulated, and comprehending key aspects associated with the creation of datasets.

Given the multifaceted nature of the task, it becomes pertinent to examine different methodologies for dataset creation. After scrutinizing the available strategies, an appropriate approach is selected that aligns best with the goals of this thesis. Subsequently, the requirements for the envisaged system are clearly defined. These requirements, rooted in the findings of existing research, serve as the blueprint for designing and implementing the DDoS dataset emulator.

The proposed prototype of an emulator is designed as a composite of interchangeable modules, each responsible for generating a specific type of network traffic. The structure of the emulator also includes a component dedicated to generating network topologies. The traffic generation in these components is modeled on existing datasets, ensuring the realism and relevance of the emulator's output.

In the final phase, the datasets generated by the emulator are meticulously compared with the datasets that served as their basis. This comparative analysis allows for a thorough evaluation of the emulator's output in terms of accuracy and fidelity. Additionally, the system's performance is evaluated in this stage. The evaluation parameters include but are not limited to, the system's efficiency and scalability. By examining these factors, the effectiveness and potential improvements of the emulator are thoroughly assessed.

1.4 Thesis Outline

This thesis is structured into six chapters. After the introduction comes Chapter 2 consisting of fundamentals on the topic. First, Section 2.1 elucidates indispensable concepts and provides requisite background knowledge. Thereafter, Section 2.2 serves to delineate related studies and elucidate existing solution strategies.

The third part, Chapter 3, is centered on explaining the solution design. This chapter presents a detailed exposition of the requirements guiding the design and the decisions made in its formulation. Building on this foundation, Chapter 4 further elaborates on the technical aspects of the solution, providing insights into the implementation process and illustrating the technology stack utilized.

Upon presenting the implemented solution, Chapter 5 follows with an evaluation and discourse on the final results. This chapter focuses on assessing the accuracy and performance of the solution. Concluding this work, Chapter 6 delivers a summary, highlights the limitations of the present solution, and offers directions for future research.

Chapter 2

Fundamentals

2.1 Background

The subsequent section aims to elucidate on essential terminology and attack approaches pertinent to DDoS, which form the foundation of this thesis. It also casts light on the concept and tools of network traffic simulation, a cornerstone of the investigation carried out in this thesis. Finally, it delves into the network topology approach adopted herein, contrasting it with its manifestation in real-world scenarios. This overview forms the backbone of the discussions and analyses in the subsequent chapters.

2.1.1 DDoS Attacks

A DDoS attack is a malicious cyber-attack in which the perpetrator seeks to disrupt the regular traffic of a targeted server, service, or network by overwhelming it with a flood of Internet traffic. Among the different types of Denial-of-Service (DoS), a DDoS is typically achieved by utilizing multiple compromised computer systems as sources of attack traffic. These exploited machines can include traditional computers and other networked resources, such as Internet of Things (IoT) devices, which are increasingly being used to amplify the scale and impact of DDoS attacks [22].

2.1.1.1 Botnets

One of the fundamental principles underlying the methodology of DDoS attacks is the use of botnets. A botnet refers to a collection of internet-connected devices, including personal computers, servers, mobile devices, and IoT devices, that have been compromised by malware and can be controlled remotely by a cybercriminal. These devices, or *bots*, can launch such an attack, amplifying the attack's scale and impact by leveraging the compromised devices' collective bandwidth and computational resources. As every bot is a legitimate internet device, distinguishing attack traffic from regular traffic can be difficult [22]. Attack tools using such botnets can employ one of two primary models: the

Agent-Handler or the IRC (Internet Relay Chat) model [5]. The Agent-Handler model is based on a master-slave relationship, where a central server, known as the *handler*, controls several *agent* computers used to launch the attack. This model allows for a high level of control and coordination, as the handler can direct the agents to target specific systems or networks. Alternatively, the IRC model utilizes public channels on an IRC network to launch attacks. In this model, the attackers use publicly available IRC channels to coordinate and launch attacks, making tracing the attack's origin more difficult. However, this model tends to be less centralized and coordinated than the Agent-Handler model.

Botnet owners resort to mimicking legitimate cyber behavior to perpetuate their botnets' existence and conceal their actions' malicious nature [56]. This approach allows them to operate undetected and evade detection by security systems. It highlights the increasing sophistication of botnets and the need for effective countermeasures to mitigate their impact.

2.1.1.2 IP Spoofing

IP spoofing is a technique to create Internet Protocol (IP) packets with a modified source address. This can hide the sender's identity, impersonate another computer system, or both. It is a core vulnerability exploited by many DDoS attacks as it makes it challenging to block malicious requests and track down the perpetrator of the attack. While IP spoofing cannot be prevented entirely, measures such as ingress filtering, outlined in BCP38 [2], can be implemented to stop spoofed packets from infiltrating a network. This involves examining incoming IP packets and rejecting those that do not match their origin or look suspicious [23].

2.1.1.3 Volumetric Attacks

Volumetric attacks are a subset of DDoS attacks in which the attacker tries to overwhelm the target network or website with much traffic to prevent legitimate users from accessing it. These types of attacks typically target the network layer of the target. The main goal of volumetric attacks is to overwhelm the target's network infrastructure, making it either slow or completely inaccessible. They achieve this goal by generating an immense volume of traffic that consumes a significant amount of bandwidth, thus limiting the available bandwidth for legitimate traffic. This massive influx of traffic comes from multiple sources, including botnets or compromised devices, making detecting and blocking the attack traffic challenging.

The taxonomy of volumetric attacks comprises various attack types, each with unique methods for overwhelming network systems.

In an Internet Control Message Protocol (ICMP) flood attack, an attacker floods a target network device with excessive ICMP echo-requests, hindering its operations [59]. IP/ICMP fragmentation attacks manipulate the IP fragmentation process by sending false fragments that resist defragmentation. This causes an accumulation in the receiver's temporary storage, ultimately leading to a denial of service [60]. A UDP flood attack overwhelms a target host by directing numerous IP packets carrying UDP datagrams to random ports. This situation overloads the host, preventing it from processing legitimate traffic [58]. Reflection attacks utilize inherent behaviors of network protocols such as UDP or TCP to amplify attack traffic. The attacker sends a request to a server using the target's IP address, causing the server to reflect the attack traffic back to the target. This process substantially magnifies the volume of traffic that the target receives, potentially causing a denial of service [57]. Recently, however, this type of attack has decreased in favour of direct-path attacks, which increased by 18% over the last three years [61].

2.1.1.4 Protocol Attacks

Protocol attacks also referred to as state-exhaustion attacks, aim to disrupt the normal functioning of a target system by depleting its resources. These attacks exploit network and transport layer weaknesses and consume server resources or network equipment resources, such as firewalls and load balancers. Because network protocols are stateful, the server must maintain the state of each connection, consuming its resources for the duration of the connection. An attacker establishing a huge number of connections, each utilizing a small amount of resources, can eventually get the server overwhelmed [22].

One example of such an attack is a SYN flood attack. The attack takes advantage of the TCP handshake, a sequence of messages exchanged between two computers to initiate a network connection, by sending many fake *Initial Connection Request* SYN packets to the target. This results in the target machine responding to each request and waiting for the final step of the handshake, which never comes, ultimately leading to the depletion of the target's resources [22]. A SYN flood attack can occur in three ways: direct, spoofed, or distributed [21]. In a direct attack, the attacker's IP is not masked, making them susceptible to discovery and mitigation. It's important to note, however, that if the attacker employs a botnet such as Mirai, there is generally less concern about masking the IPs of the infected devices, as the vast number of involved devices provides a form of anonymity. In a spoofed attack, the attacker obscures their identity by spoofing their IP address, complicating mitigation efforts. Lastly, in a distributed attack, the attacker uses a botnet, which significantly reduces the chances of tracing the attack back to its source, especially when each device in the botnet also spoofs its IP addresses.

2.1.1.5 Resource Attacks

Resource attacks, focused on the application layer, seek to overwhelm a target's server resources rather than simply increasing traffic volume. These attacks exploit the server's obligation to expend significant resources to generate web pages in response to HTTP requests [22]. Differentiating between benign and malicious traffic presents a defensive challenge due to the substantial resource influence of each request.

Several types of resource attacks exist, often focused on preserving connections in a throttled manner. Slowloris, for example, fragments HTTP headers, forcing the server to maintain open connections until all fragments arrive, impeding the processing of legitimate requests [56]. Similar strategies, like the Slow Post and Slow Read attacks, exploit the establishment of connections and the initiation of large downloads, respectively, at incredibly slow speeds, causing excessive resource utilization.

Some attacks exploit the higher resource consumption of HTTP requests on the server side. Large POST requests involve attackers uploading large files or data, monopolizing the server's connection and exhausting TCP or server resources. Similarly, HTTPS flooding involves attackers imposing high-rate HTTP/S requests, burdening the server [56].

2.1.2 Network Traffic Simulation

Traffic simulation is a technique used to model and evaluate network traffic behavior. Simulations can be performed using either real or simulated data based on previous attack characteristics [3]. In the context of intrusion detection systems, it can be used to validate the performance of models that aim to detect and respond to malicious activity [6]. Traffic can be simulated using hardware or software simulation techniques [7]. While hardware simulation creates a physical environment that closely resembles real-world conditions and therefore provides a more accurate representation of a network's behavior, it has limitations in terms of scalability, making it difficult to simulate large or complex networks. Software simulation on the other hand is highly scalable and flexible enough to create a wide variety of network scenarios. Also, as software simulation is often less expensive than hardware simulation as it only requires access to a computer and simulation software.

Within the scope of software-based simulation tools, notable contenders exist such as NS3, OMNeT++, and QualNet. NS3 is a discrete-event network simulator, valuable for simulating protocols over wired and wireless networks [63]. OMNeT++ is a modular, C++-based simulation framework, allowing the modeling of various communication protocols and systems [64]. Lastly, QualNet, a commercial solution, excels in testing large networks, providing high-speed simulations across a variety of network configurations [47].

Libpcap, a popular library in the domain of network data capture, is recognized for its extensive use by a multitude of networking tools including, but not limited to, TcpDump, WinDump, and Snort. Furthermore, the most used networking tools Wireshark and TShark, while presently favoring the next generation *pcapng* by default, also offer Libpcap support [81]. The structure of a capture file is straightforward; it commences with a file header, followed by packet records - each delineating a captured packet featuring pertinent information such as timestamp, length, and data [39]. The library's extensive adoption and robust functionality position it as a vital resource for any endeavors in network traffic analysis.

2.1.3 Network Topologies

The structure of internet topologies can be comprehended and represented through a hierarchical model, commonly referred to as *tiers*. This hierarchical approach aids in distinguishing the various levels within the internet topology, creating a clear and organized system.

Within this tier system, three main levels are conventionally acknowledged [34, 42]. Tier 3 occupies the lowest rung of this hierarchy, positioned just above individual clients or end-users. Providers at this tier are also known as access providers, characterized by their primarily local coverage. They function as gateways, connecting businesses and consumers to the broader scope of the internet. To facilitate this connection, clients use specific hardware such as modems or Network Interface Cards (NICs), enabling communication with these local Internet Service Providers (ISPs).

Upon successful connection and initiation of a message or data packet from the client, this packet is then passed up the hierarchy to the next level - tier 2. The tier 2 ISPs, also known as regional ISPs, hold an intermediate position in the tiered hierarchy. If a data packet received at this level is destined for a client within the ISP's coverage area, it is delivered directly. However, if the intended recipient is outside the regional ISP's network, the data packet is forwarded to the ISP's backbone network for further transmission.

The highest point of the hierarchy, tier 1, often referred to as the backbone level, sits at the top. Backbone networks maintain high-speed fiber-optic connections with the regional ISPs, ensuring fast and efficient data transmission. They function as a central hub within the internet topology, effectively directing data traffic and ensuring data packets reach their respective destinations, whether through direct delivery or rerouting to lower tiers for subsequent delivery. This tiered structure is instrumental in ensuring an efficient transmission and routing functionality.

In the realm of network topologies, latency plays a pivotal role in determining the efficiency and performance of data transmission. As [24] elucidates, fiber optic cables exhibit latency at 67% of the speed of light, which translates to a speed of 200,000 km/s within the fiber. The index of refraction for these cables is 1.5, resulting in light traveling 1.5 times slower through optical fiber than it does in a vacuum.

[50] highlights that this is the maximum speed achievable in real-life scenarios, with various factors often causing slower speeds. Interestingly, as distances decrease, the factors impacting speed tend to increase. This phenomenon can be attributed to the nature of connections, which are not direct or linear. Instead, data zigzags from one router to another until it reaches the target, affecting the overall latency.

For more extensive connections, such as transatlantic cables, the zigzag pattern becomes less pronounced, predominantly occurring on continents. This knowledge serves as a foundation for analyzing and designing network topologies that take latency into account, ensuring realistic latency differences between different network structures.

2.2 Related Work

In the field of cybersecurity, the validation of IDSs is an essential task, and it has been approached in various ways. One of the prevalent methods involves conducting practical tests, where systems are put to the test in real-world scenarios. This hands-on approach primarily involves the use of attack tools and traffic generators to simulate possible threat scenarios and gauge the system's ability to detect and mitigate such threats. These tools will be explored in depth in Section 2.2.1.

As an alternative to practical testing, validation can also be conducted in a more controlled, theoretical manner. This approach typically involves network traffic datasets which are used to test the IDS's capability to detect and respond to potential threats. These datasets consist of network traffic which is designed to reflect realistic network behaviour, both normal and anomalous. The limitations of this theoretical validation approach will be further discussed in Section 2.2.2.

Lastly, an integral part of synthetic datasets involves the generation of the dataset. The quality and relevance of the dataset used can greatly affect the results of the validation process. Different methodologies exist for generating these datasets, each with its own strengths and potential pitfalls. These approaches to dataset generation will be detailed in Section 2.2.3.1. In this way, a comprehensive overview of the existing methods for IDS validation will be provided, shedding light on the multi-faceted nature of IDS testing and its vital role in maintaining network security.

2.2.1 Attack Tools and Traffic Generators

The sphere of DDoS attacks has witnessed the emergence of increasingly sophisticated attack tools and traffic generators. A substantial amount of existing research has been devoted to understanding these tools [40, 46, 49, 52, 75, 80].

These attack tools can be distinguished along various dimensions. For instance, some tools offer a Command Line Interface (CLI), while others provide a Graphical User Interface (GUI). The dynamics of attack rates, too, differ in patterns and possible configurations across tools. The cross-platform compatibility of tools is another distinguishing factor, as not all tools are supported on all operating systems. Furthermore, the synchronization of the attack in a distributed setting varies between tools, as does the attack category and the protocol used.

Aging is also a pertinent concern with regard to these tools. There are many known tools from the early 2000s, such as Stacheldraht [26] and TFN Tribe Flood [26] from 1999, Mstream [31], Shaft [30], and Trinoo [26] from 2000, as well as Kaiten [40] and Knight [32] from 2001. However, there has also been the development of newer tools, such as Silent-Ddoser [5] or SEER [5], which reflect the evolving landscape of DDoS attacks.

In contrast to attack tools, traffic generators can create both attack traffic and legitimate traffic. They can be employed to test the load balancing of a system or to execute a stress test. There are numerous tools that are suitable for such tasks. Some are quite rudimentary, like Apache Bench [78], but are nevertheless effective for certain scenarios.

In addition, more sophisticated tools, such as D-ITG, are capable of creating both legitimate and attack traffic [4, 11, 13, 32]. There are also various other tools available, as shown in [5]. These tools offer diverse functionalities and can be utilized to create a range of traffic patterns depending on different input parameters.

2.2.2 Existing Datasets

In the past, researchers in the domain of DDoS attacks have increasingly utilized publicly available real datasets to validate their IDS approaches [7, 9, 15, 29, 41, 51, 65, 70, 74, 83]. Despite this trend, the suitability of the chosen dataset for the intended validation method remains a topic of active debate.

Datasets, particularly those related to DDoS, often come with several inherent challenges. For instance, the age of datasets can pose significant complications [35]. Research conducted by [3] and [6] illustrates that numerous datasets, including the FIFA World Cup Dataset and KDD among others, are considerably aged. This age issue can be especially problematic as older datasets may not reflect the current patterns and standards of network traffic. This issue becomes pronounced in the context of DDoS attacks, which have rapidly evolved over the years.

Furthermore, many datasets suffer from incompleteness, with only a limited number being available in their entirety. The extent of the network traces often represents only a brief snapshot of network activity. This short length may not capture the full complexity or diversity of potential network interactions, particularly over extended periods of time. For instance, certain types of network attacks evolve slowly, and short-term data may not reveal the full extent or progression of these attacks. In addition, the sanitization measures, such as anonymization or trimming, applied to these datasets can further toughen the challenge. Anonymization, while necessary for privacy reasons, often results in the loss of valuable information that could be crucial in identifying and understanding network attacks. Trimming, a process that reduces dataset size by selectively removing data, can result in missing crucial information and might inadvertently remove traces of subtle or slow-evolving attacks. As attackers switch from big volume, easily detected attacks to low volume stealthy attacks, this is becoming more and more of an issue [6].

Additionally, the number of datasets that are publicly accessible is rather limited, with most requiring specific access permissions. As a result, the majority of public datasets are effectively semi-restricted. Compounding this problem is the fact that available datasets often come in various formats, including some that are incompatible with quasi-standard tools like tcpdump [79]. Certain datasets even contain processed traffic data compressed into flows, thereby reducing their size. Additionally, trimmed Packet Capture (PCAP) files with removed payloads or altered headers are not uncommon, as highlighted by [6].

These datasets can vary greatly in terms of the nature and scope of attacks they include, their size, and whether they are anonymized or real or derived from simulated scenarios. For instance, CAIDA is frequently utilized in numerous research studies [7–9, 84], while also DARPA remains one of the well-known datasets [48, 77]. As time goes by, the ongoing aging of commonly used datasets has prompted researchers to increasingly turn to more recent offerings. The Canadian Institute for Cybersecurity has responded to this demand by publishing several artificially created datasets, tailored to the evolving landscape of

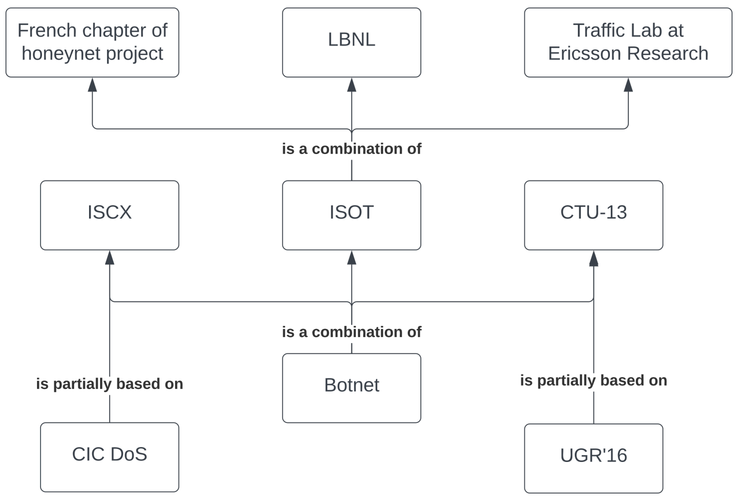


Figure 2.1: Relationships between Datasets [69]

network security. Notably, among these contributions, the CICDDoS2019 dataset has emerged as a prominent resource in contemporary cybersecurity research [72]. It presents a novel alternative to aging datasets, providing up-to-date and relevant network traffic patterns that reflect the current state of DDoS threats. These newer datasets are partly derived from real attacks, but often they are also based on existing datasets, as Figure 2.1 shows.

The choice of dataset is highly contingent on its intended use, making it a crucial factor in any research study [10]. To validate any proposed DDoS attack detection technique, it is essential for the captured network trace to contain a realistic mix of background and attack traffic, without any undue bias towards a specific type of traffic [7]. However, achieving this mix in a real experiment-driven dataset is challenging due to the lack of a known model for correctly characterizing internet traffic [66].

In contrast, if datasets are used as a data source for generating new traffic, other considerations become more pertinent. Primarily, it is important to understand the composition of traffic in the source dataset. Unlike the validation of DDoS attack detection techniques, the blending of various types of traffic is less desirable. If a filtered dataset by type isn't available, the traffic should be labeled at the very least. CAIDA is one such dataset that includes filtered DDoS traffic suitable for this purpose. Furthermore, there is a dataset collection, denoted as DDoS Packet Capture Collection, that is a compilation of various types of DDoS traffic, filtered by type [37].

2.2.3 Dataset Generation

Dataset generation plays a crucial role in developing and evaluating network security systems. To accurately represent the complexity and diversity of real-world network traffic, it is essential to consider several properties a dataset must possess. Moreover, various approaches have been proposed to generate network traffic datasets. Based on existing words, this subsection provides an overview of the properties and approaches in network traffic dataset generation, highlighting their strengths and limitations.

2.2.3.1 Approaches

In the process of generating datasets for network scenarios like DDoS attacks, four main approaches can be identified, as described in [6, 7, 73].

Mathematical models employ theoretical and symbolic representations of systems, applications, platforms, and conditions to understand network scenarios, such as DDoS attacks. By using mathematical validation, researchers can test the accuracy and reliability of these models. Although mathematical models offer a valuable foundation for understanding complex scenarios, they remain theoretical and may not accurately capture the full scope of real-world interactions and network behavior.

Simulation-based experiments offer a controllable and repeatable framework for network-based experiments on a single computer system. They provide researchers with the flexibility to rapidly prototype and evaluate potential solutions, discarding suboptimal alternatives before full implementation. Simulations use models of key operating system functions, kernel mechanisms, virtual platforms, and synthetic conditions to mimic network behavior. However, the realism of simulating attackers, targets, and network devices has been questioned [54], and the slower speed of traffic replay can hinder the evaluation of DDoS attack detection techniques, particularly for high-rate flooding attacks. Examples include NS3 [63], OMNET++ [64], QualNet [47], etc.

Emulation bridges the gap between simulation and real systems by integrating real elements of an operating system and applications with simulated network links and virtual nodes. It leverages soft routers to make connections and runs in real time, as opposed to the virtual simulated time used in simulations. While this approach offers greater realism than simulation, it faces scalability challenges, as extending the topology of computer systems beyond certain limits is difficult. Previous research has demonstrated that the intricacy involved in building and managing a network topology typically caps at around 300 nodes [7]. However, in real-life scenarios, DDoS attacks often involve thousands of nodes, making it challenging to accurately replicate such attacks in a controlled environment [68].

Real systems deliver the most realistic network conditions, operating systems, applications, and platforms for network-based experimentation. Despite their advantages, real systems pose several limitations: changes to network topology are not easily made for new experiments, live experiments involving internet worms or viruses pose significant risks, and flooding-based DDoS attacks can degrade network links. However, real systems, such as GENI [33] and PlanetLab [67], continue to be valuable tools for researchers, offering a robust platform for network experimentation and evaluations.

2.2.3.2 Dataset Requirements

The generation of realistic datasets is crucial to the validation and improvement of various defense systems. As such, [6] articulates four properties that a dataset should possess to maximize its realism and utility in this context.

Firstly, a realistic dataset should incorporate real source and destination IP addresses. This implies that clients should establish valid TCP connections with the target server, emulating genuine interactions that occur within networked environments. Additionally, these clients should be accessing real pages on the server, further augmenting the authenticity of the network traffic patterns.

Secondly, a realistic dataset should display a broad range of random source IP addresses. Both actual attacks and legitimate traffic should emanate from a diverse range of IP addresses. This level of variety accurately reflects the decentralized and widespread nature of internet traffic.

Thirdly, the generation of actual packets with valid headers is essential for the dataset's realism. Given the authentic nature of the traffic generated and captured, it is necessary for packets to maintain valid headers, aligning with the technical standards of real-world network traffic.

Lastly, the dataset should exhibit an appropriate mixture of normal and attack traffic. The simulation of network traffic should entail a balanced blend of legitimate and malicious attack traffic. This balanced representation is vital for effectively evaluating the performance of any IDS, as it replicates the multifaceted and complex environment these systems encounter in reality.

However, the degree of realism attainable in a dataset is contingent upon the selected approach, as described in Section 2.2.3.1. This suggests a nuanced approach in dataset creation, where the importance of individual properties in terms of realism is taken into account. Consequently, while all four properties are necessary for a comprehensive and realistic dataset, their significance may vary based on the intended use of the dataset and the specificities of the chosen approach.

2.2.3.3 Generation System Requirements

In crafting a system for generating network traffic datasets, the design-level requirements outlined by [35] and [7] provide a valuable roadmap.

Firstly, they advocate for reconfigurability. Such a feature facilitates the simulation of a diverse range of DDoS attacks, and allows for the alteration of network topologies as required. By ensuring that the system remains adaptable and flexible, researchers can model a wider array of scenarios.

Secondly, the system needs to emulate a complete computer network. This includes the incorporation of all necessary network equipment, from PCs and servers to routers and firewalls. Such a comprehensive setup allows for a more accurate representation of real-world networks.

Thirdly, the system must generate complete traffic, defined as a sequence of packets transmitted from a source, through routers or switches, to a destination. In the context of DDoS attacks, this involves coordinating multiple attacking machines to significantly impact a target host.

The fourth requirement concerns traffic load coordination. Each attacking host should contribute only a small fraction of the overall traffic. The goal is not to overwhelm the target with traffic from any one source, but rather to model a realistic attack where traffic comes from multiple sources. Furthermore, the system should aim for a complete capture of traffic, encompassing not just attackers, but also benign participants. Despite this, the system should still generate a labeled dataset for benign and attack traffic, enabling differentiation of the two in analysis.

Fifthly, the system should be capable of producing diverse types of traffic. This diversity extends both to protocols used and attack types launched. Such versatility enables a more comprehensive evaluation of the system's resilience and performance under various conditions.

The sixth requirement focuses on the balance between performance and cost. The system should be able to operate effectively with modest hardware, leveraging customized software to conduct meaningful experiments. While higher-end networking devices can enhance the scale of emulated attacks, substantial insights can still be gleaned from systems running on more modest setups.

Finally, the system should prioritize privacy and produce anonymous data. As such, it should not reveal any privacy-related information in the course of generating network traffic. This requirement aligns with ethical considerations and legal constraints in network research.

Chapter 3

Design

3.1 Generation Approach

Selecting one of the approaches discussed in Section 2.2.3.1 to generate a dataset is essential. For the specific scenario of creating a distributed DDoS dataset, the following criteria based on those mentioned in Section 2.2.3 are crucial:

1. Flexibility: The network topology must be easily reconfigurable, requiring minimal effort to recreate various attack scenarios.
2. Configurable Attacks: The final system should be capable of generating a wide array of different attacks, varying in DDoS attack types, distribution, and timing.
3. Realism: The network behavior should accurately reflect routers, participating devices, and a genuine mix of link bandwidth capacities and delays. The resulting dataset must possess properties stemming from this realistic network environment.
4. Scalability: The system must handle the simulation of DDoS attacks involving thousands of nodes, as these large-scale attacks are common in real-life situations.
5. Repeatability: The system should consistently produce the same output for a given input configuration, ensuring reliable and repeatable results.
6. Affordability: As financial resources for this project are limited, the system must not generate costs that exceed the project's budget constraints.

As outlined in Section 2.2.3.1, the mathematical approach offers limited realism, as it can only capture a small portion of real-world complexity through mathematical models. However, the realism questioned by [54] for simulation is not a concern in this application, as real-time simulation evaluation is not required. The slower simulation speed is not an issue; it may even provide advantages. This simulation speed is likely to be faster for non-computation-intensive tasks, allowing for quicker results. Thus, simulation can be considered realistic enough for the purpose at hand.

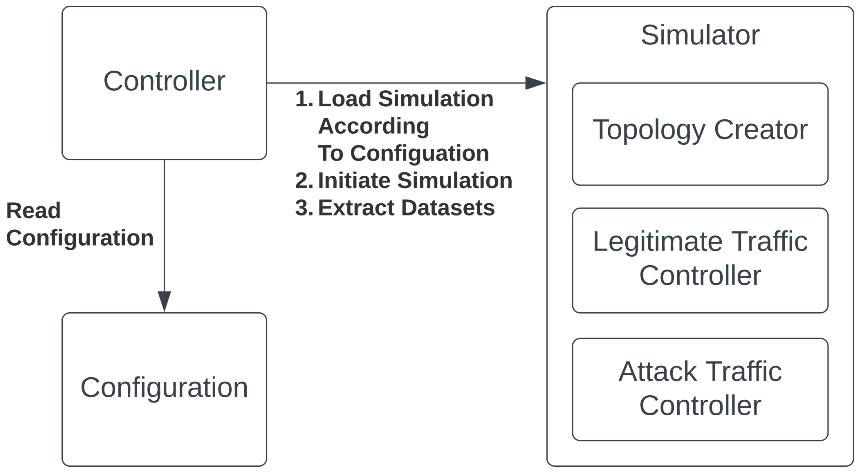


Figure 3.1: Technical Architecture - Core Components

Scalability poses a challenge when attempting to achieve such results through emulation or real-world replication. Single-system emulation proved inefficient, as demonstrated in a trial with Mininet for a large-scale network. Bigger systems replicating a DDoS attack using real systems or distributed emulation would not be affordable within the project's scope. Additionally, even if they were, such extensive systems could not be easily reconfigured for different scenarios.

Therefore, simulation is the chosen approach. It aligns well with the desired setup complexity and reconfigurability, allowing for repeatable experiments and the simulation of various attack types.

3.2 Architecture

The architecture of the EDDD is displayed in Figure 3.1. The system is organized in a centralized manner, with a controller at its core. This controller is responsible for reading and validating the system configuration. Furthermore, it interacts with the simulation engine, loading the validated configuration into the simulator as an initial step. Following this, the controller initiates the simulation process for generating the datasets. Upon completing the simulation, the controller extracts the generated datasets from the simulator and places them in an output folder readily accessible to the user.

The subsequent sections delve into the detailed design of the subcomponents *Topology Creator*, *Legitimate Traffic Controller*, and *Attack Traffic Controller*. These sections elucidate the principles these components adhere to to fulfill their respective roles within the simulation engine effectively. The controller, input file, and simulator, along with all its subcomponents and the technologies employed, will be thoroughly examined in Chapter 4. This chapter will provide an in-depth analysis and discussion of these core elements, facilitating a comprehensive understanding of their functions and interrelationships within the system.

3.3 Topology

To observe traffic in different nodes effectively, these nodes must be accurately represented within the chosen modeling approach. Therefore, it is important to consider two primary requirements when modeling network topology: accurately representing the target node, which represents the destination of network traffic, and effectively observing the behavior and activity of the network before it reaches the target through observing nodes.

A key characteristic of DDoS attacks or general traffic on large-scale servers is their distributed nature, which often leads to packets from different sources exhibiting distinct features at the target and passing through various observing nodes. Packets from remote locations tend to travel greater distances, encompassing more hops and experiencing bigger latencies before reaching their intended target. These factors are influenced by the geographic locations of the countries involved and the quality and structure of their respective infrastructures.

Modeling the distributed nature of traffic in large-scale networks can be achieved by utilizing a multi-layer network approach (*cf.* Section 2.1.3). A crucial component of this model is the broadband internet network, which connects larger regions within a country to the broadband network of the neighboring countries. The broadband internet topology level is vital for facilitating high-speed connections and achieving low latency over vast distances. This is accomplished through a strategic network design that minimizes the number of hops required to cover enormous distances, thereby ensuring efficient data traffic flow between larger regions and reducing the likelihood of potential bottlenecks or points of congestion that could adversely affect network performance.

Each node within the broadband layer delivers connectivity to a sub-network of regional routers, further refining the network's structure. The regional internet topology level focuses on distributing data traffic within smaller geographical areas, ensuring efficient connections for users within those regions. As the network hierarchy continues to narrow, each node on the regional layer provides connectivity to a sub-network of local routers. These local routers serve as the access network for nodes seeking to connect to the broader network, such as servers and clients. At the local internet topology level, the emphasis is on maintaining connections for end-users. While latency and bandwidth play essential roles in determining the overall user experience and network performance, the connections within the access network exhibit significantly lower performance than higher-level networks.

As depicted in Figure 3.2, the traffic flowing through such a network structure forms a tree-like hierarchy. The diamond-shaped symbols labeled *B* in this Figure represent backbone routers. In contrast, the octagon-shaped *R* symbols indicate regional network routers and the elliptic *L* symbols denote local network routers. However, this tree representation falls short of accurately modeling the multi-level network approach, geographic structure, and distance between nodes.

The geographic distances between nodes are reflected as latency in network traffic. Although packets may traverse countries in just a few milliseconds, these milliseconds cannot be neglected when modeling a network of this nature. Unfortunately, the tree representation like it can be seen in Figure 3.2 is unsuitable for displaying these geographical

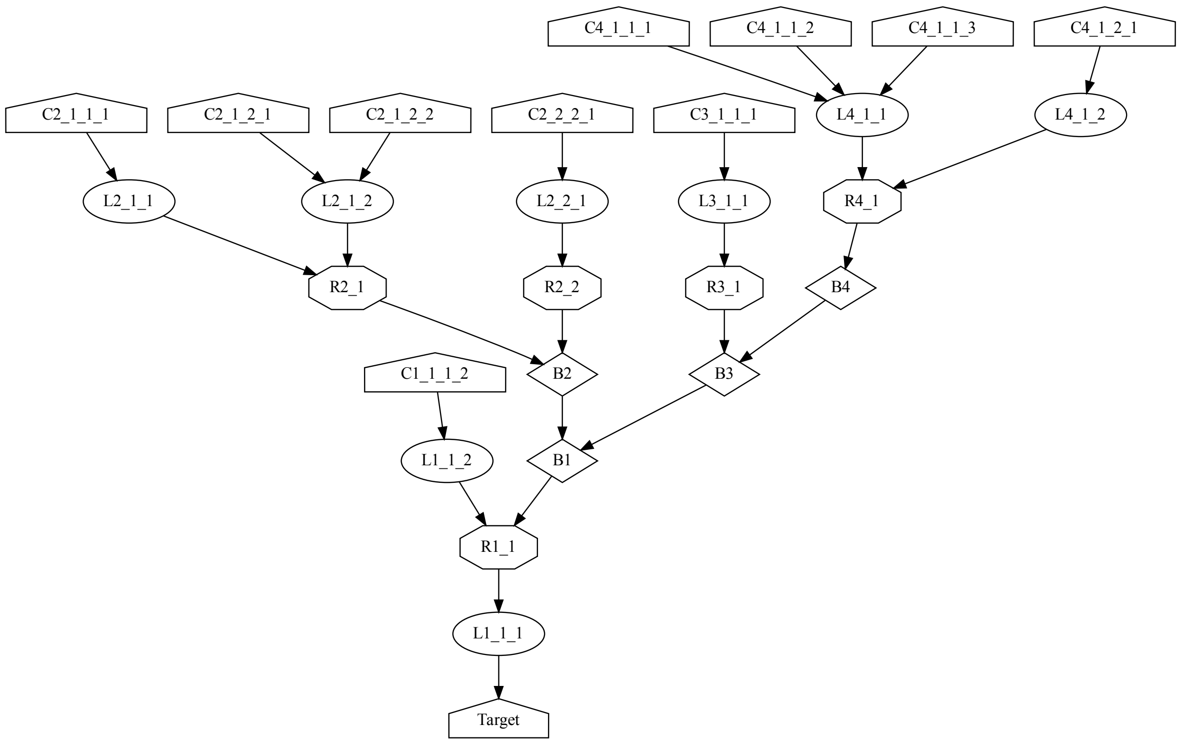


Figure 3.2: Traffic Flow Over Network Topology

distances clearly and meaningfully. Furthermore, configuring a network to resemble a country-like structure, each with its distinct subnets, presents challenges when using the tree representation. This limitation makes it difficult for an individual to easily grasp and manipulate the network structure to represent the desired geographic topology accurately.

The approach of modeling and connecting countries proves to be more intuitive for an individual. However, certain abstractions are necessary, as not all aspects influence the outcome of routed packets and their latencies in the same manner. In the real world, coverage, redundancy, population density, and general infrastructure quality vary. In contrast, the simulation in this thesis assumes a randomly distributed network and end-user coverage across a country, neglecting geographical features such as deserts, lakes, mountains, and rural areas. This simplifies the representation in a simulated model.

```cpp
struct Country {
  std::string name;  //!< Name of the country
  uint32_t nodes;    //!< Number of backbone nodes
  double area;       //!< Area of the country in sqkm
  double population; //!< Population of the country
  bool enablePcap;   //!< Is pcap recording enabled for backbone nodes
   of the country
  double attackTrafficFactor;     //!< How much of the clients produce
   attack traffic in this country
  std::vector<Country> neighbors; //!< What are the neighboring
   countries (without loops)
};
```

Listing 3.1: Country Configuration Type

A country is defined by a configuration shown in Listing 3.1, including a name for easy router node identification during traffic analysis and specifications for its area and population. When modeling, it is assumed that the number of end devices inside a country scales linearly with the population. The configuration also allows specifying the number of backbone router nodes, which are then randomly distributed across the country's area.

An abstraction assumes that a country's area is a square, facilitating distance measurements between nodes. These distance measures can be used to deduct the latency of a packet traveling from one node to another. Figure 3.3 shows what a random distribution of nodes inside an area could look like. All those nodes are then connected to ensure every node can be reached within a couple of hops. Each backbone router serves a portion of the country's area, connecting to a regional network below it. The same assumptions about square areas and equally spaced nodes apply to regional and local networks. The number of clients connected to a node is determined by the area a local node serves and the country's population density.

These countries can then be connected to neighboring countries, all modeled with the same approach but individually customizable. Neighboring countries are defined in the configuration, and their connections to the first country are determined in a way that places them as far apart as possible (*e.g.,* two neighboring countries are placed on opposite sides). Neighboring countries connect their backbone nodes to the closest backbone nodes of the first country, ensuring efficient inter-country connections.

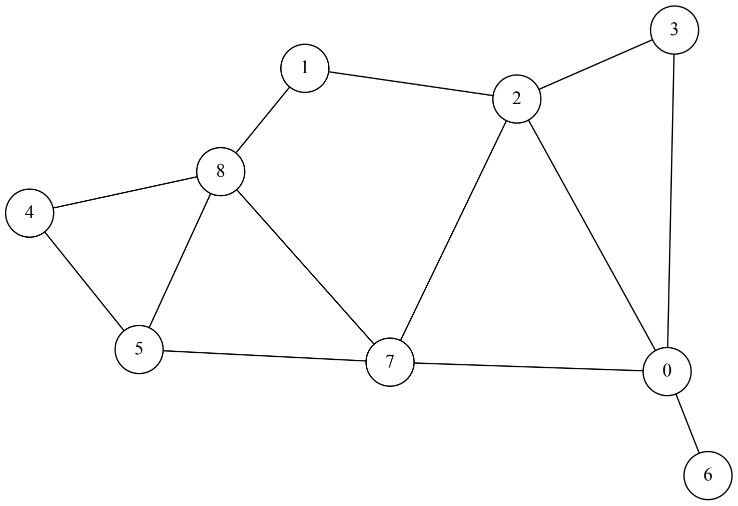


Figure 3.3: Node Distribution in Service Area

3.4 Legitimate Traffic

This section discusses legitimate traffic patterns based on dataset [28], which reveals benign traffic patterns. There, a flow-based analysis involves combining and analyzing different datasets, specifically CIC DoS dataset [18, 43], CIC-IDS2017 [17], and CSE-CIC-IDS2018 [25, 71]. A flow represents a sequence of packets with shared characteristics, such as source and destination addresses, ports, and protocols. Analyzing flows provides insights into network behavior and communication patterns between hosts. 84 features characterize each flow; however, not all are relevant and chosen for remodeling legitimate traffic in this project's scope. Not every flow within this dataset includes a complete TCP flow with SYN packets for the handshake and FIN flags.

The focus is on the sending patterns in the dataset, and the following features must be extracted:

- Flow Duration
- Idle Time between Flows
- TCP Connection Durations
- Packet Length
- Packets per Second

The dataset contains the following properties for each flow, which enable the deduction of the desired values:

- Flow Duration

- Timestamp
- Fwd Pkt Len Mean
- Fwd Pkt Len Std
- Fwd Pkt Len Min
- Fwd Pkt Len Max
- Flow Pkts/s
- SYN Flag Count
- FIN Flag Count

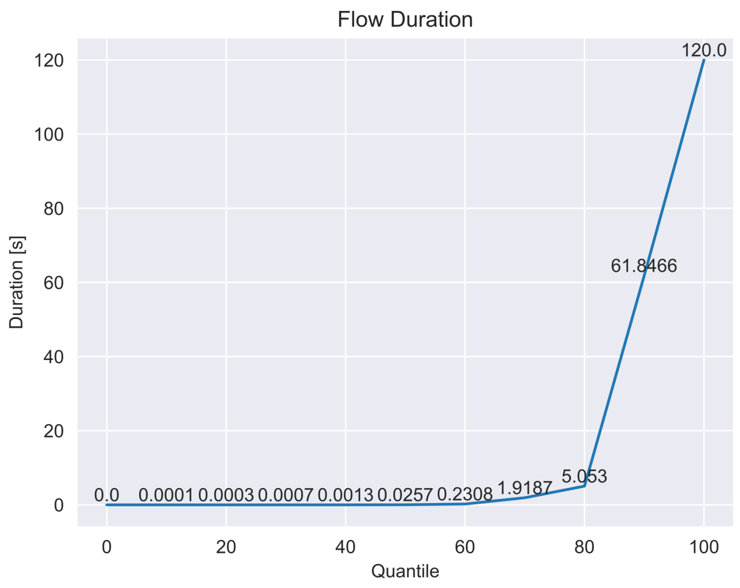


Figure 3.4: Probability of Flow Duration

Subsequent paragraphs detail how the different features are to be remodeled. Only a portion of the dataset is analyzed to remodel benign traffic according to recent traffic patterns. The dataset comprises multiple DDoS attacks with benign traffic from 2010 until 2018. As the goal is to model the traffic patterns according to the current standards, the most recent bulk of 2018 data is taken from the AWS portion of the dataset [28]. Consequently, only the traffic from April 20th, 2018, to April 21st, 2018, is examined.

Inspection of the `Flow Duration` column in the dataset facilitates direct measurement of the flow duration in microseconds. Analyzing the Flow Duration at certain probability quantiles reveals that most durations are extremely brief, taking only a fraction of a second (*cf.* Figure 3.4). However, approximately a third of the flows have significantly longer durations, ranging from several seconds to upwards of two minutes.

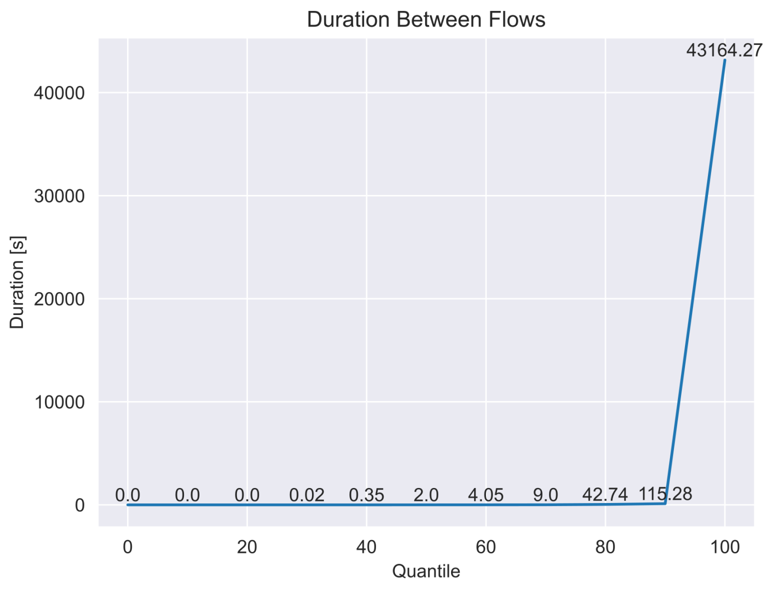


Figure 3.5: Probability of Idle Times Between Flows

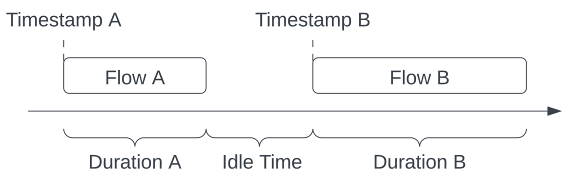


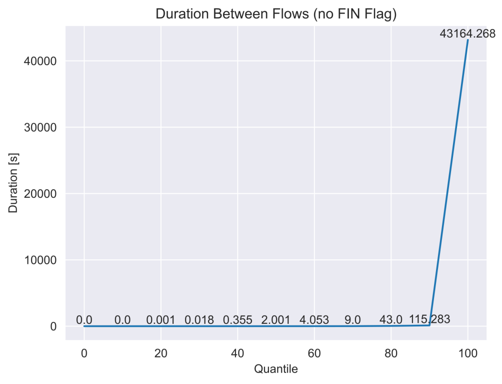
Figure 3.6: Idle Time Between Flows

In the process of recreating benign traffic, the analysis of idle times between the flows of a specific IP address is imperative. The idle times can be inferred by subtracting the duration of a flow from the time difference between two consecutive flows originating from a single IP, as illustrated in Figure 3.6. This method subtracts Timestamp A from Timestamp B, followed by Duration A.

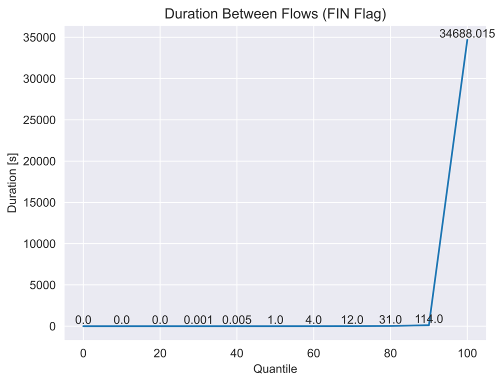
Examination of the duration between the flows (*cf.* Figure 3.5) reveals that the subsequent idle time is minimal for approximately half of the flows, lasting only fractions of a second. However, there are outliers, with idle times spanning multiple seconds and minutes and extending to several hours.

Figure 3.7 explores the potential correlation between completing a full TCP flow involving SYN and FIN flags and the idle time duration. A plausible hypothesis could posit that it is more likely to terminate a TCP connection before an extended idle time and initiate a new one post this duration. Nonetheless, this hypothesis is refuted by the data. The only observable pattern indicates that the probability of ending a TCP connection is at 0.6%, while the probability of initiating a connection is at 6%. This suggests that many connections remain open post the evaluation timeframe within the dataset, or a large proportion of TCP connections are not terminated gracefully.

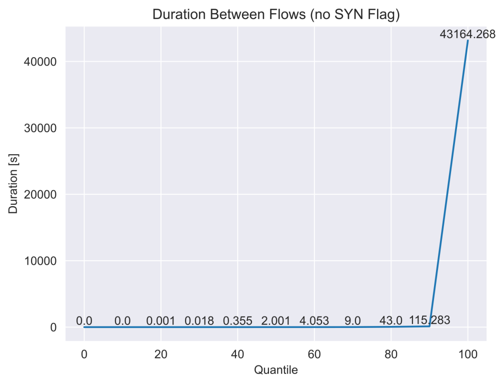
The packet lengths in the dataset can be derived from the `Fwd Pkt Len Mean`, `Fwd Pkt`



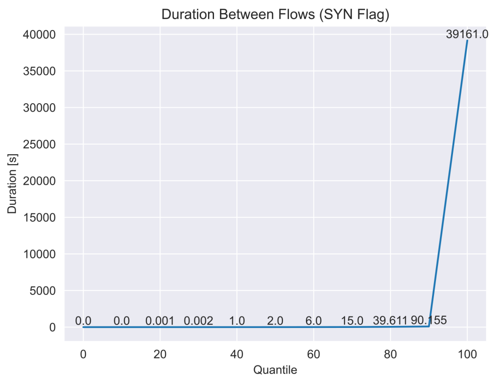
(a) Probability of Ending a Flow With No FIN Flag Before Specific Idle Times



(b) Probability of Ending a Flow With a FIN Flag Before Specific Idle Times



(c) Probability of Starting a Flow With No SYN Flag After Specific Idle Times



(d) Probability of Starting a Flow With a SYN Flag After Specific Idle Times

Figure 3.7: TCP Flags for Specific Idle Times

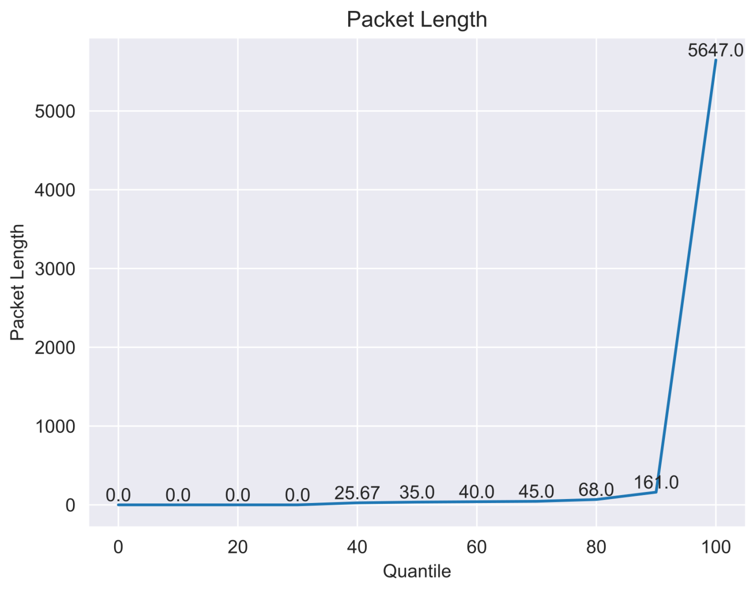


Figure 3.8: Probability of Packet Length

`Len Std`, `Fwd Pkt Len Min`, and `Fwd Pkt Len Max` columns. These four columns characterize a distribution for each flow within the dataset. However, these values cannot be simply added up to get an overall distribution, as these values describe multiple distributions. Therefore, a set encompassing a diverse range of packet lengths must be calculated first. This can be established by generating samples according to the distributions outlined for each flow and subsequently combining these samples. This distribution of packet lengths is depicted in Figure 3.8.

In this analysis, it is crucial to differentiate between the terms *packet length* and *packet size*. The term *packet size* pertains to the total size of a network packet, incorporating both the payload and any additional headers or overhead. Conversely, *packet length* refers to the size of the payload or data segment of the packet, excluding any supplementary headers or overhead. Interestingly, a considerable fraction of packets within the dataset exhibits a packet length of 0, which implies these packets do not contain a payload.

The `Flow Pkts/s` column in the dataset provides the number of packets per second for each distinct flow. This column can determine the rate at which packets are transmitted. The probability distribution of achieving specific packets-per-second rates is delineated in Figure 3.9. Upon analyzing this distribution, it becomes evident that there is a broad range of packet arrival rates. These rates extend from scenarios where packets reach the destination several seconds apart to instances where packets arrive at the same microsecond.

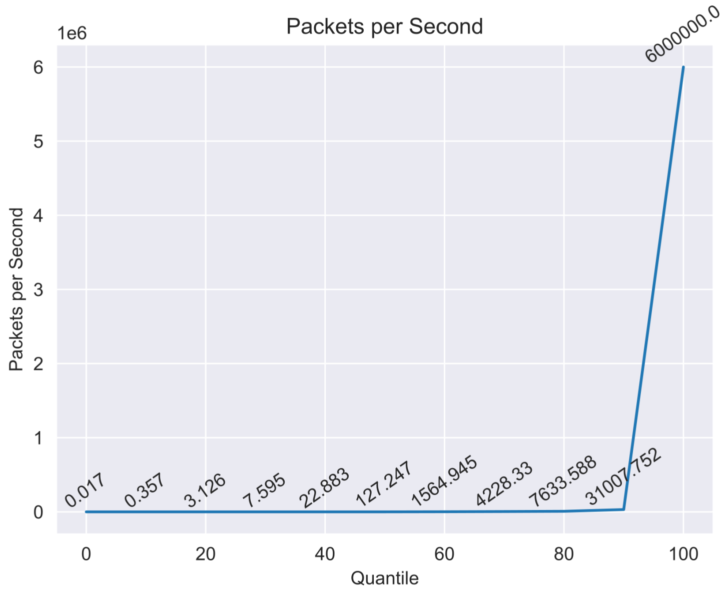


Figure 3.9: Probability of Packets per Second

3.5 Attack Traffic

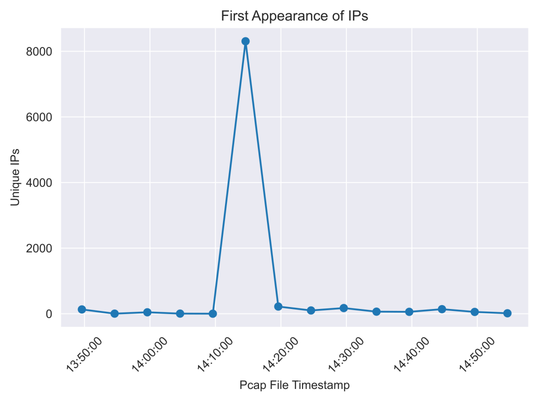
There exists a wide variety of DDoS attacks, each presenting distinctive characteristics. The primary objective of this thesis is to devise a system that facilitates the creation of distributed datasets. However, the primary focus is not necessarily remodeling the most intricate DDoS attacks. Out of the DDoS attacks discussed in Section 2.1.1, the volumetric and protocol attacks are deemed most appropriate for this study as they target the network and transport layer. Given that the system behind a target is not modeled but only the traffic arriving at it, it is not feasible to infer the effects of resource attacks.

Hence, two types of attacks are selected for modeling: an ICMP flood attack representing a volumetric attack and a SYN flood attack representing a protocol attack.

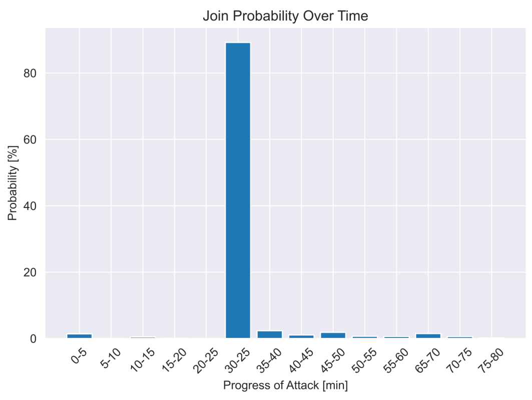
3.5.1 ICMP Flood Attack

The attack traffic is modeled according to the CAIDA DDoS dataset [14]. As this dataset only includes attack traffic, it is an excellent choice since no effort in labelling and filtering has to be made. This dataset comprises 14 files of the attack, each encapsulating a 5-minute duration. Packet timestamps have been consolidated into these 5-minute intervals to simplify evaluation and data load. From this dataset, it is discerned when various clients join the attack, their duration of participation, and the number of packets they send.

Figure 3.10a indicates that most IPs join simultaneously. Given the 5-minute segment grouping of timestamps in the analysis, it can be inferred that most of them join within the same 5-minute segment. A probability model for the joining time is constructed to

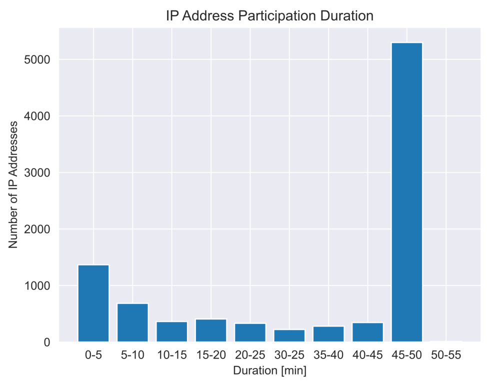


(a) Unique IP Joins Over Time

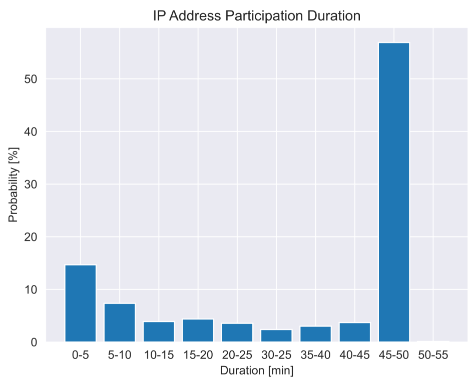


(b) Join Probability at Attack Progress

Figure 3.10: Attacker Joining Time



(a) Unique IP Participation Duration



(b) Participation Duration Probability

Figure 3.11: Attacker Participation Duration

replicate this, representing the probability of joining an attack within a 5-minute interval. These probabilities are illustrated in Figure 3.10b.

A similar approach is adopted to ascertain the duration of a client's attack. Figure 3.11a displays the number of distinct IP addresses in the CAIDA dataset and their respective attack durations. A probability model is also developed for this scenario, demonstrating the likelihood of maintaining an attack for a specific duration. This is depicted in Figure 3.11b.

The third piece of information to extract is the number of packets sent. Figure 3.12 portrays the progression of the attack in the CAIDA dataset and the average number of packets sent within a 5-minute segment of this attack. From a global attack perspective, no clear trend in the number of packets sent concerning the duration is evident. Figure 3.13 presents the attack progression from the viewpoint of a single IP and the specific client's attack duration. The packets sent appear more consistent, with some outliers attributed to the fact that in these 5-minute segments where the number of packets sent per IP is lower, many IPs do not participate in the entire 5-minute segment of an attack, thereby reducing the average. Disregarding these outliers, there is a relatively constant packet-

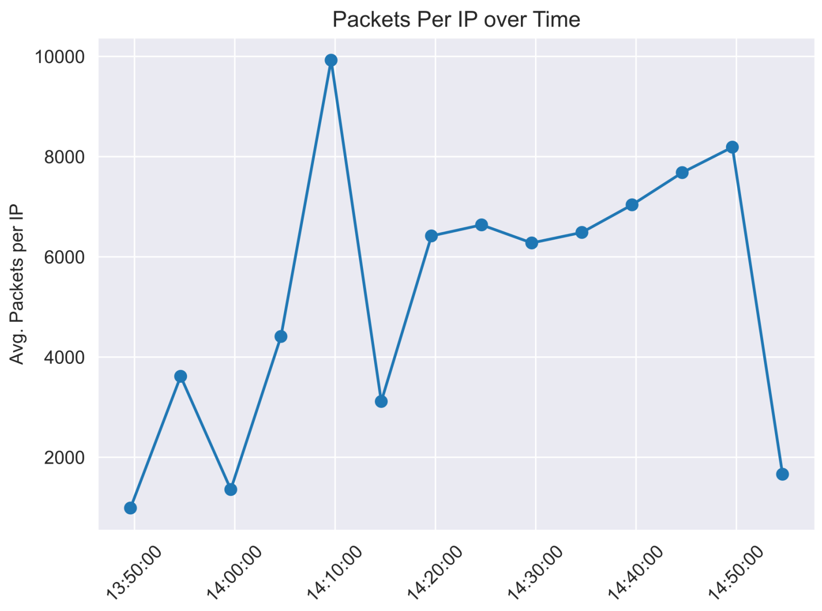


Figure 3.12: Average Packets Over Time in CAIDA Dataset

per-second rate between 16 and 27 packets per second.

3.5.2 Syn Flood Attack

The SYN Flood attack also has a corresponding dataset [37]. The SYN Flood dataset of the DDoS Packet Capture Collection contains 37,841 TCP SYN requests. However, inferring how the attack evolved regarding participating clients is not feasible due to IP spoofing. Similarly, it is impossible to ascertain each client's contribution to the attack as separating distinct clients is unattainable. The only extractable information is the number of requests arriving at the target over time, as illustrated in Figure 3.14.

Figure 3.14 shows two phases of the attack: a high-load packet phase and a low-load packet phase. In the high-load phase, up to 7,452 packets per second arrived at the target, while in the low-load phase, the rate fluctuated between 2 and 14 packets per second.

As such, these two phases should be reflected in the implementation. Figure 3.15 represent the probabilities of the number of packets per second within each specific phase, respectively.

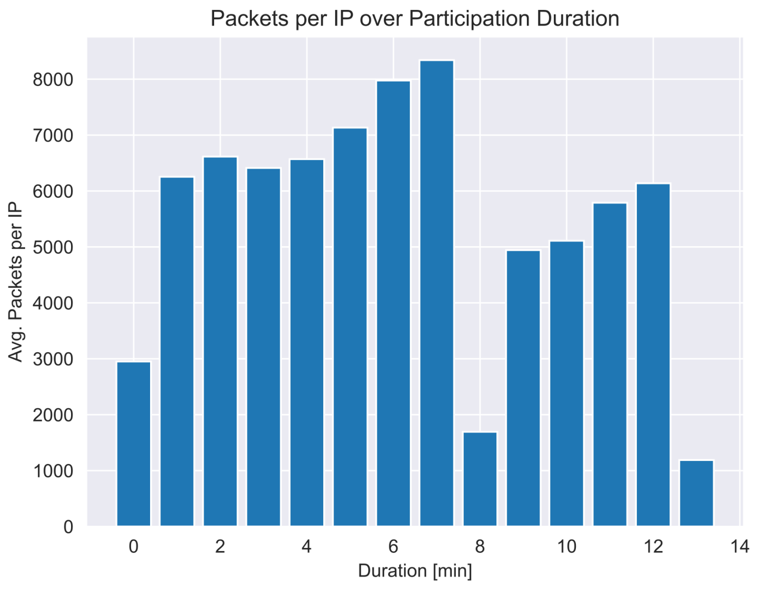


Figure 3.13: Average Packets Over Participation Duration

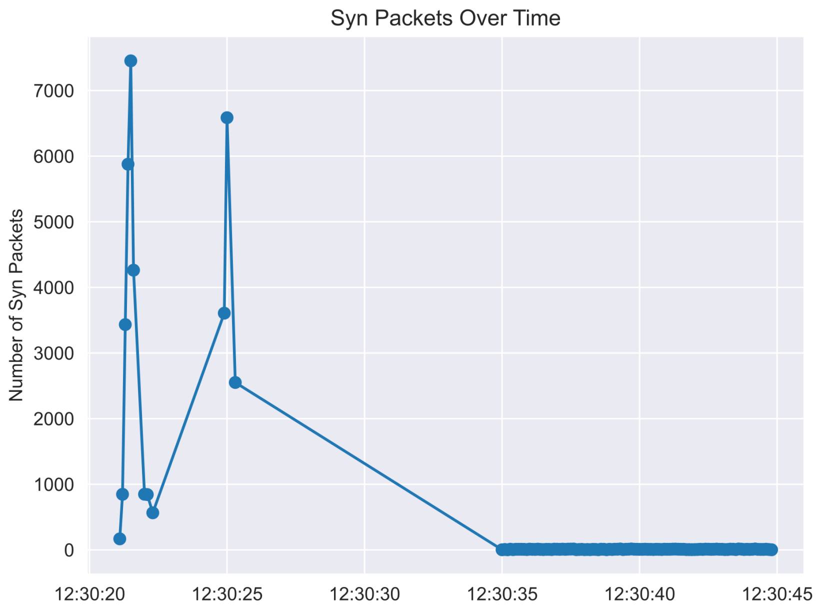
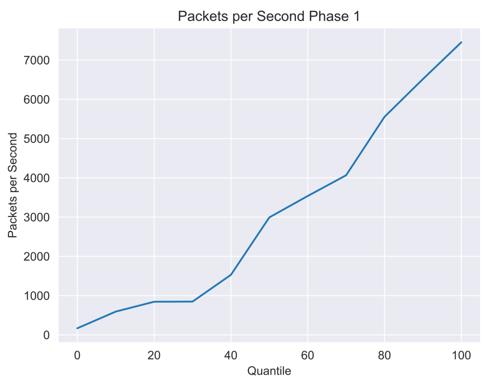
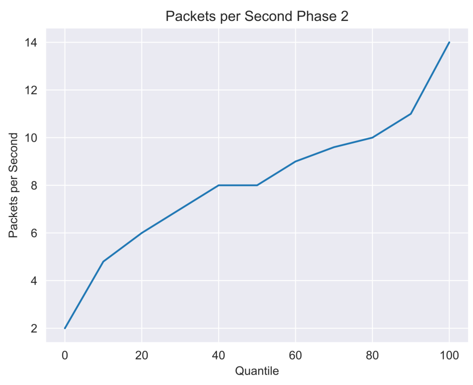


Figure 3.14: Packets Arriving at Target Within 0.1 Seconds



(a) Packets per Second Probability in Phase 1



(b) Packets per Second Probability in Phase 2

Figure 3.15: Packets per Second Probability

Chapter 4

Implementation

This chapter delves into implementing the Emulator for Distributed DDoS Datasets (EDDD), initially introduced in Chapter 3. The high-level architecture, as illustrated in Figure 4.1, encompasses three key components and employs various technologies as follows:

The configuration of the system is articulated as a JSON file. A few reasons underpin the choice of JSON as the format for the configuration file. First, its distinct file format allows easier adaptation and sharing of configurations, enhancing the flexibility and collaborative potential of the system. Second, JSON enables simple data reading and validation, simplifying user interaction and minimizing errors. Third, the accessibility of properties in JSON is straightforward, further easing the user experience. Lastly, JSON allows the representation of nested data structures. This aspect is crucial for more complex configurations, such as the topology configuration with countries, as illustrated in Section 3.3. In contrast, these configurations would be less clear and manageable if directly passed via program arguments.

As the controlling instance, a shell script is utilized. Given its primary function of reading the configuration, loading it into the simulator, initiating its start, and extracting the

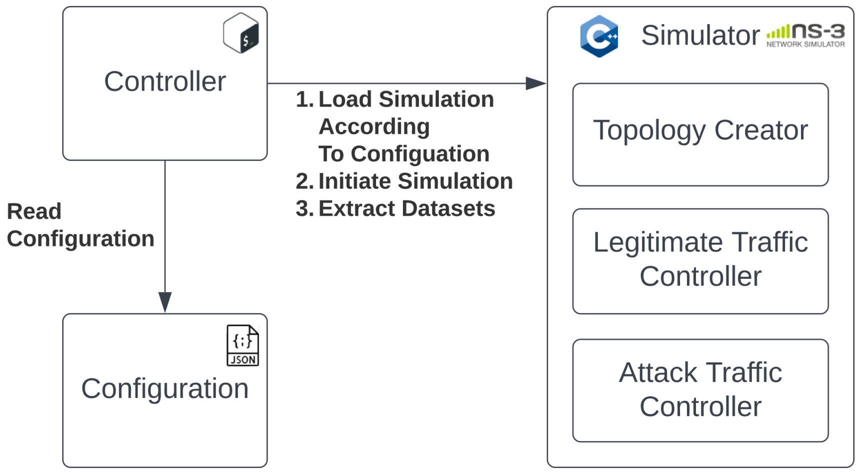


Figure 4.1: Implementation - Core Components

datasets, it is apparent that CLI access is essential. In this context, a shell script represents the most intuitive and basic system for accomplishing such a task. Moreover, it allows for automatic or remote execution, a feature that proves helpful if a simulation is desired to be executed on a different workstation.

At the heart of EDDD lies NS3, which assumes the role of the simulator. From the variety of simulators mentioned in Section 2.1.2, NS3 appears to be the most suitable. The simulator needs to meet key requirements, such as the ability to generate PCAP files. Among the simulators mentioned, namely NS3 [63], OMNet++ [64], and QualNet [47], all support this traffic data extraction and analysis method. Furthermore, the simulator had to be accessible for CLI interaction. This requirement extends beyond mere execution; it encompasses automated topology creation and configurations. Simulators OMNet++ and QualNet, while more focused on providing a user-friendly interaction via a GUI, were suitable for a use case via a GUI. The end user would never interact with the simulator directly but only through the EDDD controller. Moreover, the simulator had to be available free of charge, as paid software like QualNet would exceed the resources allocated for this thesis. In this regard, NS3 fulfills all these prerequisites. With its coding approach for the simulation and direct interaction with the simulator classes, NS3 offers the flexibility to create various scenarios and custom behaviors. Performance is another key criterion, and NS3's low-level code, written in C++, along with the ability to distribute the workload across multiple cores or even different machines via Message Passing Interface (MPI), also satisfies this requirement. In addition, a comparative study of the most widely used simulators reveals that NS3 is the predominant choice for academic purposes [62]. This fact further affirms the selection of NS3 as the simulator for this thesis. This prevalence suggests that NS3 meets the technical requirements and provides a robust and reliable platform for simulating network events, as attested by its widespread acceptance within the academic community. Consequently, the choice of NS3 brings with it a degree of assurance in its ability to meet the demands of this project.

In the following sections, a deeper exploration of each component is undertaken, highlighting the most interesting aspects of their implementation. Not only do these discussions cover the primary constituents such as the configuration file, controller, and simulator, but they also extend to the subcomponents. These include the various traffic applications and the network topology creator, each presented in detail. The ensuing discourse is designed to provide a comprehensive understanding of the system's workings.

4.1 Configuration

The configuration file utilized in the implementation is a static JSON file the user can interact with. In addition, the configuration file is responsible for setting up the required network and simulation parameters. It also allows to pass optional parameters to the system. These parameters serve as the blueprint guiding the operation of the simulation, providing crucial details that determine the behavior and outcomes of the simulation.

The required network parameters dictate the following properties of the topology and network structure:

- Countries with their properties as shown in Listing 3.1
- Bandwidth for each of the network levels, broadband, regional and local
- Latency factors for the connections on the network levels broadband, regional, and local
- Simulation representation factor in customizing the amount of network structure portrayed

Similarly, the required simulation parameter `attackType` defines the details of the simulation process itself. With this parameter, it can be decided whether an ICMP Flood Attack according to [14] or a syn flood attack according to [37] should be simulated.

In addition to these required parameters, the configuration file also allows for setting optional parameters. These optional parameters provide further customization possibilities, allowing users to fine-tune aspects of the simulation as per their specific requirements. Optional parameters include logging options to print out the pings from each client to the target (`logPings`) or printing the alignment and connections of the backbone routers in a Graphviz format (`logGraphviz`). Furthermore, it can be defined how many cores the MPI configuration uses and whether the exported PCAPs include the payload of the packets or only the headers. By offering these optional parameters, the configuration file increases the flexibility and adaptability of the simulation, allowing it to cater to a wider range of use cases and research objectives.

4.2 Shell Script Controller

The primary function of the shell script controller involves reading, validating, and parsing the configuration. However, before initiating a new dataset generation phase, it is crucial to clean up the remnants of the previous run meticulously. This involves deleting any lingering PCAP files from previous executions to guarantee a clean dataset free of outdated files. This process of tidying up is a prerequisite to avoid any potential contamination of the new dataset with residual data. The new configuration JSON is validated once this clean-up phase has been concluded. Listing 4.1 illustrates the initial validation of required parameters, namely `networkFactor`, `clientFactor`, `backboneLatencyFactor`, `regionalLatencyFactor`, `localLatencyFactor`, `backboneBandwidthGbps`, `regionalBandwidthGbps`, `localBandwidthGbps`, and `clientBandwidthGbps` ensuring they are of the correct types and formats. If this validation fails, an appropriate error message is displayed. Following this, the required country property is recursively validated for each connected country, again with appropriate error messaging if validation fails. The act of validation ensures the correctness of the inputted configuration parameters and safeguards against any possible errors during the subsequent dataset generation phase.

```bash
# Read JSON file
json_file="input-config.json"
json_data=$(cat "$json_file")

```

```bash
# Validate JSON properties
network_factor=$(echo "$json_data" | jq -r '.networkFactor')
client_factor=$(echo "$json_data" | jq -r '.clientFactor')
backbone_latency_factor=$(echo "$json_data" | jq -r '.backboneLatencyFactor')
regional_latency_factor=$(echo "$json_data" | jq -r '.regionalLatencyFactor')
local_latency_factor=$(echo "$json_data" | jq -r '.localLatencyFactor')
backbone_bandwidth_gbps=$(echo "$json_data" | jq -r '.backboneBandwidthGbps')
regional_bandwidth_gbps=$(echo "$json_data" | jq -r '.regionalBandwidthGbps')
local_bandwidth_gbps=$(echo "$json_data" | jq -r '.localBandwidthGbps')
client_bandwidth_gbps=$(echo "$json_data" | jq -r '.clientBandwidthGbps')
attack_type=$(echo "$json_data" | jq -r '.attackType')

validationSuccessful=true

if [ -z "$network_factor" ] || (($(bc <<<"$network_factor <= 0"))) || (($(bc <<<"$network_factor > 1"))); then
  validationSuccessful=false
  echo "Property networkFactor must be in the range (0,1]"
fi

# Also for client_factor

if [ -z "$backbone_latency_factor" ] || (($(bc <<<"$backbone_latency_factor < 1"))); then
  validationSuccessful=false
  echo "Property backboneLatencyFactor must be >= 1"
fi

# Also for regional_latency_factor and local_latency_factor

if [ -z "$backbone_bandwidth_gbps" ] || (($(bc <<<"$backbone_bandwidth_gbps <= 0"))); then
  validationSuccessful=false
  echo "Property backboneBandwidthGbps must be > 0"
fi

# Also for regional_bandwidth_gbps and local_bandwidth_gbps and client_bandwidth_gbps

if [ -z "$attack_type" ] || { [ "$attack_type" != "syn-flood" ] && [ "$attack_type" != "icmp-flood" ]; }; then
  validationSuccessful=false
  echo "Property attackType must be either syn-flood or icmp-flood"
fi

countries=$(echo "$json_data" | jq -r '.countries')

validate_country() {
  local json="$1"

  # Validate the JSON object
  valid=$(echo "$json" | jq '
    . |
    has("name") and (.name | type == "string") and
    has("nodes") and (.nodes | type == "number") and
    has("area") and (.area | type == "number") and
    has("population") and (.population | type == "number") and
    has("enablePcap") and (.enablePcap | type == "boolean") and
    has("attackTrafficFactor") and (.attackTrafficFactor | type == "number") and
    has("neighbors") and (.neighbors | type == "array")
  ')

  if [ "$valid" == "true" ]; then
    nodes=$(echo "$json" | jq -r '.nodes')
    area=$(echo "$json" | jq -r '.area')
    population=$(echo "$json" | jq -r '.population')
    attackTrafficFactor=$(echo "$json" | jq -r '.attackTrafficFactor')

    if (($(bc <<<"$nodes < 1"))) || (($(bc <<<"$nodes % 1 != 0"))); then
      echo "Country property nodes is invalid. Make sure, it is a positive integer."
      validationSuccessful=false
    fi

    # Validation for area, population, attackTrafficFactor

    if $validationSuccessful; then
      # Check each neighbor recursively
      neighbors=$(echo "$json" | jq -c '.neighbors | .[]')
      for neighbor in $neighbors; do
        validate_country "$neighbor"
      done
    fi
  else
    echo "Country property is incomplete. Make sure, the properties name, nodes, area, population, enablePcap, attackTrafficFactor and neighbors are present."
    validationSuccessful=false
  fi
}

validate_country "$countries"
```

Listing 4.1: JSON Validation of Required Configuration

Subsequently, the running configuration regarding the maximum number of cores to which the simulation will be distributed is defined. If a value is present, it is utilized for the running configuration. If not, the maximum number of physical cores is determined using the command in Listing 4.2 and is adopted as the configuration. When specifying the number of cores for the MPI configuration, it is essential to note that the number of physical cores, not logical ones, must be used. This distinction is critical because if a number larger than the available physical cores is selected, NS3 will encounter an error during the setting of the MPI configuration. The failure manifests in the following error message `There are not enough slots available in the system to satisfy the slots requested by the application`. This approach is based on findings from Section 5.5.1, highlighting the superior performance of multi-core MPI-based execution over single-core execution. Lastly, the optional parameters, `logPings` and `logGraphviz`, are examined. If not present, they passed to the simulation with a default value of false.

```
lscpu | grep "\^Core(s) per socket:" | awk '\{print $4\}\'
```

Listing 4.2: Determination of Maximal Number of MPI Cores

After parsing, the simulation is initiated using the command depicted in Listing 4.3. All parsed configuration properties are passed along as custom program arguments, except for the number of cores used to establish the MPI profile, as shown in Line 13 of Listing 4.3.

```
1    ./$ns3_directory/ns3 run \
2      "eddd \
3    --networkFactor=$network_factor \
4    --clientFactor=$client_factor \
5    --backboneLatencyFactor=$backbone_latency_factor \
6    --regionalLatencyFactor=$regional_latency_factor \
7    --localLatencyFactor=$local_latency_factor \
8    --backboneBandwidthGbps=$backbone_bandwidth_gbps \
9    --regionalBandwidthGbps=$regional_bandwidth_gbps \
10   --localBandwidthGbps=$local_bandwidth_gbps \
11   --clientBandwidthGbps=$client_bandwidth_gbps \
12   --logPings=$log_pings \
13   --logGraphviz=$log_graphviz \
14   --attackType=$attack_type \
15   --country=$(echo "$countries" | jq -c)" \
16     --command-template="mpiexec -np $cores %s"
```

Listing 4.3: Running the Simulation

Upon completion of the simulation, the resulting PCAP files are extracted and placed in an output folder. If the configuration specifies that only packet headers are to be retained, the PCAP files are opened, and the payload is dropped using `tshark` (*cf.* Listing 4.4).

```
if [ "$remove_pcap_payload" == true ]; then
  mkdir -p ShortDatasets
  for file in Datasets/*.pcap; do
    filename=$(basename "$file")
    tshark -r Datasets/"$filename" -w ShortDatasets/"$filename" -Y "not tcp.payload"
  done
fi
```

Listing 4.4: Removing Payload From Dataset

4.3 Simulator

Situated at the core of the EDDD architecture is the simulator. The primary file is the control center, governing the overall NS3 setup and simulation procedure.

At the outset, the main file acquires the simulation configuration from the controller, as delineated in Section 4.2. The configuration details are transmitted to the simulation via program arguments, which ensure that the simulation parameters are properly initialized and adhered to throughout the simulation process.

The following properties are passed onto the simulation:

- networkFactor
- clientFactor
- backboneLatencyFactor
- regionalLatencyFactor
- localLatencyFactor
- backboneBandwidthGbps
- regionalBandwidthGbps
- localBandwidthGbps
- clientBandwidthGbps
- country
- logPings
- logGraphviz
- attackType

Employing custom program arguments to pass the configuration presents a distinct advantage, primarily that NS3 does not require a system rebuild for each novel configuration. This approach significantly streamlines the process, making the system more efficient and less time-consuming.

The initiation of the main application marks the commencement of the NS3 simulation preparation. Initial efforts are directed towards constructing the network, inclusive of clients, routers, connections, routing tables, and more, facilitated by the NetworkTopologyCreator (*cf.* Listing 4.5). A detailed discussion of this component can be found in Section 4.4. Upon successful network creation, both attacking and legitimate clients are easily identifiable, along with the ability to extract the target IP address and the target node for future procedures.

```
NetworkTopologyCreator topologyCreator;
topologyCreator.SetNetworkSpeedFactors(backboneLatencyFactor,
                                       regionalLatencyFactor,
                                       localLatencyFactor);
topologyCreator.SetNetworkBandwidth(backboneBandwidthGbps,
                                    regionalBandwidthGbps,
                                    localBandwidthGbps,
                                            clientBandwidthGbps);
topologyCreator.SetRepresentationFactors(networkFactor, clientFactor);
if (logPings)
{
  topologyCreator.ListPings();
}
if (logGraphviz)
{
  topologyCreator.EnableBackboneGraphviz();
}
topologyCreator.CreateNetwork(parsedCountries);

NodeContainer legitimateClients
  = topologyCreator.GetLegitimateClients();
NodeContainer attackingClients = topologyCreator.GetAttackingClients();

Ptr<Node> targetNode = topologyCreator.GetTargetNode();
Ipv4Address targetAddress = topologyCreator.GetTargetAddress();
```

Listing 4.5: Setup of Topology

Subsequently, attention is turned to the three client parties. The attacking clients are the first to be addressed. Given the existence of two distinct types of attacking clients, the main process installs either the `SynFloodApplication` or the `IcmpFloodApplication` on the attacking clients, contingent upon the program argument (*cf.* Listing 4.6). This choice not only determines the global simulation duration - as only the traffic during an attack is simulated - but also relays parameters such as the seed in the form of the attacking client's index to the application.

```
for (uint32_t i = 0; i < attackingClients.GetN(); ++i)
{
  if (attackType == "syn-flood")
  {
    Ptr<SynFloodAttacker> attackerApp =
        CreateObject<SynFloodAttacker>();
    attackerApp->SetRemote(targetAddress, 8080);
    attackerApp->SetSeed(i);
    attackerApp->SetStartTime(Seconds(0));
    attackerApp->SetStopTime(Seconds(80));

    attackingClients.Get(i)->AddApplication(attackerApp);

    endTime = Seconds(80);
  }
  else
  {
    Ptr<IcmpFloodAttacker> attackerApp =
        CreateObject<IcmpFloodAttacker>(i);
    attackerApp->SetRemote(InetSocketAddress(targetAddress, 8080));
    Time clientStart = attackerApp->GetStartTime();
    Time clientEnd = attackerApp->GetEndTime();
    attackerApp->SetStartTime(clientStart);
    attackerApp->SetStopTime(clientEnd);

    attackingClients.Get(i)->AddApplication(attackerApp);

    if (clientEnd > endTime)
    {
      endTime = clientEnd;
    }
  }
}
```

Listing 4.6: Attacker Setup

By checking the attack type and creating specific attacker applications accordingly, Listing 4.6 grants flexibility to add new attack types. Both applications also incorporate randomness in determining the start and end times of attacks based on probability distributions. This randomness adds variability to the attack patterns, making them more realistic and challenging to detect or mitigate. Therefore, Listing 4.6 tracks the maximum end time (`endTime` variable) to ensure that the simulation runs for a sufficient duration to cover all attacks. This ensures comprehensive testing of the network's resilience.

Next is the configuration of the target client. A sink application is installed on this node, ensuring efficient socket functionality without the risk of buffer overflow (*cf.* Listing 4.7). This is achieved by flushing arriving packets into a sink, eliminating the need for additional processing.

```
PacketSinkHelper sinkHelper("ns3::TcpSocketFactory",
                InetSocketAddress(Ipv4Address::GetAny(), 8080));
ApplicationContainer sinkApp = sinkHelper.Install(targetNode);
sinkApp.Start(Seconds(0));
sinkApp.Stop(endTime);
```

Listing 4.7: Target Setup

In the final step, the `LegitimateTrafficApplication` is installed on the legitimate client nodes, with all the relevant parameters passed on to the application (*cf.* Listing 4.8). This systematic approach ensures a well-structured and meticulously planned simulation, setting the stage for a comprehensive and insightful analysis.

```
for (uint32_t i = 0; i < legitimateClients.GetN(); ++i)
{
  Ptr<LegitimateTrafficApplication> legitimateTrafficApp =
    CreateObject<LegitimateTrafficApplication>();
  legitimateTrafficApp->SetRemote(InetSocketAddress(targetAddress, 8080)
   );
  legitimateTrafficApp->SetSeed(i);

  legitimateClients.Get(i)->AddApplication(legitimateTrafficApp);

  legitimateTrafficApp->SetStartTime(Seconds(0));
  legitimateTrafficApp->SetStopTime(endTime);
}
```

Listing 4.8: Legitimate Client Setup

Following the setup phase, the program records the duration it took to establish the simulation construct for the packet flow. This metric is essential for analysis and provides the user with a sense of the network's magnitude that is aimed to be simulated.

Subsequently, the NS3 simulation is triggered. Upon the conclusion of the simulation, the duration required to simulate all packets is again logged for analysis purposes. Such a systematic logging approach aids in understanding the simulation process, enabling a more comprehensive evaluation with critical insights into the computational demands of the simulation, which is particularly useful when dealing with expansive networks and complex scenarios.

The ensuing sections offer a more in-depth exploration of the `NetworkTopologyCreator`, the `LegitimateTrafficApplication`, and both DDoS Traffic Applications. These subcomponents are integral to the simulation process and significant to the system's architecture. The description highlights the underpinning principles guiding their functionality within the NS3 simulation environment.

4.4 Topology

In developing a solution that accurately represents network topologies, it is essential to consider the underlying principles discussed in Section 2.1.3, particularly latency factors as highlighted by [50]. The `NetworkTopologyCreator` incorporates configurable factors for each network level to ensure a comprehensive and adaptable representation: backbone, regional, and local. It is possible to address the latency variations occurring within different network topologies by allowing customization of these factors. Consequently, this flexibility enables a more precise representation of the desired network.

Moreover, the number of nodes created per country can be adjusted. Since not every user in a country accesses the target system, simulating 100% of network coverage and end devices per country is inefficient (assuming one device per inhabitant, as outlined in Section 3.3). The `subnetRepresentationFactor` determines the proportion of subnets (regional and local) represented below each backbone router. For instance, a factor of 1 implies that every area of the country (simplified as a square, as defined in Section 3.3) is covered by a subnet. In contrast, a factor of 0.5 indicates that only 50% of the area covered by backbone routers is covered by regional routers, and only 50% of their area is covered by local routers. Finally, the `clientRepresentationFactor` controls the number of end devices within the simulated local networks' coverage area. A factor of 1 signifies that 100% of the average population/devices per area in the regions covered by the simulated subnets is represented in the simulation.

As described in Section 3.3, the simulated network comprises countries containing backbone routers serving regional networks. In turn, these regional networks consist of local networks consisting of local routers to which the end clients are connected. This section delves into constructing these topologies and how the different layers are connected.

4.4.1 Routers and Service Area

The service area of each network level, including the country, regional network, local network, or access network, is defined by a square area. The user determines the area of countries, while the area of the regional network under a backbone router is a fraction of the country's total area.

The area of a regional network can be defined as:

$$regional\ area = \frac{number\ of\ backbone\ routers}{country\ area} \tag{4.1}$$

The area a single regional router covers with its local network below is set to $900km^2$ in the code. This service area of a regional router is just an assumption and not based on actual information as real world distributions vary depending on the geographic features of an area. Consequently, the number of regional routers in a regional network under a backbone router is determined as follows:

$$number\ of\ regional\ routers = \frac{900km^2}{regional\ area} \tag{4.2}$$

The amount of local routers within the network beneath a regional router is defined similarly:

$$number\ of\ local\ routers = \frac{4km^2}{900km^2} \tag{4.3}$$

Lastly, the country's population density dictates the number of clients connected to a single local router. This relationship ensures that the distribution of clients within the network accurately represents the population distribution within the simulated area.

4.4.2 Placing Nodes

The procedure for distributing router nodes within an area remains consistent across all network layers, including the backbone, regional, and local layers. Each area is interpreted as a square, and nodes are distributed randomly within the square of the area they are placed in.

A deterministic randomness approach with a seed is chosen for the system to be reproducible. This strategy ensures that the placement of nodes, while appearing random, remains consistent and replicable across multiple simulations. As such, by using a deterministic probability function with a seed, the same node distribution can be replicated, ensuring that the simulation is consistent and reproducible. In addition, it has the benefit that any observed effects or performance variations can be attributed to the specific protocols or algorithms being studied rather than random fluctuations in node positions. Listing 4.9 provides insight into the node placement process within a given area.

```cpp
struct PositionedNode
{
  int id;
  Vector2D position;
};

std::vector<NetworkTopologyCreator::PositionedNode>
NetworkTopologyCreator::PositionNodesRandomly(uint32_t numberOfNodes,
    double area, uint32_t seed)
{
  std::mt19937 rng(seed);
  std::uniform_real_distribution<double> dist(0, std::sqrt(area));

  std::vector<PositionedNode> nodes(numberOfNodes);
  for (uint32_t i = 0; i < numberOfNodes; i++)
  {
    nodes[i] = {static_cast<int>(i), {dist(rng), dist(rng)}};
  }
  return nodes;
}
```

Listing 4.9: Node Placement Within a Given Area

In the function `PositionNodesRandomly`, the number of nodes to be placed and the area in which they are to be placed are taken as input parameters. The placement algorithm ensures that the same outcome is achieved with every execution of the program. A seed is passed to the function to generate these random positions, which are determined by the number of nodes to place and the area in which they are placed. Both properties are dependent on the countries and the topology level. This guarantees some variations in node placements. Ultimately, the positioned nodes, along with their coordinates, are returned.

As for the placement of clients, it is important to note that no further placement is made in the simulation beyond the local router level. In the simulation, a direct connection exists between a local router and a client. The local router and the connected clients comprise the access network. The area of an access network is so small that the distance between one client and the local router would not significantly impact latency compared to another client within the same access network.

4.4.3 Connecting Nodes

As described in Section 4.4.2, nodes are placed two-dimensionally within a square area in this network model. To ensure network connectivity, these nodes must be connected while satisfying two key requirements to represent a realistic network topology:

- guarantee all-connectedness: each node can reach any other node within the network through a series of connections
- connections should not cross: a connection between nodes A and B should never intersect with a connection between nodes C and D

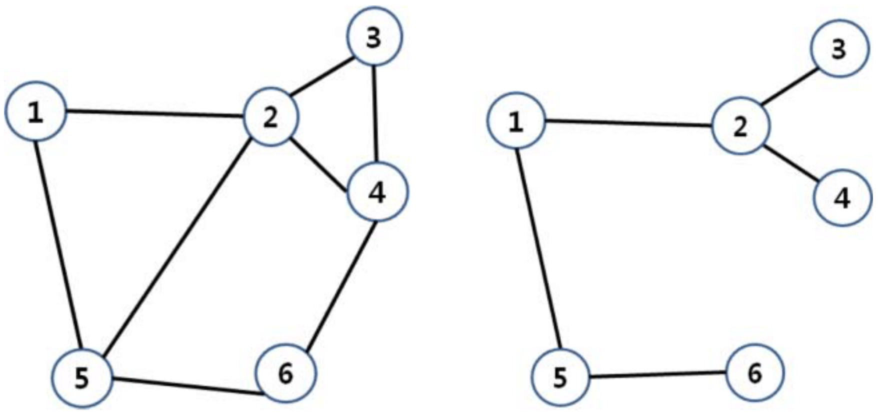


Figure 4.2: Triangulation Comparison: Gabriel Graph (left) vs. RNG (right) [38]

Graph theory can be employed to create such networks, offering multiple approaches to achieve the desired outcome. Minimum Spanning Trees (MSTs) represent one option, but they often exhibit a low edge count, translating to fewer connections and potentially requiring numerous hops to travel between nodes. Furthermore, MSTs lack redundancy or alternate routes, making them less realistic.

Triangulation serves as an alternative approach, fulfilling the requirements above. Three variants, Delaunay triangulation, Gabriel graphs, and Relative Neighborhood Graphs (RNG), implement triangulation to varying extents. Delaunay triangulation maximizes the minimum angle of all triangles within the network. At the same time, Gabriel graphs and RNGs impose distance-based constraints on the connections, resulting in sparser networks that maintain connectivity and non-crossing properties.

Figure 4.2 compares the two sparser triangulation network approaches: the Gabriel graph and RNG. Upon examination, it becomes evident that an RNG can encounter similar issues as an MST, such as requiring numerous hops to traverse from one node to another within the network. Consequently, the Gabriel graph algorithm has been selected as the preferred method for connecting nodes within a network due to its more desirable properties in addressing the issues mentioned above.

Establishing connections between nodes begins with a theoretical evaluation employing the Gabriel Graph Algorithm. After the potential edges have been ascertained, they are instantiated within the NS3 framework using point-to-point connections. Listing 4.10 delineates the connection creation process at a backbone network level. The distance between the two nodes intended to be connected is calculated in the initial phase. This distance is then used to derive the latency value that passed to the point-to-point connection via the *ChannelAttribute Delay*.

```
for (auto& edge : edges)
{ // calculate distance between nodes and set the latency (delay)
  // of the connections appropriately
  double dist =
  CalculateDistance(nodes.at(edge.first).position, nodes.at(edge.second)
   .position);
  totalDistance += dist;
  std::stringstream delay;
8    delay << (dist * m_backbone_speed_km_per_ms) << "ms";
9    p2pHelper.SetChannelAttribute("Delay", StringValue(delay.str()));
10
11   // make connection and assign ip addresses
12   NetDeviceContainer devices;
13   devices.Add(
14   p2pHelper.Install(backboneNodes.Get(edge.first), backboneNodes.Get(
       edge.second)));
15   address.Assign(devices);
16   address.NewNetwork();
17   backboneDevices.Add(devices);
18 }
```

Listing 4.10: Point-To-Point Connections and IP Assigning

After establishing the point-to-point connection, IP addresses must be assigned to the devices at the ends of such connections. In the NS3 environment, every node possesses multiple devices, each corresponding to a point-to-point connection with another node. These devices necessitate the assignment of IP addresses, enabling NS3 to populate routing tables accurately. NS3 mandates that network devices that have IP addresses of the same subnet assigned to them must be connected by a point-to-point connection. The most straightforward resolution adhering to this requirement manifests as a continual sequence of IP allocations, where each successive point-to-point connection corresponds to an incremented subnet. This can be observed in Listing 4.10 Line 14–16, where the pairs of each connection are assigned IP addresses of the same subnet, whereas afterward the subnet is incremented.

4.4.4 Connecting to Sub-Networks

Per Section 4.4.1, every router has an underlying subnet that covers a specific area. Figure 4.3 depicts this concept, displaying the area with nodes distributed randomly following the process described in Section 4.4.2. The diamond-shaped `r` represents the router one level above, accountable for managing the allocated area. This router connects to the three closest nodes within the subnet as part of the design. Placing the router at the center of the area might result in connections from the router to nodes crossing other node connections. To avoid this issue, the three most central nodes are selected as candidates for connection. Subsequently, the router is positioned in the center of these chosen nodes.

Utilizing the 2D placement of the router that serves a particular area, measuring the distance between the router and the nodes it connects to becomes advantageous. By determining this distance, one can calculate the latency a signal experiences as it travels from the router to the node. As outlined, bandwidth and latency factors are defined for each level. In this case, the bandwidth and latency factors of the subnet are applied to these point-to-point connections.

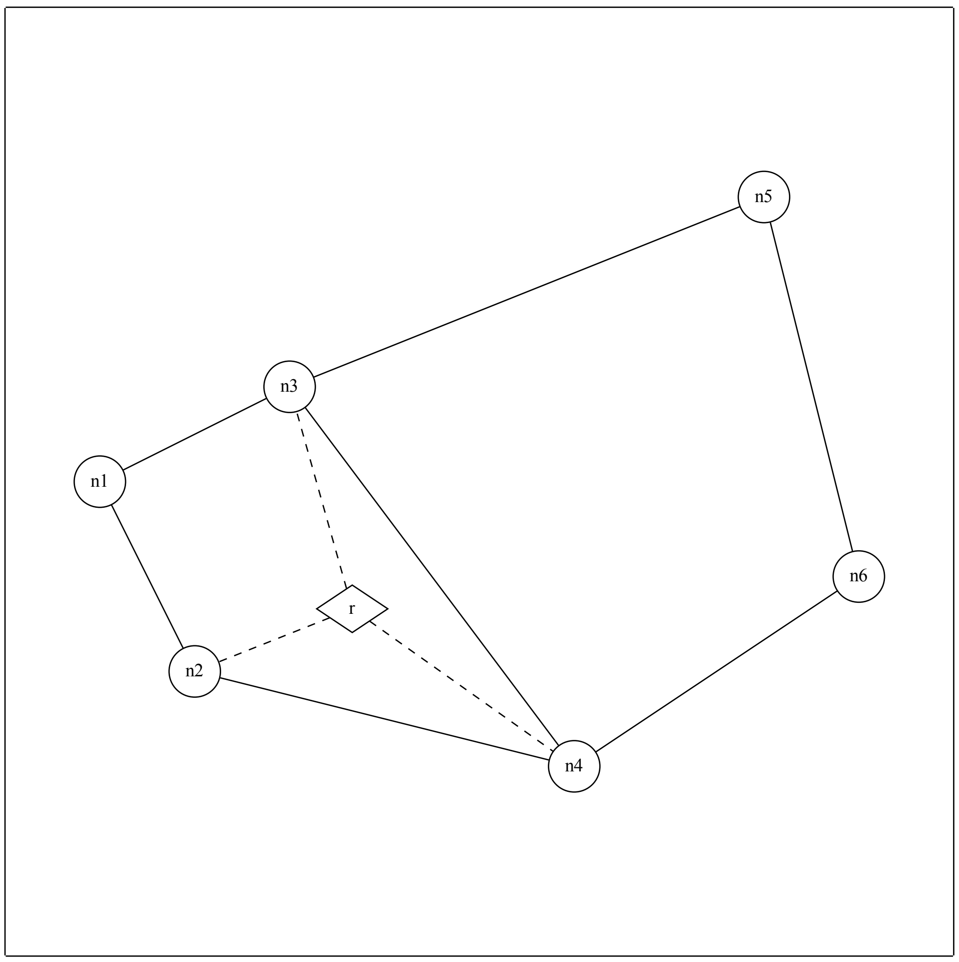


Figure 4.3: Router Connections to Subnet Nodes

4.4.5 Connecting Countries

As described in Section 3.3, countries are connected to their neighboring countries based on the topology configuration. Initially, the border nodes of a country are selected. Listing 4.11 demonstrates an approach for selecting border nodes that is both efficient and effective. First, the nodes are sorted based on their polar angle concerning the bottom-left starting node. This sorting strategy simplifies the subsequent process of identifying border nodes. Finally, Graham's scan algorithm is employed, chosen for its ability to efficiently determine convex hulls, allowing for identifying border nodes in the network.

```
std::vector<NetworkTopologyCreator::PositionedNode>
NetworkTopologyCreator::FindBorderNodes(const std::vector<PositionedNode
    >& nodes)
{
  // Find the bottom-left node as the starting point
  const PositionedNode* start = &nodes[0];
  for (const auto& node : nodes)
  {
    if (node.position.y < start->position.y ||
        (node.position.y == start->position.y
            && node.position.x < start->position.x))
    {
        start = &node;
    }
  }

  // Sort the nodes by polar angle with respect to the starting node
  std::vector<const PositionedNode*> sortedNodes;
  sortedNodes.reserve(nodes.size());
  for (const auto& node : nodes)
  {
    sortedNodes.push_back(&node);
  }
  std::sort(sortedNodes.begin(),
      sortedNodes.end(),
      [start](const PositionedNode* a, const PositionedNode* b)
  {
    double cp = crossProduct(start->position, a->position, b->position);
    if (cp == 0)
    {
      return CalculateDistanceSquared(start->position, a->position) <
             CalculateDistanceSquared(start->position, b->position);
    }
      return cp > 0;
  });

  // Find the border nodes using Graham's Scan algorithm
  std::vector<PositionedNode> borderNodes = {*start};
  for (const auto& node : sortedNodes)
  {
    while (borderNodes.size() >= 2
           && crossProduct(borderNodes[borderNodes.size() - 2].position,
                           borderNodes.back().position,
                           node->position) <= 0)
    {
      borderNodes.pop_back();
    }
    borderNodes.push_back(*node);
  }

  return borderNodes;
}
```

Listing 4.11: Select Border Nodes

From these border nodes, the k ones furthest apart are selected as demonstrated in Listing 4.12. In this context, k represents the number of neighboring countries. In most cases, this equals the number of countries within the `neighbors` parameter plus one, as the country needs to connect to the previously created one. When creating a country, a border node from the previous country is always passed on to which the new country must connect. The only exception is the root country, representing the target country, as no previous country exists. The function `findFurthestApartNodes` iteratively compares the distances between nodes to determine the furthest apart nodes, which are then added to the list of selected nodes. If the desired number of connections exceeds the number of border nodes, it selects all the nodes and repeats the selection process. This approach ensures an optimal distribution of connections among border nodes, contributing to a more realistic network topology representation.

```
std::vector<NetworkTopologyCreator::PositionedNode>
NetworkTopologyCreator::FindFurthestApartNodes(
    const std::vector<PositionedNode>& borderNodes,
    ulong numberOfConnections)
{
  if (numberOfConnections <= 0)
  {
    return {};
  }

  ulong borderNodeCount = borderNodes.size();
  std::vector<PositionedNode> selectedNodes;

  // If numberOfConnections is greater than borderNodeCount, select all
   nodes, then repeat the selection process
  while (numberOfConnections > borderNodeCount)
  {
    selectedNodes.insert(selectedNodes.end(), borderNodes.begin(),
   borderNodes.end());
    numberOfConnections -= borderNodeCount;
  }

  std::vector<bool> used(borderNodeCount, false);
  int currentNodeIndex = 0;
  used[currentNodeIndex] = true;
  selectedNodes.push_back(borderNodes[currentNodeIndex]);
  numberOfConnections--;

  while (numberOfConnections > 0)
  {
    double maxDistance = -1;
    int maxIndex = -1;

    for (ulong i = 0; i < borderNodeCount; i++)
    {
      if (!used[i])
      {
        double minDistance = std::numeric_limits<double>::max();
        for (const auto& selectedNode : selectedNodes)
        {
          double currentDistance =
                CalculateDistance(borderNodes[i].position,
                                  selectedNode.position);
          if (currentDistance < minDistance)
          {
            minDistance = currentDistance;
          }
        }

        if (minDistance > maxDistance)
        {
          maxDistance = minDistance;
          maxIndex = i;
        }
      }
    }

    used[maxIndex] = true;
    selectedNodes.push_back(borderNodes[maxIndex]);
    numberOfConnections--;
  }

  return selectedNodes;
}
```

Listing 4.12: Select Country Connection Nodes

A connection is established for the border node connecting to the previous country. Since a node from the previous country is passed on to the new country, it is possible to create this backbone connection. The bandwidth for this connection is the same as the backbone bandwidth within a single country, and since the distance between those country connecting nodes cannot be ascertained de the latency is determined by the average connection distance among the backbone nodes within a country (*cf.* Listing 4.13).

The connection to the new neighbors cannot be made yet, as they do not exist. Therefore, a new country is created, and the border node from the current country, which should connect to this new neighbor, is passed along.

4.4.6 PCAP Nodes

PCAP recording is conducted at the backbone router level within EDDD. NS3 facilitates PCAP recording at the device level associated with network nodes. EDDD mandates the inclusion of a parameter, `pcapEnabled`, for each country represented in the `connectedCountries` object. If this parameter is set to true, PCAP recording is activated for all backbone devices associated with that particular country. To facilitate the identification of the generated pcap files, they are named following the schema: `backbone-node-[country_name]-[node_id]-[device_id].pcap`.

Section 4.4.5 explicated that countries are only connected to the subsequent country once the latter has been instantiated. This implies that one of the devices created in a point-to-point connection between two countries does not belong to the current country, but rather, to the preceding one. This peculiarity must be taken into account while enabling the PCAP recording (*cf.* Listing4.13).

```cpp
const bool pcapEnabled = c.enablePcap && MpiInterface::GetSystemId() ==
    m_current_system_id;

if (hasPreviousCountryNode)
{ // make the connection to the previous country

  // set the latency (delay) according to the average distance of the
    connections in the current country
  std::stringstream delay;
  delay << (averageDistance * m_backbone_speed_km_per_ms) << "ms";
  p2pHelper.SetChannelAttribute("Delay", StringValue(delay.str()));

  // make connection and assign ip addresses
  NetDeviceContainer countryConnectionDevices =
    (p2pHelper.Install(
          backboneNodes.Get(connectingBorderNodes.at(0).id),
          previousCountryNode.value()));

  address.Assign(countryConnectionDevices);
  address.NewNetwork();

  if (pcapEnabled)
  { // enable pcap on the net device, that is in the current country
    p2pHelper.EnablePcap("backbone-node-" + country.name,
    countryConnectionDevices.Get(0));
  }

  if (prevCountryPcapEnabled && previousCountryName.has_value())
  { // enable pcap on the net device, that is in the previous country
    p2pHelper.EnablePcap("backbone-node-" + previousCountryName.value(),
                countryConnectionDevices.Get(1));
  }
}

if (pcapEnabled)
{ // enable pcap on all backbone devices of the current country
  p2pHelper.EnablePcap("backbone-node-" + country.name, backboneDevices)
    ;
}
```

Listing 4.13: Connecting Countries and PCAP Enabling

If PCAP recording was enabled for the previous country, PCAP recording must also be initiated for the connecting device associated with the previous country. Conversely, if PCAP recordings are enabled for the current country, they must only be activated for the device associated with the current country. Each pcap enabler must also be annotated with the correct country name, ensuring the resulting PCAP files can be accurately attributed to their respective countries.

Moreover, PCAP recording is enabled for traffic arriving at the target node. Given that the target node maintains only a single device (its connection to the router) merely one PCAP file is produced. This file encapsulates all packets arriving at the target, serving as a valuable resource for validation. It can be used for comparison with existing centralized datasets, providing a benchmark for evaluation. Utilizing this file also facilitates a more accurate assessment of the load imposed on the target by both the attack and legitimate traffic, thereby enhancing the understanding of network dynamics under a DDoS attack.

4.5 Legitimate Traffic

As discussed in Section 3.4, legitimate traffic is modeled based on the CIC dataset [28]. However, it is important to note that recreating the data exactly is not feasible. Consequently, selected features have been decided upon in Section 3.4, and a piecewise probability approach has been introduced to address this issue. By utilizing an array of piecewise probabilities at 10% quantile intervals, it is possible to obtain a distribution that closely resembles the original one.

Listing 4.14 demonstrates the implementation of this approach, where the input consists of an array of outcomes at different quantiles and a seed. This seed from the invocation of this method is then combined with `m_seed`, the seed unique to the instance of the application. This combination ensures that the results are different for each invocation of the method and different comparing individual application instances. It also assures that the results are deterministic, meaning that each execution of the program yields the same outcome. Nevertheless, the specific values chosen from this distribution are random.

```
double
LegitimateTrafficApplication::GetRandomDistributionNumber(
  std::array<double, 11> piecewiseProbability,
  uint32_t seed)
{ // Calculate a deterministic random number based on a quantile
    distribution
  // (piecewiseProbability) and a seed

  std::mt19937 gen(seed * m_seed);
  std::uniform_real_distribution<> dis(0, 1);

  double u = dis(gen);

  int minIndex = static_cast<int>(u * 10);
  if (minIndex == 11)
    minIndex = 10;

  return piecewiseProbability.at(minIndex) +
        (piecewiseProbability.at(minIndex + 1)
         - piecewiseProbability.at(minIndex)) * u;
}
```

Listing 4.14: Random Value According to a Peacewise Probability

The legitimate traffic controller is developed as an extension of the application class within NS3. Consequently, it possesses the `StartApplication` and `EndApplication` methods, which dictate when the application is activated and terminated. Since benign traffic is omnipresent throughout the simulation, the application's start and end timestamps correspond to the simulation's global commencement and conclusion.

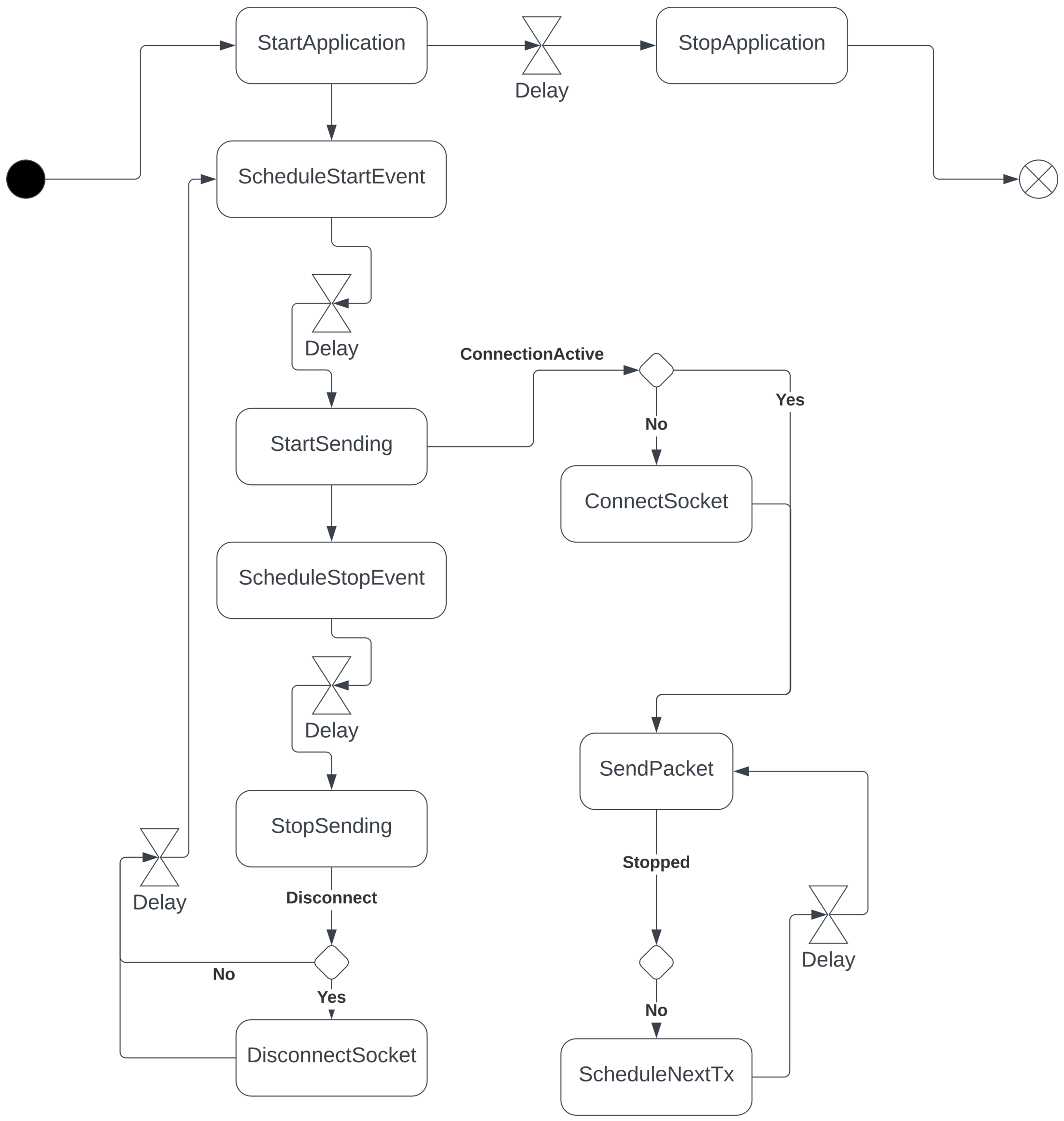


Figure 4.4: Activity Diagram of Legitimate Traffic Application

Upon activation of the application, the process aligns with the activities delineated in Figure 4.4. Initially, the flow is in an idle phase. The duration of this idle phase is established using the piecewise probability defined in Section 3.4. Subsequently, the progression within this idle phase is determined through a random linear progress distribution. These determinations are randomly generated using a seed that ensures consistent results across each execution of the program. Following a delay equivalent to the remaining idle phase's duration, a `StartSending` event is scheduled, as demonstrated in Listing 4.15.

```
void
LegitimateTrafficApplication::StartApplication()
{
  CancelEvents();

  double waitSeconds = this->GetRandomDistributionNumber(
   DISTRIBUTION_TIME_BETWEEN_FLOWS, m_seed);


  std::mt19937 gen(m_seed);
  std::uniform_real_distribution<> dis(0, 1);
  double waitProgress = dis(gen);

  m_startStopEvent =
    Simulator::Schedule(Seconds(waitSeconds * waitProgress),
         &LegitimateTrafficApplication::StartSending, this);
}
```

Listing 4.15: `StartApplication` Method of the `LegitimateTrafficApplication`

The `StartSending` method first verifies the existence of a TCP connection with the target. In the absence of an established TCP connection, it initiates a handshake. This method also instigates the scheduling of a transaction sending event via the `ScheduleNextTx` method and a `StopSending` event through the `ScheduleStopEvent` method.

The `ScheduleStopEvent` method selects a deterministically random value from the flow duration distribution, as described in Section 3.4. A `StopSending` event is scheduled upon reaching this duration. Conversely, the `ScheduleNextTx` method schedules the subsequent `SendPacket` event after a certain delay. This delay is determined through a random deterministic distribution, based on a packets-per-second probability. The random selection of a packets-per-second value within this method varies on each execution, resulting in a variable delay to the next `SendPacket` event, as it is defined as $\frac{1}{Packets/s}$.

The `SendPacket` method's primary function is dispatching packets. A packet length is randomly selected for every packet to be sent following the distribution defined in Section 3.4. Packets with a length of zero represent those without a payload. Upon sending a packet via the socket, a new `SendPacket` event is scheduled via the `ScheduleNextTx` method.

Once a scheduled `StopSending` event is triggered, the `StopSending` method is invoked. All scheduled `SendPacket` events are canceled upon reaching the end of the flow. As indicated in Section 3.4, the conclusion of a flow does not necessarily signify the termination of a TCP connection; this occurs only in 6% of instances. Therefore, within the `StopSending` method, the socket is disconnected with a 6% probability. Following this, a new `StartSending` event is scheduled via the `ScheduleStartEvent` method, which sets a new start event after a random idle time as per the distribution described in Section 3.4.

Upon reaching the scheduled end timestamp of the application, the application cancels all events, encompassing the `SendPacket` events and the events to start and stop a flow.

The implementation of the legitimate traffic controller is made available as an application for an NS3 simulation. In EDDD, a `LegitimateTrafficApplication` is established and attached for every legitimate client (*cf.* Listing 4.8). Necessary properties for the application are a seed to incorporate a degree of variation among the clients, which is utilized as `m_seed` for each stochastic decision that needs to be made, a remote socket address to which the traffic is sent, and the start and end times of the application. It's crucial to note that the start time of the application does not represent the point at which the client begins to send data but rather when the application is loaded and the first flow is scheduled. In the instance of EDDD, this is intended to occur at the beginning of the simulation at time 0. The end time is an attacking client's final interaction, which indicates that the simulation concludes when the attack is finished.

4.6 Attack Traffic

Dedicated attack traffic applications are required to model DDoS traffic for the simulation. EDDD currently supports two types of DDoS attacks, necessitating the creation of two distinct applications: the `IcmpFloodApplication`, which simulates an ICMP flood attack, and the `SynFloodApplication`, which generates a SYN flood attack.

Both applications require certain parameters to be specified upon creation. Foremost among these is the remote address, which delineates the socket address to which the traffic will be sent. In addition, a seed available to the applications as `m_seed` is needed to ensure that each client maintains a unique and deterministic random stochastic decision-making process. Furthermore, the start and end times for the applications must be defined. It should be noted that the end time for these applications is of particular importance as it dictates the global end time of the simulation.

The following subsections will provide a more detailed examination of the implementation of these two distinct applications. The unique aspects of their designs, including how they model SYN flood and ICMP flood attacks, are thoroughly discussed.

4.6.1 ICMP Flood Traffic

The IcmpFloodAttacker application is constructed on the foundation of the CAIDA dataset, as elaborated in Section 3.5.1. The yield from this dataset in terms of client-specific behaviour appears rather meager, as once a client joins the attack, it tends to send ICMP packets to the target at a rate of approximately 20 packets per second until it eventually exits the attack, as depicted in Figure 3.13. Consequently, this facilitates a straightforward implementation of the application itself, as demonstrated in Figure 4.5. The crux of this application lies in the determination of the start and end times.

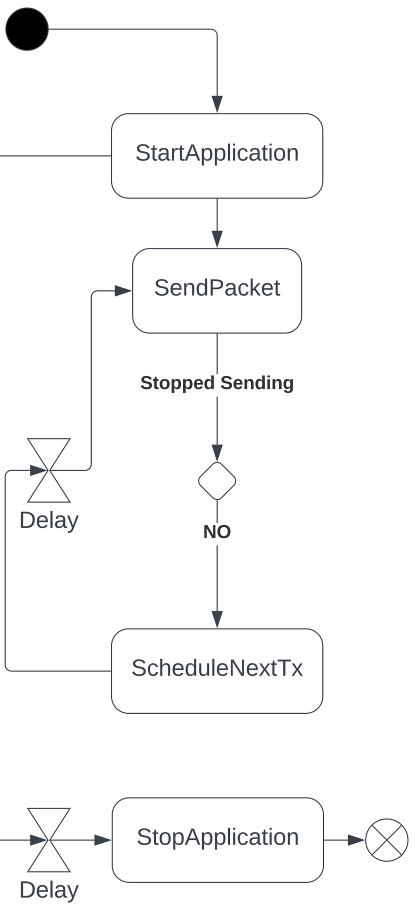


Figure 4.5: Activity Diagram of ICMP Flood Traffic Application

Drawing from the analysis in Section 3.5.1, the probabilities of starting the attack vary across different points in time. For simplification, these probabilities have been grouped into 5-minute buckets. To ascertain the start time for the application, a decision is made regarding the specific 5-minute bucket into which a particular node falls. This decision is derived from a deterministic random choice as demonstrated in Listing 4.16. Subsequently, the precise time is determined by generating another deterministic random number, representing the progress within this bucket. Given that these buckets represent 5-minute intervals, the sum of the bucket and the inter-bucket progress is multiplied by 5 minutes and results in the start time.

```
std::vector<double> start_5min_buckets_probabilities = // Probabilities
int startBucket = SelectProbabilityBucket(
    start_5min_buckets_probabilities, m_seed);

std::mt19937 gen(m_seed * 2);
std::uniform_real_distribution<double> interBucketTime(0, 1);

Time clientStart = (startBucket + interBucketTime(gen)) * Minutes(5);
```

Listing 4.16: Determination of Start Time of `IcmpFloodAttacker` Application

The application's duration is calculated in a similar fashion. The analysis in Section 3.5.1 provided probabilities for 5-minute interval durations. In this case, too, the duration bucket is selected using another seed specific to this instance. The final duration is then calculated by combining the selected duration bucket with a new value of the inter-bucket progress, multiplied by 5 minutes. The required end time for the application can then be inferred by adding the start time to the duration.

```
std::vector<double> duration_5min_buckets_probabilities = //
    Probabilities
int durationBucket = SelectProbabilityBucket(
    duration_5min_buckets_probabilities, m_seed * 3);

Time clientEnd = clientStart + ((durationBucket + interBucketTime(gen))
    * Minutes(5));
```

Listing 4.17: Determination of End Time of `IcmpFloodAttacker` Application

During the application's runtime, the `SendIcmp` method is scheduled, which recursively schedules itself until the application terminates. Within this method, an ICMP Echo packet is formed following the default configuration in the NS3 `Ping` application (*cf.* Listing 4.18). Once the ICMP Echo packet is successfully dispatched, a new packet is subsequently scheduled. This procedure is conducted at an approximate attack rate of 20 packets per second. The rate is not fixed, introducing a degree of variability in the attack pattern, but it predominantly gravitates around this value, mirroring the packet rate identified within the CAIDA dataset. Upon termination of the application, all residual events are canceled to ensure a clean state.

```
void
IcmpFloodAttacker::SendIcmp()
{
  Ptr<Packet> dataPacket = Create<Packet>(56);

  // Create an empty packet
  Ptr<Packet> icmpPacket = Create<Packet>();

  // Using IPv4
  Icmpv4Echo echo;

  // In the Icmpv4Echo the payload is part of the header.
  echo.SetData(dataPacket);

  icmpPacket->AddHeader(echo);
  Icmpv4Header header;
  header.SetType(Icmpv4Header::ICMPV4_ECHO);
  header.SetCode(0);
  icmpPacket->AddHeader(header);

  // Send the Icmp packet using the raw socket
  m_socket->SendTo(icmpPacket, 0, m_remote);

  m_pktSent++;

  ScheduleSend();
}
```

Listing 4.18: `SendIcmp` Method of `IcmpFloodAttacker` Application

4.6.2 Syn Flood Traffic

The SYN Flood Application finds its foundational basis in the SYN Flood dataset from the DDoS Packet Capture Collection, as discussed in Section 3.5.2. It is important to note the inherent complexity in replicating this dataset, especially due to the inability to deduce information about individual clients. However, as illustrated in Figure 3.15, the dataset's distinct phases can be replicated. The activity diagram reveals that upon initiating the application, phase 1 commences via the `StartPhase1` method (*cf.* Figure 4.6). This process triggers the scheduling of the first packet and determines the initial phase's stop event.

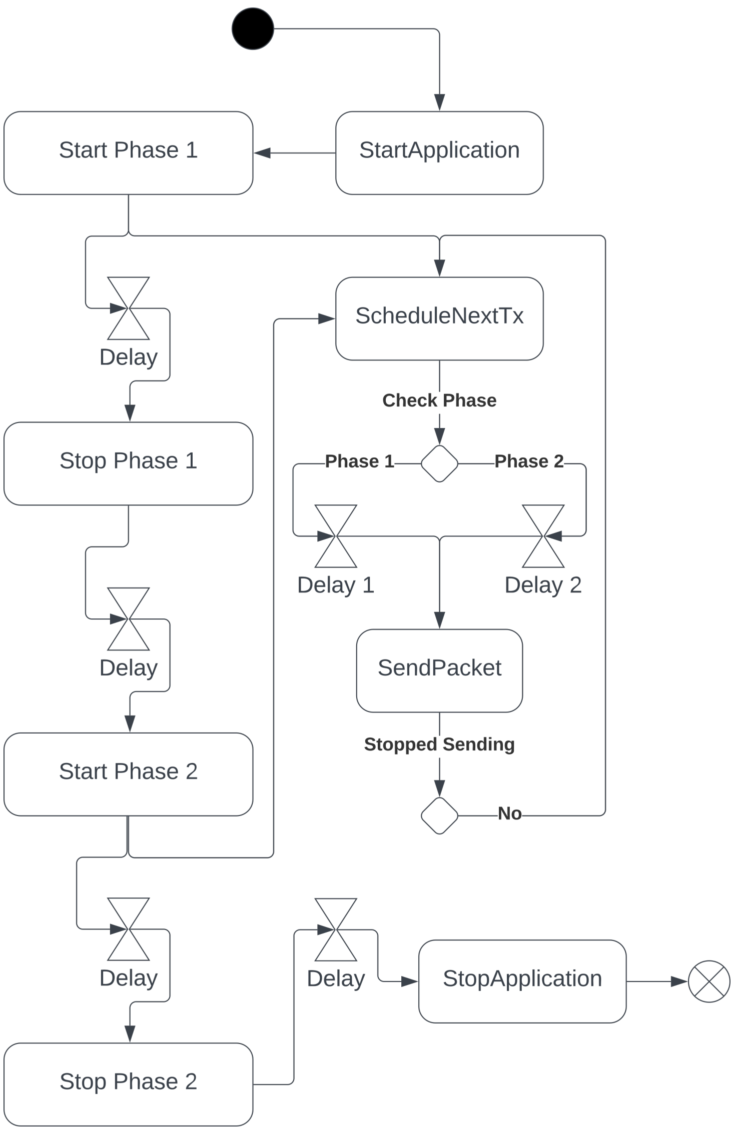


Figure 4.6: Activity Diagram of SYN Flood Traffic Application

Listing 4.19 demonstrates the method of packet scheduling. A distinct packet difference per second during the two phases in the dataset [37] necessitates a similar distinction within the ScheduleSend method. The dataset provides data for total packets per second, leading to an assumption that roughly 400 clients participated in the attack. Considering this, each client would send approximately 20 packets per second at the attack's peak. This approximation aligns with the packets sent per client in ICMP attacks, as described in Section 3.5.1. Therefore, the waiting time before sending a new packet is multiplied by 400 (`ORIGINAL_CLIENTS_ASSUMPTION`).

```cpp
Time
SynFloodAttacker::CalculateNextTime(const std::array<double, 11>& packetPerSecondProbability) const
{ // Calculation of next time a packet should be sent based on quantile distribution of packets per
  // second at target
  double packetsPerSecond = GetRandomDistributionNumber(packetPerSecondProbability, m_pktSent);
  return Seconds((1.0 / packetsPerSecond)
              * ORIGINAL_CLIENTS_ASSUMPTION);
}

void
SynFloodAttacker::ScheduleSend()
{
  Time nextTime;
  if (m_phase1)
  {
    nextTime = CalculateNextTime(DISTRIBUTION_PHASE1_PACKETS_PER_SECOND);
  }
  else
  {
    nextTime = CalculateNextTime(DISTRIBUTION_PHASE2_PACKETS_PER_SECOND);
  }

  NS_LOG_LOGIC("next packet time = " << nextTime.As(Time::S));
  m_sendEvent = Simulator::Schedule(nextTime, &SynFloodAttacker::SendSyn, this);
}
```

Listing 4.19: `ScheduleSend` Method of the `SynFloodAttacker` Application

The method of sending a packet, as shown in the `SendSyn` method in Listing 4.20, first involves constructing the headers before sending the packet via the socket. Given that SYN flood attacks employ random ports, this feature is incorporated into this method. Upon completion of the sending process, the next sending event is scheduled.

```cpp
void
SynFloodAttacker::SendSyn()
{
  // Create a TCP header with the SYN flag set
  TcpHeader tcpHeader;
  tcpHeader.SetFlags(TcpHeader::SYN);
  tcpHeader.SetSourcePort(GetRandomPort());
  tcpHeader.SetDestinationPort(m_remotePort);

  // Create an empty packet
  Ptr<Packet> synPacket = Create<Packet>();

  // Add the header to the packet
  synPacket->AddHeader(tcpHeader);

  // Send the SYN packet using the raw socket
  m_socket->SendTo(synPacket, 0, InetSocketAddress(m_remoteIp,
    m_remotePort));

  m_pktSent += 1;

  ScheduleSend();
}
```

Listing 4.20: `SendSyn` Method of the `SynFloodAttacker` Application

When phase 1 concludes, all lingering events are canceled, and phase 2 is subsequently scheduled. The initiation of the second phase involves invoking a new `ScheduleSend`. However, this phase employs the packets/second ratio from phase 2. Also, upon the commencement of Phase 2, the end of this phase is scheduled. This process involves the cancellation of all events, ensuring the effective clean-up of the application.

In the process of implementation, it emerges that certain intrinsic functionalities in NS3 induce connection resets which cannot be readily disabled. As a consequence, SYN-flood attacks, where unfinished handshakes are a significant aspect, cannot be accurately simulated in NS3 due to premature termination with a reset. This aspect poses a challenge in achieving a faithful representation of this common DDoS attack type within the NS3 environment. To mitigate this issue and ensure the final dataset does not include any unintended RST packets, a filtering mechanism is incorporated into the shell script controller (*cf.* Listing 4.21). This filter effectively sifts out all RST packets during the final phase of data preparation, thereby ensuring that the resultant dataset remains undistorted by these extraneous connection resets.

```
if [[ $attack_type == "syn-flood" ]]; then
  for file in "Datasets"/*.pcap; do
    # Check if file ends with .pcap
    if [[ $file == *.pcap ]]; then
      # Create the output file name by appending "_filtered" to the
  original file name
      output_file="${file%.pcap}_filtered.pcap"

      # Execute the tshark command to filter packets with reset flag
      tshark -r "$file" -Y "tcp.flags.reset==0" -w "$output_file"

      # rename to the old names
      rm "$file"
      mv "$output_file" "$file"
    fi
  done
fi
```

Listing 4.21: Filtering Mechanism for RST Packets

Chapter 5

Evaluation

The evaluation of the EDDD system takes its foundation from a carefully selected base scenario. In this situation, the target, situated in Switzerland, possesses a Western European reach, indicating that this server is designed to serve clients throughout Western Europe. The corresponding configuration in the configuration file, dictating the program's execution, is outlined in Listing 5.1.

```
{
  "networkFactor": 0.025,
  "clientFactor": 0.015,
  "backboneLatencyFactor": 1.25,
  "regionalLatencyFactor": 2.5,
  "localLatencyFactor": 3.75,
  "backboneBandwidthGbps": 1000,
  "regionalBandwidthGbps": 100,
  "localBandwidthGbps": 10,
  "clientBandwidthGbps": 1,
  "countries": {
    "name": "Switzerland",
    "nodes": 3,
    "area": 41285,
    "population": 8700000,
    "enablePcap":true,
    "attackTrafficFactor": 0.7,
    "neighbors": [
      {
        "name": "Germany",
        "nodes": 8,
        "area": 357558,
        "population": 83200000,
        "enablePcap":true,
        "attackTrafficFactor": 0.7,
        "neighbors": [
          {
            "name": "Belgium",
            "nodes": 2,
            "area": 30688,
            "population": 11590000,
            "enablePcap" false,
            "attackTrafficFactor": 0.5,
            "neighbors": [
              {
                "name": "UnitedKingdom",
                "nodes": 7,
                "area": 243610,
                "population": 67330000,
                "enablePcap": false,
                "attackTrafficFactor": 0.3,
                "neighbors": []
              }
            ]
          },
          {
            "name": "Netherlands",
            "nodes": 4,
            "area": 41850,
            "population": 17530000,
            "enablePcap": false,
            "attackTrafficFactor": 0.4,
            "neighbors": []
          }
        ]
      },
      {
        "name": "Italy",
        "nodes": 8,
        "area": 302073,
        "population": 59110000,
        "enablePcap": false,
        "attackTrafficFactor": 0.6,
        "neighbors": []
      },
      {
        "name": "France",
        "nodes": 9,
        "area": 551695,
        "population": 67750000,
        "enablePcap": false,
        "attackTrafficFactor": 0.7,
        "neighbors": [
          {
            "name": "Spain",
            "nodes": 5,
            "area": 505990,
            "population": 47420000,
            "enablePcap": false,
            "attackTrafficFactor": 0.4,
            "neighbors": [
              {
                "name": "Portugal",
                "nodes": 3,
                "area": 92212,
                "population": 10330000,
                "enablePcap": false,
                "attackTrafficFactor": 0.3,
                "neighbors": []
              }
            ]
          }
        ]
      },
      {
        "name": "Austria",
        "nodes": 3,
        "area": 83871,
        "population": 8956000,
        "enablePcap":false,
        "attackTrafficFactor": 0.6,
        "neighbors": []
      }
    ]
  },
  "logPings":false,
  "logGraphviz":false,
  "removePcapPayload":true,
  "attackType": "syn-flood",
  "cores": 10
}
```

Listing 5.1: Configuration for Evalation Scenario

Each country incorporated in this scenario is represented by its respective population size and area, as indicated by Google Feedback [36]. The representation factor for the network is set to `0.025` whereas the factor for the clients is set to a value of `0.015`. The *attackTrafficFactor* is set according to [12] findings that suggest a negative correlation between the ratio of attack traffic compared to legitimate traffic and the geographical distance. The exact factors of each country can be observed in Listing 5.1.

Country	**Routers**			**Clients**	
	Backbone	Regional	Local	Attack	Legitimate
Switzerland	3	3	6	9	8
Germany	8	8	56	124	44
Belgium	2	2	4	15	9
United Kingdom	7	7	35	47	93
The Netherlands	4	4	8	19	29
Italy	8	8	48	90	54
France	9	18	90	116	64
Spain	5	15	75	30	45
Portugal	3	3	15	9	21
Austria	3	3	12	11	13
	52	71	349	470	380

Table 5.1: Generated Country Topology

The resulting topology output from this configuration is summarised in Table 5.1. For each country, the number of backbone routers, regional routers, and local routers, as well as the number of attacking and legitimate clients, are presented. The configuration generates 1'323 nodes, among which 850 are clients; one serves as the target, and the remaining 472 nodes function as routers.

5.1 Topology

The approach of connecting countries has been chosen to resemble a real-world scenario. This method allows for a comprehensible arrangement and definition of countries to symbolize a map with a network infrastructure. By utilizing population and area settings, it is possible to define a country with real-world data.

For the evaluation scenario, several Western European countries have been chosen. In this scenario, a server in Switzerland is the victim of an attack originating from nodes within other European countries. The countries represented in this virtual model include Switzerland, Austria, France, Spain, Portugal, Italy, Germany, the Netherlands, Belgium, and the United Kingdom.

Figure 5.1 presents a graphical representation of the countries displayed as squares in proportion to their actual size. This figure also demonstrates how these countries are connected. This representation does not resemble a real-world map of Western Europe. However, when comparing it with a real map (*cf.* Figure 5.2) displaying the connections made in Figure 5.1, it becomes apparent that the connection paths in the real world are represented in the virtual model.

In addition to merely connecting countries, the network topology exhibits further complexity. Each country contains several backbone routers, and a multi-level network lies beneath each of these routers. This structure closely mirrors real-world network configurations (*cf.* Section 2.1.3). Connecting these nodes makes it possible to send messages from any created clients to the target server. As depicted in Figure 5.1, the target server in this scenario is located in Switzerland.

Sending a packet from any client to the Swiss target server should emulate the behavior of real packets sent from a client in the originating country to the Swiss server. This resemblance can be observed in the number of hops a packet takes before reaching the target and the transmission duration. Figure 5.3 compares real-world ping data [82], representing transmission duration, with data generated within this system using NS3.

Through the latency factor described in Section 4.1, latency can be configured for each network level. Factors 1.25 for the backbone latency (25% more latency than ideal), 2.5 for the regional latency (150% more latency than ideal), and 3.75 for the local latency (275% more latency than ideal) provided the best results in achieving the desired outcome: a ping within a similar range per country as the real-world data. This approach proved effective for most of the represented countries; however, there were outliers. For instance, Austria could not be represented as accurately as other countries. This discrepancy may be due to geographic topologies rendering an equally distributed and evolved network infrastructure less realistic.

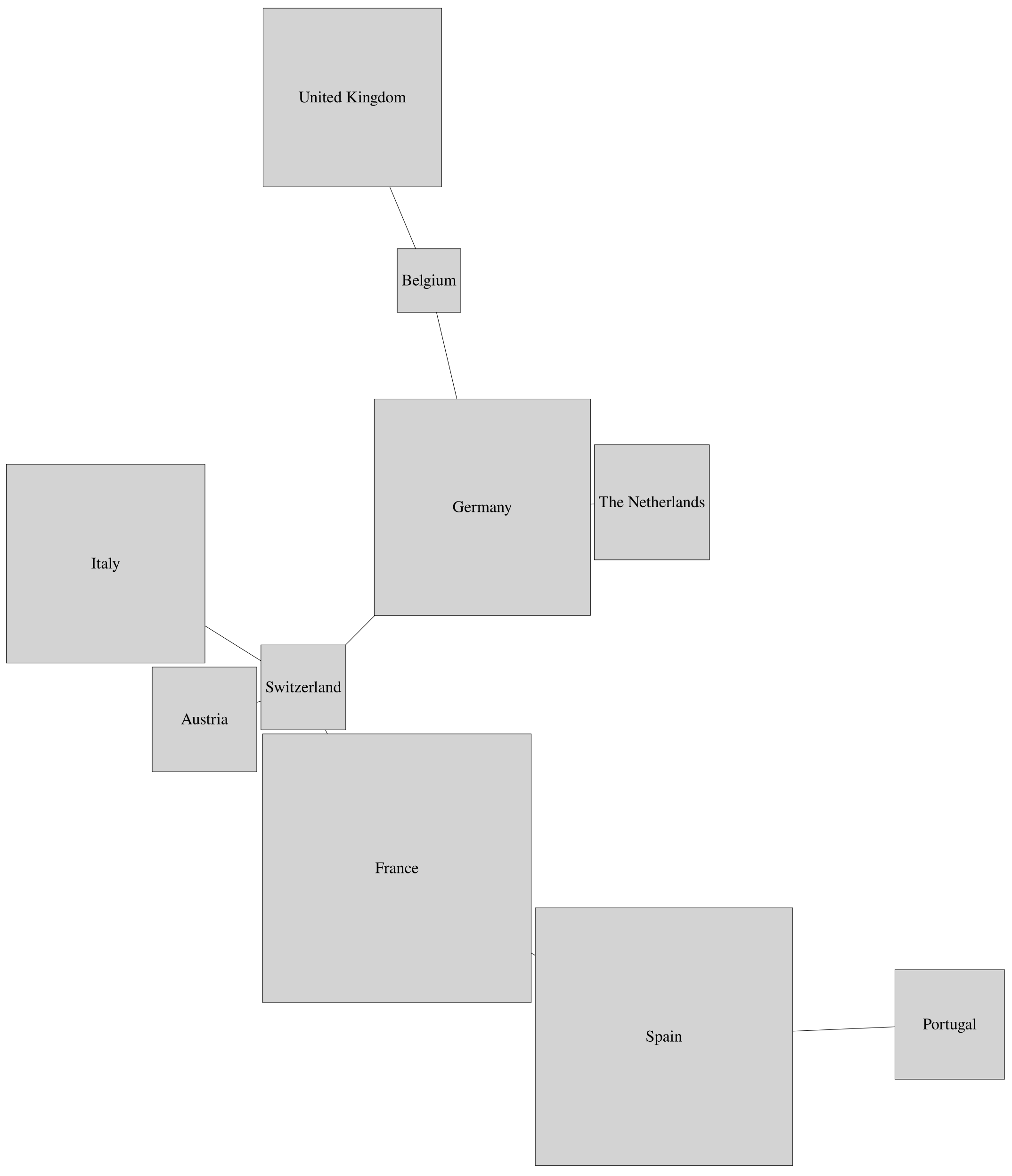


Figure 5.1: Connected Countries as Network Structure

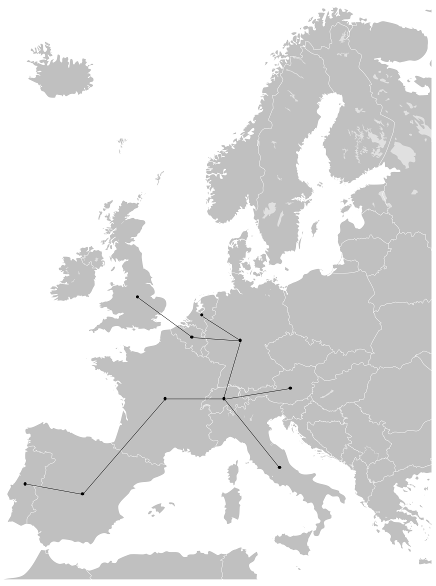


Figure 5.2: Map of Western Europe with Connections of Selected Countries

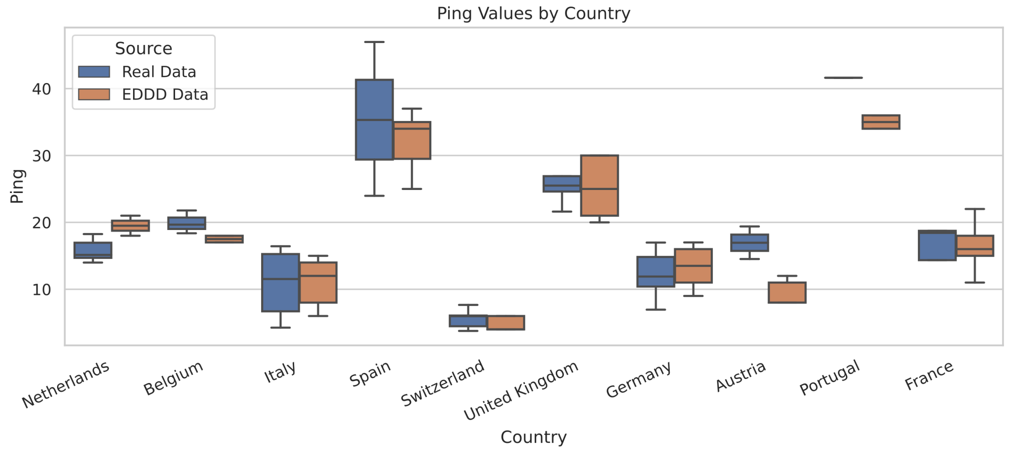


Figure 5.3: Ping Comparison: NS3 Simulation vs. Real Measures

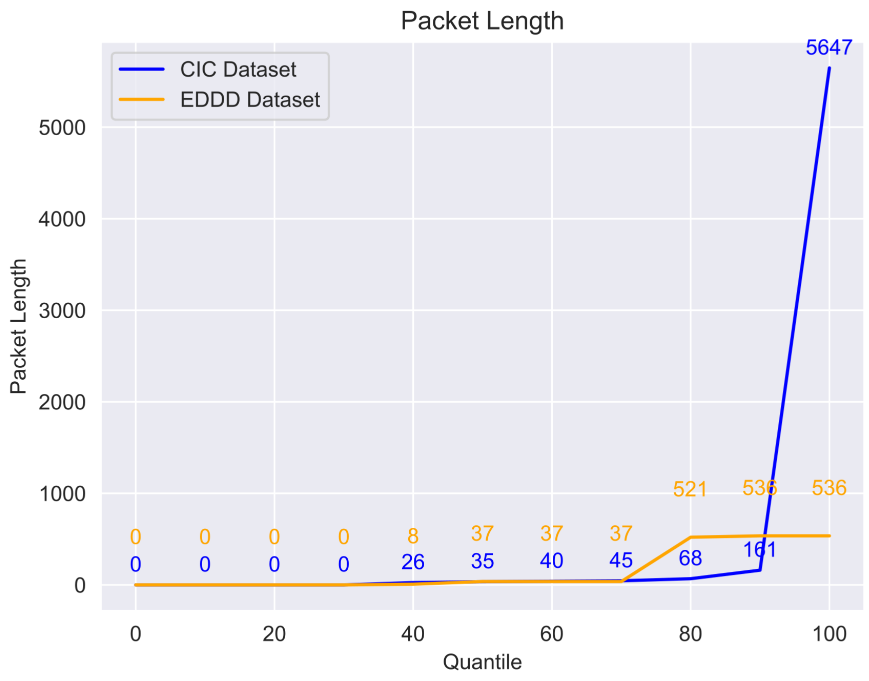


Figure 5.4: CAIDA vs. EDDD: Legitimate Packet Length

5.2 Legitimate Traffic

The modeling of legitimate traffic is based on the traffic labeled benign from the CIC dataset [28]. The subsequent approach is adopted to verify the fidelity of the generated dataset concerning the properties outlined in Section 3.4.

In the scenario detailed at the beginning of Chapter 5, a minor modification is introduced. The main function of the simulator, rather than creating the entire simulation, focuses on assigning the application for legitimate traffic to legitimate clients. With this alteration, the PCAP extracted in the target application can be analyzed without the need to classify arriving packets, as only legitimate traffic is in transit.

Evaluating the tool's effectiveness, however, presents certain complexities. The traffic modeling underlying the dataset under consideration represents data in flows, processed by the CICFlowMeter [19]. When attempting to convert the output from EDDD into flows using the CICFlowMeter, all packets are discarded, resulting in the absence of identifiable flows. A salient reason for this might be the lack of flow simulation in EDDD's design. The traffic simulated is unidirectional, composed of simple packets often devoid of any meaningful payload. For tools aiming to extract flows from realistic bidirectional traffic, the output provided by EDDD might fall short in terms of readiness for further processing. However, modeling complete bidirectional traffic to facilitate processing with tools like CICFlowMeter is not EDDD's primary aim. Hence, the evaluation is centered on analyzing only the non-flow related properties, as delineated in Section 3.4.

One property that undergoes evaluation is packet length. With the chosen approach, it

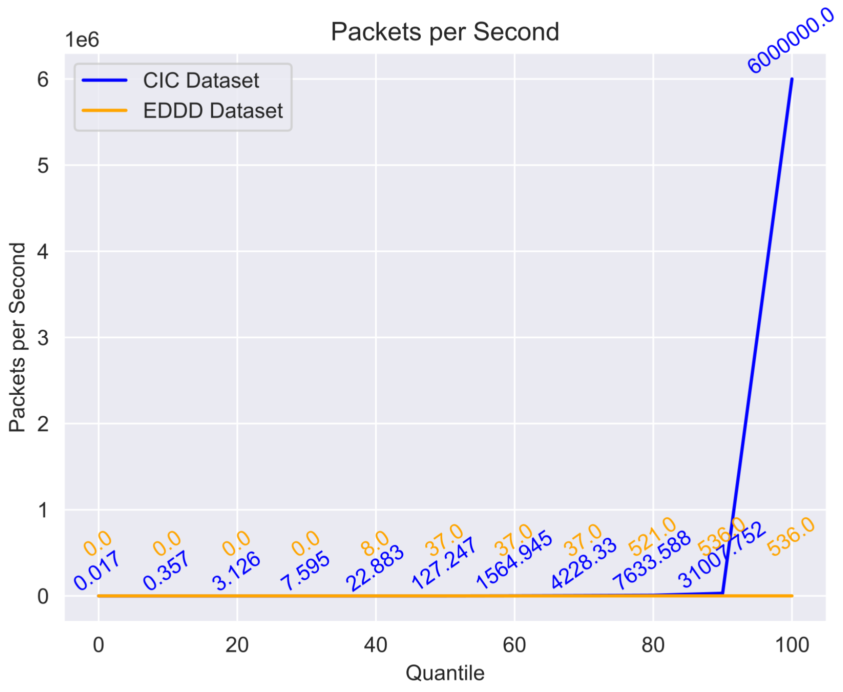


Figure 5.5: CAIDA vs. EDDD: Legitimate Packets per Second

is possible to model the ratio of non-payload packets precisely. This accuracy is evident in the quantiles, where both the original traffic from the dataset (denoted in blue) and the traffic generated by EDDD (in orange) maintain the same proportion in the charts at a packet length value of zero. However, distributions incorporating payloads display substantial deviations from the original data. The chi-square test also indicates a rejection of the assumption that the two distributions are identical. This discrepancy primarily stems from the specific scenario where the random selector frequently yields the same values. Hence a packet length of 37 and one of 536 emerge as recurrent figures.

Another attribute subject to evaluation is the packets per second. This metric per IP address is determined by the individual durations between the arrival of two packets. As EDDD's output does not identify any flows, the packets per second metric is distorted when compared to the same parameter in the original dataset, where it denotes packets per second within an individual flow. Therefore, the duration between two packets arriving in the generated dataset can be compared either to the packets per second or the duration between two flows in the original dataset. Here, too, the chi-squared test fails to confirm the similarity between the two datasets.

Upon manual inspection of the values, there is a significant discrepancy at certain quantiles. Nevertheless, it is notable that the value range of the generated datasets still falls within the original range, with a single exception at the very lowest packets per second. This discrepancy likely stems from the idle time between two flows, mirroring the longer waiting time between the arrival of two packets in the generated dataset. Given the vast range of this data, it becomes challenging to definitively ascertain whether the data is realistic. However, EDDD generates packets per second values from packets arriving several

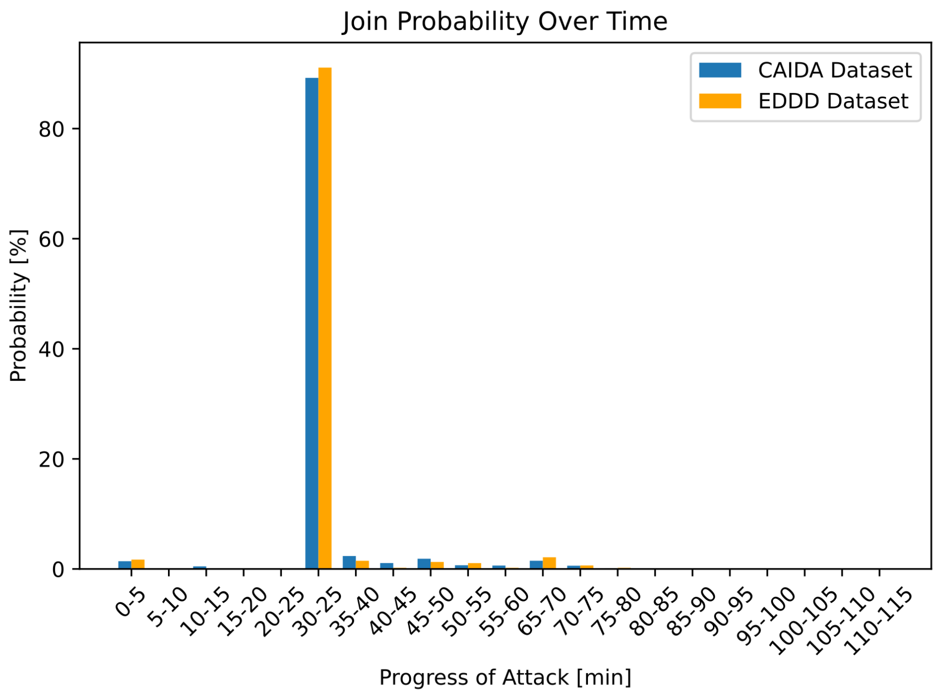


Figure 5.6: CAIDA vs. EDDD: ICMP Flood Attack Joining Time

seconds apart to those arriving at the same microsecond, in line with the original dataset.

5.3 Attack Traffic

A similar methodology to that used for for legitimate traffic is adopted to evaluate the attack traffic. The simulator's main function is slightly altered; instead of generating the entire simulation, it constructs the attack traffic by assigning traffic applications solely to attacking clients. With this modification, traffic arriving at the target can be analyzed without the need to discern between DDoS and benign traffic.

The subsequent subsections will delve into the analysis of the two possible types of attack traffic production. Section 5.3.1 centers around examining the ICMP Flood traffic, while Section 5.3.2 sheds light on the Syn Flood traffic.

5.3.1 ICMP Flood Attack

The evaluation of the synthesized ICMP Flood attack traffic leverages the structure of the CAIDA DDoS dataset, which organizes the data into digestible 5-minute intervals for more straightforward analysis. This format also serves as a logical approach for evaluating the synthesized data.

The initial property under investigation is the time of IP joining. Figure 5.6 depicts this relationship via a histogram, wherein the blue bars stand for the original dataset as

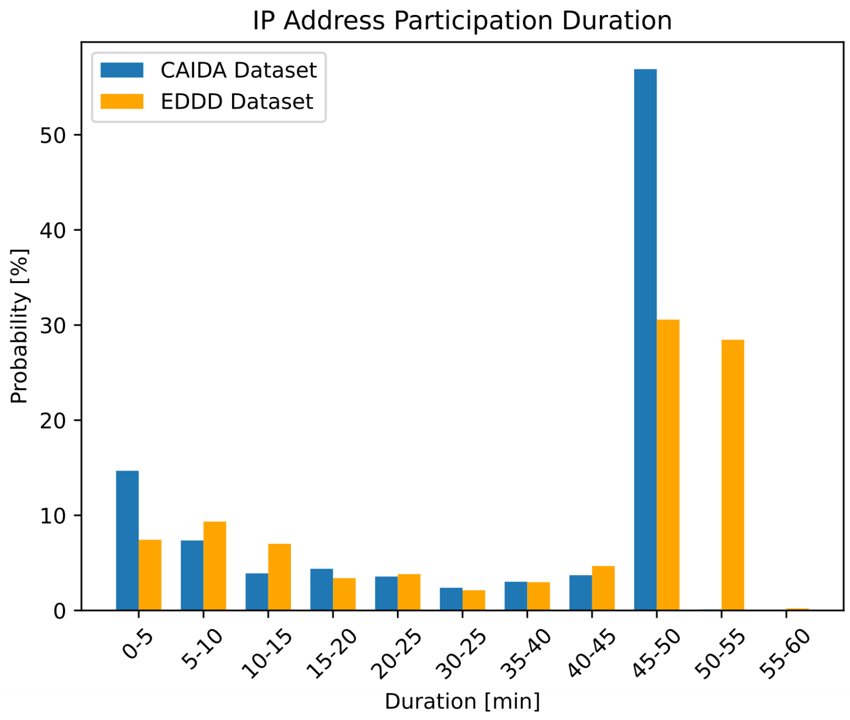


Figure 5.7: CAIDA vs. EDDD: ICMP Flood Attack Participation Duration

represented in Figure 3.10b, and the orange bars denote the equivalent probabilities from the synthesized dataset. To further validate the correspondence between the generated and original datasets, a Chi-square test is employed as a statistical measure. The results of this test provide a Chi-square statistic of 2.637 and a p-value of 0.999. The significance of the obtained p-value, which is close to 1, suggests a lack of significant evidence to reject the null hypothesis. This implies that the observed percentages align well with the expected percentages derived from the original dataset. The high p-value in the chi-square test corroborates the assertion that the join probability in the generated dataset does not significantly deviate from that observed in the original dataset. Consequently, this statistical assessment serves as an additional affirmation of the validity of the generated dataset when compared to the original.

Subsequently, the focus shifts to the duration of participation. The histogram in Figure 5.7 contrasts the original and generated data. The blue bars echo the values originally seen in Figure 3.11b from the original dataset, while the orange bars present the respective values drawn from the synthesized dataset. In further affirming the visually observed similarities between the generated and original datasets, a Chi-square test of independence is again utilized. However, the observed distortion from 45 to 55 minutes results in a very low p-value of nearly 0. This statistically indicates an inconsistency with the expected probabilities, and thus one might reject the null hypothesis. Nevertheless, a different picture emerges if this particular range of data is disregarded, and attention is only directed at the probabilities from 0 to 45 minutes. The revised computation yields a p-value of 0.81, offering no significant evidence to reject the null hypothesis in this area. This revised p-value suggests that the observed probabilities within this range are

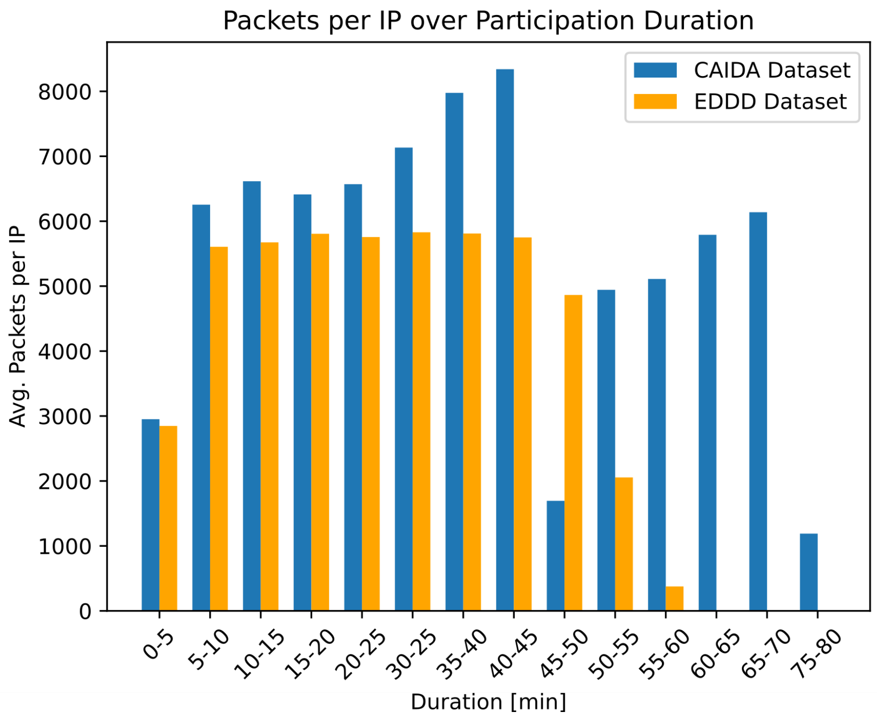


Figure 5.8: CAIDA vs. EDDD: ICMP Flood Attack Packets per IP over Participation Duration

consistent with the expected probabilities, thus enhancing the validity of the generated dataset.

The final comparison is drawn concerning the number of packets arriving at the target. Given the observed number, an estimated 16–27 packets per second was reached. This estimate is based on the observation that outliers in the diagram align with significant shifts in the number of clients participating in the attack. In Figure 5.8, the blue line corresponds to data already displayed in Figure 3.13 while the orange line represents the new data being compared. Upon careful observation, it becomes evident that parts of the duration exhibit similar behavior between the generated and original data sets. Notably, no outliers in packets per IP are found in the original data except for instances when many IP addresses leave the network within a five-minute interval. This observation validates the hypothesis that during these periods, the average packets per IP in the given five-minute interval may decrease due to an IP exiting the attack. Moreover, the original data exhibits more fluctuations than the generated data. An estimation of 16–27 packets per second derived from the data resulted in a more uniform distribution, producing more consistent data. This shows, that `uniform_real_distribution` used in Listing 4.18 to get a random attack rate indeed produces a uniform distribution.

In conclusion, the methodology employed for ICMP traffic generation has demonstrated a high degree of efficacy in replicating the characteristics of an ICMP flood attack. The patterns of clients joining the attack are well captured in the generated dataset, closely reflecting the behavior observed in the original dataset. Furthermore, the duration of a client's participation in an attack is also accurately represented. Any deviations observed

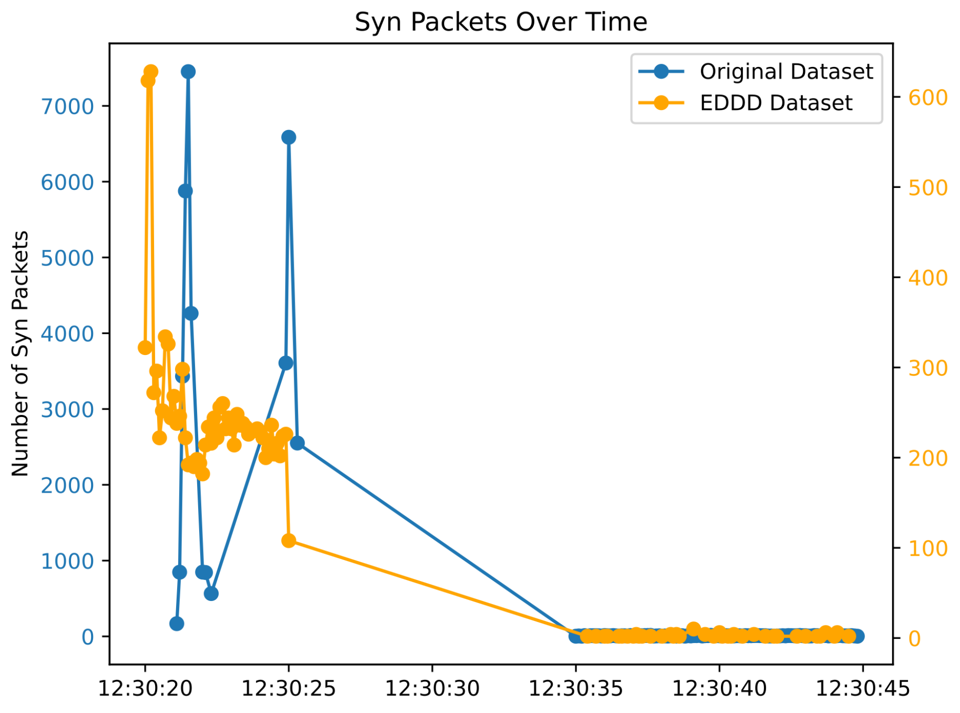


Figure 5.9: SYN Dataset vs. EDDD: SYN Flood Attack Arriving Packets Over Time

in the 45-55 minute range do not present significant concerns, as these discrepancies imply a slightly longer attack duration for certain clients in the generated dataset. Consequently, some of these clients fall into the subsequent five-minute interval for the duration measurement, slightly distorting this specific metric. The packets-per-second approach yields results that point in the right direction, even though it does not replicate the variations and outliers observed within the original data with complete accuracy. Despite this minor limitation, this approach provides valuable insights into the traffic dynamics of an ICMP flood attack. Taken as a whole, the findings affirm that this traffic application effectively emulates the characteristics inherent in an ICMP Flood attack.

5.3.2 Syn Flood Attack

As delineated in the dataset, SYN Flood attacks reveal a limited set of known properties. As articulated in Section 3.5.2, it is notably challenging to derive information about attacking clients solely based on the target's perspective since the IPs are spoofed. However, an intriguing observation can be made from Figure 3.14 - the discernible presence of two phases. This becomes the starting point for the evaluation presented herein.

Figure 5.9 visually compares the SYN packets over time, both for the original data and for the data set generated with the EDDD. As the diagram elucidates the cumulative quantity of arriving packets in 0.1-second intervals, it is not straightforwardly comparable due to the absence of information regarding the number of clients participating in the attack. Thus, comparison focuses primarily on the graphical structure rather than the raw numerical values. To manage the differences in the magnitudes of packet numbers

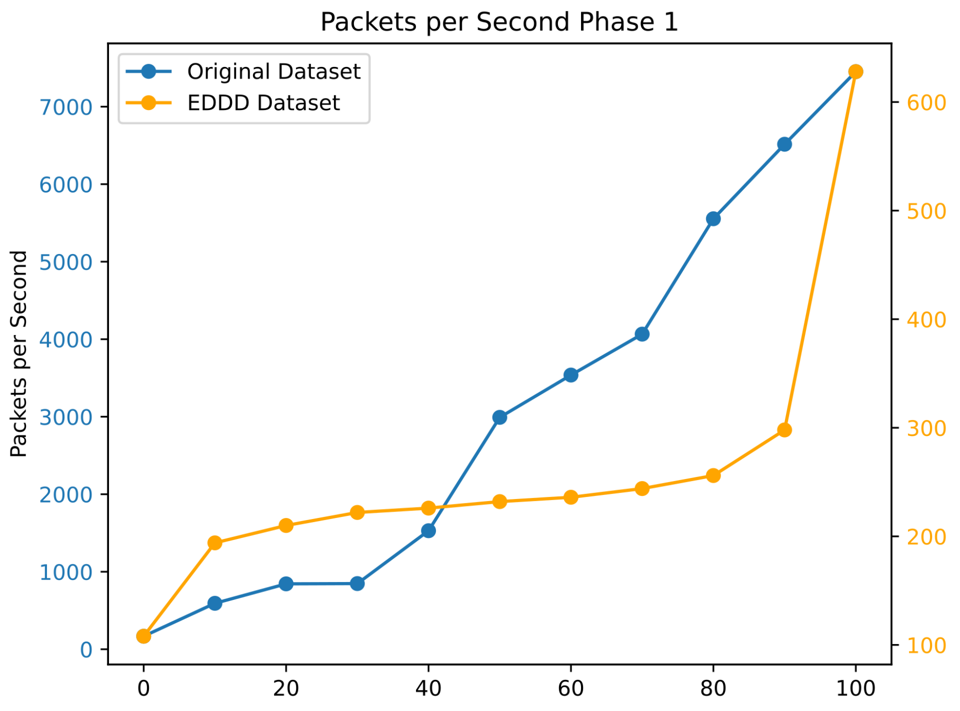


Figure 5.10: SYN Dataset vs. EDDD: SYN Flood Attack Phase 1 Arriving Packets per Second

involved, a dual y-axis approach is employed, with the left for the original data (blue) and the right for the generated data (orange). It can be observed that the generated data also features two discernible phases.

The subsequent stage of this analysis is dedicated to contrasting the packets per second within these distinct phases. Figure 5.10 offers a comparative view of phase 1 derived from both the original dataset and the data generated via EDDD. Given the challenge of differing magnitudes of packet numbers, the strategy of employing dual axes is sustained; one axis for the original data and another for the generated data. Upon inspection, it is evident that the two lines do not correspond accurately. The blue line, representing the original data, follows a rather linear distribution over the quantiles. In contrast, the EDDD-generated data exhibits a more constant rate of arriving packets during phase one. This divergence is already observable in Figure 5.9, where the data points from EDDD lie closer.

When scrutinizing the data from the second phase, it becomes apparent that the output from EDDD demonstrates a remarkable level of consistency (*cf.* Figure 5.11). Most of the time, the packets arrived at a rate of two per second, with only a few outliers marking instances where more packets arrived per second. In contrast, the original data again adhered to a more linear distribution across the quantiles. Within phase two, a substantial proportion of clients are yet to participate in the phase of the attack. The original dataset encapsulates a relatively short duration of an SYN flood attack, with each phase spanning mere seconds. Particularly in the second phase, this duration significantly impacts the packets sent, as the interval between sending successive packets is considerably

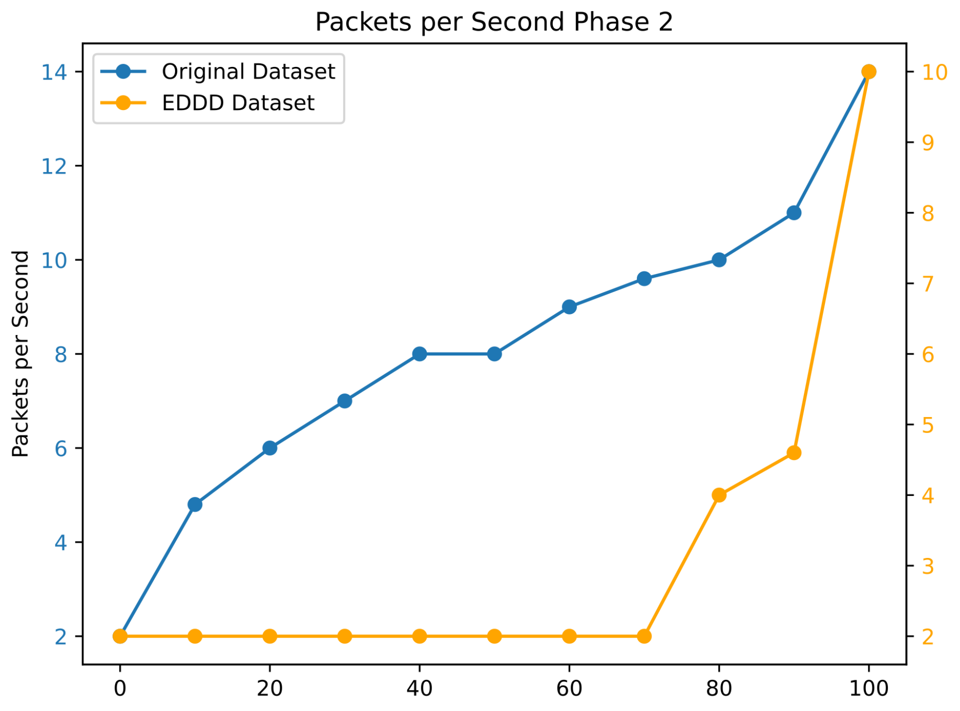


Figure 5.11: SYN Dataset vs. EDDD: SYN Flood Attack Phase 2 Arriving Packets per Second

long and may even exceed the total duration of the phase itself.

Recreating data from the original dataset presents a distinct challenge. Given the limited data available for analysis and the short duration of the attack, the generated dataset is predominantly a brief snapshot of a random distribution, which could potentially skew the overall perspective. However, it is evident that the replication of the dataset, featuring different SYN flood attack phases, was successful, even tho the packets per second did not accurately mirror the original data distributions within the phases. Despite failing to replicate the exact distribution, the replication of magnitudes remains discernible in the replicated two phases.

5.4 Distributed Attack View

The innovative feature of EDDD lies in its distributed capabilities. It presents the potential to generate DDoS datasets and assess traffic flows at multiple points in the network topology. This is especially crucial in DDoS attacks, in which data from multiple sources is sent to a single target to overwhelm it. Traditional tools that only examine data from one location, typically the target, on a network may miss crucial aspects of these complex attacks.

This section is dedicated to presenting an evaluation of the unique distributed approach that this dataset employs. Analogous to DDoS reports produced by Cloudflare [20], the

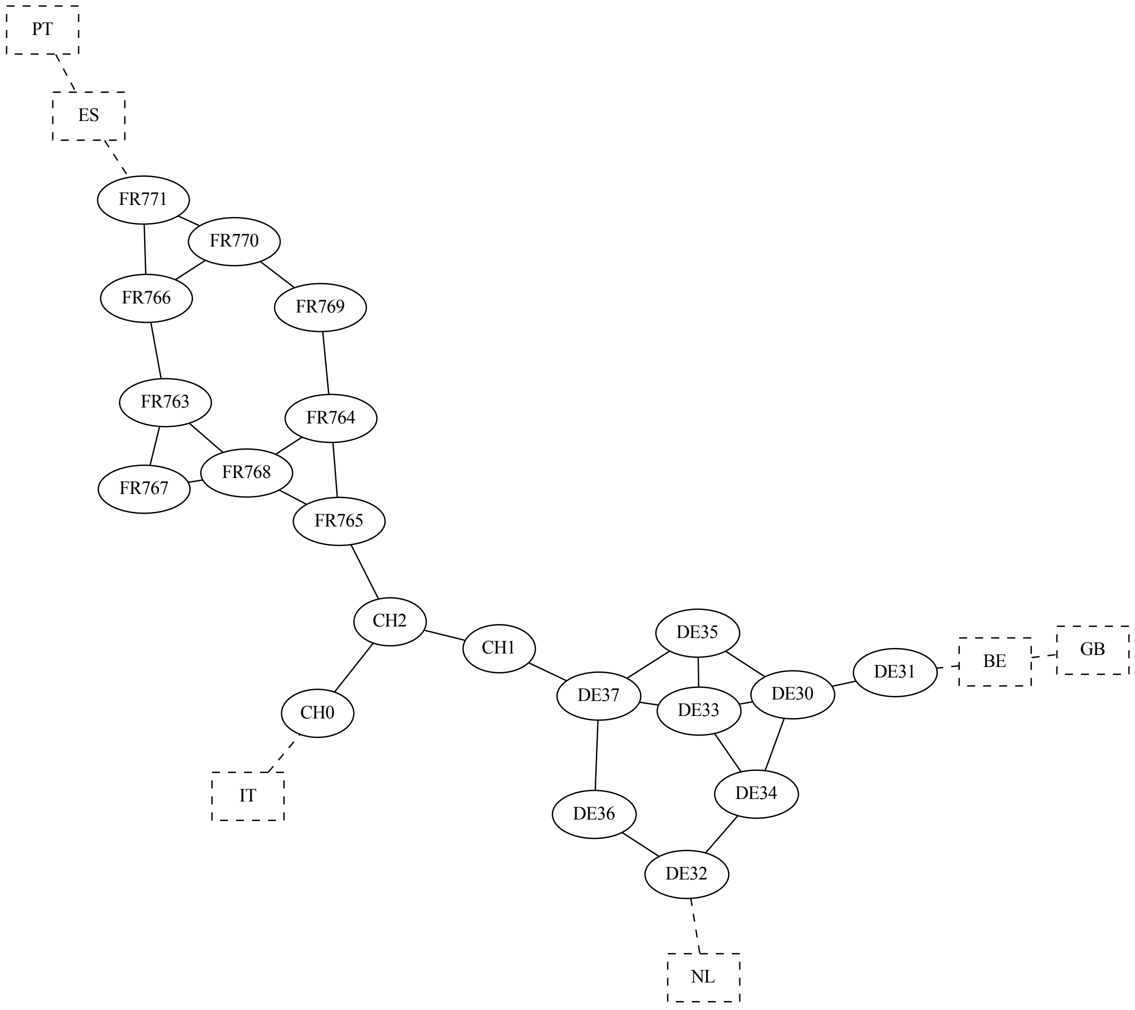
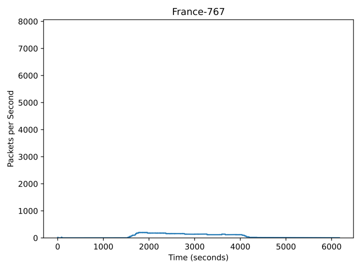


Figure 5.12: Topology of PCAP Backbone Nodes

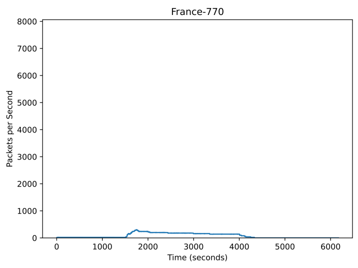
intensity of an attack in this thesis is evaluated based on the metric of requests per second, which equates to the number of packets per second.

Figure 5.13 and Figure 5.14 illustrate the packets per second overtime at backbone routers in France and Germany. The visualization encompasses all outgoing traffic destined for the target node. Due to the detailed insights the ICMP Flood attack dataset provides on the evolution of an attack, this particular attack type is chosen for the distributed analysis. The difference in the number of packets arriving at the nodes once the DDoS attack commences is considerably perceptible.

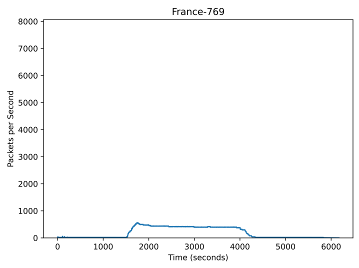
Further, Figure 5.12 offers an overview of the recording nodes and their interconnectivity. This diagram only displays the backbone routers actively engaged in PCAP recording. The other countries that connect to these backbone nodes, but do not have PCAP recording enabled, are merely depicted as dashed rectangles. The nomenclature of these nodes follows the pattern `[Country Abbreviation][Node ID]`. Here, the Country Abbreviation



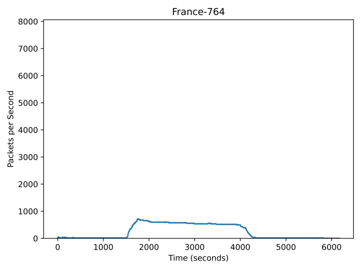
(a) Detected Packets Towards Target at FR767



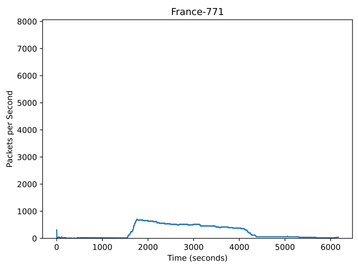
(b) Detected Packets Towards Target at FR770



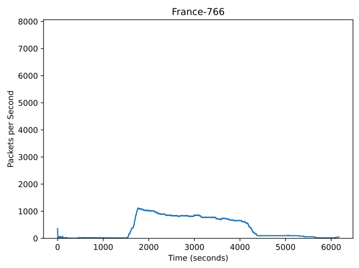
(c) Detected Packets Towards Target at FR769



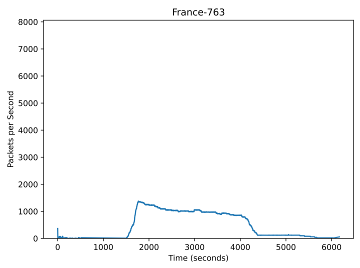
(d) Detected Packets Towards Target at FR764



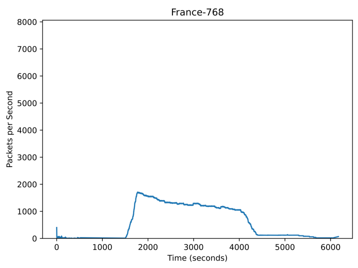
(e) Detected Packets Towards Target at FR771



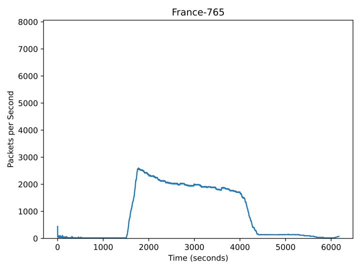
(f) Detected Packets Towards Target at FR766



(g) Detected Packets Towards Target at FR763



(h) Detected Packets Towards Target at FR768



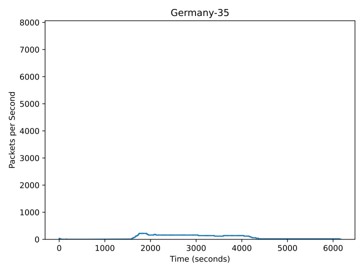
(i) Detected Packets Towards Target at FR765

Figure 5.13: Detected Packets Towards Target at Backbone Nodes in France

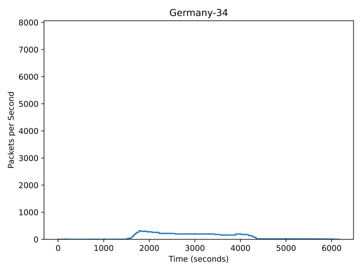
adheres to the Alpha-2 code of the respective country, as per the ISO 3166 standard [76].

Upon analyzing the distinct nodes presented in Figures 5.13 and 5.14, it is discernible that substantial disparities exist in terms of the perceivable intensity of attack traffic. Observations of nodes with low traffic, positioned towards the periphery of the topology in Figure 5.12, reveal that these nodes do not facilitate the passage of traffic through them. Consequently, the perceived intensity of an attack at a given node increases proportionally with the volume of nodes routing their traffic via this node to reach the target. As such, within a country, the highest intensity is invariably registered at the border node to Switzerland, primarily as all traffic destined for Switzerland must traverse this node.

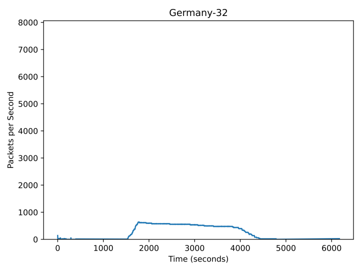
Once the traffic arrives at the target, it becomes evident that the intensity of the attack has markedly escalated compared to its last observed state at the border node leading to Switzerland, as indicated in Figure 5.13i and Figure 5.14h. This intensification can be attributed to the confluence of attack traffic delivered by various countries connecting



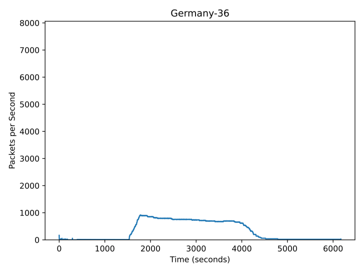
(a) Detected Packets Towards Target at DE35



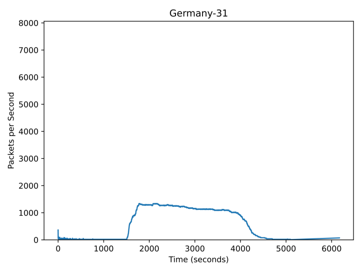
(b) Detected Packets Towards Target at DE34



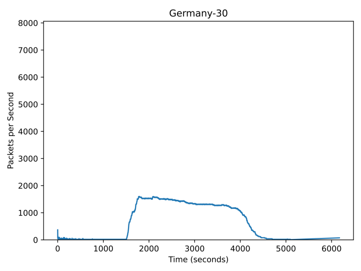
(c) Detected Packets Towards Target at DE32



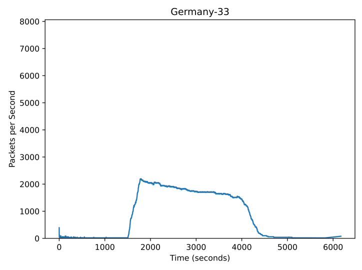
(d) Detected Packets Towards Target at DE36



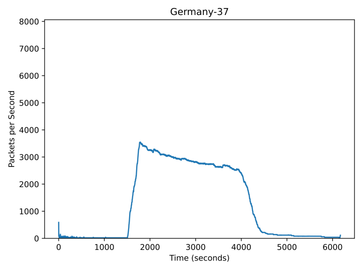
(e) Detected Packets Towards Target at DE31



(f) Detected Packets Towards Target at DE30



(g) Detected Packets Towards Target at DE33



(h) Detected Packets Towards Target at DE37

Figure 5.14: Detected Packets Towards Target at Backbone Nodes in Germany

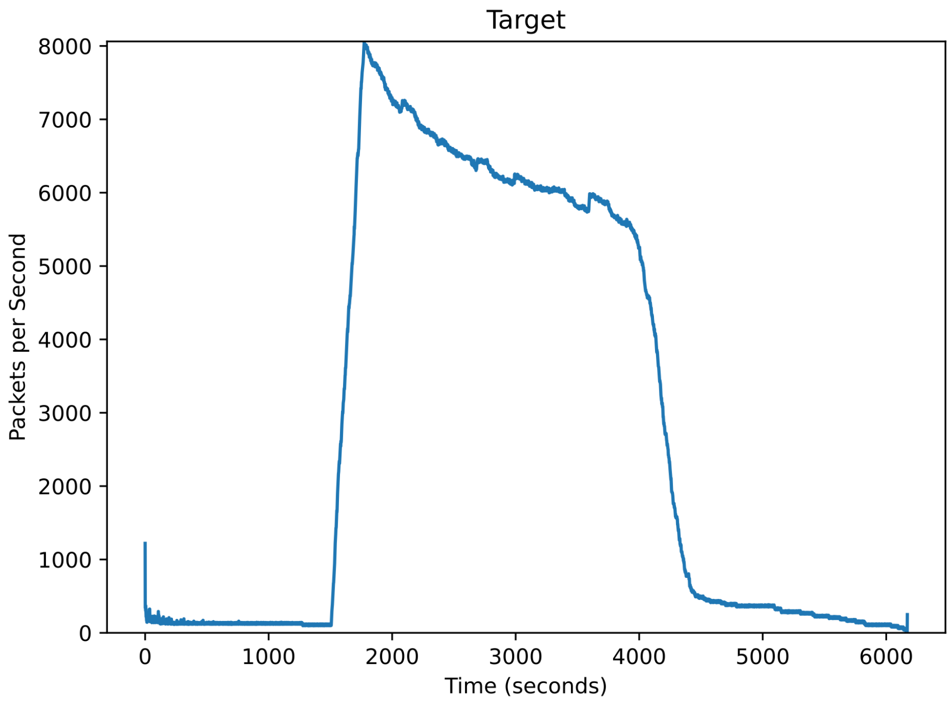


Figure 5.15: Detected Packets at Target

to Switzerland, as illustrated in Figure 5.12. France and Germany, given their roles as major connection points to Switzerland in this context, consequently accumulate the most substantial portion of the attack traffic. Observing the rapid progression of the attack's intensity across nodes is intriguing. While the peak of the attack manifests roughly 4000 packets per second at the French and German border nodes (*cf.* Figure 5.13i and Figure 5.14h), this figure doubles to approximately 8000 packets per second at the target (*cf.* 5.15).

5.5 Performance

This section is devoted to evaluating the performance of EDDD. All tests are carried out on a consistent machine with an Ubuntu 22.10 operating system for uniformity. An AMD Ryzen 9 5900X CPU powers this system and utilizes 64 GB of DDR4-RAM at 3600 MHz. Furthermore, all pertinent data is stored on a SATA-connected SSD.

The configuration employed to assess the system's performance mirrors the one used in previous evaluations for each traffic application. The sole potential variance is the number of represented nodes, dictated by the two factors regulating network and client count.

5.5.1 Parallelism

In this section, the performance of the EDDD system under different configurations is analyzed. By default, NS3 executes the entire simulation on a single core, providing robust, reproducible results in an event-based simulation context. One of the inherent advantages of utilizing NS3 is the opportunity to implement parallelism through Message Passing Interfaces (MPI). This technique allows each node in the simulation to be assigned to a specific rank, which in the context of EDDD, corresponds to different cores of the CPU.

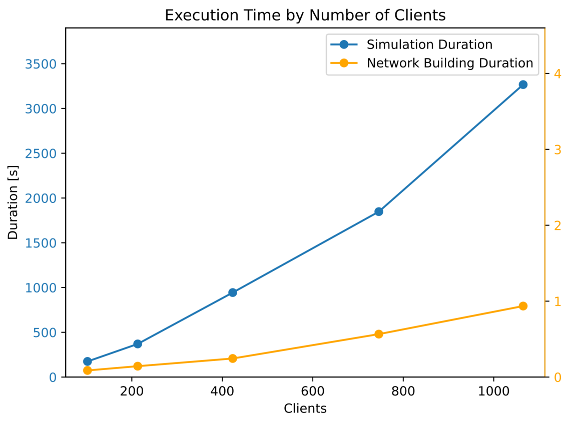
When a point-to-point link is established between nodes that do not share the same rank, a message must be passed to a separate simulation stream. It is important to note that this process is slower than a message passing between nodes of the same rank. Consequently, while it's advantageous to distribute the simulation to multiple streams, it is also beneficial to maintain a large proportion of links within the same rank to enhance simulation efficiency.

In several trials, it has been observed that assigning entire simulated countries to specific ranks proves to be an effective solution. In contrast, assigning ranks to different network topology levels (backbone, regional, local) increases cross-rank communication, negatively impacting overall performance.

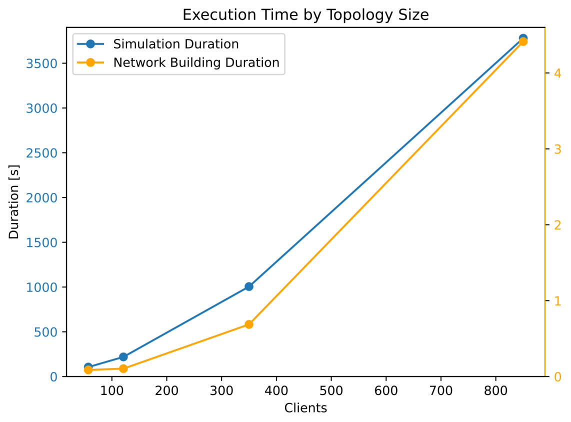
Clients		**Packets**	**Duration MPI Enabled**		**Duration MPI Disabled**	
Attack	Legitimate		Network [s]	Simulation [s]	Network [s]	Simulation [s]
22	35	1,511,475	0.088	106.115	0.209	95.823
60	61	4,448,708	0.105	219.602	0.454	341.589
197	153	14,729,593	0.687	1,005.106	5.381	2,759.167
470	380	34,394,480	4.414	3,779.267	160.320	85,672.949

Table 5.2: MPI vs. Single-Thread: Execution Times

Table 5.2 presents a comparative analysis of the network building and simulation times across various network sizes, both with and without MPI enabled. Upon examination of this table, it becomes evident that beyond a certain network size and traffic load, the EDDD variant with MPI enabled demonstrates greater efficiency than the variant without MPI. Expanding upon the simulation execution time, it is noted that this breakpoint arises when the proportion of cross-rank links is minimal compared to all the links. Observing the first row of Table 5.2, it is apparent that it involves a scant number of clients. Utilizing the same scenario as in Listing 5.1, 57 nodes are distributed across 10 countries, implying an average of merely five nodes within a country producing traffic. The result suggests that the advantage of parallelizing the traffic simulation within a country does not surmount the disadvantage associated with the requirement to synchronize and pass all messages transferred to another country. Nonetheless, this advantage outweighs the disadvantage when the involvement escalates to 121 nodes. At this juncture, it is observed that the simulation duration is lower in the MPI-enabled configuration.



(a) Performance by Number of Clients



(b) Performance by Topology Size

Figure 5.16: Performance by Network Nodes

This benefit of parallelizing a country's simulation grows as the network expands; while the execution time for the simulation for 121 nodes diminishes by 35%, for 350 nodes, it reduces by 64%. In the final example involving 850 nodes, the simulation time reduces by 96%.

Focusing on the network-building phase is invariably quicker when conducted in parallel. The network topology must be constructed on each node to build the routing tables. However, the applications, which constitute the more computationally intensive task, are only installed on the rank assigned to the node. This facilitates the parallel installation of applications, invariably leading to faster network-building times.

5.5.2 Scalability

The scalability of EDDD is a critical requirement, particularly in generating sizable datasets. The scenarios can scale in two distinct directions. The network topology can primarily expand, accommodating increasing network nodes composed of routers and clients. Secondly, the simulation duration may be extended, implicating increased transmission packets. This section aims to scrutinize the performance of both these scalability vectors.

Considering the network topology, it comprises two key elements: the number of routers and the number of clients. For evaluative purposes, an initial examination is conducted, focusing solely on the scalability of the clients. The configuration parameter, referred to as `clientFactor`, determines the number of clients on an access network. Therefore, the modification of this parameter solely affects the number of clients and leaves the number of routers untouched. Figure 5.16a illustrates the correlation between the network building and simulation time and the number of clients. The observed data demonstrates a linear relationship between the number of clients involved and the simulation and network construction durations. This suggests that as the client count increases, there is a proportional increase in the time taken for the simulation and network-building processes.

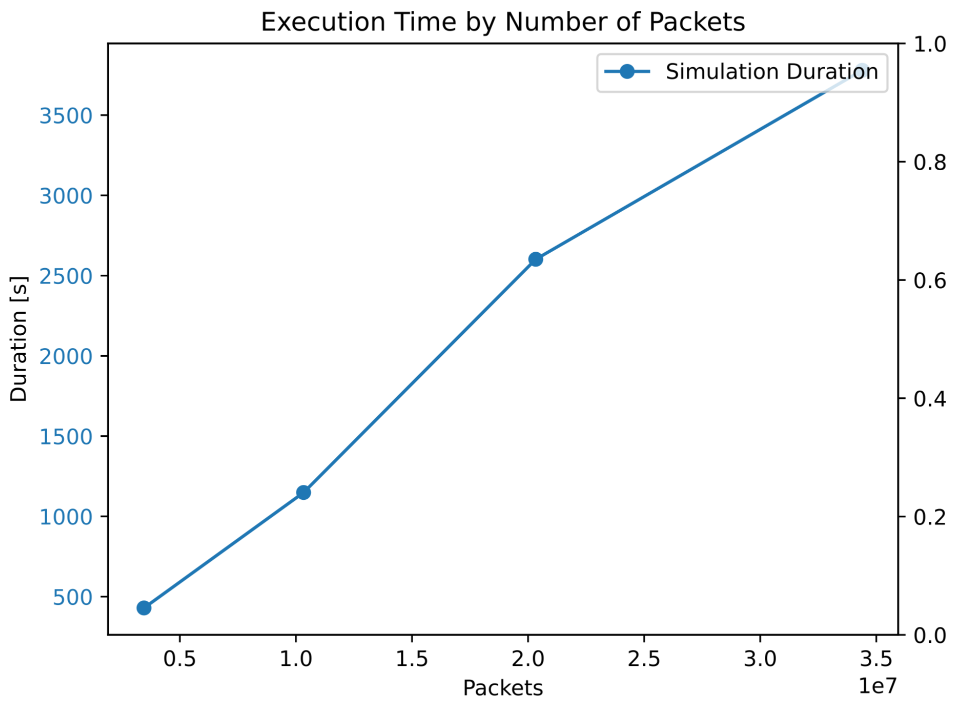


Figure 5.17: Performance by Number of Packets transmitted

However, this scenario might not realistically depict real-world circumstances. Typically, to more accurately represent a network akin to the real-world scale, the number of routers and clients would need to scale. Figure 5.16b depicts the scalability of EDDD when both these elements increase. One of the immediate observations made from these results is that the time taken for network construction is significantly elevated compared to previous results, attributable to the increased number of routers involved and the necessity for larger routing table creation. On the other hand, the increase in simulation duration is relatively nominal when additional routers are included in the process. Furthermore, the curve representing the duration of the network-building phase demonstrates a slightly disproportionate growth. On the contrary, the simulation duration appears to scale almost linearly with the number of clients and does seem to be influenced by the number of routers traversed by a packet. Given that the simulation duration is significantly longer than the building phase, the slightly disproportionate increase in the building phase is negligible. Therefore, EDDD exhibits linear scalability concerning the size of the network.

The secondary scalability direction of EDDD lies in the number of transmitted packets. The simulation duration is altered in an evaluative scenario portrayed in Figure 5.17. This leads to a scenario with identical clients and routers, albeit more transmitted packets. The simulation duration in which these packets are transmitted correlates with the number of packets sent. The evaluation performed on the mentioned hardware reveals that, on average, simulating a single packet takes approximately 8ms.

The primary influence on the execution duration of EDDD is the simulation phase, with the number of nodes within the network being a secondary influence. In summary, the simulation duration exhibits linear scalability concerning the number of involved nodes and the simulation duration.

5.6 Discussion

The primary objective of this thesis centers around constructing a system capable of generating distributed datasets. This involves the incorporation of diverse configurability to facilitate the generation of a variety of datasets. These datasets span a wide array of different topologies and traffic patterns. A key feature of the configuration allows replicating real DDoS attacks and adapting to new ones. Besides DDoS traffic, the generation of legitimate traffic is equally significant. The system must then allow the decision on which nodes the distributed datasets are recorded.

This thesis successfully fulfills these general objectives. The connected countries approach delineated in Section 3.3 proves an excellent method for creating a collection of distinct network topologies. All network components required in Section 2.2.3.3 are implemented, and the whole traffic journey is modeled with them. Only the non-existent firewall still offers room for an extension of the network model. Likewise, the requirement for anonymous IP addresses was met by assigning them to a test range to protect the privacy of the real IP addresses used, even though this contradicts a completely realistic dataset (*cf.* Section 2.2.3.2). Furthermore, the network topology configuration within the config file presents an intuitive interface for creating custom topologies. As validated in Section 5.1, this approach facilitates the creation of custom topologies and recreating real-world topologies. This adaptability permits the design of more refined targeted scenarios that yield usable and valid topologies for comparable real-world situations. A crucial advantage of this system is the scalability of the network, where two parameters manage the scaled representation of the routing network and its clients. This feature ensures that an accurate simulation of the desired network size is achievable while reflecting a real-world scenario. Specifically, even as the number of represented nodes within a country decreases through scaling, the distances and routing hops between the countries remain constant.

The configuration file additionally allows the selection of the DDoS attack type to be replicated. It controls the traffic distribution and how much each simulated country contributes to legitimate and benign traffic. Finally, this configuration file can also select the nodes at which the distributed datasets are generated. The emulator, therefore, effectively fulfills the criteria of being reconfigurable and accommodating diverse types of DDoS attacks. Its capacity to adapt network topologies provides an advantage in modeling a broad spectrum of scenarios.

However, even though two types of attacks can be replicated, they do not represent the full spectrum of these attacks. All traffic-generating applications installed on the nodes are configured and fine-tuned to match the original datasets' behaviors closely. The ICMP Flood attack relies on the CAIDA dataset [14], the SYN Flood attack on the dataset collection [37], and the legitimate traffic on the CIC dataset [28]. Notably, not every ICMP flood attack mirrors the CAIDA dataset exactly and generally, a dataset does not stand for all traffic of this type. This fact underlies why the attack duration is determined by the attack itself, with all attacker properties derived from the dataset.

However, the current implementation of EDDD presents a significant drawback in its strong coupledness. It cannot generate new datasets with variations in attack type and attacker persistence, as the applications lean more towards replicating attacks rather than producing new behaviors. Still, it satisfies the fundamental need to capture all traffic, encompassing benign and attack traffic. Moreover, it offers the flexibility to manually disable either the attack traffic or legitimate traffic, thus enabling the creation of a labeled dataset for both legitimate and attack traffic.

Performance forms another essential requirement of the system. As shown in Section 5.5.2, the simulation duration scales linearly with the number of clients involved and the number of transmitted packets. The size of the network itself when the building is very performant, making this duration negligible when compared to the simulation of the traffic. EDDD, relying on the underlying NS3, runs in $O(n)$, where n is the number of packets sent, determined by the attack type and the number of clients involved. This linear scaling enables the estimation of the simulation duration for new scenarios when having execution time data about scenarios on a smaller scale on the same runtime machine. Moreover, with parallelism, the simulation execution time can be drastically reduced. However, it is important to note that the scaling advantage of parallelism does not continue indefinitely with the increasing number of cores involved. This means that users do not need to invest in ultra-high-core CPUs for efficient simulation. Such characteristic renders EDDD predictable and a viable choice for generating new datasets. With proper planning, the creation of datasets can be scheduled to span several hours or even days, providing flexibility and foresight in the data generation process.

EDDD provides considerable ease of use. Although setting up NS3 as the underlying simulation engine and configuring everything can be challenging, the installation script simplifies this process, saving time and making EDDD accessible to a wider user base. It is also feasible to deploy it automatically in the cloud thanks to this script and no manual setup. A user-friendly configuration file allows for configuration in a comprehensible format. Should any configurations be incorrect, EDDD is programmed to abort the operation and issue a meaningful error message. This immediate feedback mechanism allows users to swiftly identify and fix misconfigurations, promoting efficient troubleshooting and contributing to a more seamless user experience.

The component-oriented design of the traffic-generating applications allows EDDD to be readily extended with new types of attacks. Modifications to the existing behavior of the system and its traffic applications can also be easily implemented. These characteristics underscore EDDD's role as a solid foundation for generating distributed datasets, offering a good level of initial capabilities. A key advantage undoubtedly rests in its straightforward and extensible structure. However, one critical prerequisite for anyone seeking to expand EDDD is understanding the principles and strategies utilized in NS3. Such knowledge is essential in manipulating and enhancing the system's functionality, as the platform's architecture is tightly integrated with NS3's methodologies. Despite this requirement, the design and flexibility of EDDD affirm its potential to be an instrumental tool in distributed dataset generation.

The novelty of EDDD resides in its ability to create distributed datasets, shifting from the current target-centric datasets. EDDD also offers a unique system for generating real-world representable network topologies with accurate pings between nodes, all configurable through a straightforward configuration file. However, the limitation of being unable to create new datasets with unseen behavior can also be perceived as revolutionary. It establishes the feasibility of generating a distributed version of a centralized dataset projected onto various network topologies and sizes. For instance, creating a distributed version of an attack related to the one portrayed in the CAIDA dataset and a distributed version for the SYN flood dataset is possible.

With its current features, EDDD can be employed for selected scenarios. Although it already presents a solid foundation for simulating additional traffic types to create further distributed datasets from custom network topology, it also displays potential for future expansions. One such potential expansion lies in its simple and extensible structure. However, a prerequisite for anyone aiming to extend EDDD is a deep understanding of the principles and approaches taken in NS3.

In summary, while EDDD is beneficial in its current form, its true value lies in its potential for future development and expansion. It demonstrates the potential for a paradigm shift in generating distributed datasets, offering a base for further advancements in this area.

Chapter 6

Final Considerations

6.1 Summary

The focus of this thesis revolves around the creation of a unique emulator, referred to as the Emulator for Distributed Denial of Service Datasets (EDDD). The development of EDDD addresses the limitations found in existing DDoS datasets, specifically the restrictions presented by their centralized approach. This work presents a marked departure from this centralized viewpoint, enabling a broader, distributed examination of DDoS attacks and their dynamics.

The EDDD is founded on the Network Simulator 3 (NS3) platform, benefitting from the inherent flexibility of an open-source system that allows for potential future expansions. Despite the complexity associated with the platform, successful navigation of these intricacies has led to the development of a versatile and adjustable emulator. The EDDD's design and architecture encompass interchangeable components, offering flexibility in the generation of varied network topologies and a spectrum of DDoS scenarios.

The current iteration of the EDDD is capable of producing SYN flood traffic, ICMP flood traffic, and legitimate traffic, based on existing datasets. Its comprehensive configuration for network structure allows for the generation of realistic topologies that can span multiple countries. Furthermore, the system captures all traffic, not only attack traffic, modeling the complete journey from origin to destination, thereby offering a complete view of the network dynamics involved in DDoS attacks.

In the process of evaluating EDDD, the emulator-generated traffic was comprehensively compared to the original centralized datasets, revealing the superior insights provided by the distributed approach. The performance analysis indicated the linear scalability of the system, with execution time scaling linearly with the number of packets sent.

EDDD outputs datasets in pcap format, facilitating straightforward analysis by widely used tools in the cybersecurity domain. The system also provides a simple command-line interface, complemented by a configuration file that allows easy adjustments to the parameters.

To summarize, this thesis has introduced EDDD as a significant contribution to the realm of DDoS research. It facilitates a more comprehensive exploration of DDoS attacks from a distributed viewpoint, marking a significant improvement over the limitations of the centralized approach. EDDD's open-source foundation and modular architecture provide a solid basis for future enhancements and add substantial value to the understanding of DDoS attack dynamics.

6.2 Conclusions

In this thesis, a tool titled *Emulator for Distributed DDoS Datasets* (EDDD) was successfully developed, that allows for the generation of distributed DDoS datasets. This accomplishment emanated from a series of meticulously made decisions, informed by prior research, meticulous design processes, and thoughtfully planned implementation strategies. These were further subjected to evaluations, all directed towards the fundamental objective of creating a solid initial version of such a tool.

One of the main takeaways from this thesis is the evident need of adopting a distributed perspective when scrutinizing DDoS attacks. Existing DDoS datasets utilize a centralized approach, which inherently limits the holistic comprehension of these intricate attack scenarios, often crucial for IDS validation. The advent of EDDD signifies a noteworthy step towards bridging this gap, offering a comprehensive view of DDoS attacks, thereby enhancing early detection and mitigative action potential before the attack culminates at its intended target.

In essence, the tool achieves its primary aim of generating DDoS datasets from a distributed viewpoint. Via its simplistic and structured configuration file, it facilitates the selection of a multitude of attack parameters, network topologies, and attack scenarios, thereby enabling the creation of diverse and realistic datasets. Nevertheless, its tight coupling with the original datasets, which serve as models for recreating attack and legitimate traffic, currently impedes the creation of entirely new, unseen attacks. Despite this limitation, it presents the opportunity to disperse these original datasets across a user-defined custom topology, thereby fulfilling its principal goal, as confirmed by the evaluation.

In the course of developing the emulation tool, the considerable potential of the open-source network simulator NS3 became strikingly evident. NS3's open-source nature was instrumental in allowing EDDD to simulate an array of DDoS attacks and legitimate traffic. Furthermore, it enabled the comprehensive configuration of network topologies, replicating real-world networks spanning multiple countries. Its component-like architecture presents an avenue for the future enhancement and extension of additional attack scenarios and features.

Additionally, the tool's straightforward command-line interface amplifies its applicability, proving useful in both academic research and practical applications. The ease of deployments, installations, and executions lends itself to automated operation of EDDD.

In conclusion, the journey of designing, implementing, and evaluating EDDD for generating distributed DDoS datasets has paved the path for a more in-depth comprehension of DDoS attack dynamics and has propelled the development of collaborative detection and mitigation techniques for DDoS attacks forward. Furthermore, it has established a sturdy foundation upon which future research in this field can build.

6.3 Future Work

Leveraging the capabilities of NS3, the simulation engine that EDDD is built upon, numerous future expansions and improvements can be envisioned. As NS3 is an open-source platform, it offers the flexibility for desired alterations and additions. However, such modifications require a detailed understanding of the inherent functionality of NS3 due to its complexity and extensive under-the-hood features.

One of the intricate aspects of NS3 pertains to the parallel execution of simulations using MPI. It offers two simulation approaches - the default DistributedSimulator and the NullMessageSimulator. The latter may potentially deliver superior performance in scenarios involving high connection counts crossing assigned nodes' ranks. However, in the context of EDDD, the NullMessageSimulator demonstrated unstable behaviour necessitating a more profound understanding of its operation. Nevertheless, overcoming this hurdle could enhance the system's performance.

Furthermore, the modular design of EDDD provides an advantageous platform for future extensions with new types of traffic. Currently, EDDD includes three implemented traffic applications - ICMP flood attacks, SYN flood attacks, and legitimate traffic - which are closely related to their source datasets. By reengineering these applications to be less reliant on specific datasets and more generic or even configurable, EDDD's functionality can be significantly extended. One such improvement could be the inclusion of new attack traffic applications that support IP spoofing and reflection attacks, even if the latter are relatively less prevalent today. However, such extensions necessitate a deeper understanding of NS3's underlying principles.

The scope of potential enhancements also extends to the modeled system. Future work could involve the implementation of firewalls with custom rules for specific nodes, or simulating the performance implications of the traffic on implicated systems.

In summary, due to its open and modular structure, EDDD offers a promising base for continued development and enhancements from its inaugural version. It poses the potential for incorporating new features, improving performance, and broadening its range of simulated scenarios.

Bibliography

[1] Yahya Al-Hadhrami and Farookh Khadeer Hussain. “Real time dataset generation framework for intrusion detection systems in IoT”. In: *Future Generation Computer Systems* 108 (2020), pp. 414–423.

[2] Paul Ferguson Alain Hebert Jay R. Ashworth. *BCP38*. URL: `http://www.bcp38.info/index.php/Main_Page` (visited on May 12, 2023).

[3] Sabah Alzahrani and Liang Hong. “Generation of DDoS attack dataset for effective IDS development and evaluation”. In: *Journal of Information Security* 9.4 (2018), pp. 225–241.

[4] Stefano Avallone, Antonio Pescape, and Giorgio Ventre. “Distributed Internet Traffic Generator (D-ITG): analysis and experimentation over heterogeneous networks”. In: *Poster at international conference on network protocols, ICNP*. 2003.

[5] Sunny Behal and Krishan Kumar. “Characterization and Comparison of DDoS Attack Tools and Traffic Generators: A Review.” In: *Int. J. Netw. Secur.* 19.3 (2017), pp. 383–393.

[6] Sunny Behal and Krishan Kumar. “Trends in Validation of DDoS Research”. In: *Procedia Computer Science* 85 (2016), pp. 7–15.

[7] Sajal Bhatia et al. “A framework for generating realistic traffic for Distributed Denial-of-Service attacks and Flash Events”. In: *computers & security* 40 (2014), pp. 95–107.

[8] Sajal Bhatia et al. “Parametric differences between a real-world distributed denial-of-service attack and a flash event”. In: *2011 sixth international conference on availability, reliability and security*. IEEE. 2011, pp. 210–217.

[9] Monowar H Bhuyan, DK Bhattacharyya, and Jugal K Kalita. “An empirical evaluation of information metrics for low-rate and high-rate DDoS attack detection”. In: *Pattern Recognition Letters* 51 (2015), pp. 1–7.

[10] Monowar H Bhuyan et al. “Detecting distributed denial of service attacks: methods, tools and future directions”. In: *The Computer Journal* 57.4 (2014), pp. 537–556.

[11] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. “A tool for the generation of realistic network workload for emerging networking scenarios”. In: *Computer Networks* 56.15 (2012), pp. 3531–3547.

[12] Jonas Brunner. *Reassembler - Towards a Global DDoS Attack Analysis Using Attack Fingerprints*. Master’s thesis. June 2023.

[13] Vít Bukac. “Traffic characteristics of common dos tools”. In: *Faculty Informat., Masaryk Univ., Brno, Czechia, Rep. FIMU-RS-2014-02* (2014), pp. 74–78.

[14] CAIDA. *The CAIDA UCSD "DDoS Attack 2007" Dataset.* URL: `https://www.caida.org/catalog/datasets/ddos-20070804_dataset` (visited on June 7, 2023).

[15] Joan Calvet et al. "Large-scale malware experiments: Why, how, and so what". In: *Proceedings of the 2010 Virus Bulletin Conference (VB).* 2010.

[16] calvin-f. *EDDD Repository.* URL: `https://github.com/calvin-f/EDDD/` (visited on June 15, 2023).

[17] Canadian Institute for Cybersecurity. *Intrusion Detection Evaluation Dataset (CIC-IDS2017).* URL: `https://www.unb.ca/cic/datasets/ids-2017.html` (visited on May 12, 2023).

[18] Canadian Institute for Cybersecurity (CIC). *CIC DoS dataset (2017).* URL: `https://www.unb.ca/cic/datasets/dos-dataset.html` (visited on May 12, 2023).

[19] Canadian Institute for Cybersecurity (CIC). *CICFlowMeter.* URL: `https://www.unb.ca/cic/research/applications.html#CICFlowMeter` (visited on May 24, 2023).

[20] Cloudflare. *Cloudflare mitigates record-breaking 71 million request-per-second DDoS attack.* URL: `https://blog.cloudflare.com/cloudflare-mitigates-record-breaking-71-million-request-per-second-ddos-attack/` (visited on June 7, 2023).

[21] Cloudflare. *SYN flood attack.* URL: `https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/` (visited on May 11, 2023).

[22] Cloudflare. *What is a distributed denial-of-service (ddos) attack?* URL: `https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/` (visited on May 11, 2023).

[23] Cloudflare. *What is IP spoofing?* URL: `https://www.cloudflare.com/learning/ddos/glossary/ip-spoofing/` (visited on May 11, 2023).

[24] Joseph Coffey. *Latency in Optical Fiber Systems.* Tech. rep. Commscope, Feb. 2017.

[25] Communications Security Establishment (CSE) & Canadian Institute for Cybersecurity (CIC). *A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018).* URL: `https://registry.opendata.aws/cse-cic-ids2018/` (visited on May 12, 2023).

[26] Paul J Criscuolo. *Distributed denial of service: Trin00, tribe flood network, tribe flood network 2000, and stacheldraht ciac-2319.* Tech. rep. California Univ Livermore Radiation Lab, 2000.

[27] Ozgur Depren et al. "An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks". In: *Expert systems with Applications* 29.4 (2005), pp. 713–722.

[28] Devendra. *DDoS Dataset.* URL: `https://www.kaggle.com/datasets/devendra416/ddos-datasets` (visited on May 12, 2023).

[29] S Renuka Devi and P Yogesh. "Detection of application layer DDoS attacks using information theory based metrics". In: *CS & IT-CSCP* 10 (2012), pp. 213–223.

[30] Sven Dietrich, Neil Long, and David Dittrich. "Analyzing distributed denial of service tools: The shaft case." In: *LISA.* 2000, pp. 329–339.

[31] David Dittrich et al. *The mstream distributed denial of service attack tool.* 2000.

[32] Christos Douligeris and Aikaterini Mitrokotsa. "DDoS attacks and defense mechanisms: classification and state-of-the-art". In: *Computer networks* 44.5 (2004), pp. 643–666.

[33] Chip Elliott. “GENI-global environment for network innovations.” In: *LCN*. 2008, p. 8.

[34] General Note. *Internet Architecture.* URL: `https://generalnote.com/Computer-Fundamental/Internet/Internet-Architecture.php` (visited on May 24, 2023).

[35] Amirhossein Gharib et al. “An evaluation framework for intrusion detection dataset”. In: *2016 International Conference on Information Science and Security (ICISS)*. IEEE. 2016, pp. 1–6.

[36] Google. *Google Feedback.* URL: `https://www.google.com/tools/feedback/intl/de/` (visited on June 7, 2023).

[37] L.F. Haaijer. *DDoS Packet Capture Collection.* 2022. URL: `https://github.com/StopDDoS/packet-captures` (visited on May 12, 2023).

[38] Chan-Myeong Han et al. “A spot-matching method using cumulative frequency matrix in 2D gel images”. In: *Biotechnology, biotechnological equipment* 28 (Nov. 2014), S37–S43.

[39] Guy Harris and Michael Richardson. *PCAP Capture File Format.* Internet-Draft draft-ietf-opsawg-pcap-02. Internet Engineering Task Force, Jan. 2023. 10 pp.

[40] Nazrul Hoque et al. “Network attacks: Taxonomy, tools and systems”. In: *Journal of Network and Computer Applications* 40 (2014), pp. 307–324.

[41] Alefiya Hussain et al. “DDoS experiment methodology”. In: *Proceedings of the DETER community workshop on cyber security experimentation*. Vol. 8. 2006.

[42] ITU Asia-Pacific Centre of Excellence Training. *Evolution of IP network core and backbone architectures.* URL: `https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/SiteAssets/Pages/ITU-ASP-CoE-Training-on-/itu-asp-coe-te-session6_internetevolution.pdf` (visited on May 24, 2023).

[43] Hossein Jazi et al. “Detecting HTTP-based Application Layer DoS attacks on Web Servers in the presence of sampling”. In: *Computer Networks* 121 (Mar. 2017).

[44] Kaspersky. *DDoS Breach Costs Rise to over $2M for Enterprises finds Kaspersky Lab Report.* URL: `https://usa.kaspersky.com/about/press-releases/2018_ddos-breach-costs-rise-to-over-2m-for-enterprises-finds-kaspersky-lab-report` (visited on June 7, 2023).

[45] Kaspersky. *Protecting your business against financial and reputational losses with Kaspersky DDoS Protection.* URL: `https://media.kaspersky.com/pdf/Kaspersky_Lab_Whitepaper_Kaspersky_DDoS_Protection_final.pdf` (visited on June 7, 2023).

[46] Harjeet Kaur, Sunny Behal, and Krishan Kumar. “Characterization and comparison of distributed denial of service attack tools”. In: *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*. IEEE. 2015, pp. 1139–1145.

[47] Keysight. *QualNet Network Simulator.* URL: `https://www.keysight.com/us/en/products/network-test/network-modeling.html` (visited on May 6, 2023).

[48] Keunsoo Lee et al. “DDoS attack detection method using cluster analysis”. In: *Expert systems with applications* 34.3 (2008), pp. 1659–1665.

[49] Cheng-Yi Liu, Chih-Hsiang Peng, and Iuon-Chang Lin. “A survey of botnet architecture and batnet detection techniques”. In: *International Journal of Network Security* 16.2 (2014), pp. 81–89.

[50] Pavel Lunin. *Network Latency: how latency, bandwidth & packet drops impact your speed.* July 2022.

[51] Xinlei Ma and Yonghong Chen. “DDoS detection method based on chaos analysis of network traffic entropy”. In: *IEEE Communications Letters* 18.1 (2013), pp. 114–117.

[52] Muhammad Mahmoud, Manjinder Nir, Ashraf Matrawy, et al. “A survey on botnet architectures, detection and defences.” In: *Int. J. Netw. Secur.* 17.3 (2015), pp. 264–281.

[53] SC Media. *Cloudflare blocked largest reported DDoS attack at 71M requests per second.* URL: `https://www.scmagazine.com/news/application-security/cloudflare-blocked-largest-reported-ddos-attack-at-71m-requests-per-second` (visited on June 7, 2023).

[54] Jelena Mirkovic et al. “How to test DoS defenses”. In: *Conference for Homeland Security, 2009. CATCH’09. Cybersecurity Applications & Technology.* IEEE. 2009, pp. 103–117.

[55] Robert Mitchell and Ing-Ray Chen. “A survey of intrusion detection techniques for cyber-physical systems”. In: *ACM Computing Surveys (CSUR)* 46.4 (2014), pp. 1–29.

[56] Netscout. *What is a DDoS Attack?* URL: `https://www.netscout.com/what-is-ddos` (visited on May 11, 2023).

[57] Netscout. *What is a Reflection Amplification Attack?* URL: `https://www.netscout.com/what-is-ddos/what-is-reflection-amplification-attack` (visited on May 11, 2023).

[58] Netscout. *What is a UPD Flood Attack?* URL: `https://www.netscout.com/what-is-ddos/udp-flood` (visited on May 11, 2023).

[59] Netscout. *What is an ICMP Flood Attack?* URL: `https://www.netscout.com/what-is-ddos/icmp-flood` (visited on May 11, 2023).

[60] Netscout. *What is an IP/ICMP Fragmentation Attack?* URL: `https://www.netscout.com/what-is-ddos/ip-icmp-fragmentation` (visited on May 11, 2023).

[61] Netsout. *Direct-Path Attacks Surge in 2022 Making Up Half of All DDoS Attacks According to Latest NETSCOUT DDoS Threat Intelligence Report.* URL: `https://www.netscout.com/press-releases/direct-path-attacks-surge-2022-making-half-all-ddos-attacks` (visited on June 7, 2023).

[62] Network Simulation Tools. *Best Network simulator for Research.* URL: `https://networksimulationtools.com/best-network-simulator-for-research/` (visited on May 14, 2023).

[63] nsnam. *ns-3 Network Simulator.* URL: `https://www.nsnam.org/` (visited on May 6, 2023).

[64] OMNeT++. *OMNeT++.* URL: `http://www.omnetpp.org/` (visited on May 6, 2023).

[65] İlker Özçelik and Richard R Brooks. “Deceiving entropy based DoS detection”. In: *Computers & Security* 48 (2015), pp. 234–245.

[66] Vern Paxson and Sally Floyd. “Wide area traffic: the failure of Poisson modeling”. In: *IEEE/ACM Transactions on networking* 3.3 (1995), pp. 226–244.

[67] Larry Peterson et al. “Experiences building planetlab”. In: *Proceedings of the 7th symposium on Operating systems design and implementation.* 2006, pp. 351–366.

[68] Moheeb Rajab et al. “My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging”. In: *Proceedings of the First Conference on*

First Workshop on Hot Topics in Understanding Botnets. USENIX Association. 2007, p. 5.

[69] Markus Ring et al. “A survey of network-based intrusion detection data sets”. In: *Computers & Security* 86 (2019), pp. 147–167.

[70] Desmond Schmidt et al. “A distributed denial of service testbed”. In: *What Kind of Information Society? Governance, Virtuality, Surveillance, Sustainability, Resilience: 9th IFIP TC 9 International Conference, HCC9 2010 and 1st IFIP TC 11 International Conference, CIP 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings.* Springer. 2010, pp. 338–349.

[71] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”. In: *International Conference on Information Systems Security and Privacy.* 2018.

[72] Iman Sharafaldin et al. “Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy”. In: *2019 International Carnahan Conference on Security Technology (ICCST).* IEEE. 2019, pp. 1–8.

[73] Iman Sharafaldin et al. “Towards a reliable intrusion detection benchmark dataset”. In: *Software Networking* 2018.1 (2018), pp. 177–200.

[74] Stavros N Shiaeles et al. “Real time DDoS detection using fuzzy estimators”. In: *computers & security* 31.6 (2012), pp. 782–790.

[75] A Srivastava et al. “A recent survey on DDoS attacks and defense mechanisms”. In: *Advances in Parallel Distributed Computing: First International Conference on Parallel, Distributed Computing Technologies and Applications, PDCTA 2011, Tirunelveli, India, September 23-25, 2011. Proceedings.* Springer. 2011, pp. 570–580.

[76] International Organization for Standardization. *ISO 3166 - Country Codes.* URL: `https://www.iso.org/iso-3166-country-codes.html` (visited on June 15, 2023).

[77] Yuan Tao and Shui Yu. “DDoS attack detection at local area networks using information theoretical metrics”. In: *2013 12th IEEE international conference on trust, security and privacy in computing and communications.* IEEE. 2013, pp. 233–240.

[78] The Apache Software Foundation. *ab - Apache HTTP server benchmarking tool.* URL: `https://httpd.apache.org/docs/2.4/programs/ab.html` (visited on May 24, 2023).

[79] The Tcpdump Group. *TCPDUMP.* URL: `https://www.tcpdump.org/` (visited on May 24, 2023).

[80] M Uma and Ganapathi Padmavathi. “A Survey on Various Cyber Attacks and their Classification.” In: *Int. J. Netw. Secur.* 15.5 (2013), pp. 390–396.

[81] Wireshark. *LibpcapFileFormat.* URL: `https://wiki.wireshark.org/Development/LibpcapFileFormat` (visited on May 11, 2023).

[82] WonderNetwork. *Ping time between Zurich and other cities.* URL: `https://wondernetwork.com/pings/Zurich` (visited on May 24, 2023).

[83] Yi-Chi Wu et al. “DDoS detection and traceback with decision tree and grey relational analysis”. In: *International Journal of Ad Hoc and Ubiquitous Computing* 7.2 (2011), pp. 121–136.

[84] Yang Xiang, Ke Li, and Wanlei Zhou. “Low-rate DDoS attacks detection and traceback by using new information metrics”. In: *IEEE transactions on information forensics and security* 6.2 (2011), pp. 426–437.

Abbreviations

CLI	Command Line Interface
DDoS	Distributed Denial of Service
DoS	Denial of Service
EDDD	Emulator for Distributed DDoS Datasets
GUI	Graphical User Interface
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IoT	Internet of Things
IRC	Internet Relay Chat
ISP	Internet Service Provider
MPI	Message Passing Interface
MST	Minimum Spanning Tree
NIC	Network Interface Card
PCAP	Packet Capture
RNG	Relative Neighborhoor Graph

List of Figures

2.1 Relationships between Datasets [69] 12

3.1 Technical Architecture - Core Components 18

3.2 Traffic Flow Over Network Topology 20

3.3 Node Distribution in Service Area 22

3.4 Probability of Flow Duration 23

3.5 Probability of Idle Times Between Flows 24

3.6 Idle Time Between Flows 24

3.7 TCP Flags for Specific Idle Times 25

3.8 Probability of Packet Length 26

3.9 Probability of Packets per Second 27

3.10 Attacker Joining Time 28

3.11 Attacker Participation Duration 28

3.12 Average Packets Over Time in CAIDA Dataset 29

3.13 Average Packets Over Participation Duration 30

3.14 Packets Arriving at Target Within 0.1 Seconds 30

3.15 Packets per Second Probability 31

4.1 Implementation - Core Components 33

4.2 Triangulation Comparison: Gabriel Graph (left) vs. RNG (right) [38] . . . 45

4.3 Router Connections to Subnet Nodes 47

4.4 Activity Diagram of Legitimate Traffic Application 53

4.5 Activity Diagram of ICMP Flood Traffic Application 56

4.6 Activity Diagram of SYN Flood Traffic Application 58

5.1 Connected Countries as Network Structure 65

5.2 Map of Western Europe with Connections of Selected Countries 66

5.3 Ping Comparison: NS3 Simulation vs. Real Measures 66

5.4 CAIDA vs. EDDD: Legitimate Packet Length 67

5.5 CAIDA vs. EDDD: Legitimate Packets per Second 68

5.6 CAIDA vs. EDDD: ICMP Flood Attack Joining Time 69

5.7 CAIDA vs. EDDD: ICMP Flood Attack Participation Duration 70

5.8 CAIDA vs. EDDD: ICMP Flood Attack Packets per IP over Participation Duration 71

5.9 SYN Dataset vs. EDDD: SYN Flood Attack Arriving Packets Over Time . 72

5.10 SYN Dataset vs. EDDD: SYN Flood Attack Phase 1 Arriving Packets per Second 73

5.11 SYN Dataset vs. EDDD: SYN Flood Attack Phase 2 Arriving Packets per Second 74

5.12 Topology of PCAP Backbone Nodes 75

5.13 Detected Packets Towards Target at Backbone Nodes in France 76

5.14 Detected Packets Towards Target at Backbone Nodes in Germany 77

5.15 Detected Packets at Target 78

5.16 Performance by Network Nodes 80

5.17 Performance by Number of Packets transmitted 81

List of Tables

5.1 Generated Country Topology 63

5.2 MPI vs. Single-Thread: Execution Times 79

Listings

3.1 Country Configuration Type 21
4.1 JSON Validation of Required Configuration 35
4.2 Determination of Maximal Number of MPI Cores 38
4.3 Running the Simulation 38
4.4 Removing Payload From Dataset 38
4.5 Setup of Topology 39
4.6 Attacker Setup 40
4.7 Target Setup 41
4.8 Legitimate Client Setup 41
4.9 Node Placement Within a Given Area 44
4.10 Point-To-Point Connections and IP Assigning 45
4.11 Select Border Nodes 48
4.12 Select Country Connection Nodes 49
4.13 Connecting Countries and PCAP Enabling 51
4.14 Random Value According to a Peacewise Probability 52
4.15 `StartApplication` Method of the `LegitimateTrafficApplication` 54
4.16 Determination of Start Time of `IcmpFloodAttacker` Application 56
4.17 Determination of End Time of `IcmpFloodAttacker` Application 57
4.18 `SendIcmp` Method of `IcmpFloodAttacker` Application 57
4.19 `ScheduleSend` Method of the `SynFloodAttacker` Application 59
4.20 `SendSyn` Method of the `SynFloodAttacker` Application 59
4.21 Filtering Mechanism for RST Packets 60
5.1 Configuration for Evalation Scenario 61

Appendix A

Contents of the CD

The following deliverables are submitted for this thesis:

- **Code:**
  - Contains the source code for EDDD, together with guidelines for its installation and usage. For convenience and enhanced accessibility, this is also made available on GitHub [16].
- **Thesis:**
  - ZIP file containing the source of the thesis
  - PDF of the thesis
  - Plain text files of the abstract in English and German

Appendix B

Installation Guidelines

Detailed guidelines for the installation and usage of EDDD can be found in the `README.md` of the respective repository [16].

1. Clone the repository:

   ```
   git clone https://github.com/calvin-f/EDDD.git
   cd EDDD
   ```

2. Install NS3 with its dependencies and integrate the EDDD module by executing:

   ```
   install.sh
   ```

3. Adapt the configuration if desired by changing properties in `input-config.json`.
4. Run EDDD by executing:

   ```
   run.sh
   ```