University of Zurich[UZH]

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

Reassembler - Towards a Global DDoS Attack Analysis Using Attack Fingerprints

Jonas Brunner
Zurich, Switzerland
Student ID: 17-709-650

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

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

ifi

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

Abstract

In recent years, the frequency and scale of Distributed Denial-of-Service (DDoS) attacks have increased significantly, yet they remain an unsolved problem. Many intrusion detection systems employ shared attack fingerprints and signatures as a standard practice to detect cyber attacks. For DDoS attacks, a single attack fingerprint typically is not enough to detect other attacks of the same kind. However, attack fingerprints can still be useful, especially when the same attack is observed from multiple locations.

Thus, this work proposes Reassembler, a tool that enables global analysis of DDoS attacks based on attack fingerprints recorded at different locations. For this, multiple attack scenarios are specified using a custom-built simulated network. Based on the attack scenarios, the Reassembler solution is implemented, analyzing and aggregating attack fingerprints into a global view.

Reassembler is evaluated based on four simulated and one real use case (based on real DDoS network traces), demonstrating that the Reassembler can derive interesting properties such as the number of intermediate nodes or the estimated percentage of spoofed IPs. Based on different experiments, it is shown under which circumstances the Reassembler performs best and where improvements are needed.

Zusammenfassung

In den letzten Jahren haben die Häufigkeit und das Ausmass von Distributed Denial-of-Service (DDoS) Angriffen erheblich zugenommen, trotzdem bleiben sie ein ungelöstes Problem. Viele Intrusion Detection Systeme verwenden geteilte Angriffs-Fingerabdrücke und -Signaturen als Standardpraxis zur Erkennung von Cyberangriffen. Bei DDoS-Angriffen reicht ein einzelner Angriffsfingerabdruck in der Regel aber nicht aus, um andere Angriffe derselben Art zu erkennen. Dennoch können Angriffsfingerabdrücke nützlich sein, insbesondere wenn derselbe Angriff von mehreren Standorten aus beobachtet wird.

Daher schlägt diese Arbeit den Reassembler vor, ein Werkzeug, das eine globale Analyse von DDoS-Angriffen auf der Grundlage von Angriffs-Fingerabdrücken ermöglicht, die an verschiedenen Standorten aufgezeichnet wurden. Hierfür werden mehrere Angriffsszenarien mit Hilfe eines eigens erstellten simulierten Netzwerks definiert. Basierend auf diesen Angriffsszenarien wird die Reassembler-Lösung implementiert, die Angriffs-Fingerabdrücke analysiert und zu einer globalen Sicht zusammenfasst.

Reassembler wird anhand von vier simulierten und einem realen Anwendungsfall (basierend auf realen DDoS Aufzeichnungen) evaluiert, wobei demonstriert wird, dass der Reassembler interessante Eigenschaften wie die Anzahl der Zwischenknoten oder den geschätzten Prozentsatz von gefälschten IPs ableiten kann. Anhand verschiedener Experimente wird gezeigt, unter welchen Umständen der Reassembler am besten funktioniert und wo Verbesserungen notwendig sind.

Acknowledgments

I want to thank all the people who were involved in this thesis. My special thanks go to my supervisor Dr. Bruno Rodrigues who provided invaluable feedback and advice throughout the thesis. I would also like to thank Prof. Dr. Burkhard Stiller for giving me the opportunity to be part of this project at the Communication Systems Group at the University of Zurich.

Contents

Chapter 1

Introduction

Distributed Denial-of-Service (DDoS) attacks pose a severe risk to Internet availability. Attacks happen often, as they are relatively simple to launch but hard to detect and mitigate efficiently. DDoS-for-Hire services enable anyone to launch an attack within minutes. Search activity for so-called "stresser" or "booter" services has increased so much that the UK government even launched an ad campaign seeking to deter Cybercrime [34]. In recent years, DDoS attacks have not only become more accessible, but they have also grown in terms of bandwidth. Cloudflare, one of the leading DDoS mitigation companies, reported that they had mitigated multiple attacks which exceeded 1Tbps in bandwidth [69]. It is also not uncommon for attackers to demand a ransom payment to stop an existing attack or not attack in the first place [33]. With the increasing digitization trend of society, it becomes clear that DDoS attacks threaten businesses and individuals. Despite multiple commercial and research efforts, DDoS attacks remain an unsolved problem.

One primary reason is the growing number of devices connected to the Internet that are poorly secured or not secured. This allows attackers to take control of devices ranging from security cameras to smart fridges. In large numbers, such infected devices form a botnet that can be used to launch Denial-of-Service attacks in a distributed manner. A famous botnet called "Mirai" was estimated to have infected over 100'000 devices [10]. Although the original creators of the botnet were convicted, variants of the botnet are still active and launching attacks with up to 2.5Tbps [69].

Naturally, an ideal counter strategy to distributed attacks also involves a distributed defense to mitigate attacking traffic at different points, possibly closer to their origin. In this regard, sharing information is essential for equally distributed DDoS attack defense. Numerous Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), which typically rely on their internal knowledge, already employ fingerprints or signatures as a standard practice for sharing attack data [42]. In this regard, the DDoSDB [18] project was proposed as a central repository where various organizations can share their DDoS fingerprints, thereby expanding the knowledge of a single organization. However, due to the distributed nature of DDoS attacks, different DDoS fingerprints may present different views of the same attack, posing a challenge and an opportunity to analyze and aggregate such fingerprints.

1.1 Motivation

Many existing DDoS defense strategies are primarily implemented at the attack target [31, 35, 46]. However, unlike with other security threats, the sharing of DDoS attack fingerprints and signatures is not widely adopted as a method for collective defense. This is because the fingerprint of a specific DDoS attack cannot be universally applied to detect other attacks of the same kind, thus serving more as a means for analysis rather than detection.

Compared to other security threats, the distributed nature of DDoS attacks makes them observable by other network nodes. Even though some sharing mechanisms for DDoS attack fingerprints exist [18], they have rarely been leveraged to gain further insights into a DDoS attack from a global perspective. However, attack fingerprints recorded at different nodes provide a unique perspective on a certain DDoS attack. With attack fingerprints from many cooperating entities, this provides a novel data foundation for a global post-mortem analysis of DDoS attacks.

A global post-mortem analysis can provide a more accurate view of a DDoS attack than an analysis solely at the attack target. For example, the origin of network layer attacks not using a handshake mechanism is almost impossible to locate when the attack employs IP spoofing techniques. While large cloud providers can roughly estimate the origin of an attack by analyzing the ingress traffic per edge location, this still does not provide a fully accurate picture because traffic can be routed through various ISPs or even countries [71]. In this regard, shared observations from multiple locations (including backbone routers) could provide further insights into the true origin of the attack. Moreover, novel metrics, such as the count of intermediate nodes observing an attack, or key nodes aiding in the execution of the attack, can be derived from attack fingerprints.

To the best of the author's knowledge, no existing research attempts to create a comprehensive analysis of a DDoS attack based on fingerprints recorded at different locations. Consequently, the challenge addressed in this thesis is filtering and aggregating DDoS attack fingerprints to form a global analysis, thereby assessing whether such a solution can enrich DDoS attack analysis and fingerprinting methods in general.

1.2 Thesis Goals

The goal of this master thesis is to design and implement an algorithm for a global analysis of DDoS attacks based on attack fingerprints collected in a distributed manner. This involves research about existing fingerprinting and detection methods and requires that use cases are defined on which a global analysis can be designed and evaluated.

Consequently, the following thesis goals have been defined.

- **Overview of Fingerprinting Methods and Existing Solutions:** Research on existing DDoS attack fingerprinting solutions. Overview of existing approaches for gathering a global DDoS attack view.

- **Definition of Use Cases:** Definition of use cases considering fingerprints presenting different information of the same DDoS attack. Further, considering a mix of intentional and unintentional adversarial fingerprints.
- **Design and Implementation of a Solution for Global Analysis:** Design and implementation of an algorithm for filtering and aggregating fingerprints to a global DDoS attack view based on characteristics inherent to the attack fingerprints.
- **Evaluation and Discussion:** Evaluation and discussion of the produced results aiming to answer how and whether the proposed solution can enrich DDoS-related information and improve the quality of DDoS attack analyses.

1.3 Thesis Outline

Following the introduction, fundamental concepts and related work are presented in Chapter 2. Thereon, Chapter 3 introduces the design of the proposed solution along with the considered attack scenarios. The implementation of the solution is further elaborated in Chapter 4. After that, Chapter 5 evaluates the proposed solution using real and simulated attack data. Lastly, Chapter 6 concludes this thesis and presents future work.

Chapter 2

Fundamentals

2.1 Background

This background section provides an introduction to the fundamental concepts on which this thesis is founded. First, an overview of the definition, types, and techniques of DDoS attacks is presented. Thereafter, different approaches to network monitoring are discussed. Lastly, the general idea of fingerprinting methods in the context of DDoS attacks is explained.

2.1.1 Distributed Denial of Service Attacks

Distributed Denial-of-Service attacks are a growing security threat that aims to disrupt the availability of online services by overwhelming them with large volumes of traffic. Attacks are usually launched in a distributed fashion and make use of botnets. By infecting insecure devices, malware can build an army of bots. Often, botnets are quickly adapted to use the latest security vulnerabilities. An example is the Mirai botnet [5], which was reported to use the Log4j vulnerability quickly after it had been discovered [9].

The motivation behind DDoS attacks is often financial or economic gain. In this regard, it is no surprise that ransom DDoS attacks continue to rise [70]. However, DDoS attacks are also launched as part of cyberwarfare [58] or simply for revenge [37]. Since attackers do not always have the resources to build a botnet, an entire business model about renting botnets for DDoS attacks has emerged [32]. Such botnets can launch a variety of different attack types. While different classifications exist for DDoS attacks [41, 73], this thesis follows the classification of [37], where two different categories of attacks are distinguished.

2.1.1.1 Bandwidth depletion attacks

The goal of bandwidth depletion attacks is to consume the available bandwidth of a victim's server so that the online presence becomes unavailable for regular users. These

attacks are often launched from botnets and can additionally use amplification to increase their impact further. Amplification attacks leverage different amplification vectors to increase traffic volumes directed to a victim server. A prime example is reflection attacks, where an attacker sends a request to a server with a spoofed source IP address to reflect the response to the victim. Reflectors that return a larger response than the initial request can be used to amplify a DDoS attack. For example, this has been done with DNS (Domain Name System) queries where the response was shown to be 73 times larger than the initial request [49].

Besides amplification attacks, certain network and transport layer protocols can also exploit bandwidth depletion attacks. The most prevalent bandwidth depletion attack is the UDP flood attack. If a server receives a UDP packet on a specific port, it tries to look up an associated process. However, for an arbitrarily chosen port with no corresponding process, the server returns an ICMP message stating that no corresponding application was found. The sheer mass of UDP traffic can overwhelm a server or even a firewall in large traffic volumes. Also, the ICMP protocol can be exploited to launch DDoS flood attacks.

2.1.1.2 Resource depletion attacks

Resource depletion attacks aim to crash the CPU or memory of a victim system, often termed application-layer attacks. In the early days, single malformed packets were able to crash systems by causing infinite loops [37] or buffer overflows [27]. Today, malformed packet attacks such as the ping of death are less common because they are patched on new systems or discarded by the Web Application Firewall (WAF) rules [13]. Protocol exploits such as the TCP SYN attack are more frequent. In an SYN attack, all available connections of a server get consumed by intentionally not finishing the three-way handshaking mechanism.

Along the same lines, HTTP flood attacks exploit the HTTP protocol to exhaust server resources. This is usually achieved by making a large number of HTTP GET or POST requests which are known to trigger expensive computation on the server (*e.g.,* downloading a large file). Such requests are almost similar to real traffic and, thus, extremely hard to detect.

2.1.1.3 IP Spoofing

Many of the previously discussed attacks are launched using IP spoofing techniques. With IP spoofing, an attacker intentionally falsifies the source address field of the packet header, thus making it hard to track the true source of an attack. This is only possible because of the lack of source address validation (SAV) in the TCP/IP protocol, which allows the source IP of a packet to be easily manipulated.

Network ingress filtering [54] tries to address the problem of spoofed source IPs by checking whether incoming packets are consistent with the routing table of a network device. However, universal implementation remains challenging due to its complexity, which is one of the main reasons why IP spoofing remains an unsolved problem.

2.1.2 Network Monitoring

Network monitoring is essential for detecting and mitigating DDoS attacks. It can generally be divided into active and passive approaches [30]. Active methods perform measurements by sending traffic to the network using tools such as Ping or Traceroute [63]. Passive approaches monitor existing traffic that transits through a measurement point without generating additional network traffic. The following sections further explain two often-used examples of passive network monitoring, packet capture, and flow-based monitoring.

2.1.2.1 Packet Capture

Packet capture represents the most comprehensive form of passive network monitoring. It entails recording the entire packet data, demanding significant computational resources and storage capacity. Two main approaches for accessing the network traffic exist.

- **Test Access Ports (TAP)** are designed to duplicate all network traffic between two network devices [66]. TAPs can be used without introducing latency, but they require additional hardware and, thus, access to the physical link of the network to observe.
- **Switched Port Analyzers (SPAN)** mirror network traffic from one or multiple ports to another. Such functionality is implemented in most packet-forwarding devices nowadays [30], and no additional hardware is required. The main drawback is that it can cause delays and jitter [75].

Multiple tools can produce packet capture (PCAP) files from the observed network traffic. Tcpdump [61], and Wireshark [67] are among the most popular [25]. Typically, these tools operate on a dedicated capture device that receives the traffic via TAP or SPAN. While Tcpdump is only a command-line tool, Wireshark provides a graphical user interface (GUI) and more advanced filtering capabilities. Wireshark can also be used without GUI with the bundled `tshark` command-line interface (CLI).

2.1.2.2 Flow-Based Monitoring

Flow-based monitoring offers a less exhaustive yet more scalable solution for network monitoring [30]. A Flow is a set of packets with common properties recorded at an observation point within a certain time interval [2]. Flows are effective for insights into traffic patterns and network usage. However, due to the aggregation of multiple packets into one flow, they are not suitable for in-depth inspection and analysis of network traffic.

Typically, flows are captured directly on network devices (*i.e.,* routers and switches) that support the Netflow [12] standard. For network devices that do not support Netflow, it is also possible to use a 3rd-party Netflow probe (*i.e.,* capturing device) connected

to a SPAN port. Netflow probes can also be configured to use a sampling mechanism (*e.g.,* every n-th packet), reducing the number of packets to analyze and increasing the scalability.

Flow-based monitoring and packet capture lay the foundation for network monitoring and are essential for the generation of attack fingerprints and signatures, as described in the next section.

2.1.3 Attack Fingerprints / Signatures

Attack fingerprints, often called attack signatures, are frequently used to recognize known cyber attacks. Fingerprints contain characteristic information about an attack and are extracted from network traces. The characteristics of an attack fingerprint or signature (*e.g.,* target port) can be used to mitigate the attack. Many Intrusion Detection and Prevention systems (IDS / IPS) utilize fingerprints or signatures as a common practice to combat cyber attacks. To expand the view of a single entity, central repositories allow attack fingerprints to be shared. Such repositories often use a proprietary fingerprint format, as there are different use cases and interpretations of what information a fingerprint must contain. For example, when installing Snort [51], one can download several known fingerprints/signatures from a centralized (and remote) database [57].

In the context of DDoS, the interpretation of a fingerprint ranges from a dedicated attack [15, 31] to a simple means to detect an attack [35, 65], including observed characteristics such as target ports, used protocols, source and destination IP addresses, and other features. A detailed overview of fingerprinting methods concerning DDoS attacks is presented in Chapter 2.2.

One limitation of fingerprint or signature-based detection techniques remains: They can only be used to detect known attack patterns. For novel attacks (zero-days), where no fingerprint or signature is available in a shared repository, signature-based techniques fail, and different approaches, such as anomaly detection, must be used.

2.2 Related Work

The detection and mitigation of DDoS attacks have been an active research topic for several years, starting when the first attacks hit the University of Minnesota in 1999 [20]. Nowadays, DDoS attacks employ a variety of different techniques and protocols. Therefore, it is not surprising that also a variety of DDoS mitigation mechanisms exists which can be deployed in different locations: Source-based mitigation techniques installed on network infrastructure [40], and in the cloud, [29] can be effective under certain conditions. Still, they require software to be installed on networking devices that lie on the path of the attack traffic. Hence they are not simple to deploy and require a coordinated effort. Destination-based mitigation techniques are deployed on edge, or access routers of the attack target (*i.e.,* victim) [73]. Here, fingerprint-based mitigation techniques can be applied, which are relevant to the scope of this work.

2.2.1 DDoS Fingerprinting

The literature on DDoS attack fingerprinting does not consistently define what a fingerprint means. Lee and Shieh [35] consider the route a packet has traversed between source and destination to be its fingerprint. They propose a filtering scheme that recognizes and drops packets with a spoofed IP address based on their path fingerprint.

Osanaiye [46] denotes a fingerprint as the operating system of an attacking device. A combination of active and passive fingerprints detects whether an IP packet is spoofed. The passive fingerprint is generated by analyzing incoming packets, the active fingerprint is by probing the source IP. An IP packet is considered valid if both fingerprints extracted from operating systems match.

Wang et al. [65] employ the hop count of an IP packet as a fingerprint against which future requests are compared to filter out packets with spoofed IP addresses. They infer the hop count from the IP header's Time-to-Live (TTL) field for each packet. In the learning state, the *Hop-Count-Filterig* (HCF) technique generates and collects fingerprints. Once packets that do not match the stored fingerprint for that IP are detected, HCF changes to the filtering state and discards those packets.

The Bot-Finger-Printing (BotFP) method by Blaise et al. [6] creates fingerprints that model the communication patterns of hosts in a network. For each host, a fingerprint is derived based on the behavior in the network. Afterward, using a clustering algorithm, normal and bot communication are categorized. New hosts are then classified according to their distance to the clusters.

Van Hove [31] defines a fingerprint as a set of source IP addresses, destination and source ports, transport layer protocol, application layer protocol, and specific application layer attributes. He proposes a tool (DDoS Dissector) that generates attack fingerprints from packet-based measurement data. Conrads [15] further improved the DDoS Dissector by allowing it to analyze flow-based measurement data.

The fingerprint definition of [31] and [15] is closest to the fingerprint definition of this thesis, as the fingerprint represents a standalone view of a distinct DDoS attack. The problem with other presented fingerprinting methods is that they are only a means to detect an attack, but they do not uniquely characterize an attack. As such, they cannot be used to comprehensively analyze a certain DDoS attack. Furthermore, not all methods can be deployed directly but require a change in the underlying Internet protocol (*e.g.,* path fingerprint [35]).

2.2.2 A Global DDoS Attack View

All of the above-mentioned fingerprinting methods are destination-based. They help to detect and mitigate an attack on the target's side. This is certainly important from a victim's perspective. However, more global views of DDoS attacks are also required to mitigate such threats efficiently. In this regard, Akella et al. [3] propose a detection method that helps ISP networks detect attacks on themselves and external attacks that

use a certain ISP network. Their detection method relies on stream-sampled profiles of normal traffic and applies anomaly detection. Based on traffic profiles at each router, Akella et al. [3] define several fingerprints. Aggregated fingerprint data is then shared among the routers in the ISP networks using a simple consensus mechanism based on a predefined confidence threshold. However, a recent study by [59] revealed that ISPs are often not interested in joining a collaborative defense due to a lack of financial incentives.

Another approach to get a global view of DDoS attacks is looking at darknet traffic (*i.e.,* routable but unused Internet addresses). This idea was initially proposed in [43] and was based on the assumption that DDoS attackers randomly generate a spoofed source IP for each packet. When observing a large enough IP range (*e.g.,* 1/256 of the IPv4 space), Moore et al. [43] were able to sample an attack overview of the whole Internet space using backscatter analysis. Following these footsteps, Fachkha et al. [22] extend this idea without relying on backscattered analysis. They conduct a flow-based traffic analysis on DNS queries to the darknet space. Their results show that the global increase of DNS queries of type *ANY* is caused by DNS amplification attacks.

Also observing 1/256 of the IPv4 Space, Du and Shanchieh Jay [21] follow a different approach to get a global view and detect collaborative cyber attacks. They define an Attack Social Graph (ASG) from the recorded traffic, borrowing concepts from social network analysis. Using Principal Component Analysis (PCA) and hierarchical clustering, they reduce the ASG to discover collaborative attacks.

As presented, approaches exist to gather a global view of DDoS attacks. However, these reports are gathered in hindsight over months up to years [22, 43]. In this regard, federated learning provides another approach for detecting DDoS attacks on a global scale. The FLDDoS solution proposed in [74] uses federated learning to improve the local modal of a client with the learnings of other clients, without actually sharing the data. The authors further address the data-imbalance problem in federated learning using a clustering algorithm based on K-means. Clustering techniques have also been employed by other researchers in the context of DDoS attacks, a topic that is discussed in the following section.

2.2.3 Clustering DDoS Attacks

In recent DDoS literature, clustering has been mainly applied to low-level data points (*i.e.,* packet or flow data) and not DDoS attacks. For example, Lee et al. [36] use cluster analysis to separate DDoS attacks into different phases, allowing them to identify precursors of DDoS attacks. For features, they mainly rely on the concept of entropy (*e.g.,* how diverse are source IP and port numbers). With nine features of the IP packet in total, they use Euclidean distance to measure dissimilarities between data points. After that, the clustering is performed using Ward's minimum-variance method.

Another clustering approach of low-level data points is presented by Gu et al. [26]. Their proposed BotMiner framework aims at detecting botnets by performing a cross-cluster correlation between two clustered data planes. The first plane (C-Plane) is based on TCP and UDP flow records and represents "who is talking to whom" [26]. The clustering of

the C-Plane is done using the *X-means* [48] algorithm. The second plane (A-Plane) is based on Snort data [51], representing malicious communication activities. It is clustered by activity type and activity features. Overall, the authors show that the cross-cluster correlation can successfully detect botnets with a low false positive rate.

Cluster analysis has also been applied to application-level attacks. Ye et al. [68] propose a method to cluster a user's sessions to capture browsing behavior. Based on four features per session, such as page popularity or average transition probability, they create clusters from normal browsing behavior data. Similar to [36], new data is assigned to a cluster using Euclidean distance and Ward's minimum-variance method.

Thoma and Hadjicostis have proposed a less common clustering method [62]. They use a Hidden Markov Model to create a time-dependent analysis of cyber attacks. Their proposed approach models both malicious and normal traffic packets to detect DDoS attacks.

2.2.4 Datasets

There exist different datasets for the evaluation of DDoS attack analysis and mitigation methods. A often used dataset is the KDD'99 Cup dataset [47], which contains simulated SYN flood attack data. Its drawback is the imbalance of data, with 80% being attack traffic [4].

The DARPA-2009 dataset contains real attack data of SYN floods targeting a single IP address from more than 100 sources [24]. Besides the attack traffic, the dataset also contains background traffic.

A more recent dataset is provided by Sharafaldin et al. [55]. The CIC-DDoS2019 evaluation dataset provides raw network capture data (PCAP) for different DDoS attack types executed on a testbed architecture. Similar to the other mentioned datasets, CIC-DDoS2019 only provides a single view of the DDoS attack using one monitoring location. In that regard, a lack of datasets that provide multiple views of the same attack can be observed.

This Chapter has provided an overview of the existing work on methods for DDoS attack fingerprinting and the different approaches for detecting and analyzing DDoS attacks. It has been shown, that there is no unified definition of an attack fingerprint and various detection methods employ different fingerprint formats [6, 31, 35, 46, 65]. However, not all of them are ready to be deployed, as they require a change to the underlying Internet protocol [35]. While most of the discussed fingerprinting and detection methods are destination-based, several approaches for globally analyzing DDoS attacks have been presented [3, 22, 43]. They provide interesting insights on how DDoS attacks can be observed, focusing on different characteristics. What still needs to be explored is a global analysis using attack fingerprints which inherently provide value for the creator of the fingerprint. As such, the fingerprint format proposed by [31] is of special interest. Lastly, the presented datasets show that real and simulated attack data can be used for evaluation [24, 47, 55]. However, they also highlight the lack of datasets providing several views of the same attack using more than one recording location.

Chapter 3

Design

Different approaches to observe and aggregate global views of DDoS attacks have been introduced in the last Chapter. While the presented approaches usually apply algorithms on low-level network-capture data, this Chapter presents a novel solution for deriving global insights into a DDoS attack directly from attack fingerprints recorded at different locations. A high-level architecture of this solution can be found in figure 3.1.

The underlying concept is that each node in the network analyzes its network traffic to create unique DDoS attack fingerprints. After calculating fingerprints, participating nodes share their collection with a central point. With fingerprints from multiple nodes, the so-called *Reassembler* analyzes and aggregates DDoS attacks globally. In the end, the Reassembler provides an output file that describes the global key characteristics of a potential DDoS attack.

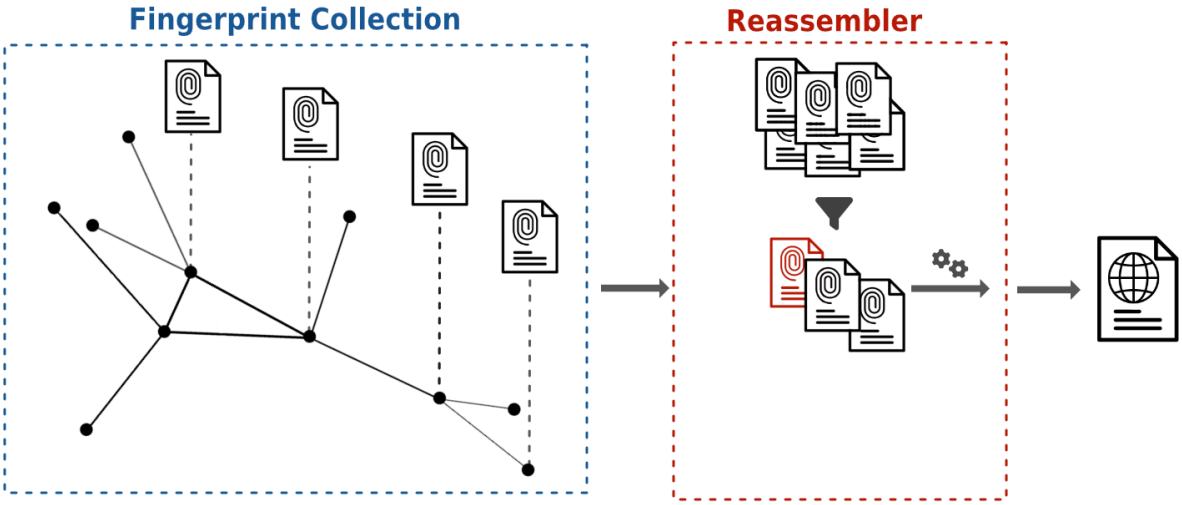


Figure 3.1: High-level architecture of the Reassembler process

The DDoS attack fingerprint format proposed by [31] is used and extended for fingerprint collection. However, a custom scenario is developed to generate realistic attack fingerprints due to the lack of real data sets containing DDoS attack traffic recorded at different nodes

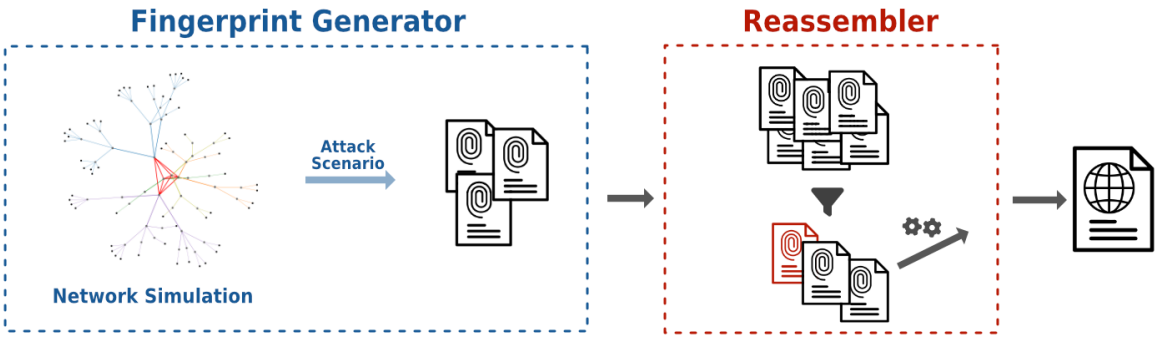


Figure 3.2: High-level architecture using a Fingerprint Generator

(*cf.* Chapter 2.2.4). In this regard, the fingerprint collection (*cf.* Figure 3.1) is replaced by a *Fingerprint Generator* (*cf.* Figure 3.2).

The following subchapters outline the design of the presented high-level architecture as follows: Chapter 3.1 presents the Fingerprint Generator and discusses how the simulation is modeled to imitate real attack fingerprints. Thereafter, Chapter 3.2 introduces the Reassembler component used to produce DDoS global attack statistics in a compact file.

3.1 Fingerprint Generator

Existing DDoS data sets are often limited in their applicability for deriving global insights since they are usually only recorded at one node. While a single node may cover a large subnet [24], it only provides one point of view. However, multiple points of view are required to derive a more precise view of a globally distributed attack using underlying attack data. One possible approach is to modify the captured data of a single node manually, thereby creating multiple new points of view from one initial point of view. The disadvantage is that manually splitting up an existing data set to multiple points of view is time-consuming because values such as TTL or timestamps have to be manipulated in a way that makes sense in the context of an IP network. For example, a router sending attack traffic to a victim should see the traffic with a larger TTL value than the victim itself. Even though it is possible to manually create the underlying capture data for simple scenarios, the approach does not scale when increasing the complexity of the network topology. In this regard, a custom scenario directly on the attack fingerprint level is used to model attacks within more complex network topologies.

3.1.1 Simulated Network Topology

The IP network topology of the current Internet is complex as it connects numerous devices daily. There are more than 100'000 registered Autonomous Systems (AS) [1], each containing different-sized and non-consecutive blocks of IPs. It becomes clear that a simulated network topology must abstract the underlying complexity while preserving the

most important aspects relevant to DDoS attack fingerprint generation. In that regard, the following requirements for a simulated network topology have been considered:

1. **Hierarchical structure**
   Similar to the IP network topology of the current Internet, the simulated network topology must be hierarchical with a dynamic amount of layers.

2. **Multiple Subnets**
   The simulated network topology must span multiple subnets with different Classless Inter-Domain Routing (CIDR) prefixes.

3. **Valid IPs**
   Each node in the network topology must have a valid IP address that belongs to an existing subnet.

4. **Realistic Backbone architecture**
   Subnets must be connected such that a valid route between any two nodes in the topology exists.

5. **Non-trivial routing algorithm**
   Different routes must have different costs to apply a realistic routing algorithm. This also includes that the network cannot be fully connected to avoid trivial routing.

3.1.1.1 Basic Topology

The most basic network topology that fulfills all of the above-mentioned requirements is shown in Figure 3.3. It consists of the following elements:

- **Subnets**
  In the TCP/IP architecture, a subnet is a subdivision of an IP network used to enhance routing efficiency and simplify administration. The simple scenario in figure 3.3 consists of two subnets (green and purple) using a 16-bit network prefix. For the sake of simplicity, a subnet in the scenario represents an Autonomous System.

- **Intermediate Nodes**
  Nodes within a subnet that are not on the lowest level in the tree are denoted as intermediate nodes (gray). In reality, such nodes represent different types of network hardware (*e.g.,* core or edge routers). However, this distinction is omitted to keep the scenario manageable.

- **Backbone connections**
  Backbone connections (red) connect the topmost nodes of each subnet. In the simplest scenario (*cf.* Figure 3.3), there are only two nodes to connect. Other backbone topologies for more complex scenarios are discussed in the following Chapter (*cf.* Table 3.1).

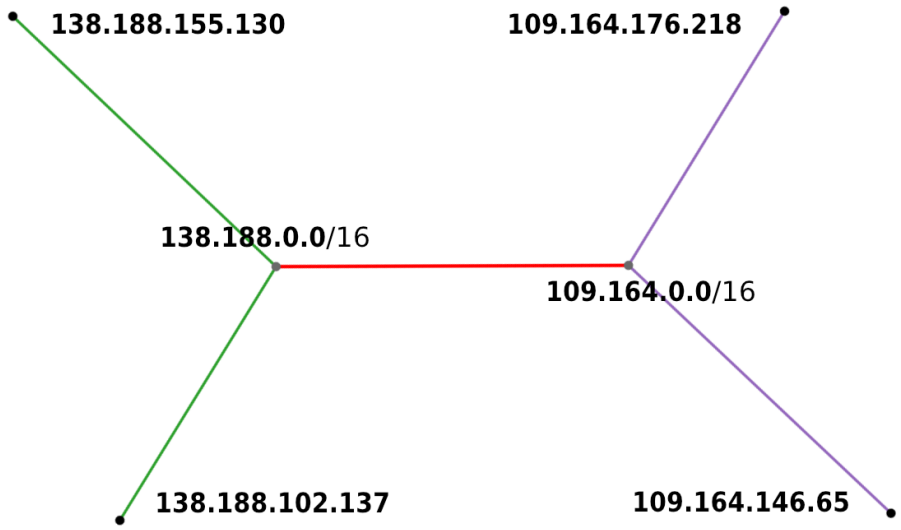


Figure 3.3: Most simple network topology

- **Clients**
  Nodes on the lowest level of the subnet are denoted as clients (black). All clients have a random IP based on the subnet they belong to.

While this architecture fulfills the requirements for simulated network topology, it is unsuitable for a more complex scenario as it is too small and does not provide interesting routes between nodes. An extension of this basic scenario, which provides a more sophisticated foundation for the global evaluation of DDoS attacks, is presented in the next section.

3.1.1.2 Extending the Topology

Due to its hierarchical structure, the simple topology (*cf.* Figure 3.3) can easily be extended with additional subnets and more levels per subnet. Figure 3.4 shows a more sophisticated architecture with five subnets. Each subnet can have a different amount of levels, thereby forming a tree structure. For example, the blue subnet has six levels, whereas the orange subnet has only four. Since the number of nodes per level is randomized, one can see that the orange subnet contains more nodes than the blue subnet, even though the blue one has more levels.

With multiple levels per subnet, the prefixes along the path must be adapted to simulate a realistic network topology. In other words, nodes on the same subnet tree on different levels cannot have the same network prefix. Otherwise, the same IP address might occur at different levels of the subnet tree. A simple approach to prevent this issue is to increase the network prefix on each level of the subnet tree. The number of bits to increase per level can either be a pre-defined value (*e.g.,* 4 bits per level) or dynamic based on the height of the subnet tree. Those approaches are further elaborated in Chapter 4.

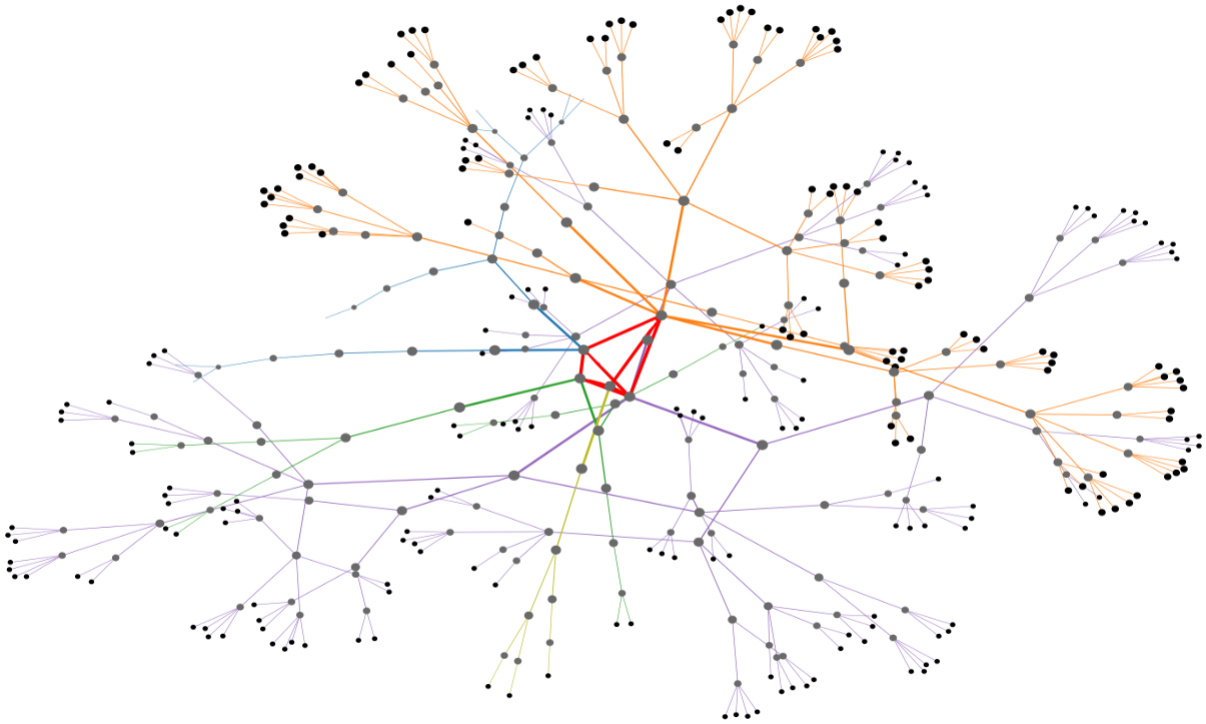


Figure 3.4: Sophisticated network topology

While the backbone connection with two subnets is trivial, there are different possibilities for network topologies with more than two subnets. An example of a backbone topology with multiple subnets is shown in Figure 3.4. It uses a partially connected mesh backbone network where each subnet root node is connected to two or more other root nodes. Besides the partial mesh topology, other approaches are discussed in Table 3.1. Even though ring or star topologies are simpler to implement, the partial mesh topology (*cf.* Figure 3.4) is used for the network simulation because it closely reflects a realistic global network topology.

3.1.1.3 Routing

Routing protocols of the current Internet are based on hierarchical structures involving several interconnected networks. On the highest level, Autonomous Systems make use of Exterior Gateway Protocols (EGPs) such as the Border Gateway Protocol (BGP) [50]. Within ASes, Interior Gateway Protocols (IGPs) facilitate routing. Examples include the Routing Information Protocol (RIP) [38] and the Open Shortest Path First (OSPF) [44] protocol.

In a simulated scenario, the routing path between two nodes is decided from the outside with a global network topology perspective. The routing path within a hierarchical subnet is straightforward since only one path exists between two nodes. Routing across multiple subnets is more interesting because there are multiple paths through the backbone network. The routing path selection is made based on the shortest path algorithm. A random weight property per edge is introduced to make the selected paths less trivial.

	Advantages	Disadvantages	Applicability for Scenario
Ring	• Simple to implement in scenario • Access to all nodes with few connections	• Not realistic for large networks	Simple to implement but an unrealistic simulation of a real network
Star	• Simple to implement in scenario • Access to all nodes with few connections	• Single point of failure in the real world	One central node which collects all inter-subnet traffic.
Mesh	• High redundancy and fault tolerance in the real world • Simple to implement in scenario	• Not realistic for large networks • Quadratic growth of connections	All routes between subnets are trivial and thus less interesting for a scenario.
Partial Mesh	• Closest reflection of a real global network topology	• Higher complexity	Randomly connecting all subnets such that a path between every node exists is more complex.

Table 3.1: Backbone Topologies

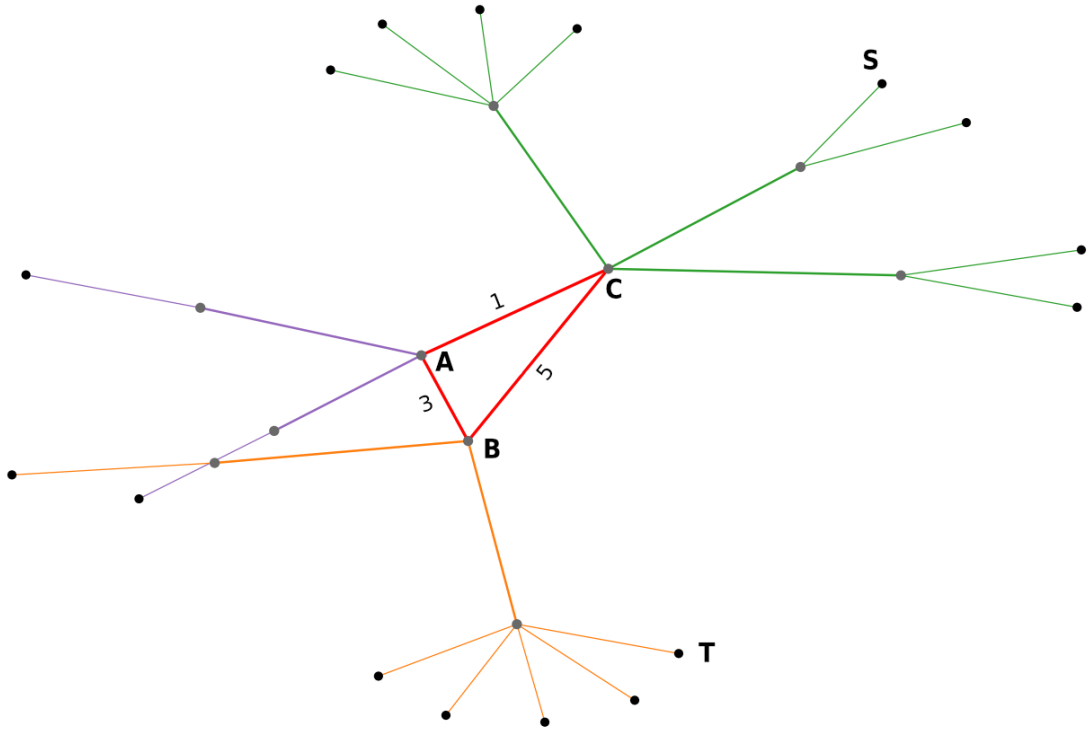


Figure 3.5: Network topology with weighted edges

This results in shortest paths between nodes, which are not always minimal in hops. An example is shown in figure 3.5 where the path between source S and target T is shorter via an additional node A due to the large edge weight between B and C.

The presented network topology for the simulated scenario abstracts the complexity of today's internet infrastructure while still maintaining key aspects such as subnets and backbone architecture. It has to be noted that the presented topology is not used for simulating low-level packet data but directly on the DDoS attack fingerprint level. The design of attack fingerprints is discussed in the next section.

3.1.2 Attack Fingerprint Format

In literature, different definitions of DDoS fingerprints exist (*cf.* Chapter 2.2). This thesis adopts the attack fingerprint definition proposed in [31]. According to this perspective, an attack fingerprint represents a DDoS attack originating from one or multiple sources and directed toward a singular target IP address. Table 3.2 shows the full list of properties of an attack fingerprint along with their description. The list of `attack_vectors` is an important property of attack fingerprints. Attack fingerprints contain one or multiple `attack_vectors`, each characterized by a unique combination of source port and protocol pairs (*cf.* Table 3.3).

The generation of such attack fingerprints happens with a dedicated program, the so-called *DDoS Dissector* [17]. The Dissector analyzes PCAP or NetFlow files and outputs JSON data containing a single fingerprint in the format mentioned above. As can be seen

Field name	Description	Datatype
`attack_vectors`	List of attack vectors describing the attack (*cf.* Table 3.3)	Array of objects
`target`	IP address or subnet of the attack target	String
`tags`	Tags assigned to this attack, *e.g.,* "TCP SYN flag attack"	Array of strings
`key`	MD5 hash of the fingerprint, which serves as identifier and file name of the fingerprint	String
`time_start`	Start timestamp of the attack (time zone local to the attack target)	DateTime
`time_end`	End timestamp of the attack (time zone local to the attack target)	DateTime
`duration_seconds`	Duration of the attack in seconds	Integer
`total_megabytes`	Total volume of the attack in megabytes (MB)	Integer
`total_packets`	Total number of packets in the attack	Integer
`total_ips`	Count of unique source IP addresses	Integer
`avg_bps`	Average number of bits/s during the attack	Integer
`avg_pps`	Average number of packets/s during the attack	Integer
`avg_Bpp`	Average number of Bytes per packet	Integer

Table 3.2: Attack Fingerprint format proposed by [31]

from the `target` property, the dissector automatically derives a target from the underlying capture data. This functionality employs a straightforward detection mechanism, identifying IP addresses that received more than 50% of the analyzed traffic. If the 50% threshold is unmet, the Dissector looks for a target subnet. Alternatively, if automatic detection is not desired, it is possible to override the detection mechanism with a custom target.

While the above-mentioned mechanism of the DDoS dissector is optimal for analysis at a single point (*i.e.,* at the victim), it poses different problems when fingerprints from multiple recorded locations are combined. To overcome the challenges, this thesis proposes an extension of the existing fingerprint format.

3.1.2.1 Extended Fingerprint Format

The existing DDoS Dissector solution [31] is well suited for quickly gathering an overview of attack vectors for a target that is attacked with high certainty. However, for a global analysis, it is also desirable to have attack fingerprints with less certainty as they can be interesting in a global context. For example, with a high threshold of 50%, only nodes that are very close to the target (and the target itself) report a fingerprint, but with high certainty. Targets that receive less than 50% are discarded, and no fingerprint is

Field name	Description	Data type
`service`	Name of the service used in the attack vector, determined by the source port and protocol	String
`protocol`	IP protocol (*e.g.,* TCP, UDP, ICMP)	String
`fraction_of_attack`	The fraction of the entire DDoS attack that the attack vector accounts for $[0, 1]$	Float
`source_port`	Source port of the attack vector if source port and protocol are associated with a specific service	Integer or ”random”
`destination_ports`	List of outlier destination ports with the corresponding fraction of the traffic	Map<String, Float> or ”random”
`tcp_flags`	List of outlier TCP flags with the corresponding fraction of the traffic. Null if the protocol is not TCP, or there are no outliers	Map<String, Float>
`nr_packets`	Count of packets in the attack vector	Integer
`nr_megabytes`	Number of megabytes of traffic caused by the attack vector	Integer
`time_start`	Start timestamp of the attack vector: the first packet of this attack vector (timezone local to the attack target)	DateTime
`duration_seconds`	Duration of this attack vector in seconds (last timestamp - first timestamp)	Integer
`source_ips`	List of unique IP addressed that sent traffic to the target within the attack vector	Array of strings

Table 3.3: Attack Vector format proposed by [31]

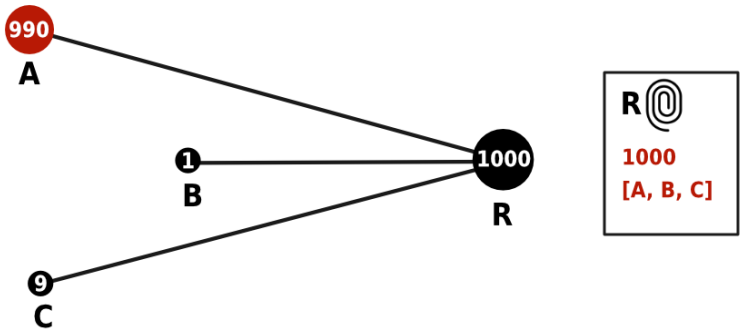


Figure 3.6: Loss of source context in fingerprint aggregation

generated. With a lower threshold (*e.g.,* <10%), nodes farther away might also produce a fingerprint with lower certainty. While a single fingerprint with low certainty cannot identify or describe an attack, it helps build a global picture. This thesis proposes several extensions to the existing fingerprint format of [31].

When comparing multiple fingerprints recorded at different locations, it becomes nearly impossible to map fingerprints to their recording location without additional information. Although some assumptions about the recording location can be drawn from the observed IP addresses and TTL values, it is not precise enough for a global analysis. In this regard, a new `location` property that identifies the location where the fingerprint was captured is proposed.

Another shortcoming of the existing fingerprint format for a global analysis is the loss of context along an attack path. Since the number of packets and the source IPs are aggregated on every node that produces a fingerprint, much information is lost. An example of this can be found in Figure 3.6. Router R receives traffic using the same protocol and port from three different sources. Client A maliciously sends a lot of packets (*e.g.,* SYN flood), while client B and C solely try to access the website. Due to the aggregation per port and protocol in the fingerprint, router R calculates a fingerprint with three attack sources and a total number of 1'000 packets. This means that non-malicious source IPs are included in the fingerprint if they use the same protocol and port within the same timeframe. Such behavior can be detected using the reported TTL values to a certain extent. However, if all arriving packets at router R have the same TTL, the fingerprint makes no distinction between malicious or non-malicious source IPs.

The loss of source context can be prevented by including a mapping from source IPs to the number of packets. A mapping from source IPs to observed TTL values is further proposed for additional analytics. As mentioned before, such properties in the fingerprint are not required when generating a fingerprint at a single location with high certainty. However, the additional information provides an interesting basis for further analysis of a global picture. The additional proposed properties are summarized in Table 3.4.

Level	Field name	Description	Data type
Fingerprint	`location`	The IP of the location where the fingerprint was recorded	String
Attack Vector	`nr_packets_by_source`	Number of packets in the attack vector grouped by source IP	Map<String, Integer>
Attack Vector	`ttl_by_source`	Observed TTLs in the attack vector grouped by source IP	Map<String, Integer[]>
Attack Vector	`detection_threshold`	Percentage of attack traffic compared to all observed traffic	Float

Table 3.4: Additional proposed properties for attack fingerprints

3.1.3 Attack Scenario

The Fingerprint Generator generates a set of unique Attack Fingerprints based on a global view of the above-presented topology (*cf.* Chapter 3.1.1). Unlike the DDoS dissector, which relies on low-level captured network data to generate fingerprints, the Fingerprint Generator produces fingerprints directly from an underlying attack scenario. In its most basic form, an attack scenario consists of one or multiple attack sources and one single target. Unsurprisingly, an attack scenario must consider more parameters to generate a set of interesting attack fingerprints. In this regard, the additional properties of an attack scenario are discussed in the following sections.

3.1.3.1 Attack Types

DDoS attacks are constantly evolving, and they come in various forms. The overall taxonomy of DDoS attacks is discussed in Chapter 2.1.1. For the Attack Scenario, this thesis focuses on one of the most prevalent DDoS attack types, the SYN flood. The idea of an SYN flood is to exploit TCP's three-way handshake mechanism by intentionally not responding with an ACK packet. Compared to bandwidth depletion attacks (*cf.* Chapter 2.1.1), the SYN flood provides more possibilities for an interesting analysis since attack traffic and regular traffic might look very similar. For example, a regular visitor of a website has to complete the same TCP handshake that an attacker is actively trying to exploit. Conversely, for bandwidth depletion attacks, packets can be clearly distinguishable from normal traffic, but they still fulfill the goal if they exhaust the physical link to the target.

In the real world, SYN flood attacks can occur in three different ways [14]:

- **Direct Attack**
  One single attacker sends a flood of SYN packets without masking (*i.e.,* spoofing) its own IP address. Because such an attack is fairly easy to mitigate, it is rarely used in practice.

- **Spoofed Attack**
  This attack is similar to a direct attack, but the attacker actively hides his identity by spoofing the IP address of each packet.

- **Distributed Attack**
  A distributed attack involves many attack sources. Based on the number of attacking sources, an attacker can additionally mask the true origin by spoofing the IP address. However, for large botnets, such an overhead is unnecessary since it is not possible anyways to locate the source of the attack.

For the attack scenario of this thesis, a distributed SYN attack with partially spoofed sources is selected. A reference attack fingerprint based on the Dataset of Sharafaldin et al. [55] can be found in Listing 3.1. The reference attack fingerprint describes an SYN attack with a duration of approximately 6 hours. Both `source_port` and `destination_ports` are reported as *random* and thus, provide no value for further analysis. Interestingly, only one `ttl` value has been observed despite the occurrence of multiple `source_ips`. Since it is very unlikely that all sources have exactly the same distance to the target, the attacker most likely made use of IP spoofing techniques. Another characteristic of an SYN attack fingerprint can be found in the `tcp_flags` object. The string "`.....S.`" represents a TCP packet header where the SYN flag is set to true. From the fingerprint, it can be seen that over 99% of arriving packets at the target contained this exact header (*cf.* Listing 3.1, Line 9ff.).

The attack scenario requirement for the Fingerprint Generator can be derived based on the fingerprint characteristics of the reference fingerprint (*cf.* Listing 3.1). First, an attack aims only at a single target and may originate from multiple sources. Furthermore, attack sources likely make use of IP spoofing techniques. In this regard, it is also required to have realistic TTL values even in a simulated scenario. While selecting a single attack target and one or multiple attack sources is relatively straightforward, the next section discusses how the spoofing of IP addresses can be modeled.

3.1.3.2 Spoofed Addresses

The lack of authentication in the TCP/IP protocol allows anyone to modify the source field of an IP packet. This practice, known as IP spoofing, is frequently employed by malicious actors when launching DDoS attacks. The malicious actors use this to hide their identity by masking the true source of an attack packet. To obtain a realistic attack scenario, the Fingerprint Generator has to replicate this malicious behavior. Based on the presented topology in Chapter 3.4, a spoofed IP is denoted as an address of a leaf node that does not belong to the subnet where the node is located. In this regard, a node in the simulated topology can communicate using its valid IP (belonging to the subnet) or a random spoofed IP.

```
{
    "attack_vectors": [
        {
            "service": null,
            "protocol": "TCP",
            "fraction_of_attack": 1.0,
            "source_port": "random",
            "destination_ports": "random",
            "tcp_flags": {
                "......S.": 0.997
            },
            "nr_packets": 2093500,
            "nr_megabytes": 127,
            "time_start": "2018-11-03T15:28:00.776482+00:00",
            "duration_seconds": 22067,
            "source_ips": [
                "172.16.x.x",
                ...
                "52.84.x.x"
            ],
            "ethernet_type": {
                "IPv4": 1.0
            },
            "frame_len": {
                "60": 0.998
            },
            "fragmentation_offset": {
                "0": 1.0
            },
            "ttl": {
                "243": 0.997
            }
        }
    ],
    "target": "192.168.x.x",
    "tags": [
        "TCP flood attack",
        "TCP",
        "TCP SYN flag attack"
    ],
    "key": "60764fd912f7bd9f047ca885f652758f",
    "time_start": "2018-11-03T15:28:00.776482+00:00",
    "time_end": "2018-11-03T21:35:48.097688+00:00",
    "duration_seconds": 22067,
    "total_packets": 2093500,
    "total_megabytes": 127,
    "total_ips": 25,
    "avg_bps": 46066,
    "avg_pps": 94,
    "avg_Bpp": 60
}
```


Listing 3.1: DDoS Dissector Reference Attack Fingerprint based on CIC-DDoS2019 [55] Dataset

One of the significant benefits of having a global view of a DDoS attack is the ability to detect and analyze spoofed addresses more effectively. By examining the attack from multiple points (*i.e.,* intermediate nodes), it is possible to identify patterns and inconsistencies that indicate the presence of spoofed addresses, providing valuable insights into the attacker's methods and potentially aiding in traceback efforts. In this regard, IP traceback approaches have been widely explored in literature [52, 60, 72].

The primary obstacle lies in the Fingerprint Generator directly producing attack fingerprints without simulating individual network packets. Considering this aspect, it is impossible to generate a spoofed IP address for each packet sequentially sent, but a node's spoofed address must be declared beforehand. Simply assigning a single spoofed IP address to a node does not produce realistic attack patterns in a fingerprint, as it serves as an alias but does not influence the number of observed source addresses at the target. Conversely, choosing a new random address for each sent packet in the scenario bloats the attack fingerprints with thousands of addresses. From a detection point of view, this is also not optimal because IP collisions have shown to be a valid approach for detecting spoofed Addresses [45]. Since the pool of all available IP addresses is extremely large, it is unlikely to run into an IP collision in a small-scale simulated scenario.

The Fingerprint Generator aims to produce fingerprints that represent realistic attack patterns while still allowing proven detection algorithms to work, even in small-scale scenarios. Spoofed addresses are not randomly generated from the entire set of available IP addresses but sampled from a smaller subset. This allows IP collisions to occur with a higher probability. An example can be found in Figure 3.7. The two nodes with real IP A and B both draw a sample of the pool of spoofed IP addresses (S1-S9). It can be seen that both nodes have the spoofed IP $S4$ in their list of spoofed addresses. Observing the traffic at router R, the IP $S4$ can be observed with different TTL values because two different nodes use it. How this fact is used for detection and analysis is further discussed in Chapter 3.2. Furthermore, the implications of sample and pool size are discussed in Chapter 5.2.

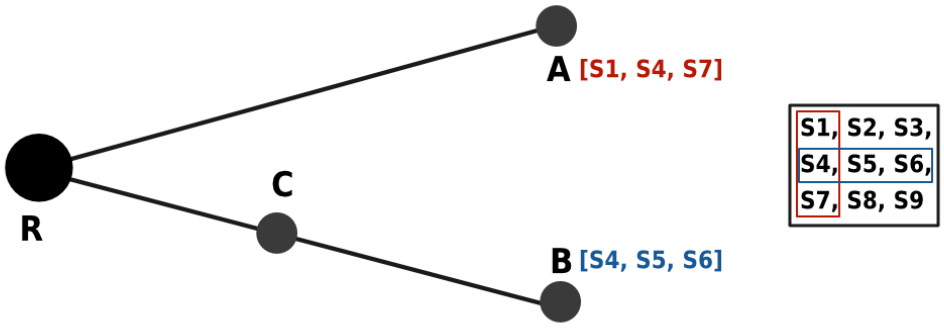


Figure 3.7: Assigning spoofed IPs from a pool

3.1.3.3 Background Traffic

In reality, DDoS attacks do not occur in an isolated environment. When dealing with advanced attack types, it is even challenging to distinguish the attacking traffic from the legitimate traffic. Thus, simulating an attack scenario also requires including background traffic unrelated to the attack itself. For this scenario, three different types of background traffic are considered.

- **Legitimate traffic to the attack target**
  While a DDoS attack is ongoing, the regular traffic to the attack target remains. Examples include regular users who want to visit a web server that is under attack. As mentioned above, such traffic can look almost similar to attack traffic.

- **Spoofed traffic to other nodes**
  Network traffic to other nodes (attack target excluded) uses spoofed source addresses. Examples include other ongoing DDoS attacks that are not directed to the observed attack target.

- **Legitimate traffic to other nodes**
  General traffic in the network that does not involve packets to the attack target. Examples include any web browsing or other online activities of regular users.

3.1.4 Attack Scenario Parameters

Based on the above-discussed challenges and requirements, the simulated attack scenario can be customized in the following way (*cf.* Table 3.5). The number and depth of subnets can be customized on a topology level. These values determine the size of the network and, thus, the average number of hops that are required from an attack source to the target. Because a partial mesh is used as backbone topology, the depth of a subnet and the number of subnets in the network influences the path length between two leaf nodes. The number of nodes per level determines the network's overall number of nodes. Increasing this number makes the network larger while preserving the average hops between the attack source and target.

The attack sources and the target can be configured on an attack level. The number of attacking sources influences the generated fingerprint at the attack target. It also impacts the overall number of fingerprints in the network that observe the same attack from a different angle. Furthermore, changing the probability for a node to communicate using a spoofed IP is possible. If more nodes use a spoofed IP address, the count of attack sources perceived at the target increases, despite the constant network size.

Lastly, the amount of background traffic can be configured with two parameters: The number of unique paths used for background traffic and the probability that a path is directed to the attack target.

Scope	Parameter	Description
Topology	Subnets	A list of subnets along with their IP and network prefix
Topology	Subnet depth	The maximum depth of a subnet. For the hierarchical topology, this equals the max height of the subnet tree
Topology	Nodes per level	The maximum number of nodes per level in a subnet tree.
Attack	Sources	A list of attack sources (by their real IP)
Attack	Target	A single attack target IP address
Attack	Spoofed IP %	The probability that a leaf node communicates using a spoofed IP
Background Traffic	Amount	The number of unique paths in the network that are used for background traffic
Background Traffic	Amount to target	The probability of background traffic being directed to the attack target

Table 3.5: Configuration parameters of the simulated attack scenario

3.2 Reassembler

The Reassembler module generates a global overview of a DDoS attack given a set of attack fingerprints recorded at different locations. A global view of these attacks is crucial to identify patterns and efficiently develop countermeasures. In this regard, the global overview aims to obtain insights that cannot be derived from observing the DDoS attack at a single point. Such insights include, for example, the number of intermediate nodes and their level of involvement in an attack. For example, the number of intermediate nodes helps to understand an attack's scale and assess attackers' resources and strategies. In addition, the level of involvement of intermediate nodes allows for assessing the degree to which each node contributes to the attack, which may be targeted for mitigation efforts and improvement on detection points based on gained insights.

On a high level, the process of the Reassembler can be structured into four steps.

1. Pre-process attack fingerprints
2. Identify Attack
3. Analyze Attack
4. Aggregate Output

While the overall process seems simple, each step contains its challenges. In this regard, the four steps are further elaborated in the following Chapters.

3.2.1 Pre-process Attack Fingerprints

Pre-processing attack fingerprints includes two main actions.

- **Read/Load Attack Fingerprints** from a shared location (*e.g.,* an online repository) Since this thesis uses a simulated scenario, the shared location is emulated by a local folder.
- **Convert Fingerprints** to a format that allows for further processing. In order to make the lookup and filtering of data efficient, fingerprints are converted to a flat (*i.e.,* non-nested) data format. The technical details are discussed in the Implementation (*cf.* Chapter 4.2.1)

While one could argue that loading attack fingerprints falls outside the pre-processing stage, it is included as the purpose of pre-processing is to provide the data for the next step such that it can be analyzed efficiently.

3.2.2 Attack Identification

The notion of an attack fingerprint implies that the fingerprint describes a specific attack and was generated by a previously identified attack. Within the scope of this work, this assumption is weakened so that a fingerprint does not necessarily describe an attack but serves as a simple observation of traffic. On the one hand, this removes the complex logic and computational overhead of identifying an attack from the network nodes. On the other hand, it requires identifying attacks later by looking at a large set of possible attack fingerprints.

This design decision is required because the current implementation of the DDoS Dissector (*cf.* Chapter 3.1.2) does not allow the generation of fingerprints as they result from the simulated attack scenario (*cf.* Chapter 3.1). As discussed, the DDoS Dissector identifies an attack whenever a relatively high threshold (50% of all traffic) is met. Clearly, as opposed to the simulated scenario, nodes further away from the attack target would not produce an attack fingerprint when running the DDoS Dissector. To align the DDoS Dissector's functionality with the simulated scenario results, this thesis proposes several contributions to the DDoS Dissector software (*cf.* Chapter 4.3).

Having many possible attack fingerprints using a low detection threshold leaves the detection of attack targets, intermediate nodes, and attack sources to the Reassembler module and takes away the responsibility from individual nodes.

3.2.2.1 Infer Attack Target

Considering a use case where many participants in the network contribute their recorded attack fingerprints in a shared repository, there are several possibilities for how an attack target is inferred.

- **Attack target is actively reported with fingerprint**
  A network participant actively signals that he has been targeted by an attack because, for example, a web server went offline for several hours. There is no further work needed in identifying an attack as all observed parameters (from the target) are reported in an attack fingerprint.
- **Attack target is not actively reported but submits fingerprint**
  There is no active announcement of a network participant to be targeted. However, attack fingerprints are submitted by using the DDoS Dissector in the background.
- **Attack target and fingerprint not reported**
  The target of the DDoS attack does not announce an attack nor submit an attack fingerprint. Even though other network participants might observe the attack, key observations from the attack target are missing.

For the first two possibilities, the attack fingerprint at the target is collected and shared with others. In the case of a node actively reporting its attack, no additional detection mechanism is needed. Essentially, a global analysis can be created using a pre-defined target. In the second case, a node under attack shares its attack fingerprint without reporting an attack. This leaves the detection of the attack to the Reassembler. Lastly, the third approach does not provide any information about an attack from the target point of view. A global analysis of the attack with the proposed solution is not possible, as the baseline (*i.e.,* the fingerprint at the attack target) is missing.

While the first two possibilities are supported by the proposed design of the Reassembler, this thesis focuses on the second variant, where the attack is automatically inferred from a set of attack fingerprints. This is achieved by applying a detection mechanism comparable to the original detection mechanism of the DDoS Dissector proposed in [31]. The main idea is to use a relatively high detection threshold while leveraging the fact that a node under attack also submits fingerprints. In that regard, an attack can be identified by searching for fingerprints that have recorded an attack on themselves while having a high enough detection threshold.

For a large network with many observed fingerprints, it is possible that multiple fingerprints fulfill the attack detection requirements. In that case, either the most targeted node (*e.g.,* in terms of nr. of packets) or all of the detected target nodes can be evaluated. When evaluating all nodes that record an attack on themselves, the attack threshold for fingerprint generation is of high importance. A too-low threshold results in many possible attack targets that have to be evaluated on a global level, which is computationally expensive. A too-high threshold might lead to overlooking an attack. On a target level (*i.e.,* leaves in the network tree), such a threshold optimization is left for future work, and the single most targeted node that reported being attacked is considered the global attack target.

3.2.2.2 Intermediate Nodes

Detecting intermediate nodes involves the selection of all fingerprints that recorded traffic to the attack target from a different location than the attack target itself. Only excluding

Parameter	Description	Finality
Attack Target	One single attack target of a globally observed attack	Final
Intermediate Nodes	List of nodes that (potentially) observed the attack.	Requires Refinement
Attack Sources	List of attack sources (not filtered for spoofed addresses)	Requires Refinement

Table 3.6: Attack Identification

fingerprints at the attack target results in the possibility that a fingerprint recorded at an attack source is considered an intermediate node. In the real world, such behavior is unlikely as, especially for botnets, the attacking device is neither interested nor computationally capable of generating and sharing a fingerprint. However, from an adversarial point of view, there is the possibility of transmitting a false fingerprint as a cover for a real source of attack.

In that regard, attack fingerprints of attack sources cannot be trusted. Fingerprints from attack sources are already discarded by the Fingerprint Generator for the simulated scenario. In general, not only fingerprints of attack sources but every fingerprint can be manipulated by an adversary. This is further discussed in Chapter 5.2.1.3.

Intermediate nodes include all nodes that observed any form of traffic to the attack target. Thus, the reported number of intermediate nodes is too high at an initial stage since there is no exclusion of intermediate nodes that only observed real (*i.e.,* non-attack) traffic.

3.2.2.3 Sources

As discussed before, fingerprints of attack sources are not available. This means that all attack sources have to be derived from intermediate and target attack fingerprints. In a case where the attack target can capture all network traffic, the information in the fingerprint is enough to identify all attack sources. However, some packets may be lost for resource exhaustion attacks because the physical link is exhausted. This makes the fingerprint recorded at the attack target incomplete, meaning some source IPs are missing. In such a case, the global picture can be restored by looking at the source IPs recorded at intermediate nodes. Since this Thesis uses a TCP SYN attack as a scenario, it does not consider resource exhaustion with dropped packets.

In the scope of this work, source IPs are derived from the fingerprint recorded at the attack target. At the first stage of attack identification, there is no distinction between spoofed and non-spoofed addresses. Therefore, the number of true attack sources is likely decreased when analyzing the attack in a further step (*cf.* Chapter 3.2.3).

In summary, the attack identification step of the Reassembler produces the three outputs parameters summarized in Table 3.6. It can be seen that both intermediate nodes and

attack sources are not final but serve as a basis for further analysis in a third step. This is further elaborated in the next Chapter.

3.2.3 Attack Analysis

The attack analysis is the main step in the Reassembler process. In essence, the analysis aims to refine the detected attack parameters (*cf.* Table 3.6) and derive further insights in the context of a global DDoS attack view. It has to be noted that derived insights from an aggregated fingerprint cannot be as precise as when directly looking at the packet data. However, for a global attack analysis, data must be aggregated into a much smaller format before it can be shared with others to make it computationally feasible globally. With this in mind, the following Chapters present different approaches to building a global picture of an attack, which is not meant to be 100% accurate. The parameters that influence the precision of the presented methods are further discussed in the Evaluation Chapter (*cf.* Chapter 5).

3.2.3.1 Total Attack Percentage

An attack fingerprint indicates the absolute number of packets to a target and the detection threshold used to capture the fingerprint. The detection threshold aids in putting the number of packets into context, depending on the node capacity. Following the same idea, the percentage of the total attack is calculated to put a network node into a global attack context. This is achieved by using the attack fingerprint recorded at the target as a reference point and determining the proportion of the total attack contributed by each intermediate node. In this regard, the total attack percentage is a metric that can only be derived in a global analysis. However, it requires the availability of an attack fingerprint at the attack target.

When evaluating the contributions of intermediary nodes to a global attack, it is crucial to put the detection threshold into perspective. For example, a router from a local ISP might have a conspicuously high detection threshold of over 75%. However, if the total attack percentage reveals that this node accounts for less than 0.5% of the total attack, it becomes less interesting again. On the other hand, nodes with a low detection threshold might still be of interest if they account for a large percentage of the total attack. Examples include large DNS servers that can be exploited for a reflection attack.

3.2.3.2 Distance to Target

Assessing the distance of an intermediate node to the attack target is instrumental in building a global picture of the attack. Intuitively, nodes nearer to the attack target will likely observe a larger portion of the total attack. Combined with the computed attack percentage (*cf.* Chapter 3.2.3.1) and the detection threshold, this approach facilitates the identification of anomalous intermediate nodes that deviate from expected patterns.

Operating System	Default Initial TTL Value
Microsoft Windows	128
macOS	64
Linux	64
FreeBSD	64
Cisco IOS	255
OpenWRT	64
BusyBox	64

Table 3.7: Popular operating systems and their default initial TTL values [16, 53]

The TTL information in the attack fingerprints is leveraged for the distance calculation. One key advantage of the TTL field is that it cannot be forged along the path, as each (honest) network node decreases its value by one. However, it is important to note that an IP packet has no standardized initial value for the Time to Live field. Different operating systems employ various initial values (*cf.* Table 3.7). Theoretically, an attacker can set an arbitrary initial value for the TTL field of an IP packet. In practice, the randomization of TTL values during DDoS attacks has rarely been observed [45].

Concerning distance, two different measures are of interest: *(i)* The distance between an attacker and the target and *(ii)* the distance between an intermediate node and the target. While both measures are based on the TTL values of the IP packet, they use different methods for the calculation.

Distance of attacker
The distance of an attacker to the attack target is calculated based on the TTL values recorded at the target. For the calculation, the knowledge of the initial TTL value of an attack packet is required. Borrowing the idea employed in Hop-Count Filtering [65], the initial TTL value of the packet is derived by finding the smallest common initial TTL value that is higher than the TTL value recorded at the target. This is possible because commonly observed initial TTL values are relatively far apart, but paths between two internet hosts rarely exceed 30 hops in the distance [11, 64]. This thesis considers $32, 64, 128$, and 255 as common initial TTL values, and both the Fingerprint Generator and the Reassembler employ this list for generation and calculation.

For example, an attack target (blue) receives an attack packet from an unknown attacker (red) with a TTL value of 42 (*cf.* Figure 3.8). Along the path, the packet has also been observed by an intermediate node with a TTL value of 44. However, there is no further observation of the TTL value closer to the target. Based on the list of commonly observed initial TTL values, the smallest value higher than 42 is 64. In this regard, the distance of the attacker is calculated as $64 - 42 = 22$ hops.

Distance of an intermediate node
Compared to calculating the attacker's distance, determining the distance between intermediate nodes and the target does not require knowing the initial TTL value. In this

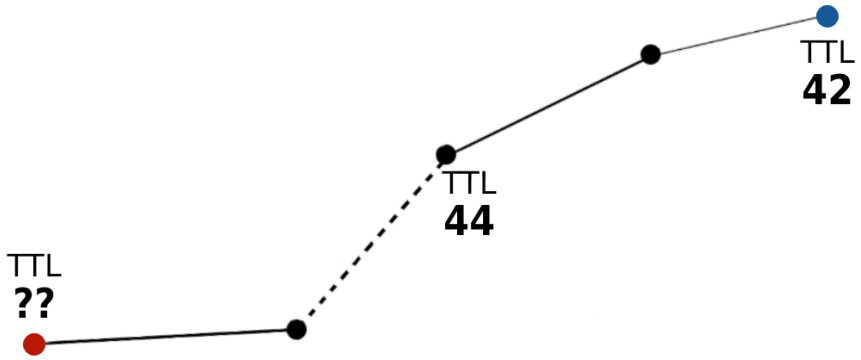


Figure 3.8: Calculating the distance in hops of an attacker

context, calculating the distance is more straightforward, as it simply involves finding the relative difference between observed TTL values for a packet sent to the same target. Following the example in Figure 3.8, the distance between the participating intermediate node and the attack target is trivially calculated ($44 - 42 = 2$).

While the calculation is trivial, some factors can make this calculation imprecise. A network packet is not always guaranteed to take the same route in the real world. This means the TTL values at intermediate nodes and the target may fluctuate slightly. For this thesis, that factor is neglected as the simulated network scenario always returns a stable shortest path between any two nodes. However, fluctuations in the observed TTL values from a single source address can also occur when an attack source employs spoofing mechanisms. While the detection of spoofed sources is discussed later (*cf.* Chapter 3.2.3.3), the main takeaway for the distance calculation is that a node might observe multiple TTL for the same address. This is the case when two nodes use the same spoofed IP.

An example to illustrate this behavior can be found in Figure 3.9. Two attacking nodes (red) employ IP spoofing mechanisms and happen to send one or more packets with the same IP. One of the attacking nodes uses 128 as the initial TTL value, and the other uses 64. As a consequence, the participating intermediate node B observes two different TTL values for packets from the same IP to the attack target (blue). Two different TTL values

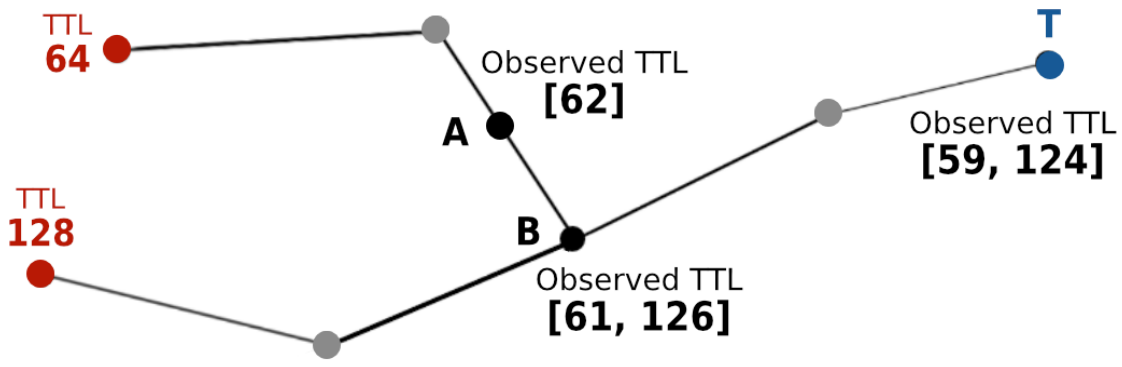


Figure 3.9: Calculating the distance in hops between intermediate nodes and target

for the same source IP are also observed at the target. Calculating the distance between node B and target T seems trivial at first by subtracting the mean TTL of T from the mean TTL of B. The calculation $\frac{61+126}{2} - \frac{59+124}{2} = 2$ yields the expected distance of 2, for node B and target T.

However, when calculating the distance between intermediate node A and target T, it can be observed that the distance calculation does not yield the expected result. The calculation $\frac{62}{1} - \frac{59+124}{2} = -29.5$ results in a negative value, which is not valid as a distance measure. The challenge lies in the fact that an intermediate node may observe a subset of the TTL values detected at the target, and with widely separated initial TTL values, calculating the mean does not yield valid outcomes.

The solution is explicitly selecting a value from the observed TTLs at the target when calculating the distance for one TTL value at an intermediate node. Following the idea of [65], the largest TTL value at the target that is smaller than the TTL value at the intermediate node is selected. For example, when calculating the distance between node A and target T, the TTL value 59 at the target is selected for comparison, as it is the largest value that is smaller than 62 (*i.e.,* the TTL at the intermediate node). The calculation $\frac{62}{1} - \frac{59}{1} = 3$ yields the correct results for the distance between node A and T.

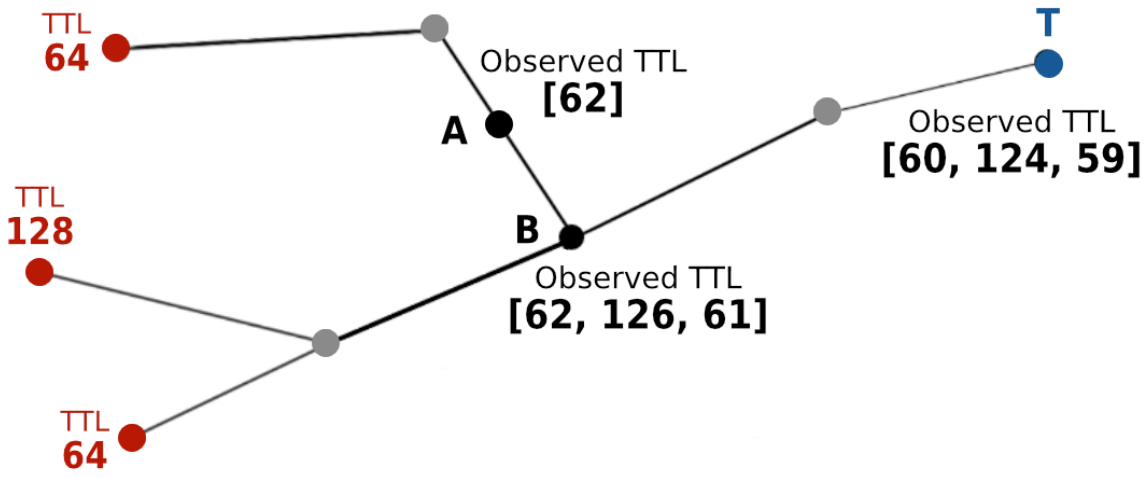


Figure 3.10: Calculating the distance in hops between intermediate nodes and target

Still, the proposed approach can yield inaccuracies in extreme cases with many spoofed addresses. Figure 3.10 shows an example with a third added attacking node that uses 64 as the initial TTL value. When calculating the distance from node A to target T again, the result is no longer accurate because the target has an additional TTL observation. The largest TTL value at T that is smaller than the TTL value at A is now 60, which results in a calculated distance of $\frac{62}{1} - \frac{59}{1} = 2$. Such inaccuracies cannot be omitted when only using aggregated fingerprint data for distance calculation. Nonetheless, they only occur with spoofed IPs. The impact of the percentage of spoofed IPs on the accuracy of distance calculation is further discussed in the Evaluation (*cf.* Chapter 5.2.1.1).

3.2.3.3 Detecting Spoofed Sources

Detecting spoofed addresses is crucial for analyzing the attack from a global perspective, as it helps understand the number of involved attack sources. Utilizing the TTL field of an IP packet for identifying spoofed addresses has proven effective, as shown by Wang *et al.* [65]. Their proposed solution consists of two phases: a learning phase and a filtering phase. The learning phase can be replaced with the fingerprint aggregation process with the extended fingerprint format (*cf.* Chapter 3.1.2.1). Rather than comparing TTL values with previously recorded values in a database, the detection can be performed solely by examining the fingerprint data. If a fingerprint contains more than one TTL value per source IP, the IP address has likely been spoofed.

The detection of spoofed IP addresses in [65] happens directly at the victim. In this regard, a global view of an attack from multiple attack fingerprints yields a new perspective on this detection problem. First, a confidence score can be built based on how many other nodes made the same observation along the path. Correlating this data with the distance from each node to the target, it is possible to derive at which distance from the target an IP address was first detected as spoofed.

Naturally, the precision of these detection techniques relies on the number of spoofed addresses and the frequency of IP collisions. In a simulated scenario, IP collisions seldom occur on a smaller scale than the global internet unless a predefined set of spoofed addresses is used. The influence of the percentage of spoofed addresses and the size of the spoofed IP pool are further discussed in the Evaluation (*cf.* Chapter 5.2).

3.2.3.4 Normalizing Intermediate Nodes

The attack identification step of the Reassembler produces a list of intermediate nodes. Those intermediate nodes are selected because they observed traffic to the attack target within a given time interval. However, at this stage, there is no distinction between attack and background traffic. For example, in the case of a TCP SYN attack, legit traffic includes regular website visitors who want to visit a website under attack.

Deciding whether a fingerprint represents attack or background traffic is not part of the loosened attack fingerprint definition. This is because a sophisticated attack detection mechanism is computationally impossible at each network node. At the same time, no fingerprint should be prematurely discarded because of a too-high detection threshold. In this context, it is the responsibility of the Reassembler to filter fingerprints of background traffic from fingerprints of attack traffic.

Filtering background traffic from attack traffic is only possible when it is not mixed within one fingerprint. The aggregation within the fingerprint makes a later distinction impossible, as only the number of packets and the TTL values are available per IP. Hence, only the fingerprints that exclusively observed background traffic can be discarded. This is achieved using a duration-based threshold, as background traffic is assumed to contact the attack target less often and with a shorter duration (*cf.* Table 4.3).

Normalizing the number of packets at the attack target based on the count of packets from discarded fingerprints is unfortunately not possible. The same background traffic packets could potentially be observed by multiple network nodes along the path to the attack target. Therefore, without knowing the exact topology of the network, the sum of discarded packets cannot be used to normalize the number of attack packets at the target. While the normalization of intermediate nodes cannot be used to refine the packet count at the attack target, it still aids in building a more accurate picture of an attack.

3.2.4 Global Fingerprint

The aggregation to build a global fingerprint is the fourth and last step in the Reassembler process. Based on the insights derived in the previous step, the goal is to provide a compact file format (*i.e.,* fingerprint) that includes all the interesting properties of the attack from a global view. As can be seen in Table 3.8, the global fingerprint groups the attributes into four different categories.

Attack properties include information about the attack itself. Attributes, such as *Attack Service* and *Attack Protocol*, are derived from the local fingerprint of the detected attack target. The start and end-time of the attack are based on a global view that only includes filtered intermediate nodes (*cf.* Chapter 3.2.3.4). By calculating the duration of the attack based on intermediate nodes, the attack duration can be derived even if the network link to the attack target is exhausted and the fingerprint is incomplete. Even though the attack attributes of the global fingerprint are similar to a local fingerprint, the increased robustness due to the extended network view is an advantage of the global fingerprint.

The *Target* properties consist of attributes that uniquely identify the attack target (IP Address) and configuration parameters of the attack target. In a real scenario, the *Detection Threshold* is a configuration parameter of a target node running the DDoS Dissector. For the simulated scenario, the detection threshold corresponds to a theoretical value representing the minimum required threshold needed to generate a local fingerprint at the target.

The *Intermediate Nodes* group accounts for the largest part of the global fingerprint. The *Number of Nodes* attribute describes the filtered amount of intermediate nodes that observed attack traffic. Intermediate nodes that observed genuine traffic to the target during the attack interval are filtered from the other statistics and summarized in the *Discarded Nodes* attribute. Additionally, the *Detection Threshold Percentiles* attribute provides information about the distribution of the detection threshold of the intermediate nodes. In the simulated scenario, this value is also theoretical, and it essentially signifies the minimum detection threshold configuration of intermediate nodes required to reach a certain percentage of global attack coverage. It has to be noted that in this context, full coverage refers to every participating intermediate node submitting a fingerprint. The influence of the scenario configuration on the detection threshold percentiles is further explored in the Evaluation (*cf.* Chapter 5).

The *Key Nodes* attribute is of special importance, as it contains a list of the most relevant nodes for the attack. Each key node contains the attributes listed in Table 3.9. The *Hops*

Category	Attribute	Data Type
Attack	Start Time	Timestamp
	End Time	Timestamp
	Duration (s)	Float
	Attack Service	String
	Attack Protocol	String
Target	IP Address	String
	Detection Threshold	Float
Intermediate Nodes	Nr. Nodes	Integer
	Discarded Nodes	Integer
	Detection Threshold Percentiles	Map<Integer, Float>
	Key Nodes	Object (*cf.* Table 3.9)
Sources	Nr. Sources	Integer
	Pct. Spoofed	Float

Table 3.8: Global Fingerprint format

to Target value is calculated based on the analysis in Chapter 3.2.3.2. The *Number of packets*, *Detection Threshold*, and the duration are based on the local fingerprint of the intermediate node. In contrast, the *Fraction of the Total Attack* is based on the global view (*cf.* Chapter 3.2.3.1). For evaluation (only in simulated scenarios), a *Distance* attribute also exists, which represents the ground truth data for the *Hops to Target* attribute. The error in distance estimation is reported in the *Inferred Distance Difference* attribute.

The attributes in the *Sources* group characterize the attack's origin. The *Number of Sources* attribute is based on the number of observed IPs to the attack target during the interval. Besides removing duplicates, there is no normalization for the number of sources. However, using the spoofed addresses detection mechanism (*cf.* Chapter 3.2.3.3), the *Percentage of Spoofed Addresses* is estimated. This enables assessing whether the reported number of sources is likely too high or accurate.

The four phases of the Reassembler module produce a compact format of a global attack view of a DDoS attack. This is achieved by pre-processing a set of local attack fingerprints and identifying the attack on a high level. The global attack view is refined using different analysis approaches in the next step. The last step produces a global fingerprint representing an entire globally observed DDoS attack. A global fingerprint allows the efficient sharing and comparison of DDoS attack fingerprints and enables further use cases, such as clustering similar attack types. The possibilities for further enhancements are explored in Chapter 5.3 and Chapter 6.3.

Attribute	Data Type	Scopes
Number of Packets	Float	Simulated, Dissector
Hops to Target	Float	Simulated, Dissector
Detection Threshold	Float	Simulated, Dissector
Start Time	String	Simulated, Dissector
End Time	String	Simulated, Dissector
Distance	Integer	Simulated
Inferred Distance Difference	Float	Simulated
Fraction of Total Attack	Float	Simulated, Dissector
Duration in Seconds	Float	Simulated, Dissector

Table 3.9: Key node attributes in Global Fingerprint

Chapter 4

Implementation

This Chapter presents an implementation of the previously discussed components. Chapter 4.1 shows the generation of fingerprints based on a dynamic attack scenario. Thereafter, Chapter 4.2 discusses the implementation of the Reassembler and shows how the generated fingerprints are aggregated to a global view of a DDoS attack. Lastly, Chapter 4.3 presents the contributions to the DDoS Dissector proposed to enable a full workflow for a global fingerprint-based DDoS attack analysis. The complete source code for the Generator, Reassembler and DDoS Dissector can be found on GitHub [7, 8].

4.1 Fingerprint Generator

The Fingerprint Generator is implemented in Python and consists of two main components: A network graph that simulates a network topology as described in Chapter 3.1.1, and a generator method that produces fingerprints based on the underlying network graph. Both components are discussed in the following sections.

4.1.1 Network Graph

To model the network, a graph data structure is used. This is possible because routing decisions are made from the outside in the simulated scenario. In contrast to the global internet infrastructure, there is no routing abstraction needed in a way that each node forwards packets to the next best node of his knowledge. Thus, a node does not need an internal routing logic but merely serves as an additional hop on a path between two network nodes. In that regard, the network simulation is implemented using the `Graph` class in Python from the *networkx* package.

The network is built bottom-up based on the provided subnet configuration (*cf.* Table 3.5). First, each hierarchical subnet is generated on its own. Thereafter, all subnets are connected using a partial mesh backbone architecture. This procedure is further elaborated in the next sections.

4.1.1.1 Hierarchical Subnets

Hierarchical subnets are randomly generated based on the configuration parameters of the attack scenario. As shown in Listing 4.1, a hierarchical subnet is created recursively starting from a `root` node. The `root` node is an instance of the `IPNetwork` class from the *netaddr* package, and it consists of an IP address and a network mask. The network mask is increased by the `prefixlen` parameter for each added level in the tree. This is achieved by calling the `create_subnet_nodes` method recursively (*cf.* Listing 4.1, Line 31). The `create_subnet_nodes` method increases the bits of the network mask on each level by the amount defined in the `prefixlen` parameter (*cf.* Line 21). The number of nodes per level is defined by the `max_clients` parameter and is randomly selected on each level. This makes the topology of the subnets more heterogenous and, thus, more realistic. The base case of the recursive `create_subnet_nodes` method (*cf.* Line 6ff.) samples a random number of clients from all possible IPs of the parent subnet using the `iter_hosts` method. The clients are then added using the `add_node` method and connected to their parent using the `add_edge` method. Furthermore, only for the clients (*i.e.,* base case), the `spoofed` attribute is set based on the probability in `spoofed_pct` (*cf.* Line 12).

```
1  def create_subnet(root, levels, prefixlen, max_clients, spoofed_pct):
2      g = nx.Graph()
3      g.add_node(root.ip, ip=root.ip, level=1, client=False,spoofed=False)
4
5      def create_subnet_nodes(parent, level):
6          # base case
7          if level == levels:
8              ips = sample(list(parent.iter_hosts()),
9                           randint(1, max_clients))
10             for p in ips:
11                 g.add_node(p, ip=p, level=level + 1, client=True,
12                            spoofed=random() <= spoofed_pct,
13                            spoofed_ips=sample(SPOOFED_IP_POOL,
14                                               randint(7, 25)))
15                 g.add_edge(parent.ip, p, color=color, level=level + 1,
16                            ms=random.uniform(1, 100))
17             return
18
19         nr_nodes = randint(1, max_clients)
20         i = 0
21         for s in parent.subnet(parent.prefixlen + prefixlen):
22             i += 1
23             if i > nr_nodes:
24                 break
25             s = s.next()
26             g.add_node(s.ip, ip=s.ip, level=level + 1,
27                        client=False, spoofed=False)
28             g.add_edge(parent.ip, s.ip, color=color, level=level + 1,
29                        ms=random.uniform(1, 100))
30             # recursive call
31             create_subnet_nodes(s, level + 1)
32
33     create_subnet_nodes(root, 0)
34     return graph
```

Listing 4.1: Recursive structure of the create_hierarchical_subnet method

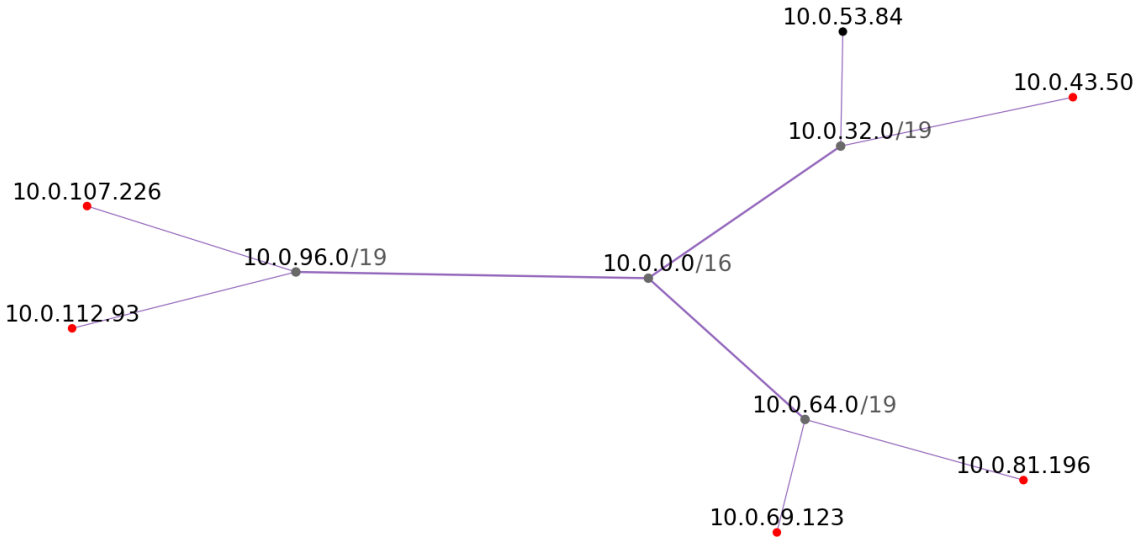


Figure 4.1: Single hierarchical subnet

An example of a resulting subnet can be found in Figure 4.1, where the root node uses a 16-bit network mask, and nodes on the first level utilize a 19-bit network mask. First-level nodes have three child nodes in their subnet. On the second level, the subnet consists of two client nodes each. The color of the client nodes signifies whether a node communicates with a masked IP or not. Red nodes use a spoofed IP; black ones use their real IP (*cf.* Figure 4.1). This decision is taken for each node during the subnet generation.

Nodes and edges have different attributes that are leveraged to route and generate fingerprints at a later stage. The `ip` attribute of a node is used as ground truth data and for routing (*cf.* Table 4.1). `Level` and `client` attributes are used to efficiently access certain nodes of the network (*e.g.,* all leaves or all backbone routers). Most important for fingerprint generation are the `spoofed` and `spoofed_ips` attributes. While each node is assigned a random list of spoofed IP addresses, the `spoofed` flag decides whether a node communicates with a spoofed IP.

Compared to nodes, edges require fewer attributes (*cf.* Table 4.2). The `ms` attribute defines the duration for a packet to traverse a certain edge. Again, for efficient access, edges include a `level` attribute that reflects the parent's level in the subnet tree.

4.1.1.2 Backbone Connection

After the generation of one or multiple subnets, the subnets are connected to form a global network. As discussed in Chapter 3.1, a partial mesh is used as a backbone topology. The partial mesh is built by first connecting every subnet in a ring topology. This ensures that all subnets are connected and a valid path between every node in the network exists. Afterward, additional randomized edges (*i.e.,* shortcuts) are added to form a partial mesh topology. It has to be noted that for a small number of subnets (*i.e.,* $<= 3$), the ring topology also equals a full mesh topology. Increasing the number of subnets makes the

Attribute	Data Type	Description
ip	IPAddress	The real IP address of the node
level	Integer	The level of the subnet tree
client	Boolean	Flag that indicates if a node is a leaf node of its subnet tree
spoofed	Boolean	Flag that indicates if a node communicates with a spoofed IP
spoofed_ips	List<IPAddress>	A list of spoofed IP addresses randomly drawn from a pool of available IPs

Table 4.1: Node attributes

Attribute	Type	Description
ms	Integer	The delay that occurs when a packet is routed through this edge
level	Integer	The level of the parent node in the subnet tree

Table 4.2: Edge Attributes

partial mesh more sparse since the number of randomly added edges grows linear $(n/2)$, but the number of all possible connections grows quadratically. This is intended and reflects the real world, where a single AS is only connected to a small fraction of all other ASNs.

```
def create_network(subnets, max_levels, max_clients, spoofed_pct):
    subgraphs = [create_subnet(s, levels=randint(1, max_levels),
                               max_clients=randint(2, max_clients),
                               spoofed_pct=spoofed_pct, prefixlen=4)
                 for i, s in enumerate(subnets)]

    G = nx.compose_all(subgraphs)
    n = len(subnets)

    # Ring topology
    edges = [(subnets[i].ip, subnets[(i + 1) % n].ip,
              {'level': 0, 'ms': random.uniform(1, 100)})
             for i in range(n)]

    # Add random connections to create a partial mesh
    m = round(n / 2)
    mesh = [(u.ip, v.ip, {'level': 0, 'ms': uniform(1, 100)})
            for u, v in sample(list(combinations(subnets, 2)), m)]

    edges.extend(mesh)
    G.add_edges_from(edges)
    return G
```

Listing 4.2: Connecting all subnets

The `create_network` method (*cf.* Listing 4.2) combines the subnet and backbone generation using the above-presented strategy. The returned network graph forms the basis

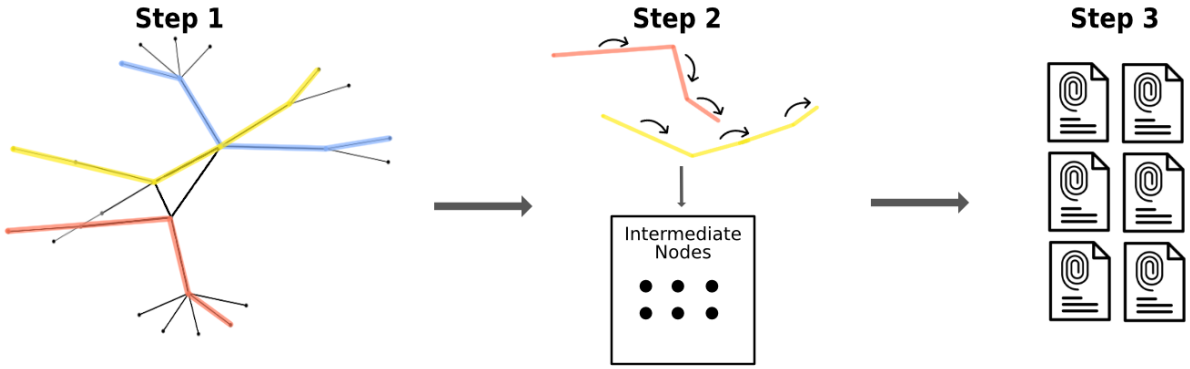


Figure 4.2: Three-step generation process

for the generation of fingerprints, as discussed in the next Chapter.

4.1.2 Generator

The generation of fingerprints based on a network graph is a three-step process (*cf.* Figure 4.2).

1. Calculate paths for attack and background traffic.
2. Walk all calculated paths and aggregate the traffic along the path for each intermediate node.
3. Iterate over all intermediate nodes and generate a fingerprint per observed target.

While this approach still requires iterating over intermediate nodes for each traffic source, it is much more efficient than simulating the traffic on a packet level. No matter how many packets an attack source sends to a target, the path must only be traversed once.

4.1.2.1 Traffic Paths

The first step of the fingerprint generation process includes computing attack paths and selecting background traffic paths. As previously discussed, routing decisions are determined using an outside perspective, applying Dijkstra's shortest path algorithm [19] on the network graph. The `shortest_path` method of the *networkx* library with weighted edges is used for the implementation. The weight of the edge is defined by the `ms` attribute (*cf.* Table 4.2). Overall, this results in one attack path per attack source which is minimal in terms of time delay between the source and attack target.

Background traffic is simulated by randomly selecting pairs of nodes in the network according to predefined rules. Concerning the traffic target, two cases exist: *(i)* Background traffic to the attack target and *(ii)* background traffic to an unrelated node. The amount of unique background traffic paths can be dynamically adjusted along with the fraction

directed to the attack target (*cf.* Listing 4.3). Because some nodes are marked as spoofed at the initialization of the network graph, cases *(i)* and *(ii)* fulfill all three types of background traffic discussed in Chapter 3.1.3.3.

```
def generate_background_traffic(G, amount, target, targeted_pct=0.2):
    unrelated = [(x, y) for x, y in combinations(G.nodes, 2)
                 if x != y and y != target]
    unrelated_sample = sample(unrelated, int(amount * (1-targeted_pct)))

    targeted = [(n, target) for n, data in G.nodes(data=True)
                if not data.get('spoofed', False)]
    targeted_sample = sample(targeted, int(amount * targeted_pct))

    return [(src, tgt, shortest_path(G, src, tgt, weight='ms'), False)
            for src, tgt in unrelated_sample + targeted_sample]
```

Listing 4.3: Generating background traffic paths

Attack and background paths are stored in a tuple of the form `(source, target, path, is_attack)`. The boolean flag `is_attack` differentiates background traffic from attack traffic paths. This allows the aggregation of intermediate node data using a combined list of attack traffic.

4.1.2.2 Aggregating Data per Intermediate Node

The data aggregation per intermediate node is achieved by iterating over all attack and background paths. However, before data can be aggregated, each path needs properties that define the characteristics of the traffic. The properties include TTL, start time, duration, the number of packets, and the number of megabytes of traffic. Since the background traffic is randomly generated, a single source can have traffic paths to multiple other nodes (attack target or unrelated). For this reason, attack characteristics are defined on a path level, not on the source level. This means that a source node A can generate background traffic to an unrelated node B, but also to the attack target X at a later start time.

The selection of attack parameters differs between attack traffic and background traffic. However, for the duration and the number of packets, the attack and background ranges overlap so that it is not trivial to distinguish attack traffic from background traffic. The traffic characteristics for both types of traffic are summarized in Table 4.3. The start time and the nr. of Megabytes have the same range for attack and background traffic. However, the nr. of Megabytes depends on the nr. of packets, whereas the latter is different for attack and background traffic.

When iterating over attack paths with the above-mentioned traffic characteristic, the aggregated data is stored in a nested dictionary using the attributes in Table 4.4. The dictionary captures all traffic information per intermediate node and target. Since an intermediate node can be part of many traffic paths (attack and background), certain nodes and target combinations are visited multiple times. This is also why all attributes are of type list or dictionary. While walking the traffic paths, intermediate values are appended to the list or dictionary such that no fingerprint-relevant data is lost. For

Attribute	Attack Range	Background Range
Start Time	[0s, 10s]	
Duration	[60s, 600s]	[10s, 70s]
Nr. of Packets	$[10^3, 10^6]$	$[10^2, 10^5]$
Nr. of Megabytes	*Nr. packets* $* [\frac{1}{1000}, \frac{1}{5'000}]$	

Table 4.3: Simulated traffic characteristics

example, the start time for each observed source at an intermediate node is stored in a list. In this way, it is possible to extract different information at a later stage (*e.g.,* earliest attack, bursts, etc.).

Special handling is required when the attack source spoofs its IP address. Since one node can send traffic using multiple spoofed addresses, `nr_packets_by_source` and `nr_megabytes` must be normalized. For example, if a node sends 1'000 packets and uses a pool of ten spoofed IPs, 100 packets are counted towards each spoofed source IP address. Furthermore, a mapping between spoofed IPs and the real IP address is stored for traceability. In the real world, this information is not available. However, for the simulated scenario, it is used as ground truth data for the evaluation at a later stage.

TTL values and timestamps must be calculated along the different traffic paths for the data to be realistic. Starting from a random common TTL value (*i.e.,* 64, 128, 255), the TTL value is decreased on each step while iterating over a path. Furthermore, the timestamps are shifted back based on the weight (*i.e.,* delay) of the edge. The delays are accumulated along the path in a way that the timestamps of the intermediate nodes reflect real routing behavior.

4.1.2.3 Fingerprint Generation

The generation of fingerprints is based on the dictionary of intermediate nodes. By definition of [31], an attack fingerprint only includes traffic to a single target. In that regard, for each node in the intermediate nodes dictionary, a fingerprint is generated for every observed traffic destination. Naturally, this results in a large number of fingerprints. While some of the fingerprints can be discarded at generation (*e.g.,* fingerprints at an attack source), the filtering of relevant fingerprints is left to the Reassembler module (*cf.* Chapter 3.2).

A reference fingerprint format of a real SYN flood attack is shown in Listing 3.1. Some values are trivial to derive with the aggregated node data from the simulated scenario. For example, the `time_start` corresponds to the earliest observed timestamp per node and target. On the other hand, some values require additional calculation. For example, the observed `ttl` values have to be normalized to a probability value between 0 and 1. Furthermore, the `detection_threshold` value requires comparing all other observed traffic at the node. It is calculated by dividing the number of packets to the target by the number of observed packets at the node.

Attribute	Data Type	Description
node (key)	IPAddress	Intermediate Node where traffic is observed
target (key)	IPAddress	Target to which traffic is directed
ttl	Dict[int, int]	Mapping of TTL value to number of packets
ttl_by_source	Dict[IPAddress, List[int]]	Mapping IP address to one (or multiple) observed TTL values
sources	List[IPAddress]	List of observed traffic sources
sources_real	Dict[IPAddress, IPAddress]	Mapping of spoofed IP to real IP (ground truth data)
time_start	List[int]	List of start times for observed traffic
duration_seconds	List[int]	List of duration for observed traffic
nr_packets	List[int]	List of observed number of packets for each traffic source
nr_packets_by_source	Dict[IPAddress, int]	Mapping of source address to observed number of packets
nr_megabytes	List[int]	List of observed number of megabytes for each traffic source

Table 4.4: Data type for intermediate node aggregation

A unique key for each fingerprint is generated before the list of generated fingerprints can be stored in individual JSON files. This is achieved by creating an md5 hash of the fingerprint content. The resulting key serves as a unique identifier to the fingerprint as it is proposed in the format of [31]. Finally, the fingerprints are stored in a predefined output folder using their calculated key as the filename. Such a folder simulates the sharing of attack fingerprints in a central repository as it is done, for example, in the DDoSDB [18].

4.2 Reassembler

Based on the proposed design in Chapter 3.2, this Chapter outlines the implementation of the Reassembler module. Similar to the Fingerprint Generator, Python is used as a programming language for the Reassembler. Python provides the advantage of easing the work with JSON data (*i.e.,* Fingerprints), and popular libraries can be used for data analysis. Examples include the data analysis and manipulation tool Pandas [39] or the NumPy [28] package for scientific computing.

The structure of this Chapter follows the proposed design (*cf.* Chapter 3.2). First, the reading and pre-processing of fingerprints is discussed in Chapter 4.2.1. Thereafter, the

steps for attack detection and analysis are introduced together in Chapter 4.2.2. Lastly, the file generation of global fingerprints is presented (*cf.* Chapter 4.2.3).

4.2.1 Pre-process Fingerprints

The first step of the pre-processing includes reading the fingerprints. Fingerprints can be provided via JSON files in a folder or directly in memory by passing the data in a Python dictionary, with the latter option mainly used for evaluation (*cf.* Chapter 5).

The `read_fingerprints_from_folder` method in Listing 4.4 shows how the JSON files are programmatically opened and read from a folder. Per default, Python reads JSON data into a regular dictionary (*cf.* Line 9). However, having a list of thousands of dictionaries (*i.e.,* fingerprints) is not optimal for data analysis as it is impossible to formulate complex queries or computations easily. For that reason, the attack fingerprints are loaded into a Pandas [39] DataFrame. Through integrated functionalities, Pandas DataFrames allow the efficient manipulation and analysis of large datasets.

```
1  def read_fingerprints_from_folder(path, simulated):
2      # create an empty list to store the DataFrames
3      dfs = []
4      # loop over each JSON file in the folder
5      for filename in os.listdir(path):
6          if filename.endswith('.json'):
7              with open(os.path.join(path, filename)) as f:
8                  # load the JSON data into a Python dictionary
9                  data = json.load(f)
10                 flat_data = flatten_fingerprint(data, simulated=
       simulated)
11                 df = pd.json_normalize(flat_data, 'attack_vectors')
12                 dfs.append(df)
13     return pd.concat(dfs, ignore_index=True)
```

Listing 4.4: Read fingerprints to Pandas DataFrame

DataFrames work best with flat tabular data. As can be seen in the reference fingerprint (*cf.* Listing 3.1), the format contains many nested structures. While Pandas can normalize nested structures to a certain extent, the `json_normalize` function does not work well with dynamic keys in nested structures. For example, the observed percentage of the TTL value `243` (*cf.* Listing 3.1) would be normalized to a column name of `attack_vectors.ttl.243`. Having a dynamic key in the column name does not work for efficient analysis, as column names have endless possibilities.

A custom method is used to flatten the attack fingerprints to solve the problem of dynamic keys in nested structures. The `flatten_fingerprint` method (*cf.* Listing 4.5) takes a fingerprint dictionary as input and returns the flattened data as Pandas DataFrame. Instead of storing an entire attack fingerprint in each row of a DataFrame, the `flatten_fingerprint` converts the data so that each row contains one attack vector from a single source IP. Naturally, copying the attributes located outside the `attack_vectors` in the fingerprint to each row in the DataFrame (*cf.* Line 13) is required. This generates an

overhead in terms of space, but it allows the efficient querying of fingerprint data using built-in Pandas functionality.

Flattening attack fingerprints by attack vector and source IP removes the need for dynamic keys. This is because the dynamic key is usually the source IP itself (*cf.* Chapter 3.1.2.1). For example, the mapping `nr_of_packets_by_source` (*cf.* Table 4.4) can be replaced with a simple integer value per attack vector and source IP. The total `nr_packets` can be calculated again using built-in Pandas functionality.

```
1  def flatten_fingerprint(data, simulated):
2      flat_avs = []
3
4      for av in data['attack_vectors']:
5          for src_ip in av['source_ips']:
6              flat_av = {
7                  'key': data['key'],
8                  'source_ip': src_ip,
9                  'source_ip_real': av['source_ips_real'][src_ip],
10                 'ttl': av['ttl_by_source'][src_ip],
11                 'nr_packets': av['nr_packets_by_source'][src_ip],
12                 **{k: attack_vector[k] for k in ['service', 'protocol',
13                     'duration_seconds', 'time_start',
14                     'detection_threshold']},
15                 **{k: data[k] for k in ['target', 'location']}
16             }
17             if simulated:
18                 flat_av['distance'] = data['distance']
19                 flat_av['is_attack'] = av['is_attack']
20                 flat_av['source_ip_real'] =av['source_ips_real'][src_ip]
21             flat_avs.append(flat_av)
22
23     data['attack_vectors'] = flat_avs
24     return data
```

Listing 4.5: Flatten fingerprints before analysis

The `simulated` parameter of the `flatten_fingerprint` method is used to set additional attributes that are only available in a simulated environment. When the parameter is set to *true*, ground truth data, such as `distance`, `is_attack`, and `source_ip_real` are added to each flattened attack vector (*cf.* Line 18ff.). For non-simulated attack fingerprints from the DDoS Dissector, the `simulated` parameter must be set to *false*.

After flattening all fingerprints, the list of flat attack vectors is merged to a single DataFrame using Pandas' `concat` method (*cf.* Listing 4.4, Line 13). This DataFrame represents an unanalyzed global view of an attack consisting of many vectors from a source IP to a destination IP. Besides the `key` property, there is no separation of individual attack fingerprints, but all the available information can be queried using a single DataFrame. Of course, the data of a single fingerprint can be retrieved by grouping the data by `key`. The analysis performed on the global DataFrame is discussed in the next Section.

4.2.2 Attack Analysis

The attack analysis is at the heart of the Reassembler module. In the first step, the attack target is detected, and all possible intermediate nodes that observed the attack are selected. After that, multiple analyses to refine the global attack view are made.

The Reassembler module is built using an object-oriented approach. Listing 4.6 shows the constructor of the Reassembler class. Upon initialization of a Reassembler instance, the previously discussed `read_fingerprints` (*cf.* Listing 4.4) method is leveraged to read a folder of fingerprints to a flattened DataFrame. After the conversion of timestamps, the `find_target` method is called to infer the attack target. This is further discussed in the next section.

```
1  class Reassembler:
2      def __init__(self, fingerprint_folder=None, fingerprint_data=None,
       simulated=True):
3          if len(fingerprint_data) > 0:
4              self.fps = pd.concat(
5                  [pd.json_normalize(
6                      flatten_fingerprint(x, simulated=simulated),
7                      'attack_vectors')
8                  for x in fingerprint_data], ignore_index=True)
9          elif fingerprint_folder is not None:
10             self.fps = read_fingerprints_from_folder(fingerprint_folder,
       simulated=simulated)
11         else:
12             raise ValueError("No Fingerprint Data provided")
13
14         self.fps['time_start'] = pd.to_datetime(self.fps['time_start'])
15         self.fps['time_end'] = (self.fps['time_start'] + pd.to_timedelta
       (self.fps['duration_seconds'], unit='s'))
16         self.target = self.find_target
17         self.simulated = simulated
18         ...
```

Listing 4.6: Reassembler Class

4.2.2.1 Infer Attack

As outlined in Chapter 3.2.2.1, the attack target is selected by finding the most targeted network node from all globally submitted attack fingerprints. Similar to the detection mechanism in the DDoS simulator, a minimum detection threshold of 50% is assumed. Since the attack fingerprints are stored in a flat DataFrame, the attack vectors are grouped by location to find the target (*cf.* Listing 4.7, Line 5).

If no target is found, for example, because the detection threshold is set too high, an error is thrown. It can be solved by lowering the detection threshold or manually overriding the attack target. More details on the usage of the Reassembler can be found in the Use Case Evaluation (*cf.* Chapter 5.1).

```
def find_target(self) -> str:
    possible_targets = self.fps[
        (self.fps['target'] == self.fps['location']) &
        (self.fps['detection_threshold'] >= 0.5)]
        .groupby(['location'])
        .agg(
            {'nr_packets': 'sum', 'key': 'min', 'detection_threshold': '
    min'})
        .sort_values('nr_packets', ascending=False)

    if len(possible_targets) == 0:
        raise ValueError("No Target found with the desired attack
    threshold")

    # returns IP of most targeted location
    return possible_targets.index[0]
```

Listing 4.7: Find most targeted location

4.2.2.2 Analyze Intermediate Nodes

A crucial part of the analysis happens in the `reassemble` method that can be called on any `Reassembler` instance. For the analysis of the intermediate nodes, the attack traffic observed at the target forms the baseline. Hence, the first part of the `reassemble` method consists of selecting all traffic to the attack target (*cf.* Listing 4.8). This is accomplished with Pandas by selecting all fingerprints that have the same location and target as the identified attack target (*cf.* Lines 2f.). The resulting `entries_at_target` DataFrame allows aggregating values such as the total number of packets at the target (*cf.* Line 6).

```
entries_at_target = self.fps[
                    (self.fps['location'] == target)
                    & (self.fps['target'] == target)].copy()
entries_at_target['ttl_count'] = (entries_at_target['ttl']
                                  .apply(lambda x: len(x)))
total_nr_packets_at_target = entries_at_target['nr_packets'].sum()
```

Listing 4.8: Reassemble method: Select entries at target

The next step is the selection of all fingerprints that observed the attack from a location other than the attack target (*cf.* Listing 4.9). The `observing_fp` DataFrame is afterward merged with the baseline TTL values, such that the hops to the target can be calculated (*cf.* Lines 4ff.).

```
observing_fp = self.fps[(self.fps['target'] == target) &
                        (self.fps['location'] != target)].copy()
# Merge with the baseline
observing_fp = observing_fp.merge(ttls_at_target,
                                  how='left', on='source_ip')
observing_fp['hops_to_target'] = observing_fp.apply(
    calculate_hops_to_target, axis=1)
```

Listing 4.9: Reassemble method: Find intermediate nodes and merge with baseline

The calculation of hops to the target follows the approach presented in Chapter 3.2.3.2. It works by comparing each observed TTL value with the highest TTL value at the target that remains below the observed TTL value (*cf.* Listing 4.10). The distance is then calculated using the mean over all TTL values recorded for a specific source IP and target IP combination. It has to be noted that multiple TTLs per source IP only occur when multiple nodes use the same spoofed IP but are at varying distances from the attack target. The impact of this is further explored in the Evaluation (*cf.* Chapter 5.2.1.1).

```
def calculate_hops_to_target(row):
    ttl = row['ttl']
    ttl_on_target = row['ttl_on_target']

    distances = []

    for t in ttl:
        # Get the largest ttl_on_target value that is smaller than t
        valid_targets = [x for x in ttl_on_target if x <= t]
        target = max(valid_targets)
        distance = t - target
        distances.append(distance)

    # Calculate the mean distance
    mean_distance = sum(distances) / len(distances)
    return mean_distance
```

Listing 4.10: Method for calculating hops to target

From the `observing_fp` DataFrame with calculated distances to the target, the aggregation into intermediate nodes happens (*cf.* Listing 4.11). For this, all observing entries are grouped by their location (*cf.* Line 6). Depending on whether the scenario is simulated or not, ground truth data about the actual distance to the target can be added to the target (*cf.* Lines 3f.).

```
agg_config = {'nr_packets': 'sum', 'hops_to_target': 'mean', '
    detection_threshold': 'min', 'time_start': 'min',
                        'time_end': 'max'}
if self.simulated:
    agg_config['distance'] = 'mean'

intermediate_nodes = observing_fp.groupby('location').agg(agg_config).
    copy()
```

Listing 4.11: Aggregating intermediate nodes

Together, the `entries_at_target` and `intermediate_nodes` DataFrames form the basis for the generation of the global fingerprint. The implementation is discussed in the following Chapter.

4.2.3 Global Fingerprint Generation

A global fingerprint represents a compact form of a globally observed DDoS attack. Following the proposed (local) attack fingerprint format of the DDoS Dissector [31], the

global fingerprint is stored as a JSON file. For the later identification of globally observed DDoS attacks, the fingerprint contents are md5 hashed and stored in an additional `key` property inside the global fingerprint. In this regard, the form of the global fingerprint is rather similar to a local fingerprint. This is intended to facilitate integrations with other ecosystem services (*e.g.,* DDoSDB) and their existing technologies.

Listing 4.12 shows how the global fingerprint is created from the aggregated DataFrames (*cf.* Chapter 4.2.2.2). The values are stored in the `summary` dictionary and can be saved to a JSON format using the `save_to_json` method provided by the `Reassembler` class. The generated fingerprint follows the format in Table 3.8. It can be seen that properties related to the attack and target category are derived from the `entries_at_target` DataFrame. Properties of the intermediate nodes category are derived from the aggregated `intermediate_nodes` DataFrame (*cf.* Chapter 4.2.2.2).

```
summary = {
    'attack': {
        'start_time': entries_at_target['time_start'].min().isoformat(),
        'end_time': entries_at_target['time_end'].max().isoformat(),
        'duration_seconds': entries_at_target['duration_seconds'].mean()
        ...
    },
    'target': {
        'ip': target,
        'detection_threshold': (entries_at_target['detection_threshold']
                                .mean())
    },
    'intermediate_nodes': {
        'discarded_intermediate_nodes': (len(intermediate_nodes)
                                - len(filtered_intermediate_nodes)),
        'nr_intermediate_nodes': len(filtered_intermediate_nodes),
        'detection_threshold': {p: v for p,v in zip(DEFAULT_PERCENTILES,
                                threshold_percentiles)},
        'key_nodes': (filtered_intermediate_nodes
                    .sort_values('nr_packets', ascending=False)
                    .sort_values('hops_to_target')
                    .to_dict('index'))
    },
    'sources': {
        'nr_sources': len(entries_at_target),
        'pct_spoofed': pct_spoofed
    }
}
```

Listing 4.12: Assembling a global fingerprint from aggregated DataFrames

A complete global fingerprint of a simulated attack scenario can be found in Appendix C.1. It includes ground truth data such as the list of actual attack sources and the actual distance of each intermediate to the target. Contrarily, the global fingerprint in Appendix C.2 is based on real attack traces and does not include ground truth data. The usage of the `add_ground_truth_data` method is further shown in Chapter 5.1.2.2.

4.3 DDoS Dissector Contributions

The DDoS Dissector [31] software, along with the proposed attack fingerprint format, forms the base for this thesis. However, not all proposed changes in this thesis are compatible with the current implementation of the DDoS Dissector. This Chapter outlines the contributions to the DDoS Dissector proposed in this work. While some extend the functionality, others aim to fix Bugs encountered when using the Software.

4.3.1 Generating Multiple Fingerprints

As previously discussed, the detection mechanism of the DDoS Dissector is based on the percentage of observed packets per target IP address. The high detection threshold of 50% does not work well with network nodes observing a DDoS attack from farther away. Usually, a few hops away from the target nodes observe the attack with a threshold of less than 50% (*cf.* Chapter 5.2.2). A sophisticated attack detection on the DDoS Dissector level that detects attacks with low thresholds exceeds the scope of this work. In some cases, a clear detection from farther away may not be possible only by looking at the observed traffic. For that reason, a simpler detection is proposed by generating a fingerprint for the top n most targeted IPs.

For this, a new `infer_target_by_index` method is created, which selects the n-th most targeted IP. Instead of only processing one single target, the main process of the DDoS Dissector is wrapped in a loop that generates the configured amount of fingerprints.

```
def infer_target_by_index(attack: Attack, index: int) -> IPNetwork:
    LOGGER.debug("Inferring attack target by index %d" % index)
    packets_per_ip = (attack.data
                      .groupby('destination_address')['nr_packets']
                      .sum()
                      .sort_values(ascending=False))
    most_traffic_address, nr_packets = list(packets_per_ip.items())[
    index]
    return IPNetwork(most_traffic_address)
```

Listing 4.13: Infer attack target by index

This functionality is only invoked, for compatibility reasons, when the corresponding arguments are passed. Two arguments are added for this reason:

- **-t, --threshold**
  The threshold argument defines the minimum detection threshold for an IP to be considered an attack target. If the argument is not set, the default value of 0.5 (*i.e.*, 50% of packets in the PCAP) from the current DDoS Dissector implementation remains.

- **-n, --nfingerprints**
  The number of fingerprints argument defines how many fingerprints should be generated. If this argument is set, the detection threshold is overwritten, and fingerprints for the n most targeted IP addresses are generated.

The suggested modifications facilitate the usage of the DDoS Dissector for a global attack view rather than a localized attack detection. However, for full compatibility with the Reassembler, additional changes to the fingerprint format are required.

4.3.2 Extending the Fingerprint Format

The extended fingerprint format (*cf.* Table 3.1.2.1) enables global analysis of attack fingerprints without loss of context along the path (*cf.* Figure 3.6). The additional properties in Table 3.4 are added to the DDoS Dissector to achieve this.

The values are either derived from the observed network data directly or from the configuration attributes when the Dissector is started.

- **location**
  The location in the fingerprint represents the IP where the fingerprint is collected. Since the recording location cannot trivially be derived from a PCAP file, it is implemented using an additional configuration attribute `-l, --location`. This further allows the later calculation and submission of fingerprints at a location independent from where the PCAP was initially recorded.

- **nr_packets_by_source & ttl_by_source**
  Packets and TTL by source are derived from the underlying PCAP data. No configuration attribute is needed to extend the fingerprint with this information.

- **detection_threshold**
  The detection threshold is derived from different values based on the given configuration of the DDoS Dissector. If the argument `-n` is set, the detection threshold represents the percentage of packets to the fingerprint target in comparison with all observed traffic at the node. If the `-t` argument is set, the detection threshold represents the configuration of the DDoS Dissector. If none of the arguments is set, the detection threshold is set to the default value of 0.5.

As can be seen, most of the proposed changes to the DDoS Dissector can be activated on an individual level. In this regard, it is also important that the extended fingerprint can be activated using an opt-in mechanism. For this, a new argument `-e, --extended-format` is introduced. The opt-in mechanism allows the current behavior of the DDoS Dissector in the future for nodes that do not want to participate in a global analysis. This is especially important since the extended fingerprint format requires more storage, particularly in combination with a high `-n` value. In the Deployment Considerations (*cf.* Chapter 5.3.1), this aspect is further discussed.

4.3.3 Fixing Parallelization Bug

When evaluating real PCAP datasets at the beginning of this work, a parallelization bug in the DDoS dissector was encountered. For performance reasons, the DDoS Dissector

reads a PCAP file in parallel. To enable parallelization, the input file is split into chunks of 5MB and temporarily stored in the `/tmp` folder. Thereafter, all chunks in the folder are read using multiple threads.

```
 elif filetype == FileType.PCAP:
     if filename.stat().st_size < (5 ** 6):
         return read_pcap(filename)
     LOGGER.debug(f'Splitting PCAP file {filename} into chunks of 5MB.')
+        # Create directory in tmp
+        os.makedirs('/tmp/dissector_chunk', exist_ok=True)
+        # Clean up directory in case previous run was aborted
+        old_chunks = [Path(rootdir) / file for rootdir, _, files in os.
    walk('/tmp/dissector_chunk')
+                      for file in files if file.startswith('chunk')]
+        if len(old_chunks) > 0:
+            for chunk in old_chunks:
+                os.remove(chunk)
+
+        # Splitting into chunks
+        process = subprocess.run(['editcap', '-c', '100000', filename,
    '/tmp/dissector_chunk/chunk.pcap'],
+                                 capture_output=True)
+        if process.returncode != 0:
+            LOGGER.critical("Splitting not successful", process)
+        chunks = [Path(rootdir) / file for rootdir, _, files in os.walk
    ('/tmp/dissector_chunk')
+                  for file in files if file.startswith('chunk')]

     pool = multiprocessing.Pool(nr_processes)
     results = pool.map(read_pcap, chunks)
```

Listing 4.14: Improved implementation for parallel processing

The encountered problem is related to the splitting of the PCAP files. The existing implementation of the DDoS Dissector uses the `tcpdump` command-line tool with the `-C 5` option to generate chunks of 5MB. For reasons not yet identified, the chunking prematurely terminated after the first chunk on certain devices. An error at this stage is not recognized by the DDoS Dissector, leading to a silent failure of the chunking process. Nonetheless, the fingerprinting process proceeds normally, but it only processes the initial 5MB of the PCAP file. The silent failure of the chunking, which results in an incomplete fingerprint, is fixed by the following proposed changes (*cf.* Listing 4.14):

- **Remove old chunks before processing**
  In the case that a previous run of the DDoS Dissector was not successful and failed before the `/tmp` folder was cleaned up, a new analysis may read additional unrelated chunks. This is resolved by storing chunks in a dedicated `/tmp/dissector_chunk` folder, which is cleaned up before every run (*cf.* Line 7ff.).

- **Use *editcap* instead of *tcpdump***
  On some devices, the `tcpdump` command failed for PCAPs that were previously edited and merged with `mergecap`. This issue is resolved by using the `editcap` [56] command-line tool with the `-c` option (*cf.* Line 15ff.).

- **Log errors**
  The main drawback of the existing DDoS Dissector implementation is that an error in the chunking process is silent and not handled at all. The proposed improved implementation checks for a successful return code (*cf.* Line 17) and logs an error if the chunking was not successful.

Chapter 5

Evaluation

This Chapter evaluates the proposed solution from different angles and discusses different adversarial attack vectors. First, a use case evaluation explains how the proposed solution can be used. This is shown based on four test scenarios of increasing size. Thereafter, the Reassembler module is evaluated using different experiments, thereby exploring the limits of the proposed solution. Lastly, the results of this work are discussed, focusing on achieving the set goals and considerations for deployment.

5.1 Use Case

The use case evaluation describes how the two proposed modules can be used for attack simulation (Fingerprint Generator) and global analysis (Reassembler). Additionally, the proposed changes to the DDoS Dissector are evaluated on a manually crafted small-scale scenario based on real DDoS attack data [55]. Lastly, Reassembler and Fingerprint Generator are evaluated based on four test scenarios.

5.1.1 Attack Simulation

The simulation of a globally observed DDoS attack depends on two main parts: *(i)* The simulated network topology and *(ii)* the attack characteristics (*e.g.,* target, sources). Both parts can be configured using the fluent interface [23] provided by the `Generator` class. Each instance method returns the instance itself at the end of the operation. This allows for readable and flexible chaining of methods. An example can be found in Listing 5.1.

The first part, the generation of the simulated network, is done in the constructor (*cf.* Listing 5.1, Line 1ff.). The maximum depth and number of nodes per level can be configured with optional arguments. Other network characteristics besides the topology can be updated after the initialization, for example, using the `set_spoofed_pct` method (*cf.* Line 4). By altering network characteristics after the initialization, multiple evaluation

# Nodes in Network	337
# Generated Fingerprints	382
Total Fingerprint Size	1.7MB
Smallest Fingerprint Size	734B
Largest Fingerprint Size	21.6 KB

Table 5.1: Simulated scenario output

runs can be performed using identical topology but varying configurations. This is important because network generation always includes a randomization factor for generating subnets.

```
1 Generator([IPNetwork(f"{10 + i}.0.0.0/8")
2                 for i in range(10)],
3                  max_levels=3, max_clients=5)
4             .set_spoofed_pct(0.8)
5             .set_random_attack_sources(10)
6             .add_background_traffic(50)
7             .simulate_attack()
8             .save_to_json("./fingerprints")
```

Listing 5.1: Fluent Interface of the Generator Class

The second part, the attack configuration, and simulation, is achieved by calling instance methods of the `Generator` class. A single random attack target and one attack source are selected at the initialization of the `Generator` instance. The number of attack sources can be flexibly configured after the initialization using the `set_random_attack_sources` method. The attack simulation is completed by calling the `simulate_attack` method. This generates fingerprints as discussed in the Implementation (*cf.* Chapter 4.1.2.3). The resulting fingerprints are stored in the instance variable `fingerprints` by default. The fingerprints can either be accessed directly using the instance, or they can be saved to files. The `save_to_json` method is used (*cf.* Line 8), which writes every fingerprint to a JSON file in the specified folder. The ability to access the fingerprints of the simulation directly on an instance level omits unnecessary file reads and writes when the fingerprints are directly evaluated using the Reassembler afterward.

The results of the above code Listing (*cf.* Listing 5.1) are summarized in Table 5.1. The configuration of ten subnets, with a maximum of three levels per subnet and five clients per level, results in a total simulated network of 337 nodes. It must be noted that the outputs are still subject to randomization, as the configuration is only an upper bound to the network size but does not specify a distinct topology. Since the `save_to_json` method is used, the resulting 382 fingerprints are stored in the specified output folder (*cf.* Listing 5.1, Line 8). The content of the output folder is visualized in Listing 5.2. It consists of a flat list of JSON files named by their derived md5 hash key (*cf.* Chapter 4.1.2.3).

The Fingerprint Generator Module's discussed outputs are based only on one network and attack configuration. The influence of the configuration parameters provided by the `Generator` class is further discussed in Chapter 5.1.4.

```
./fingerprints/
├── 0a0f91b88b2ee64531489fe8ab4b9f7b.json
├── 0b3e8f3f9ed807b20a07761570ca35da.json
├── 0d779f8b67007c18fabbfaaaa603c8e1.json
│   ...
├── fad0a2355ca500bba9f9dc9bf0f7b842.json
├── fb65d9068d1e7007b05917730ca46be1.json
└── fd796b5de70bbece27ed1eae77a2c07c.json
```

Listing 5.2: Output folder containing fingerprints

5.1.2 Global Analysis

For the global analysis of a DDoS attack, the Reassembler module is also provided as a Class with a fluent interface, similar to the Fingerprint Generator (*cf.* Chapter 5.1.1). The Reassembler can be configured to read from different data sources, and the global fingerprint output is configurable depending on the use case. This is further discussed in the next two Chapters.

5.1.2.1 Data Sources

The `Reassembler` module can be used with two different types of data sources. Listing 5.3 shows the usage of the Reassembler with in-memory data. Instead of writing the generated fingerprints persistently to a file, the fingerprints are directly accessed (*cf.* Line 4) and passed to the Reassembler using the `fingerprint_data` argument (*cf.* Line 6).

```
1 fingerprints = (Generator([IPNetwork(f"{10 + i}.0.0.0/8")
2                  for i in range(5)])
3                  .simulate_attack()
4                  .fingerprints)
5
6 summary = (Reassembler(fingerprint_data=fingerprints)
7            .reassemble()
8            .summary)
```

Listing 5.3: Reassembler with in-memory input

The second method to pass data to a `Reassembler` instance is based on persistently stored files in a local folder. When passing the optional `fingerprint_folder` argument to the constructor, the `Reassembler` iterates over all the files in the provided folder and reads them into an in-memory data structure. The example in Listing 5.4 shows that the `Generator` instance is only used to save the fingerprints to a folder, but the instance itself is not stored for later access (*cf.* Lines 1-4). Indeed, this is not needed since the Reassembler reads the fingerprints directly from the local path that is passed using the `fingerprint_folder` argument (*cf.* Lines 6-8).

Both options to pass fingerprint data to the Reassembler serve different purposes. The in-memory method can be used for quick iterations with the Fingerprint Generator, as it allows better performance due to fewer file reads and writes. The folder method yields

the greatest flexibility as it can also be used, for example, with real attack fingerprints generated by the DDoS Dissector. The two proposed data sources are not meant to be comprehensive. In the future, it is also possible to integrate, for example, remote data sources such as the DDoSDB [18].

```
(Generator([IPNetwork(f"{10 + i}.0.0.0/8")
            for i in range(5)])
 .simulate_attack()
 .save_to_json("./fingerprints"))

summary = (Reassembler(fingerprint_folder='./fingerprints')
           .reassemble()
           .summary)
```

Listing 5.4: Reassembler with folder input

5.1.2.2 Output

The output of the Reassembler is a global summary (or global fingerprint) of the observed DDoS attack based on the different attack fingerprints. Like the Generator module, the global `summary` can be accessed directly or persistently saved to a JSON file in memory on the instance. An example that does both can be seen in Listing 5.5. After calling the `reassemble` method (*cf.* Line 6), which analyzes all fingerprints and builds an in-memory summary, the global fingerprint is persisted in a file using the `save_to_json` method. At the same time, the summary is stored in a variable by accessing the `summary` property on the instance directly. The two output approaches serve different purposes, similar to the different data sources. Accessing the global fingerprint in memory allows for further processing of the summary or comparing multiple runs of the same Reassembler instance with modified data. The persistent storage of files can be used to share with others or for later analysis in a different context.

```
scenario = (Generator([IPNetwork(f"{10 + i}.0.0.0/8")
                       for i in range(5)])
            .simulate_attack())

summary = (Reassembler(fingerprint_data=scenario.fingerprints)
           .reassemble()
           .add_ground_truth_data(scenario.target, scenario.sources)
           .save_to_json('./global-fp')
           .summary)
```

Listing 5.5: Reassembler output configuration

To evaluate the accuracy, the `Reassembler` module allows adding ground truth data to the global fingerprint output. Ground truth data refers to information not available in a real-world scenario but is known for the provided training data. In the case of the Reassembler, such data includes the attack target and the real (non-spoofed) IP addresses of all attack sources. Using the `add_ground_truth_data` method, the global fingerprint is extended with only known information in a simulated context.

The generated global fingerprint from Listing 5.5 is shown in Listing 5.6. Since the default values without background traffic were used for the `Generator`, the resulting global fingerprint shows trivial data for some properties. For example, the `detection_threshold` percentiles all have the value 1.0 since there is no background traffic. In that regard, there are also no intermediate nodes to filter out so that the calculated `nr_intermediate_nodes` (*cf.* Line 12) corresponds with the `nr_location_observing_attack` (*cf.* Line 41) from the ground truth data. The impact of spoofed IPs and the background traffic on the accuracy of the global attack fingerprint are further discussed in Chapter 5.2.


```
{
  "attack": {
    "start_time": "2023-05-22T06:35:41.811895",
    "end_time": "2023-05-22T06:38:54.218793",
    "duration_seconds": 192.406898
  },
  "target": {
    "ip": "11.49.119.34"
  },
  "intermediate_nodes": {
    "discarded_intermediate_nodes": 0,
    "nr_intermediate_nodes": 8,
    "detection_threshold": {
      "25": 1.0,
      "50": 1.0,
      "75": 1.0
    },
    "key_nodes": {
      "11.49.0.0": {
        "nr_packets": 993801,
        "hops_to_target": 1.0,
        "detection_threshold": 1.0,
        "time_start": "2023-05-22T06:35:41.760191",
        "time_end": "2023-05-22T06:38:54.167089",
        "distance": 1,
        "inferred_distance_diff": 0.0,
        "fraction_of_total_attack": 1.0,
        "duration_seconds": 192.406898
      }, ...
    }
  },
  "sources": {
    "nr_sources": 1,
    "pct_spoofed": 0.0
  },
  "ground_truth": {
    "nr_attack_av": 9,
    "nr_background_av": 0,
    "nr_participating_nodes": 9,
    "nr_locations_observing_attack": 8,
    "sources": ["14.33.33.168", ...],
    "target": "11.49.119.34"
  },
  "key": "2aab1f4934a20ed8c6da54cc82424cde"
}
```


Listing 5.6: Global Fingerprint JSON output

Location	Attack Traffic	Background Traffic	TTL	Delay
Target	100%	0%	-	0ms
Router B	100%	20%	+1	2ms
Router A	50%	50%	+2	7ms

Table 5.2: Mixing attack and background traffic for evaluation scenario

5.1.3 Real Attack Fingerprints from DDoS Dissector

With the proposed extensions to the DDoS Dissector, running the Reassembler with attack fingerprints from real attacks is also possible. As mentioned, no public datasets are available that observe a single DDoS attack from different locations. In that regard, a highly specific custom scenario demonstrates the workings of the Reassembler process using actual attack data.

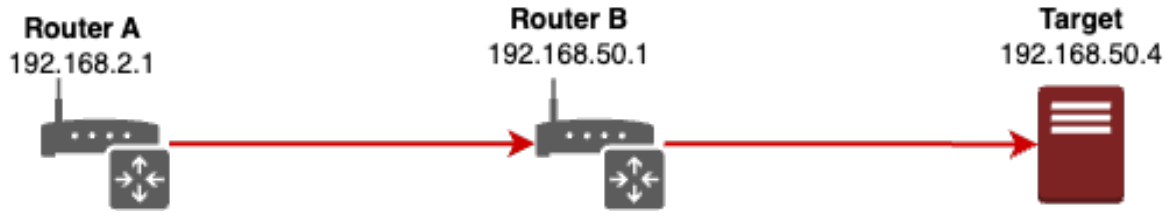


Figure 5.1: DDoS Dissector evaluation scenario

The CIC-DDoS2019 dataset [55] serves as a basis for the evaluation scenario shown in Figure 5.1. For the analysis, the SYN attack is filtered out from the rest of the dataset using the provided timestamps in [55]. Since the dataset contains traffic observed from only one location, the PCAP files for the three network participants in the evaluation scenario (*cf.* Figure 5.1) must be manually created. This is achieved by separating the PCAP into attack traffic directed toward the target and the remaining background traffic. Different amounts of attack and background traffic are blended to build a unique PCAP file for each of the three nodes (*cf.* Table 5.2). Furthermore, to simulate different recording locations for the PCAPs, the TTL values of the attack traffic are manually increased using *editcap* [56], and an artificial delay of a few milliseconds is added. This method creates two additional, though synthetic, perspectives of a real DDoS attack, which can be used for evaluation.

In the next step, the three resulting PCAP files are processed by the enhanced DDoS Dissector (*cf.* Chapter 4.3) to generate the extended attack fingerprint format necessary for the Reassembler. Listing 5.7 shows the CLI command to generate an extended fingerprint. The `main.py` file of the DDoS Dissector is executed with the argument `-e` to generate a fingerprint of the extended format. With the `-l` argument, the location in the fingerprint is set. Fingerprints are generated for the five most targeted IP addresses, defined by the argument `-n`.

```
python ./src/main.py -f ./target/traffic.pcap -e -l "192.168.50.4"  -n 5
    --output ./reassembler
```

Listing 5.7: Creating extended fingerprint using the DDoS Dissector

Network	Nr. Subnets	Max Levels	Max Clients	Nr. Nodes
N1	5	3	5	268
N2	15	6	5	9'356

Table 5.3: Different network configurations for the evaluation

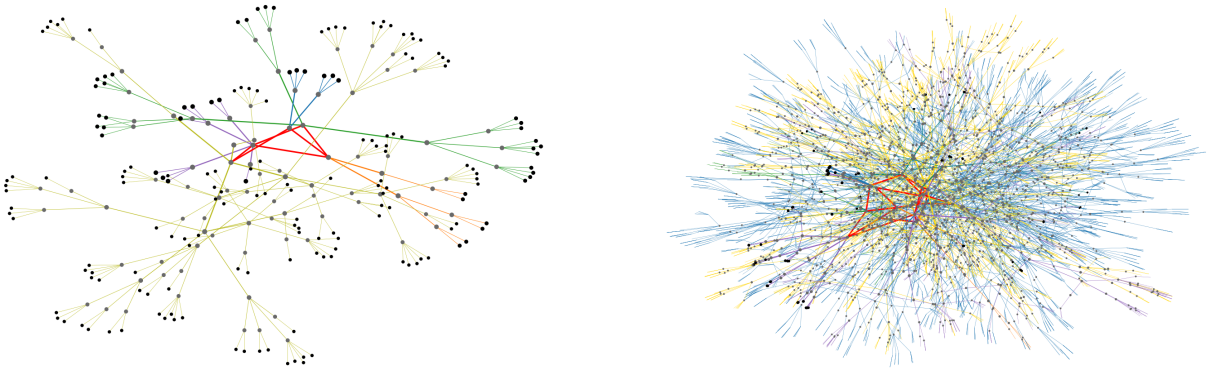
Given that the background traffic is not manually adapted for each network node of the evaluation scenario, duplicated fingerprints of background traffic can occur when processing the data of all three network nodes with the DDoS Dissector. The evaluation scenario results in 12 unique fingerprints that describe either background traffic or the attack from different locations.

Running the generated fingerprints through the Reassembler process generates a global fingerprint of the attack (*cf.* Appendix C). The result is positive, as the Reassembler correctly infers the attack target from all submitted fingerprints. Moreover, the two intermediate nodes (Router A and Router B), which also observed the attack, are accurately identified with the correct distance to the target. Since there are no nodes that only observe background traffic, none of the intermediate nodes are discarded in the global fingerprint.

Even though the evaluation scenario with real attack data offers limited complexity, the complete flow of generating fingerprints and analyzing them with the Reassembler works. Simulated test scenarios are used for further analysis of the Reassmbler process with increased complexity, as described in the next section.

5.1.4 Test Scenarios

The evaluation of the Reassembler module is based on different network and attack scenarios. From a network topology perspective, two scenarios exist (*cf.* Table 5.3). The small network *N1* consists of 268 network nodes (*cf.* Figure 5.2a). The large network *N2* consists of almost 10'000 network nodes grouped in 15 subnets (*cf.* Figure 5.2b).



(a) N1 (b) N2

Figure 5.2: Visualization of the two networks in Table 5.3

Scenario	Network	Nr. Sources	Nr. Background	Pct. Spoofed	Fingerprints
S1	N1	5	10	25%	85
S2	N1	20	50	25%	343
S3	N2	100	200	25%	2'777
S4	N2	500	1000	25%	12'712

Table 5.4: Different attack configurations for the evaluation

Based on the two networks, four different attack scenarios are created with increasing attack size (*cf.* Table 5.4). For each network (N1, N2), a small and a large attack are simulated. With the percentage of spoofed IPs constant at 25%, the number of resulting fingerprints mainly depends on the number of sources and background routes. For the large scenarios (S2, S4), it can be observed that the number of fingerprints generated exceeds the number of nodes in the respective network. For example, scenario S4 generates 12'712 fingerprints in a network (N2) of 9'356 nodes. This means that some of the network nodes submit more than one attack fingerprint.

5.1.4.1 Ground Truth Data

In a simulated environment, the ground truth data is known but not used for the Reassembler process. It provides insights into how the configuration parameters of the scenario and the proposed simulated network topology influence the generated fingerprints. With the background traffic configuration of the scenarios (*cf.* Table 5.4), it is clear that not all nodes observe actual attack traffic. Still, many observe unrelated (*i.e.,* background) traffic. Figure 5.3 shows the percentage of nodes observing the attack out of all nodes submitting at least one fingerprint. The percentage values range from 28% to 35%. This aligns with the expected results from the attack configuration, where the number of attack sources is about half of the background traffic for all scenarios. The observed variations can be explained by the randomness in the scenario, where an attack traffic path might have a different length (*i.e.,* involves fewer nodes) than a background traffic path.

Similar fluctuations can also be observed when analyzing the composition of observed attack vectors (*i.e.,* flattened fingerprints). Figure 5.4 shows a significantly lower percentage of true attack vectors in Scenario 1. This is because three of the five randomly assigned attack sources are located in the same subnet as the attack target, thus producing much shorter attack traffic paths that involve fewer nodes. In the other scenarios, the true and background attack vectors are roughly split into 40% and 60%. The deviation from the expected value of 33% can be attributed to the difference in path length between attack and background traffic paths. Attack sources are only selected from leaf nodes in the subnet tree, but background traffic paths are randomly sampled from the whole network. This means that, on average, attack traffic paths are longer than background traffic paths and involve more nodes.

Overall, the ground truth data of the scenario aligns with the provided attack configuration. Nevertheless, the inherent randomness within the scenario can cause significant fluc-

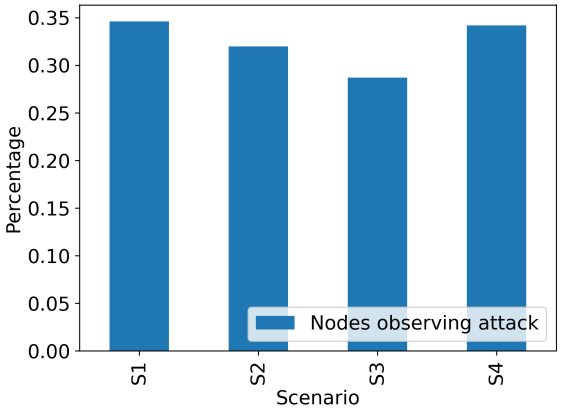


Figure 5.3: Percentage of nodes observing the attack

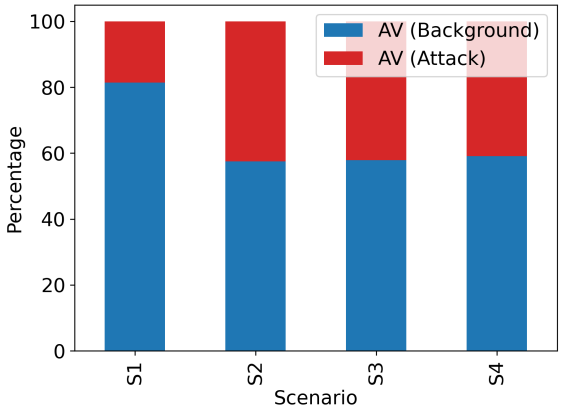


Figure 5.4: Composition of observed attack vectors within each scenario

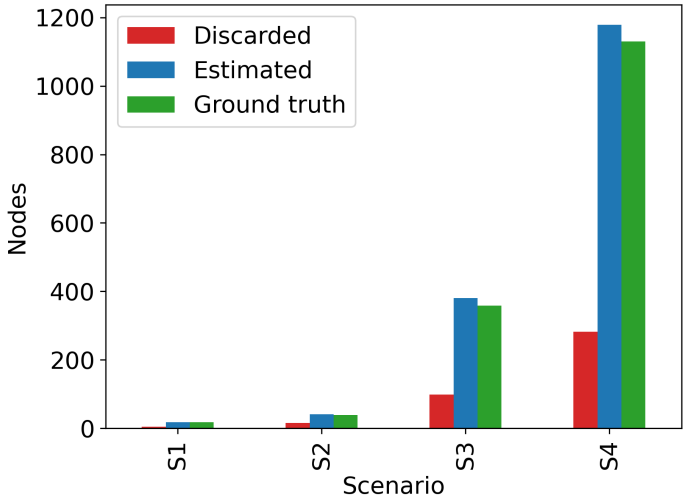


Figure 5.5: Intermediate nodes estimation

tuations for smaller network topologies or attack configurations with only a few sources.

5.1.4.2 Reassembler Evaluation

The Reassembler module aims to derive the previously discussed ground truth data and the attack configuration from the information in the attack fingerprints. This reliably estimates intermediate nodes observing an attack (*cf.* Figure 5.5). By discarding nodes that only monitor background traffic (*cf.* Chapter 3.2.3.4), the Reassembler estimates the number of nodes as slightly too high compared to the ground truth data. Even though the absolute values increase from scenario 1 to scenario 4, the relative number of discarded nodes lies between 19% and 28% for all scenarios. This is unsurprising since the background and attack traffic ratio is comparable across the four scenarios (*cf.* Table 5.4).

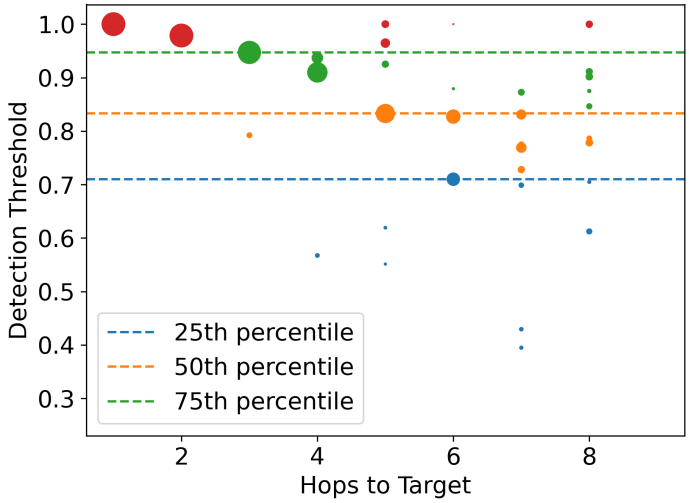


Figure 5.6: Detection thresholds for scenario 2

Another insight derived by the Reassembler is the global distribution of detection thresholds of the participating nodes. A scatter plot of the detection thresholds from the nodes in scenario 2 is shown in figure 5.6. The x-axis represents the distance in hops to the attack target. The size of each marker signifies the total count of packets observed by a node, helping in determining whether it is a high-volume node or a comparatively smaller one.

It stands out that up to the fifth hop on the x-axis, one single node per hop is always significantly larger than the others. This aligns with the hierarchical network topology, where, within a subnet, only one path from the root node to a leaf node (*i.e.,* the attack target) exists. At a single hop distance to the target, the detection threshold corresponds to 1 (*cf.* Figure 5.6), indicating that every packet observed is considered part of the attack. Based on the ground truth data of the simulated scenario, this assumption is incorrect as some of the background traffic is also directed toward the attack target. However, since the DDoS Dissector does not perform sophisticated attack detection, such imprecision cannot be omitted.

Overall, the detection thresholds shown in Figure 5.6 are rather high, with only a few outliers below 50%. This is beneficial because a standard DDoS Dissector instance would detect the attack and generate an attack fingerprint. However, the detection thresholds strongly depend on the background traffic amount. This is further explored in Chapter 5.2.

Using attack fingerprint data, the Reassembler also detects spoofed IP addresses. This is done based on the idea of [65] (*cf.* Chapter 3.2.3.3) and relies on IP collisions with different TTL values. Each of the four scenarios (*cf.* Table 5.4) employs a probability of 25% that an attacking node uses IP spoofing techniques. Figure 5.7 shows the estimated count of (real) attack sources calculated by the Reassembler. In general, it can be observed that the estimated number of sources for all scenarios is many times too high. For S2 and S3, the number of observed sources at the attack target is four times bigger than the actual

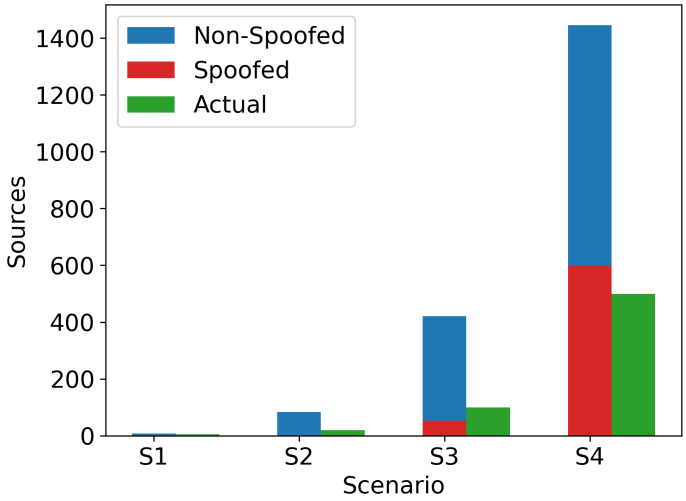


Figure 5.7: Spoofed sources estimation for scenarios

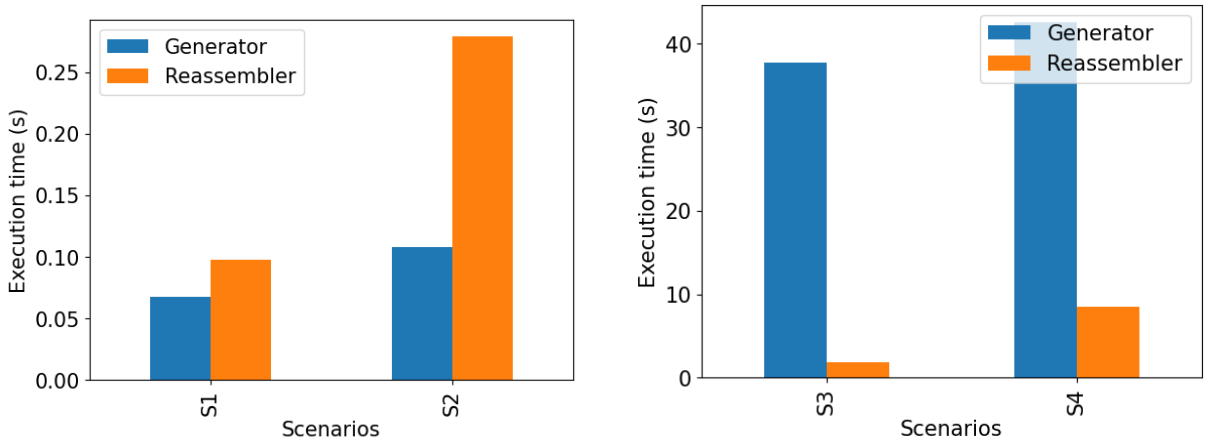
count of attack sources. In S4, the estimation is slightly lower yet almost three times higher than the actual count.

The goal of the Reassembler is to normalize the observed count of attack sources by estimating how many of the observed sources are spoofed. This is indicated with the red bar in the plot (*cf.* Figure 5.7). It can be seen that the Reassembler identifies a considerable number of sources as spoofed in the scenarios that have a high count of attack sources (S3, S4). In Scenario 4, more than 40% of the observed attack sources are identified as spoofed. However, even when deducting 40% of the sources, the estimated count is still almost twice the actual count. This shows that even a moderate IP spoofing probability (25%) can substantially inflate the number of sources observed at the target. Furthermore, the Reassembler cannot detect every spoofed IP address. This is expected to a certain extent as the detection mechanism is based on IP collisions that are not guaranteed to occur. The limits of the detection mechanism are further explored in the Experiments Chapter (*cf.* Chapter 5.2).

Overall, the gathered insights confirm that interesting properties of a DDoS attack can be derived by the Reassembler when analyzed on a global scale. Nevertheless, it has been demonstrated that many parameters influence the accuracy of the analysis and do not work in every scenario. For this reason, Chapter 5.2 conducts a series of experiments, each manipulating a single parameter to test its influence on the accuracy of the Reassembler analysis.

5.1.4.3 Performance Evaluation

Running the four scenarios in Table 5.4 reveals interesting findings in terms of execution duration (*cf.* Figure 5.8). All scenarios use in-memory fingerprint data without saving it to permanent storage, thus eliminating file read/write operations. For Scenario S1 and S2 in the smaller network, the Reassembler takes more time than the Generator



(a) Scenarios based on N1

(b) Scenarios based on N2

Figure 5.8: Execution duration by scenario

(*i.e.,* network simulation and fingerprint generation). In S2, the execution time of the Generator exceeds the Reassembler by more than double. Nevertheless, the total execution time for each scenario in the smaller network remains quick, taking less than a second. Contrary behavior is observed for the scenarios in the large network (N2). The Generator requires a significantly longer execution duration than the Reassembler, which brings the total scenario execution time to almost one minute. The key observation is that the Reassembler scales better with an increasing number of fingerprints than the Generator's performance with increasing network size. However, this is only relevant when iterating with the Reassembler in a simulated scenario. In the real world, the computational load to generate fingerprints is distributed among the participating network nodes.

5.2 Experiments

With the simulated network topology and the many possible attack configurations, there are numerous parameters that influence the output of the Reassembler. As shown by the four attack scenarios in Chapter 5.1, it is not always trivial to see which parameter of the attack scenario influenced the outcome the most. Often, it is necessary to run multiple iterations of the same scenario with minor modifications to certain parameters to build an intuition of how specific parameters influence the outcome. This Chapter aids in developing an understanding of the most influencing parameters based on different experiments. First, different adversarial scenarios are discussed, focusing on the parameters that are influenced by an attacker's tactic. Following this, detection thresholds are explored, which affects how participating nodes need to configure their DDoS Dissector instance.

5.2.1 Adversarial

This Chapter explores adversarial attacks on reassembling a global picture of a DDoS attack. In this context, any strategy that attempts to hinder the detection or the analysis

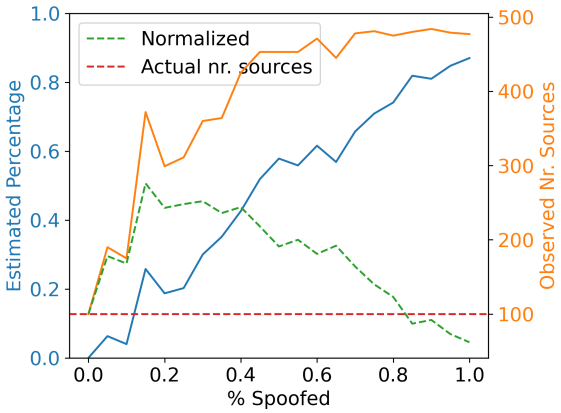
process of the attack is considered adversarial. The following sections explore the effects of IP spoofing techniques (*cf.* Chapter 5.2.1.1) and the impact of missing fingerprint data (*cf.* Chapter 5.2.1.2) on the basis of different experiments. Lastly, Chapter 5.2.1.3 discusses remaining adversarial threats and possible solutions.

5.2.1.1 IP Spoofing

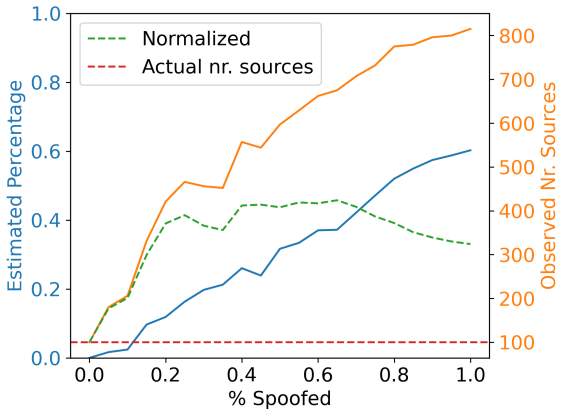
The use case with the four scenarios has shown that the estimation of attack sources is too high for certain cases (*cf.* Chapter 5.1.4.2). Following this observation, this Chapter aims to assess the impact of the percentage of spoofed addresses and the size of the spoofed address pool on the accuracy of the attack sources estimation. This is achieved using an isolated scenario without background traffic built on the N2 network configuration (*cf.* Table 5.3). The attack scenario consists of 100 randomly chosen attack sources. Across the different runs, only the spoofed percentage is updated, while the target nodes and the network stay the same. Due to the randomness in setting nodes that use a spoofed IP, it is possible that an attack source uses spoofed IPs in one run, but a real IP in the next one. While the attack sources stay the same, the set of nodes that use a spoofed IP is randomly drawn on each run.

The results of the experiment are shown in Figure 5.9. Within each plot, the spoofed percentage is increased in 5% steps from 0% to 100% using the same network and spoofed IP pool size. Each subplot uses a different IP Pool Size, ranging from 500 (*cf.* Figure 5.9a) to 10'000 (*cf.* Figure 5.9d). The red horizontal dashed line represents the actual number of sources, which remains consistent across all experimental runs. The orange line denotes the number of observed sources at the attack target. It includes all observed IPs, including spoofed IPs that, for example, have only been observed once. The blue line signifies the estimated percentage of spoofed addresses based on the detection mechanism discussed in Chapter 3.2.3.3. Finally, the green dashed line denotes the normalized count of observed sources, calculated by subtracting the estimated spoofed percentage (blue line) from the observed count (orange line).

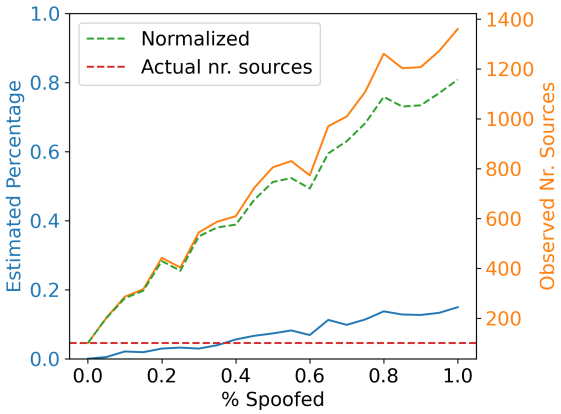
From the plot in Figure 5.9a, it can be observed that the number of observed sources rapidly grows with an increasing percentage of spoofed IPs. However, as the estimated percentage (blue line) practically increases linearly with the spoofed percentage on the x-axis, the normalized count (green) settles around 2.5 times the actual count. More than that, the normalized count falls below the actual count for spoofed percentages above 80%. Such behavior is only observed in Figure 5.9a, which has the smallest spoofed IP pool size. The decrease of the normalized count with increasing spoofed percentage can be explained by the selection of spoofed IPs in the network simulation. A network node that employs IP spoofing draws a random sample between 7 and 25 IPs from the pool of spoofed addresses. With 100 attack sources drawing at least 7 spoofed IP addresses (assuming a spoofed percentage of 100%), the pool of available spoofed addresses is quickly exhausted. This makes the detection of spoofed IP addresses very accurate, as almost every spoofed IP is used by more than one node, thereby triggering the detection mechanism. With such a high estimated percentage of spoofed addresses, the normalized count is expected to fall below the actual count. The exhaustion of available spoofed IPs explains why the count of observed sources (orange) converges to 500.



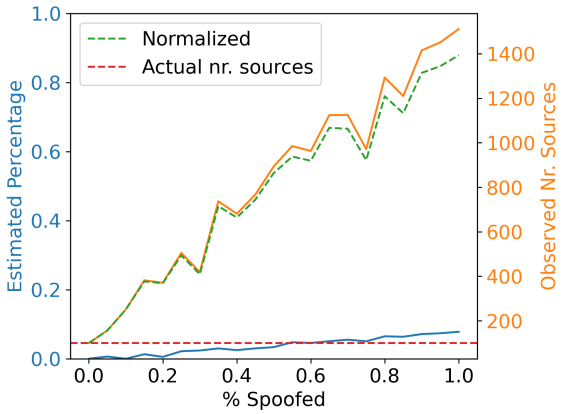
(a) Spoofed IP pool size: 500



(b) Spoofed IP pool size: 1'000



(c) Spoofed IP pool size: 5'000



(d) Spoofed IP pool size: 10'000

Figure 5.9: Spoofed sources estimation with different spoofed IP pool size

The normalized count does not fall below the actual count for scenarios with a larger pool size of spoofed IPs (>=1000). While a decrease in the normalized count can still be observed with a pool size of 1000 (*cf.* Figure 5.9b), it is less extreme than the smallest pool size of 500. With an even larger pool size (*cf.* Figures 5.9c,5.9d), the estimated percentage of spoofed IPs never exceeds 20%. As a consequence, the normalized count practically increases equivalent to the observed count of sources. This leads to an overestimation of actual sources by more than 10 in extreme cases.

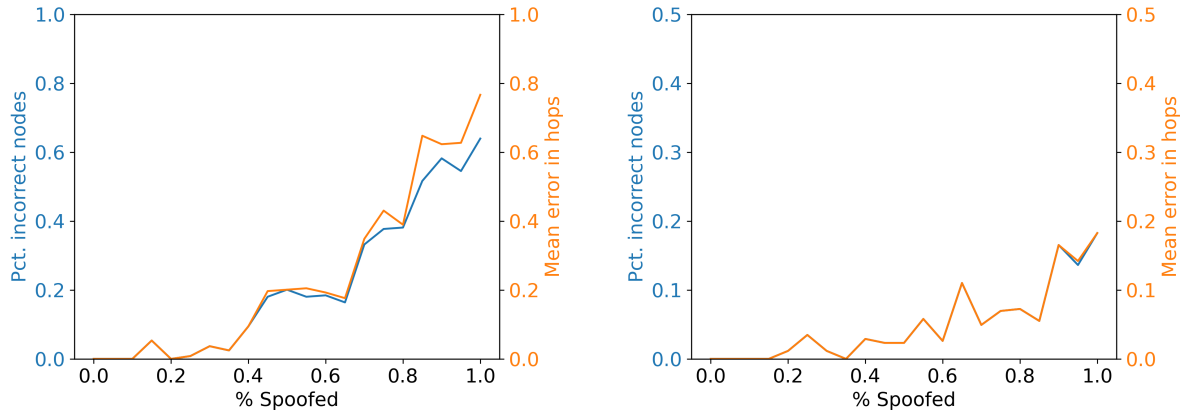
In summary, the isolated scenarios show the interplay between the pool size of available spoofed IPs and the overestimation of source IPs in the attack analysis. By increasing the number of available IPs that can be used for spoofing, the overestimation of attack sources grows dramatically. Moreover, increasing the percentage of nodes using IP spoofing techniques intensifies the problem further. The main challenge lies in artificially constraining the number of available spoofed IPs does not correspond to behavior that can be observed in the real world but is required to make the simulated scenario work. Further work is required to evaluate whether such IP spoofing techniques behave differently in a real, non-simulated environment.

Besides the estimated count of attack sources, IP spoofing also affects the distance estimation for intermediate nodes. As discussed in Chapter 3.2.3.2, the distance from an intermediate node to the attack target is calculated based on the observed TTL values. Observing different TTL values for a source IP at a single node can lead to inaccurate distance calculation. The impact of this problem is explored in Figure 5.10. The spoofed percentage increases over multiple runs based on the N2 network (*cf.* Table 5.3) with 100 attack sources and no background traffic. Through this process, the impact on the percentage of incorrect distance estimates (blue line) and the mean error in terms of hops (orange line) are observed. The experiments are conducted twice with different spoofed IP pool sizes.

For a spoofed IP pool size of 500 (*cf.* Figure 5.10a), the percentage of incorrectly estimated distances is low for a spoofed percentage below 40%. For higher spoofed percentages, the percentage of incorrectly estimated distance increases up to 80% in extreme cases. Remarkably, the mean error in hops remains under one, even with 100% spoofed IP addresses. In that regard, there are many incorrect distance estimations. However, the majority of estimations are only one hop off.

A large improvement in accuracy can be observed when increasing the spoofed IP pool size to 1'000 (*cf.* Figure 5.10b). With the larger pool size, the percentage of incorrectly estimated distances stays at under 20% in all cases. The blue and red lines are almost congruent, which signifies that nearly all incorrectly estimated distances are off by 1. Only for the case of 95% spoofed IPs, the mean error is slightly higher than the percentage of incorrectly estimated distances, which signifies that at least one distance was off by more than one hop.

In terms of distance estimation, the impact of the spoofed IPs pool size is inversely proportional. With an increasing pool size, the distance estimation becomes more accurate. This is contrary behavior to the attack source count estimation discussed before. In that regard, the distance estimation is expected to behave better in real-world scenarios where



(a) Spoofed IP Pool Size: 500

(b) Spoofed IP Pool Size: 1'000

Figure 5.10: Distance estimation of intermediate nodes with increasing spoofed percentage

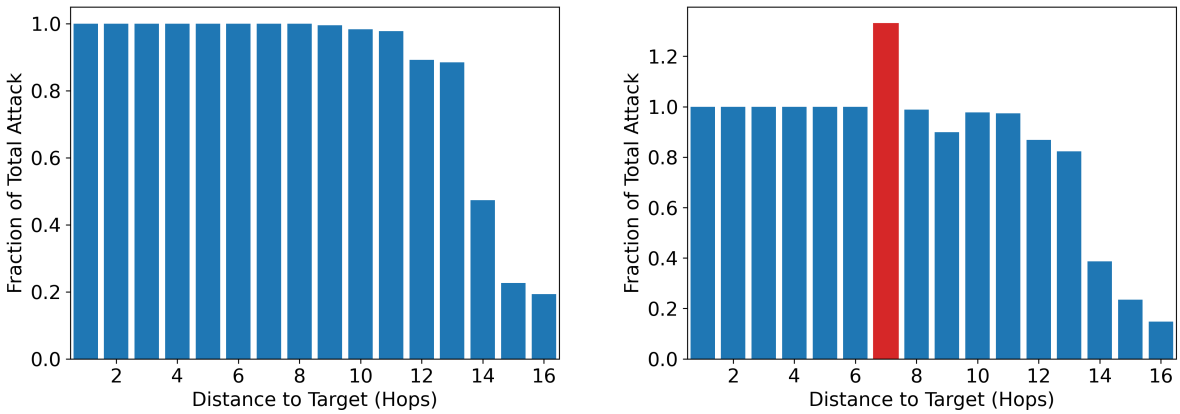
the number of available spoofed IPs is not artificially limited. Still, future work is needed to confirm these observations in a non-simulated scenario.

5.2.1.2 Missing Fingerprints

In the simulated scenario, every network node that observes any kind of traffic submits a fingerprint, except the attack sources. Only looking at fingerprint data, an overview of how the attack was globally observed can be built by grouping fingerprints based on their estimated distance to the target and calculating the sum of the `fraction_of_total_attack`. With perfect coverage of the attack, it is possible to see from what distances most of the attack traffic emerged. This is visualized for scenarios S3 and S4 (*cf.* Table 5.4) in Figure 5.11. The bar chart shows the proportion of the total attack observed at any given distance away from the attack target. For the scenario, S3 (*cf.* Figure 5.11a), most of the attack traffic is first observed between 13 and 15 hops away from the attack target. From 9 hops away, almost 100% of the attack is on each hop to the attack target. As mentioned, such an analysis is only possible when all fingerprints along the attack traffic path are submitted to the Reassembler.

As previously demonstrated, the accuracy of distance estimation can vary depending on the percentage of addresses that are spoofed (*cf.* Chapter 5.2.1.1). Even minor deviations of just one hop in the distance can result in an attack coverage chart that is not logical for certain distances. An example of this is shown in Figure 5.11b, where the observed coverage 7 hops away from the target is over 100% (indicated by the red bar). Naturally, a value over 100% does not make sense and indicates that some distance estimations are incorrect. As a consequence, some nodes are attributed to the wrong group (*i.e.,* distance), which distorts the result. The attack coverage chart can indeed provide useful insights into assessing the accuracy of the distance estimation. This is especially beneficial in real-world scenarios where ground truth data about the network topology is unavailable.

The ideal scenario of perfect coverage, where every network node contributes fingerprints, is unlikely for two reasons. Firstly, not every network node (*e.g.,* backbone routers) is



(a) Attack Coverage in S3

(b) Attack Coverage in S4

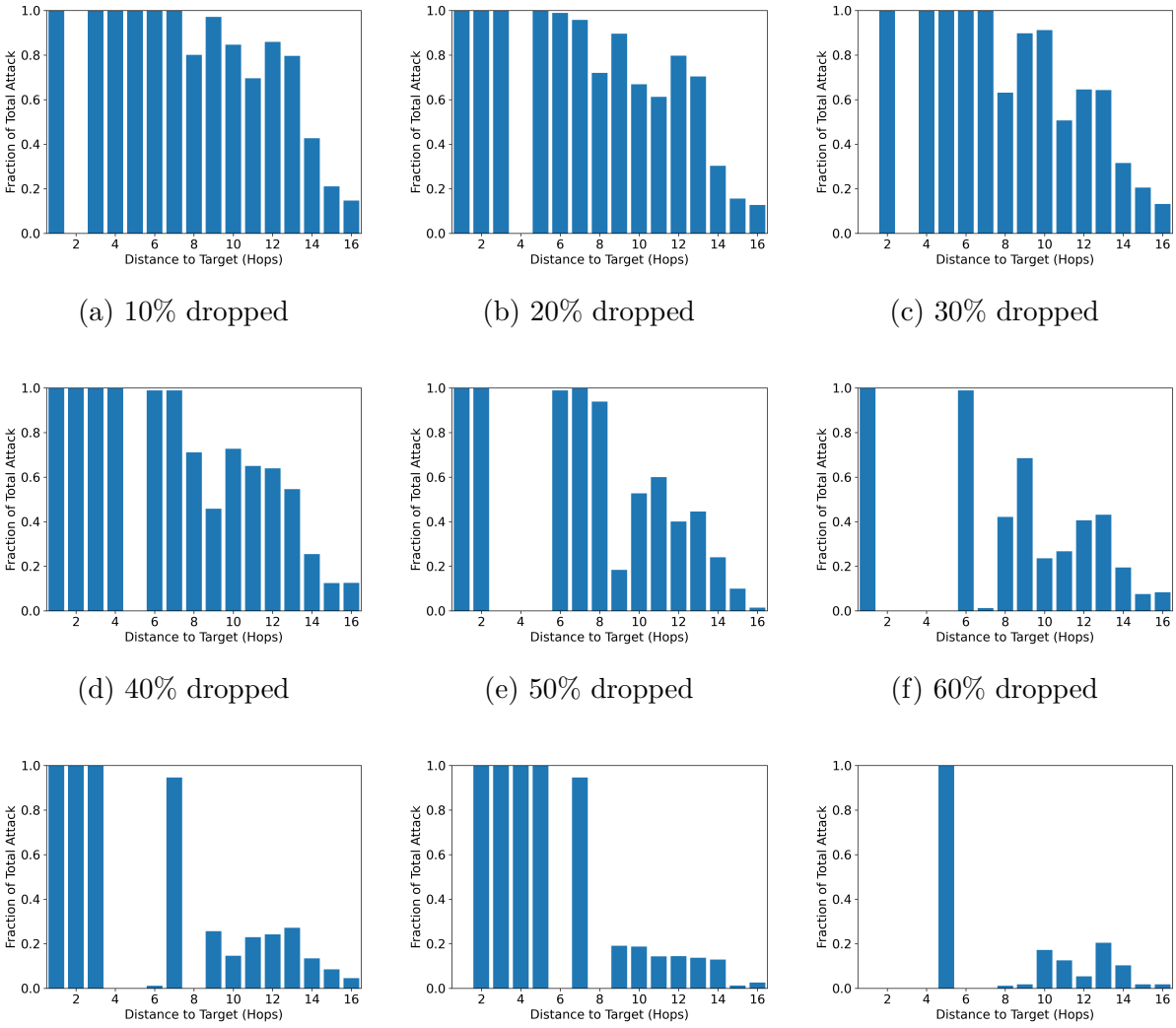
Figure 5.11: Attack coverage per distance for scenarios S3 and S4

guaranteed to participate in the process and run the DDoS Dissector software. Secondly, network nodes with malicious intent might deliberately withhold fingerprints to hinder detecting and analyzing a certain attack. In this regard, the following experiment evaluates how the Reassembler analysis is affected by missing fingerprints and what minimum amount of network nodes should participate in the process.

The experimental setup is based on the configuration of scenario S3 (*cf.* Table 5.4). Figure 5.11 displays the attack coverage chart from nine separate runs of the scenario, each with an increasing amount of dropped (*i.e.,* not available) fingerprints. It is worth mentioning that the sample of discarded fingerprints is drawn independently for each run. Thus, one fingerprint might be discarded in the chart with 10% of dropped fingerprints (*cf.* Figure 5.12a) but still included in the chart with 20% dropped fingerprints (*cf.* Figure 5.12b).

The key observation across all runs is that the attack coverage near the attack target consistently shows extreme values, either at 100% or directly at 0% with no intermediate values. This can be explained by the hierarchical network structure, where only one path exists from a root to a leaf node in a subnet. In this context, the observed fraction of total attack close to the attack target only consists of a single fingerprint, which explains the extreme values of either 100% or 0%. The observed attack coverage farther away decreases in slower steps as it is comprised of several smaller fingerprints instead of a single large one. To summarize, removing random fingerprints greatly affects the coverage close to the target, while its impact diminishes with increasing distance to the target.

The question remains how the rest of the analysis, for example, the estimated number of intermediate nodes, is affected by missing fingerprints. This is evaluated by comparing the estimated number of intermediate nodes with the ground truth data across multiple runs with an increasing number of dropped fingerprints (*cf.* Figure 5.13). It can be observed that the number of estimated nodes decreases linearly as the number of excluded fingerprints increases. This trend is positive in that there are no unexpected outliers caused by the exclusion of specific fingerprints. In that regard, there is no minimum amount of participating network nodes required to make a global analysis work. However, the estimated size of an attack strongly depends on the number of participating nodes.



(a) 10% dropped (b) 20% dropped (c) 30% dropped

(d) 40% dropped (e) 50% dropped (f) 60% dropped

(g) 70% dropped (h) 80% dropped (i) 90% dropped

Figure 5.12: Scenario S3 with increasing amount of dropped fingerprints

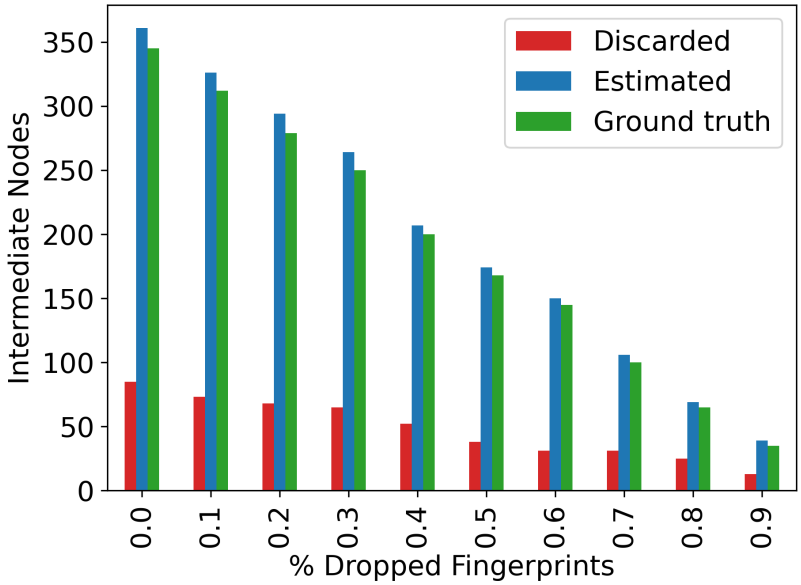


Figure 5.13: Intermediate nodes estimation with an increasing amount of dropped fingerprints

Further research could explore whether the attack coverage chart can be used to refine the estimated number of intermediate nodes. For instance, a low attack coverage at a close distance to the target implies missing fingerprints. This concept can also be applied to nodes farther away, given the understanding that the total attack coverage should never decrease when moving closer to the target.

5.2.1.3 Additional Adversarial Attack Vectors

In addition to the previously discussed adversarial attack methods on the Reassembler process, further threats exist to an accurate global analysis. This section overviews additional attack vectors and discusses strategies to address them.

For adversarial attacks, the primary concern is participating network nodes that do not act honestly or even attack sources that try to act as intermediate nodes to falsify the analysis. From an intermediate node perspective, the following threats are considered:

- **Submitting forged fingerprints**
  An adversarial intermediate node could forge fingerprints by hiding specific source IPs or by adding unrelated source IPs to divert attention to someone else. For example, when adding the IP of an honest intermediate node as a source IP to an attack fingerprint, it can make the honest intermediate node less trustworthy. Currently, the proposed Reassembler solution does not explicitly filter fingerprints from attack sources, because this mechanism can be abused by submitting forged fingerprints. Preventing such behavior is hardly possible. However, suspicious behavior could be monitored using outlier detection, for example, if a source IP is only reported in one fingerprint.

- **Withholding fingerprints**
  Intentionally withholding fingerprints has the same effect as not participating in the first place. The impact of missing fingerprints has already been discussed in Chapter 5.2.1.2. In general, such behavior cannot be prevented, but it would be possible to penalize a network node (*i.e.,* decreasing its trustworthiness) for not submitting fingerprints. Although withholding a fingerprint might slightly impact the accuracy of the analysis, it cannot compromise the overall analysis process.

From an attack source perspective, the following strategies pose a threat:

- **Submitting a fingerprint for a different node**
  Following the idea of a spoofed source address in a TCP/IP packet, a fingerprint can also be submitted with an incorrect location field. An adversary can abuse this to spam the Reassembler process with many incorrect fingerprints, making it hard to distinguish which fingerprints are real and which are forged. The problem can be addressed by introducing an authentication mechanism before submitting a fingerprint. On the one hand, this increases the entry barrier for network nodes to participate in a global DDoS detection effort. Nevertheless, such a registration process also allows gathering data about the network topology (*e.g.,* neighboring AS of a network node).

- **Customized TTL values**
  Different analysis steps of the Reassembler are based on the observed TTL values. While an attack source cannot influence the TTL values along the path, it is possible first to explore the path to a target (*e.g.,* using traceroute [63]) and then calculate an initial TTL value so that the detection of spoofed IPs is not triggered. As detecting spoofed IPs based on TTLs has been shown to not work in every case, further research in this area is required to explore how a global DDoS analysis can be improved to better handle customized TTL values.

Overall, most of the threads either require complicated synchronization with other attacking nodes to be effective or have a rather small impact on the global analysis. However, further research is required to better explore and evaluate such threads in a non-simulated environment.

5.2.2 Detection Thresholds

The fraction of total attack that is observed decreases when moving away from the attack target. This intuition has already been addressed in Chapter 5.2.1.2, wherein it was discussed how the total observed fraction of an attack further away from the target decreases slower when dropping fingerprints (*cf.* Figure 5.12). The observation can also be confirmed by directly looking at a scatter plot of the observed fractions of the total attack by distance to the attack target (*cf.* Figure 5.14). A simple linear regression shows that the fraction of the total attack behaves inversely proportional to the distance from the attack target. It has to be noted, that the regression line does not serve as a general

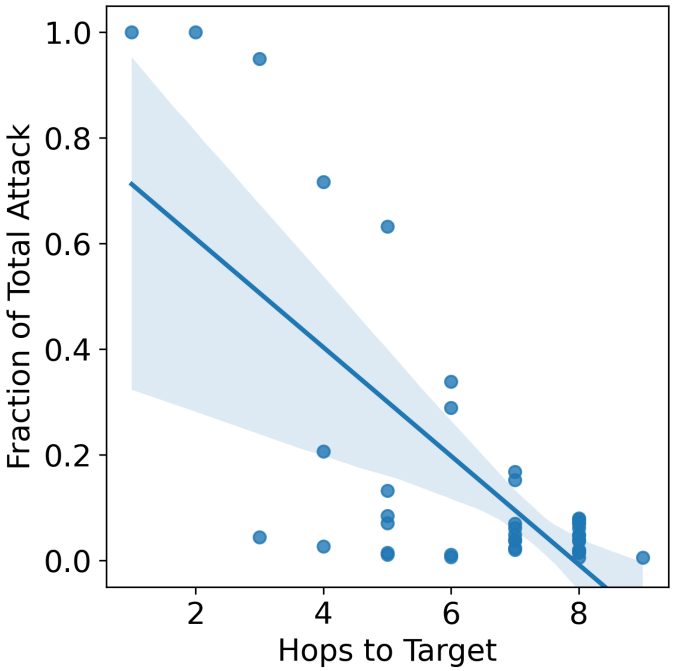


Figure 5.14: Fraction of total attack vs. hops to target for scenario S2

indicator for the expected fraction of the total attack at a certain distance. Rather, it shows the trend for a specific network topology, in this case, the Network N2 (*cf.* Table 5.3) with a maximum path length between two nodes of 9.

Furthermore, it can be observed that the detection threshold is also related to the observed fraction of the attack. In the simulated scenario, the detection threshold describes the percentage of attack traffic observed compared to unrelated traffic. In the real world, a DDoS Dissector node only detects an attack if the configured threshold is lower or equal to the observed attack traffic. As Figure 5.15 shows, the fraction of total attack behaves proportionally to the detection threshold in Scenario S2. In this regard, it confirms the intuition that nodes located further away from the attack target not only perceive a reduced proportion of the total attack but also require a lower detection threshold to identify the attack. For the attack scenario S2 (*cf.* Table 5.4), the detection thresholds are rather high in general (*cf.* Figure 5.15). As previously discussed, the detection threshold percentiles depend on the network's background traffic. This is further explored in the next experimental setup.

To evaluate the impact of background traffic on the required detection thresholds, several runs of Scenario S3 are performed, progressively increasing the background traffic. The four runs with 500, 2'000, 5'000, and 10'000 unique background traffic routes are depicted in Figure 5.16. Each marker's size represents the network node's size regarding observed packets. With 500 background routes, 2.5 times the number of attack routes, the median detection threshold is slightly over 80% (*cf.* Figure 5.16a). The 75th percentile is nearly at a 100% detection threshold, suggesting that a large number of nodes observed only the attack and no background traffic. With 10 times the number of background traffic compared to attack traffic, the median detection threshold decreases to around 50% (*cf.*

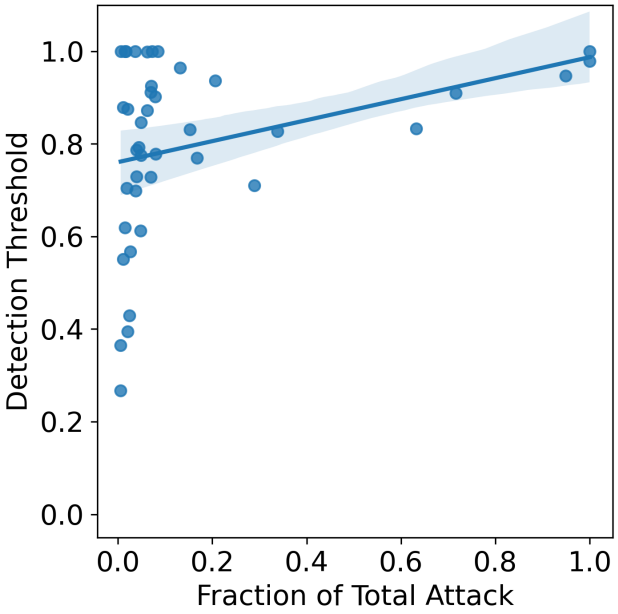


Figure 5.15: Detection threshold vs. fraction of total attack for scenario S2

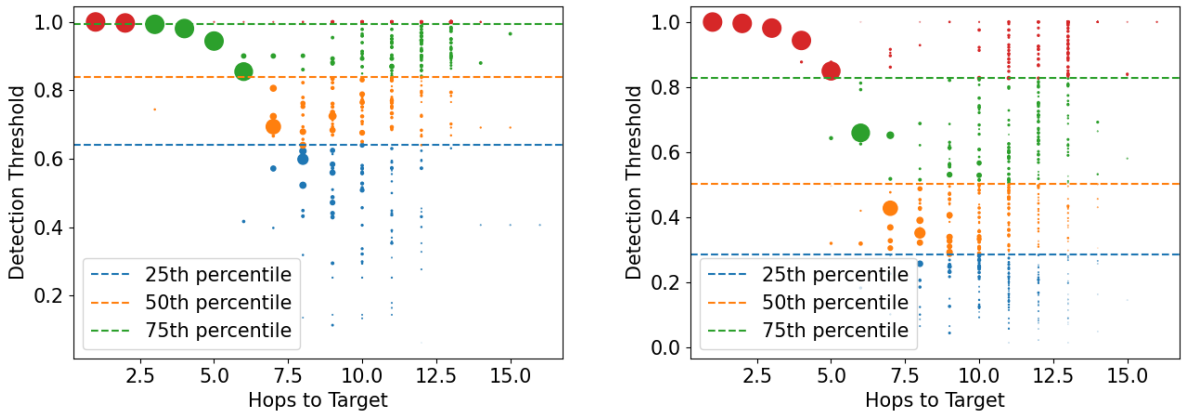
Figure 5.16b). In comparison, the 75th percentile with 2'000 background routes remains relatively high at over 80%. Only when increasing the number of background routes to 5'000, all three percentiles for the detection threshold fall below 50% (*cf.* Figure 5.16c). For 10'000 background routes, which is 50 times the number of attack routes, the mean detection threshold falls below 20% (*cf.* Figure 5.16d).

In general, it can be observed that nodes close and within the same subnet as the attack target have a very high detection threshold. A node can detect attacks in the same subnet even when running the DDoS Dissector software with a high threshold, thus producing fewer fingerprints. Other nodes with a high detection threshold are usually located further away and are rather small regarding observed traffic (*cf.* Figure 5.16). This behavior is related to the simulated scenario, where, due to randomization, some nodes don't observe much other traffic than the attack traffic.

In Summary, the experiment confirms that the detection threshold of nodes decreases with a growing amount of background traffic. However, large nodes close to the attack target are less affected than smaller nodes. The experiment shows that a reasonable value for the detection threshold is highly dependent on the network topology and the composition of observed traffic. In that regard, future work is needed to explore what percentages of detection threshold are reasonable in a real-world setting, considering different DDoS attack strategies.

5.3 Discussion

The previous Chapters have discussed and evaluated the proposed solution in detail. Several positive aspects are worth mentioning. First, the proposed Reassembler solution



(a) 500 Background Routes

(b) 2'000 Background Routes

(c) 5'000 Background Routes

(d) 10'000 Background Routes

Figure 5.16: Detection thresholds for scenario S3 with increasing amount of background traffic

performs effectively for real and simulated data. This shows the broad applicability of the proposed solution. Second, the introduced fingerprint generator (*cf.* Chapter 3.1) facilitates rapid iterations of the whole process. This is highly beneficial as it encourages further research in this area because changes to the fingerprint format can be tested and evaluated easily. Lastly, with the implemented changes to the DDoS Dissector, it is possible to generate attack fingerprints with the newly proposed extended format (*cf.* Chapter 3.1.2.1) based on real network capture data. The changes to the DDoS Dissector further include bug fixes and are backward compatible so that existing installations of the DDoS Dissector keep working.

Several aspects were identified that require further improvement based on the conducted evaluation and experiments (*cf.* Chapter 5.2). While the simulated scenario covers the requirements discussed in Chapter 3.1.1, it still presents limited complexity regarding the mix of attack and background traffic. The findings from the experiments (*cf.* Chapter 5.2) confirm that certain parameters of the simulated scenario highly impact the outcome of the analysis produced by the Reassembler. However, finding realistic values for the parameters is challenging, as no real-world reference data sets cover a single DDoS attack from multiple angles. Therefore, additional research is necessary to determine whether the scenario configurations are applicable and realistic in real-world situations.

Another area that requires improvement is the loosened attack fingerprint definition proposed in this thesis. The loosened definition initially addressed the problem that nodes further away from the attack would not recognize an attack because the DDoS Dissector detection threshold of 50% is not reached. For the limited scope of this thesis, the naive solution of considering an attack fingerprint only as a *possible* attack and moving the attack detection to the Reassembler worked effectively in the context of the simulated scenario. However, this comes at the cost of increased storage and data-sharing requirements for each participating network node, which this thesis did not consider. In this regard, further research is needed to evaluate the implications of loosening the definition of an attack fingerprint and to see whether more sophisticated attack detection mechanisms on each network node are computationally feasible.

A last aspect that can be improved concerns the detection of spoofed IPs. The detection of spoofed IPs highly depends on the pool size of available spoofed IPs. Constraining the number of available spoofed IPs is necessary in a small-scale simulated scenario, but it does not reflect the real world. In this regard, the proposed solution presents limited applicability to the real world. However, it highlights the open challenges in this field of research as detecting spoofed IPs is usually a sophisticated process that analyzes various aspects of an IP packet [45]. Due to performance constraints when sharing fingerprints globally, only a minimal amount of data can be stored within an attack fingerprint. Novel approaches for detecting spoofed IPs, considering a global view but limited data, would greatly benefit the work proposed in this thesis.

Several key points can be noted in terms of achieving the thesis goals. The initial goal to cluster fingerprints had to be abandoned because attack fingerprints already clearly identify attack traffic to a certain target. Thus the grouping of fingerprints turned out to be a trivial problem under the assumption that an underlying attack detection of the DDoS Dissector is precise. The proposed Reassembler achieves the adapted goal of

utilizing multiple attack fingerprints to form a global view of a DDoS attack. With the fingerprint generator (*cf.* Chapter 3.1) simulating an SYN attack and the manually crafted attack scenario (*cf.* Chapter 5.1.3) based on real attack data [55], two different use case scenarios have been proposed. Furthermore, adversarial attacks have been addressed by proposing a spoofed IP detection based on aggregated TTL values and by assessing the extent of fingerprints that can be maliciously withheld. A thorough evaluation of the proposed solution has been presented as a final goal, including various experiments that discuss important attack parameters in detail.

5.3.1 Deployment Considerations

In addition to the previously discussed areas for improvement, some deployment considerations should be considered. Like all strategies that demand global collaboration, finding network operators willing to participate in a joint DDoS mitigation and analysis process is challenging. Hence, it is crucial that a monitoring and analysis solution such as the DDoS Dissector can operate without impacting the performance of the network node in general. Especially when using the extended fingerprint format, high-traffic network nodes might run into performance limitations because TTL values and the number of packets per IP are recorded in the fingerprint. This can drastically increase the size of a fingerprint for large amounts of observed IPs, for example, due to IP spoofing. Addressing this issue, the extended fingerprint format has intentionally been added to the DDoS Dissector in an opt-in fashion. Nonetheless, the size and quantity of the resulting fingerprints at each network node are possible limiting factors when operating the full Reassembler workflow.

Chapter 6

Final Considerations

6.1 Summary

Within this thesis, the global analysis of DDoS attacks based on attack fingerprints has been researched. Thereby, related work covering existing fingerprinting and detection methods has been presented, and the research gap in the global analysis of DDoS attacks has been discussed. Additionally, existing datasets and their applicability as an attack use case in the context of this thesis have been discussed.

The general design of a solution for the global analysis of DDoS attacks has been presented. Due to the lack of suitable datasets, a fingerprint generation module was proposed based on simulated network topology. Thereafter, the generated fingerprints, simulating a TCP SYN attack, formed the basis for designing and implementing the Reassembler module. Within the scope of this work, the definition of an attack fingerprint was loosened, and additional attributes have been proposed to the existing fingerprint format, resulting in an extended fingerprint format. As a consequence, the DDoS Dissector software has been improved to support the new fingerprint format and to be compatible with the Reassembler process.

The proposed global DDoS attack analysis solution has been thoroughly evaluated using four simulated attack cases. Additionally, a manually crafted attack scenario based on a real attack dataset was used to present the successful interaction of all newly proposed and adapted components. Attack parameters with interesting observations were further examined using multiple experiments. The results of the evaluation and the experiments have been extensively discussed, highlighting the positive aspects of the solution and discussing further characteristics requiring improvement.

The outcome of this thesis includes two code repositories that contain a working implementation for the proposed changes to the DDoS Dissector [7] and the proposed Reassembler process [8]. All evaluation scripts are included as a package in the Reassembler repository, facilitating further work and more experiments with the simulated architecture.

6.2 Conclusions

In conclusion, the feasibility of a global DDoS attack analysis based on attack fingerprints has been confirmed. Benefits include a more precise analysis of DDoS attacks due to multiple perspectives that can be reassembled to a global view.

In reference to the use cases, several lessons have been learned. Firstly, a realistic dataset containing multiple views of the same attack was not available, and attempts to create a reasonably complex scenario from real data were unsuccessful. The fingerprint generator proposed as a substitute proved beneficial in enabling rapid iterations, particularly in extending and testing the new fingerprint format. However, drawbacks to the simulated scenario could also be observed. One such drawback is the lack of complexity regarding the mix of attack and background traffic. At the same time, it has also been demonstrated that specific scenario parameters can strongly impact the resulting analysis. For instance, the pool of spoofed IPs significantly affected the accuracy of identifying spoofed attack sources, and the amount of background traffic in the network strongly influenced the detection thresholds of participating network nodes. In that regard, the influence of scenario parameters on the resulting analysis was successfully evaluated, but finding scenario parameters that correspond to the real world was challenging due to the lack of suitable datasets.

With respect to the proposed Reassembler, existing methods for detecting spoofed IPs could successfully be applied directly on the fingerprint level using the extended fingerprint format. Furthermore, novel parameters in the context of DDoS attack analyses could be derived, such as the number of intermediate nodes. With the derived distance to the attack target and the observed detection threshold, it has been shown that conclusions can even be drawn about the network topology. On the other hand, the evaluation also revealed that there is still room for improvement regarding the precision of detecting spoofed IPs. The loosened fingerprint definition also impacts the number and size of attack fingerprints.

Eventually, various perspectives were used to examine potential adversarial attacks. It was demonstrated that the Reassembler is not significantly impacted by missing or intentionally withheld fingerprints. Additional adversarial attack vectors were discussed theoretically, revealing that authentication is an important aspect of addressing forged fingerprints and that further research is still required in this area.

6.3 Future Work

The proposed solution in this thesis provides a first basis for analyzing DDoS attacks on a global level using attack fingerprints. Still, various aspects can be improved and would benefit from future research.

The generated fingerprints based on a simulated network topology are limited in complexity. Future work could explore how the mix of attack and background traffic can be made more realistic. Furthermore, considering other attack types besides the TCP SYN

attack could improve the fingerprint generator. With the main benefit of the simulated scenario being the possibility for fast iteration, future work could research what scenario parameters make a simulation realistic and, thus, better applicable to the real world.

The detection of spoofed IPs based on TTL values has been shown to work in a simulated environment with a small pool of spoofed IPs. Future work could explore how this detection can be more precise without artificially constraining the number of possible spoofed IPs. This also includes further research about the cost and benefit of additional parameters in the attack fingerprint. In this regard, it would also be interesting to explore whether different fingerprint formats can be combined into a global attack analysis, thus giving network operators the freedom to choose between smaller and richer fingerprint formats.

Eventually, further research could apply such a process in a real-world scenario, thereby testing whether the benefits of a global DDoS attack analysis in a simulated environment can also be applied in the real world.

Bibliography

[1] 2ip.io. *ASN list*. URL: https://2ip.io/analytics/asn-list/ (visited on Apr. 10, 2023).

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

[3] Aditya Akella et al. "Detecting DDoS attacks on ISP networks". In: *Proceedings of the Workshop on Management and Processing of Data Streams*. 2003, pp. 1–2.

[4] 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.

[5] Manos Antonakakis et al. "Understanding the Mirai Botnet". In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1093–1110.

[6] Agathe Blaise et al. "Botfp: Fingerprints clustering for bot detection". In: *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2020, pp. 1–7.

[7] Jonas Brunner. *DDoS Dissector*. URL: https://github.com/j0nezz/ddos_dissector (visited on June 8, 2023).

[8] Jonas Brunner. *Reassembler*. URL: https://github.com/j0nezz/reassembler (visited on June 8, 2023).

[9] Larry Cashdollar. *Mirai Botnet Abusing Log4j Vulnerability*. 2022. URL: https://www.akamai.com/blog/security/mirai-botnet-abusing-log4j-vulnerability (visited on Jan. 20, 2023).

[10] Center for Internet Security. *The Mirai Botnet - Threats and Mitigations*. URL: https://www.cisecurity.org/insights/blog/the-mirai-botnet-threats-and-mitigations (visited on Jan. 20, 2023).

[11] Bill Cheswick, Hal Burch, and Steve Branigan. "Mapping and visualizing the Internet." In: *USENIX Annual Technical Conference, General Track*. Citeseer. 2000, pp. 1–12.

[12] Benoit Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. Oct. 2004. URL: https://www.rfc-editor.org/info/rfc3954.

[13] Cloudflare Learning Center. *Ping of death DDoS attack*. URL: https://www.cloudflare.com/learning/ddos/ping-of-death-ddos-attack/ (visited on Jan. 20, 2023).

[14] Cloudflare Learning Center. *SYN flood attack*. URL: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/ (visited on Apr. 24, 2023).

[15] Jessica G Conrads. "DDoS Attack fingerprint extraction tool: making a flow-based approach as precise as a packet-based". MA thesis. University of Twente, 2019.

[16] Noah Davids. *Initial TTL values*. URL: http://noahdavids.org/self_published/TTL_values.html (visited on May 31, 2023).

[17] DDoS Clearing House. *DDoS Dissector*. URL: https://github.com/ddos-clearing-house/ddos_dissector (visited on June 7, 2023).

[18] DDoS Clearing House. *DDoSDB*. URL: https://github.com/ddos-clearing-house/ddosdb/ (visited on June 1, 2023).

[19] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numer. Math.* 1.1 (Dec. 1959), pp. 269–271.

[20] David Dittrich. *The DoS Project's 'trinoo' distributed denial of service attack tool*. 1999.

[21] Haitao Du and Shanchieh Jay Yang. "Discovering collaborative cyber attack patterns using social network analysis". In: *Social Computing, Behavioral-Cultural Modeling and Prediction: 4th International Conference, SBP 2011, College Park, MD, USA, March 29-31, 2011. Proceedings 4*. Springer. 2011, pp. 129–136.

[22] Claude Fachkha, Elias Bou-Harb, and Mourad Debbabi. "Fingerprinting internet DNS amplification DDoS activities". In: *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE. 2014, pp. 1–5.

[23] Martin Fowler. *Domain-specific languages*. 2010.

[24] Manaf Gharaibeh and Christos Papadopoulos. "Darpa-2009 intrusion detection dataset report". In: *Tech. Rep.* (2014).

[25] Piyush Goyal and Anurag Goyal. "Comparative study of two most popular packet sniffing tools-Tcpdump and Wireshark". In: *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE. 2017, pp. 77–81.

[26] Guofei Gu et al. "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection". In: *USENIX Security Symposium*. 2008.

[27] Brendon Harris and Ray Hunt. "TCP/IP security threats and attack methods". In: *Computer communications* 22.10 (1999), pp. 885–897.

[28] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362.

[29] Zecheng He, Tianwei Zhang, and Ruby B Lee. "Machine learning based DDoS attack detection from source side in cloud". In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE. 2017, pp. 114–120.

[30] Rick Hofstede et al. "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix". In: *IEEE Communications Surveys & Tutorials* 16.4 (2014), pp. 2037–2064.

[31] KW Hove. "Automated DDoS Attack Fingerprinting by Mimicking the Actions of a Network Operator". B.S. thesis. University of Twente, 2019.

[32] Brian Krebs. *DDoS-for-Hire Service Webstresser Dismantled*. 2018. URL: https://krebsonsecurity.com/2018/04/ddos-for-hire-service-webstresser-dismantled (visited on Jan. 20, 2023).

[33] Brian Krebs. *Powerful New DDoS Method Adds Extortion*. 2018. URL: https://krebsonsecurity.com/2018/03/powerful-new-ddos-method-adds-extortion/ (visited on June 11, 2023).

[34] Brian Krebs. *UK Ad Campaign Seeks to Deter Cybercrime*. 2020. URL: `https://krebsonsecurity.com/2020/05/uk-ad-campaign-seeks-to-deter-cybercrime` (visited on Jan. 20, 2023).

[35] Fu-Yuan Lee and Shiuhpyng Shieh. "Defending against spoofed DDoS attacks with path fingerprint". In: *Computers & Security* 24.7 (2005), pp. 571–586.

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

[37] Tasnuva Mahjabin et al. "A survey of distributed denial-of-service attack, prevention, and mitigation techniques". In: *International Journal of Distributed Sensor Networks* 13.12 (2017).

[38] Gary S. Malkin. *RIP Version 2 Carrying Additional Information*. RFC 1388. Jan. 1993. URL: `https://www.rfc-editor.org/info/rfc1388`.

[39] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56 –61.

[40] Jelena Mirkovic, Gregory Prier, and Peter Reiher. "Attacking DDoS at the Source". In: *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* IEEE. 2002, pp. 312–321.

[41] Jelena Mirkovic and Peter Reiher. "A taxonomy of DDoS attack and DDoS defense mechanisms". In: *ACM SIGCOMM Computer Communication Review* 34.2 (2004), pp. 39–53.

[42] 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.

[43] David Moore et al. "Inferring internet denial-of-service activity". In: *ACM Transactions on Computer Systems (TOCS)* 24.2 (2006), pp. 115–139.

[44] John Moy. *OSPF Version 2*. RFC 2178. July 1997. URL: `https://www.rfc-editor.org/info/rfc2178`.

[45] Masayuki Ohta et al. "Analysis of spoofed IP traffic using time-to-live and identification fields in IP headers". In: *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*. IEEE. 2011, pp. 355–361.

[46] Opeyemi A Osanaiye. "Short Paper: IP spoofing detection for preventing DDoS attack in Cloud Computing". In: *2015 18th International conference on intelligence in next generation networks*. IEEE. 2015, pp. 139–141.

[47] Atilla Özgür and Hamit Erdem. "A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015". In: *PeerJ Preprints* (2016).

[48] Dan Pelleg and Andrew W. Moore. "X-Means: Extending K-Means with Efficient Estimation of the Number of Clusters". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. 2000, pp. 727–734.

[49] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. "Survey of network-based defense mechanisms countering the DoS and DDoS problems". In: *ACM Computing Surveys (CSUR)* 39.1 (2007).

[50] Yakov Rekhter, Susan Hares, and Tony Li. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. Jan. 2006. URL: `https://www.rfc-editor.org/info/rfc4271`.

[51] Martin Roesch. "Snort: Lightweight intrusion detection for networks." In: *Lisa*. Vol. 99. 1. 1999, pp. 229–238.

[52] Stefan Savage et al. “Practical Network Support for IP Traceback”. In: *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2000, pp. 295–306.

[53] Albin Sebastian. *Default Time To Live (TTL) values*. URL: `http://www.binbert.com/blog/2009/12/default-time-to-live-ttl-values/` (visited on May 31, 2023).

[54] Daniel Senie and Paul Ferguson. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. RFC 2827. May 2000. URL: `https://www.rfc-editor.org/info/rfc2827`.

[55] 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.

[56] Richard Sharpe. *editcap - Edit and/or translate the format of capture files*. URL: `https://www.wireshark.org/docs/man-pages/editcap.html` (visited on June 11, 2023).

[57] Snort. *Snort Rules and IDS Software Download*. URL: `https://www.snort.org/downloads/#rule-downloads` (visited on June 6, 2023).

[58] Alissa Starzak. *The latest on attacks, traffic patterns and cyber protection in Ukraine*. 2022. URL: `https://blog.cloudflare.com/ukraine-update/` (visited on Jan. 20, 2023).

[59] Jessica Steinberger et al. “Distributed ddos defense: A collaborative approach at internet scale”. In: *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2020, pp. 1–6.

[60] Minho Sung and Jun Xu. “IP traceback-based intelligent packet filtering: A novel technique for defending against Internet DDoS attacks”. In: *IEEE Transactions on parallel and Distributed Systems* 14.9 (2003), pp. 861–872.

[61] The Tcpdump Group. *TCPDUMP & LIBCAP*. URL: `https://www.tcpdump.org` (visited on June 5, 2023).

[62] Marios Thoma and Christoforos N Hadjicostis. “Detection of collaborative cyber-attacks through correlation and time dependency analysis”. In: *2016 18th Mediterranean Electrotechnical Conference (MELECON)*. IEEE. 2016, pp. 1–6.

[63] Michael C. Toren. *tcptraceroute - Linux man page*. URL: `https://linux.die.net/man/1/tcptraceroute` (visited on May 31, 2023).

[64] Yves Vanaubel et al. “Network fingerprinting: TTL-based router signatures”. In: *Proceedings of the 2013 conference on Internet measurement conference*. 2013, pp. 369–376.

[65] Haining Wang, Cheng Jin, and Kang G Shin. “Defense against spoofed IP traffic using hop-count filtering”. In: *IEEE/ACM Transactions on networking* 15.1 (2007), pp. 40–53.

[66] Joy Weber. “The fundamentals of passive monitoring access”. In: *Net Optics, Inc., Santa Clara, CA, USA* (2006).

[67] Wireshark Foundation. *Wireshark*. URL: `https://www.wireshark.org/` (visited on June 5, 2023).

[68] Chengxu Ye, Kesong Zheng, and Chuyu She. “Application layer DDoS detection using clustering analysis”. In: *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*. IEEE. 2012, pp. 1038–1041.

[69] Omer Yoachimik. *Cloudflare DDoS threat report 2022 Q3*. 2022. URL: `https://blog.cloudflare.com/cloudflare-ddos-threat-report-2022-q3` (visited on June 11, 2023).

[70] Omer Yoachimik. *Cloudflare DDoS threat report for 2022 Q4*. 2023. URL: `https://blog.cloudflare.com/ddos-threat-report-2022-q4` (visited on Jan. 20, 2023).

[71] Omer Yoachimik. *DDoS Attack Trends for 2022 Q1*. 2022. URL: `https://blog.cloudflare.com/ddos-attack-trends-for-2022-q1/` (visited on June 11, 2023).

[72] Shui Yu et al. "Traceback of DDoS attacks using entropy variations". In: *IEEE transactions on parallel and distributed systems* 22.3 (2010), pp. 412–425.

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

[74] Jiachao Zhang et al. "Flddos: Ddos attack detection model based on federated learning". In: *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE. 2021, pp. 635–642.

[75] Jian Zhang and Andrew Moore. "Traffic Trace Artifacts due to Monitoring Via Port Mirroring". In: *2007 Workshop on End-to-End Monitoring Techniques and Services*. 2007, pp. 1–8.

Abbreviations

AS	Autonomous System
BGP	Border Gateway Protocol
CIDR	Classless Inter-Domain Routing
CLI	Command-Line Interface
DDoS	Distributed Denial-of-Service
DNS	Domain Name System
EGP	Exterior Gateway Protocol
GUI	Graphical User Interface
HCF	Hop-Count-Filterig
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IGP	Interior Gateway Protocols
IP	Internet Protocol
IPS	Intrusion Prevention System
OSPF	Open Shortest Path First
PCA	Principal Component Analysis
RIP	Routing Information Protocol
SAV	Source Address Validation
SPAN	Switched Port Analyzer
TAP	Test Access Ports
TCP	Transmission Control Protocol
TTL	Time-to-Live
UDP	User Datagram Protocol
WAF	Web Application Firewall

List of Figures

List of Tables

Listings

Appendix A

Contents of the CD

The following deliverables are submitted for this thesis:

- **Code:** ZIP file containing the source code of the following repositories
  - Reassembler Repository: Contains the source code for the Reassembler, Fingerprint Generator, and evaluation scripts. The source code can also be found on GitHub [8].
  - DDoS Dissector: Contains the extended version of the DDoS Dissector, including the proposed changes in Chapter 4.3. The source code can also be found on GitHub [7].
- **Thesis:**
  - ZIP file containing the source code of the thesis
  - PDF of the thesis
  - Plain text files of the Abstract in English and German

Appendix B

Installation Guidelines

A comprehensive installation guide with usage examples can be found in the `README.md` file of the respective repositories [7, 8].

B.1 Reassembler

1. Clone the Reassembler repository

   ```
   git clone https://github.com/j0nezz/reassembler
   cd reassembler
   ```

2. Create a python virtual environment or conda environment for the Reassembler and install the python requirements

   Venv:
   ```
   python -m venv ./python-venv
   source python-venv/bin/activate
   pip install -r requirements.txt
   ```

   Conda:
   ```
   conda create -n reassembler
   conda activate reassembler
   conda install pip
   pip install -r requirements.txt
   ```

3. Run the Reassembler

   ```
   python main.py
   ```

B.2 DDoS Dissector

1. Clone the extended DDoS Dissector repository

   ```
   git clone https://github.com/j0nezz/ddos_dissector
   cd ddos_dissector
   ```

2. Create a python virtual environment or conda environment for the DDoS Dissector and install the python requirements

   Venv:
   ```
   python -m venv ./python-venv
   source python-venv/bin/activate
   pip install -r requirements.txt
   ```

   Conda:
   ```
   conda create -n dissector
   conda activate dissector
   conda install pip
   pip install -r requirements.txt
   ```

3. Run the DDoS Dissector with the `-e` flag to produce fingerprints of the extended format

   ```
   python src/main.py -f data/attack_traffic.pcap -e -l 178.0.x.x
   ```

Appendix C

Global Fingerprints

C.1 Global Fingerprint of Simulated Scenario


```
{
  "attack": {
    "start_time": "2023-06-10T20:15:35.243738",
    "end_time": "2023-06-10T20:25:18.189005",
    "duration_seconds": 582.9452670000001,
    "service": null,
    "protocol": "TCP"
  },
  "target": {
    "ip": "12.172.45.2",
    "detection_threshold": 1.0
  },
  "intermediate_nodes": {
    "discarded_intermediate_nodes": 6,
    "nr_intermediate_nodes": 40,
    "detection_threshold": {
      "25": 0.6975658707277652,
      "50": 0.8897746820948915,
      "75": 0.9438572376547147
    },
    "key_nodes": {
      "12.160.0.0": {
        "nr_packets": 112689654.0,
        "hops_to_target": 1.0,
        "detection_threshold": 0.9958792178351743,
        "time_start": "2023-06-10T20:15:35.236116",
        "time_end": "2023-06-10T20:25:18.181383",
        "distance": 1,
        "inferred_distance_diff": 0.0,
        "fraction_of_total_attack": 1.0,
        "duration_seconds": 582.945267
      },
      "12.0.0.0": {
        "nr_packets": 112689654.0,
        "hops_to_target": 2.0,
        "detection_threshold": 0.9515403153645143,
```

```
        "time_start": "2023-06-10T20:15:35.185994",
        "time_end": "2023-06-10T20:25:18.131261",
        "distance": 2,
        "inferred_distance_diff": 0.0,
        "fraction_of_total_attack": 1.0,
        "duration_seconds": 582.945267
      },
      ...
    }
  },
  "sources": {
    "nr_sources": 115,
    "pct_spoofed": 0.02608695652173913
  },
  "ground_truth": {
    "nr_attack_av": 732,
    "nr_background_av": 1026,
    "nr_participating_nodes": 109,
    "nr_locations_observing_attack": 36,
    "sources": [
      "12.143.220.186",
      "14.68.148.10",
      ...
    ],
    "target": "12.172.45.2"
  },
  "key": "78898757027057115e3aefb12d836f62"
}
```


Listing C.1: Global Fingerprint based on simulated scenario S2

C.2 Global Fingerprint of Real Attack Data


```
{
  "attack": {
    "start_time": "2018-11-03T15:28:00.776482+00:00",
    "end_time": "2018-11-03T21:35:47.776482+00:00",
    "duration_seconds": 22067.0,
    "service": null,
    "protocol": "TCP"
  },
  "target": {
    "ip": "192.168.50.4",
    "detection_threshold": 0.7249111223621241
  },
  "intermediate_nodes": {
    "discarded_intermediate_nodes": 0,
    "nr_intermediate_nodes": 2,
    "detection_threshold": {
      "25": 0.5277473421668619,
      "50": 0.5503294973955934,
      "75": 0.572911652624325
    },
    "key_nodes": {
      "192.168.0.1": {
        "nr_packets": 2093500,
        "hops_to_target": 1.0,
        "detection_threshold": 0.5954938078530566,
        "time_start": "2018-11-03T15:28:00.776282+00:00",
        "time_end": "2018-11-03T21:35:47.776282+00:00",
        "fraction_of_total_attack": 1.0,
        "duration_seconds": 22067.0
      },
      "192.168.2.1": {
        "nr_packets": 1046531,
        "hops_to_target": 2.0,
        "detection_threshold": 0.5051651869381303,
        "time_start": "2018-11-03T15:28:09.666466+00:00",
        "time_end": "2018-11-03T21:35:31.666466+00:00",
        "fraction_of_total_attack": 0.4998953904943874,
        "duration_seconds": 22042.0
      }
    }
  },
  "sources": {
    "nr_sources": 25,
    "pct_spoofed": 0.4
  },
  "key": "30c8aa201ceff1847db9513a49a8d88f"
}
```


Listing C.2: Global Fingerprint of evaluation scenario based on [55]