University of Zurich[UZH]

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

CH^{2}TF – Collaborative Heavy Hitter Traffic Filtering

Fabian Küffer
Zürich, Switzerland
Student ID: 15-931-421

Supervisor: Dr. Bruno Rodrigues, Katharina Müller, Prof. Dr. Burkhard Stiller
Date of Submission: December 14, 2022

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
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: https://www.csg.uzh.ch/

Abstract

The emergence of DDoS attacks posed a paramount problem in the advent of the internet's growth. Indeed, DDoS attacks occur regularly and lead to major service outages that incur high costs in various dimensions. Thus, this thesis explores the topic of collaboration between parties to enable a *distributed* defense approach. Hence, CH²TF proposes a collaborative signaling protocol to enable heavy hitter (HH) traffic filtering. However, while HH are large traffic flows in the network, their *global* visibility is often unclear. Consequently, this collaborative approach clears up the otherwise opaque visibility of *global* HH. While related work has continuously been addressing DDoS defense techniques, there currently exists a research gap regarding a collaboration effort to identify HH.

A prototype has been designed and implemented to showcase the workings of the signaling protocol. Moreover, the evaluation results of the prototype showed that HH of specific ongoing attack cases (*i.e., volumetric attacks*) are successfully detected in a collaborative manner with sufficiently high accuracy (0.85), though the prototype does not fare that well in other specific attack detection scenarios (*i.e., botnets*, 0.42). Additionally, the evaluation results showed that the analyses to detect attacks and HH are performant and are expected to scale well.

Zusammenfassung

Seit dem Aufkommen des Internets haben sich DDoS Angriffe als ein schwerwiegendes Problem dargestellt. In der Tat, DDoS Angriffe treten regelmässig auf und führen zu Kosten in verschiedenen Dimensionen, durch Serviceunterbrüche und Mitigationskosten. Deshalb untersucht diese Thesis das Thema der Kollaboration um eine *verteilte* Verteidigung zu erzielen. CH^{2}TF schlägt also ein kollaboratives Signalisierungsprotokoll vor um *Heavy Hitter* Netzwerk-Traffic Filtering zu ermöglichen. Folglich wird durch diesen kollaborativen Ansatz Klarheit geschaffen bezüglich der *globalen* Visibilität von Heavy Hitters, die ansonsten intransparent ist. Während die Verteidigungsmittel gegen DDoS Angriffen kontinuierlich erforscht wurden, existiert derzeit eine Forschungslücke im Bereich von kollaborativen Vorgehen um Heavy Hitters zu identifizieren.

Zusätzlich wurde ein Prototyp entworfen und implementiert um das Vorgehen des Signalisierungsprotokolls zu zeigen. Auch haben die Evaluationsergebnisse gezeigt, dass Heavy Hitters während einem laufenden Angriff in gewissen Szenarien (*i.e., volumetrischer Angriff*) erfolgreich, sprich mit mit genügend hoher Präzision (0.85), kollaborativ aufgedeckt werden können. Jedoch hatte der Prototyp Mühe in anderen Szenarien (*i.e., Botnets*, 0.42). Ergänzend zeigten die Evaluationsresultate auch, dass die Analysen um Angriffe und Heavy Hitters zu entdecken performant sind und voraussichtlich gut skalierbar sind.

Acknowledgments

I would like to take this opportunity to express my gratitude to everyone that made this thesis possible. Especially, my supervisors Dr. Bruno Rodrigues, Katharina Müller, and Prof. Dr. Burkhard Stiller who gave me the opportunity to work on this thesis. In particular, I especially express my gratitude towards Dr. Bruno Rodrigues for the support and invaluable inputs, insights, and discussions that enabled this thesis. Of course, a big 'Thank You!' to the Communication Systems Group at the UZH as well, which has always provided me with opportunities to work on my theses and projects and was invaluable to my studies. Also, I would like to thank Pascal Kiechl for his proofreading inputs.

Contents

Chapter 1

Introduction

Since the advent of the internet, Denial-of-Service (DoS) attacks have posed a problem and have been considered by the research community and security professionals [96]. However, the emergence of Distributed Denial-of-Service (DDoS) attacks in 1999 led to this attack vector growing substantially in prominence and remains, despite high efforts by academia and industry, currently unsolved [96, 69]. These attacks lead to a disruption in access to services and unavailability of internet platforms, incurring high costs not only from the disruption of the services but also from its mitigation processes [69, 96]. Hence, various aspects in the realm of DDoS attacks have been researched, *inter alia*, also to approach the mitigation by analyzing and dissecting the incentives and motives behind an attack to prevent them from the start, though research has shown that the objectives and motives of attackers are not homogeneous, and thus, not solvable from this perspective [96]. Furthermore, complicacies in solving this topic arise from a large number of inherently insecure devices connected to the internet and are commonly exploited to conduct DDoS attacks on a large scale [20]. As such, it is clear that DDoS attacks remain a complex topic that has yet to be solved.

Thus, given these paramount challenges, this thesis builds upon the current research and identifies that the distributed nature of DDoS attacks calls for a *distributed* defense strategy that involves various mitigation points, which are possibly closer to the attack's origin. Nevertheless, *distributed* and *collaboration* strategies bring forth additional, multi-dimensional challenges, such as the interplay of technical, legal, economic, and social, that must be carefully considered and evaluated [69].

Besides, heavy hitters (HH) can be seen as large flows of traffic that consume a high amount of network resources [74], thus, their identification and traffic differentiation at a local level would lead to a Quality-of-Service (QoS) improvement. For instance, by handling HH in a special manner, the overall network utilization can be optimized and congestions can be prevented [31]. However, in an ongoing DDoS attack, the *global* visibility of HH is opaque, which this thesis clears up by collaboratively identifying HH.

Thus, this thesis explores the concept of building *heavy hitter tables* based on different dimensions collaboratively to enable the filtering of *HH* based attack traffic. In this sense,

this thesis proposes and implements a communication protocol to enable collaboration between multiple parties on an ongoing attack.

Furthermore, related work has covered adjacent DDoS defense topics such as detection [23], mitigation [53, 54], though the major, cooperation-related literature is covered in [96, 54]. Nevertheless, no existing work has approached the concept of building HH tables *collaboratively*.

The major contributions of CH^{2}TF are threefold: (*i*) a **Collaboration Protocol** to facilitate DDoS defense, (*ii*), the policies to enable the **Detection** of *attacks* and *HH*, and (*iii*), the **Open Source** publication of the developed prototype.

1.1 Thesis Goals

Motivated by the work description that has been outlined, the thesis goals are as follows:

- Design of a signaling protocol to exchange information on HH traffic. Such traffic has to be based on different topics (*e.g.,* regional, national, and international).
- Design of a data-oriented and time-efficient algorithm to calculate global HH in different dimensions and return the information, upon request, to each participating instance.
- A proof-of-concept has to be developed and evaluated to satisfy the proposed protocol and algorithms to validate the proposed concept or collaborative HH.
- In detail, full-fledged documentation of related work evaluations, design decisions, background, design, prototyping, evaluations, and discussions of findings has to be completed.

1.2 Methodology

The methodology of this thesis includes literature research on the state-of-the-art DDoS detection systems, as well as research into how collaboration and cooperation are employed in the realm of DDoS defenses and the discussion thereof. Furthermore, a theoretical foundation in the topics relevant to this thesis (*i.e.,* Pub/Sub, Bloom Filters, DDoS Attacks) is built and researched upon using literature research. Given these foundations, CH^{2}TF is designed and implemented by employing applied research. To evaluate the proposed work, CH^{2}TF is evaluated from multiple dimensions by various experiments.

1.3 Thesis Outline

The structure of the thesis has been divided in the following manner: In this chapter, the topic of the thesis is introduced. Subsequently, in Chapter 2, the relevant background information regarding DDoS attacks and Pub/Sub systems is explored. Following the discussion and comparison of the related work in Chapter 3, the design and implementation of the system are documented in Chapter 4 and 5, which make up the main section of this thesis. In Chapter 6 the results of the system evaluation are presented, and in Chapter 7 a conclusion is drawn, and future work is presented.

Chapter 2

Background

This chapter explores the theoretical background and fundamentals of DDoS attacks, the Pub/Sub pattern and gives an introduction to Bloom filters and heavy hitters (HH). Thus, Section 2.1 talks about DDoS attacks and gives a short classification of DDoS attacks and their mitigation techniques. Next, HH are introduced in Section 2.2. Further, various Pub/Sub frameworks are presented and discussed in Section 2.3. Lastly, Bloom filters are explored in Section 2.4.

2.1 Distributed Denial of Service Attacks

Essentially, Denial of Service (DoS) attacks attempt to prevent legitimate users to access and use specific web services, resources, or systems [96, 75]. Although attacks of this kind do not lead to a compromise of user data, the resulting outages can lead to loss of user trust [37]. While DoS attacks already appeared in the early 1980s, it was not until 1999 that the first *Distributed* Denial of Service (DDoS) attack occurred [96, 75]. Since this incident, most DoS attack occurrences can be classified as distributed [96].

Furthermore, DDoS attacks have also grown substantially in attack volume. Figure 2.1 shows the largest known DDoS attacks and trends in attack volumes [37]. However, this figure's data only includes data collected from Google Cloud and other (undisclosed) sources [37]. While the exponential trend appears alarming, [37] notes that the internet itself is also growing exponentially, making the result look more problematic than it actually is.

Generally, DDoS attacks can be classified based on various criteria, *inter alia*, the attacker's motivation, leading to the following enumeration [96]:

- **Financial/ Economical Gain:** Attacks that are motivated by financial or economical gain are a significant predicament for corporations. They are very technically affine and therefore also hard to defend against.

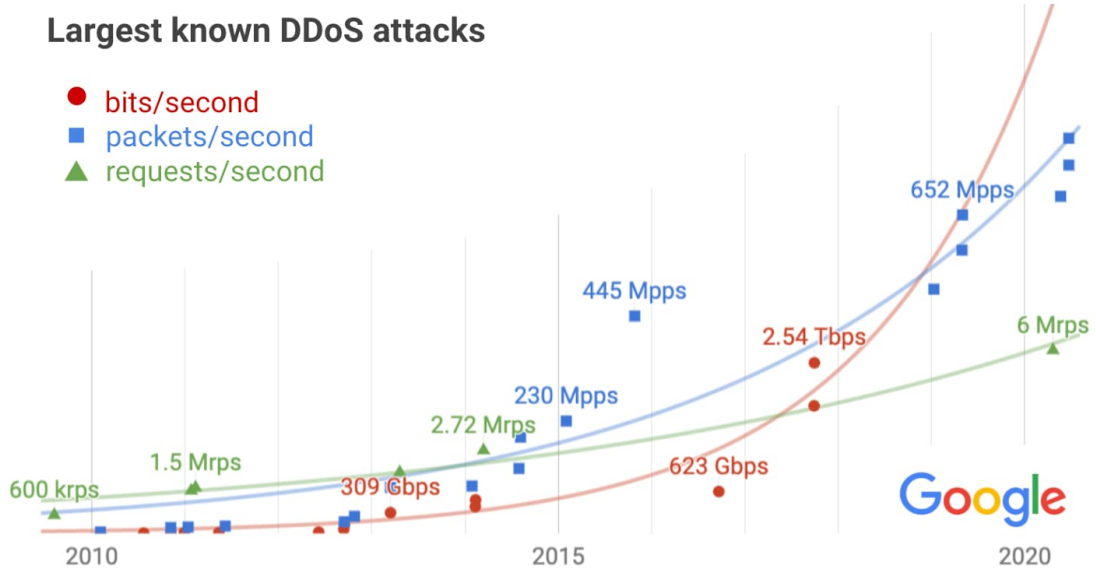


Figure 2.1: Largest known DDoS attacks and trends in attack volumes [37]

- **Revenge:** Attacks from this category are motivated by the feeling of injustice, leading to revenge attacks.
- **Ideological Belief:** This is one of the most prevalent incentive, with politically motivated attacks in Estonia (2007), Iran (2009), and WikiLeaks (2010) falling under this category.
- **Intellectual Challenge:** This category includes attacks that are motivated by the attacker's inclination of wanting to learn and extend their understanding of attacks, or knowledgeable attackers that want to show their capabilities.
- **Cyberwarfare:** These attacks stem from organizations belonging to a country and target other countries, for political reasons. Targets of these attacks can include critical infrastructures and services and may have an enormous impact.

A more *technical* classification of attacks is introduced in Section 2.1.1.

2.1.1 DDoS Attacks & Attack Classification

Fundamentally, based on the targeted protocol level, DDoS attacks can be distinguished between **Network/transport-level DDoS attacks** and **Application-level DDoS attacks** [96]. In the case of network/transport-level DDoS attacks, usually, TCP, UDP, ICMP, and DNS protocol packets are exploited to conduct the attack, while application-level attacks target exploits HTTP, DNS, or Session Initiation Protocol (SIP) [96]. This results in application-level attacks being very similar to legitimate traffic, while also consuming less bandwidth [96].

Reflection-Based Attacks

In reflection-based attacks the attacker spoofs the target's IP address and contacts legitimate servers with a request, *e.g.,* DNS requests [40, 49]. The response to the request is in turn sent to the target, since the server in the middle believes the origin of the spoofed IP address is the target [49]. In the case of network/transport-level DDoS attacks, the attackers often send requests such as ICMP echo requests to the reflectors, resulting in a denial of service [96].

Amplification-Based Attacks

This type of attack employs services for traffic amplification, such that each message sent, results in either a generation of large or multiple messages, which in turn are targeted towards the victim [96]. Hence, this results in an *amplification* of traffic [96]. Furthermore, amplification attacks are usually combined with **Reflection Attacks** [96] to hide the origin of the attack while amplifying the attack [49]. An instance of this attack in the category of network/transport-level attacks was the Smurf attack, which used ICMP echo requests, and spoofing. It exploited IP broadcasting to generate high traffic from all devices inside the network [14, 96]. However, the Smurf exploit can be considered as "solved" nowadays [14].

Botnet-Based Attacks

This attack type differs from the other types that were presented, since it can be used as a *mechanism* to enable and conduct DDoS attacks [96]. Furthermore, [96] notes that using botnets to facilitate DDoS attacks is very prominently used, also for the precarious DDoS cases. By employing a botnet, not only can the DDoS attack scale in size extremely well, but also appear opaque, by using spoofing techniques [96].

Figure 2.2 shows the structure of such a botnet-based attack. The attackers use so called *handlers* to communicate *indirectly* with the bots to conduct an attack on a DDoS victim [96]. The handlers themselves can be programs that run on compromised devices, for instance, network servers [96]. The bots, also known as 'zombies', are devices that were compromised, using viruses such as worms or Trojan horses or by exploiting backdoors, by the handlers [96]. The bots themselves are then the DDoS attackers [96]. Additionally, the bots can be controlled by their masters using *IRC-based*, *Web-based*, and *P2P-based* techniques [96].

Furthermore, the rise in popularity of Internet of Things (IoT) devices, which are prominently *insecure*, lead to botnet-based attack to be highly viable [20]. For instance, the Mirai botnet, which conducted a highly effective DDoS attack against Dyn DNS infrastructure, controlled 100′000 devices in 2016, where many of those devices were in fact CCTV cameras [20]. Additionally, botnets can include techniques to detect and evade their capture by erasing their data to offer resiliency [96].

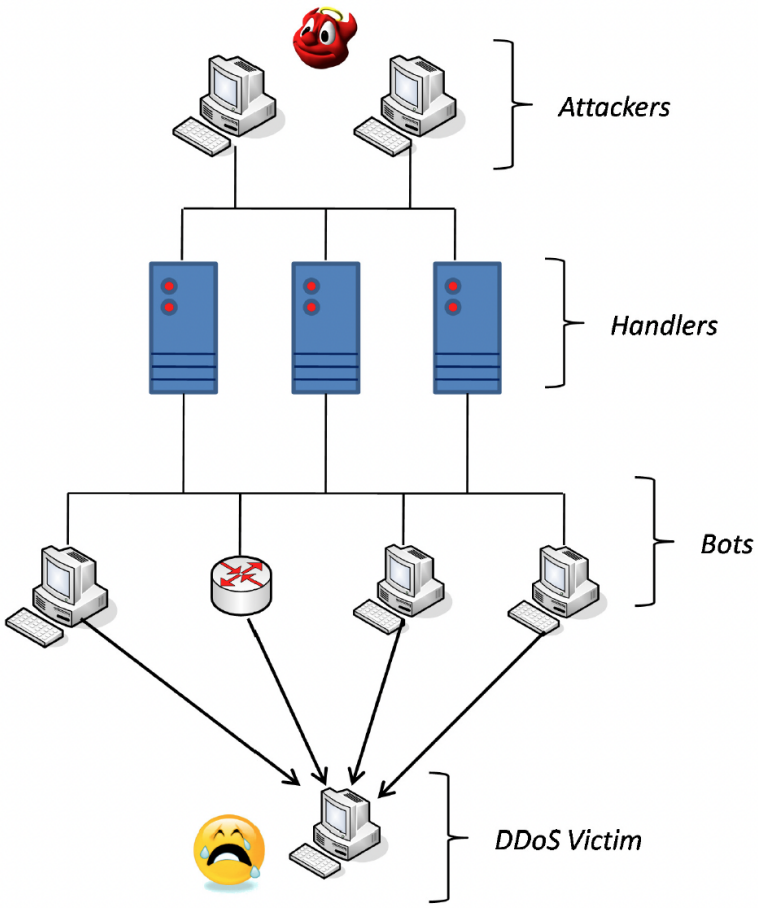


Figure 2.2: Botnet-Based DDoS Attack [96]

2.1.2 DDoS Defense & Attack Mitigation

Regarding the DDoS attack detection, [8] note that separating *legitimate* and *illegitimate* traffic is not trivial, since attacks can mimic the behavior of *legitimate* traffic to avoid detection. In general, a successful and effective mechanism for DDoS prevention and mitigation should fulfill the following properties and requirements as outlined by [8]:

- Legitimate users of a service should not be impeded by the use of a protection mechanism.
- Attacks from within, but also from the outside of the network should be preventable by the use of a protection mechanism.
- Mechanisms must be performant and scalable, according to modern data centers, and also offer robustness, adaptiveness, and flexibility.
- DDoS mitigation should be offered by the mechanism.
- The mechanism should avoid false-positives and have high detection rates.

In this context, a mechanism can be seen as a DDoS attack protection solution [8].

Attack Detection

Common techniques to detect DDoS attacks (using SDNs) had been reviewed by [8]. A selection of such mechanisms is presented here:

- **Entropy:** This technique uses probability distributions to calculate the entropy. The entropy is then compared against predefined threshold values to detect an attack. Features such as source IP address, destination IP address, or port numbers are used to compute the entropy.
- **Machine Learning:** This technique is about detecting anomalies using ML. This has also been employed to detect IoT-based DDoS attacks (*cf.* Section 3.10). ML techniques for anomaly detection include Bayesian networks, self-organizing maps (SOM), and fuzzy logic. Using such techniques, the traffic flow can be classified into *legitimate* and *illegitimate*. However, the performance of such techniques depends on the training sets that were used.
- **Traffic Pattern Analysis:** This technique is about analyzing the behavioural traffic patterns to differentiate between *legitimate* and *illegitimate* traffic. For instance, bots in a botnet behave differently to benign hosts, and this fact can be used to detect an attack.
- **Connection Rate:** This term includes multiple techniques. For instance, the 'connection success ratio' can be used to detect an attack. The rationale is, that a benign host would have a much higher number of successful connection attempts than a malicious host. Using a threshold value for this ratio can be used to detect malicious hosts.

However, [8] notes that the aforementioned techniques employ *thresholds* in some sort to detect an attack. Such thresholds act as a baseline for what constitutes as *illegitimate* traffic [8]. Further attack detection literature is presented in Chapter 3.

Attack Signaling

Attack signaling is about the *real-time* signaling of DDoS attack information and telemetry [28]. This has been proposed by [28] in a standardization attempt (*cf.* Section 3.13). Using a signaling protocol enables the communication between an attack target and the mitigators [28]. Thus, this allows the mitigation of an ongoing DDoS attack [28, 69]. Existing work also employs Blockchains (BC) to enable collaborative signaling (*cf.* Section 3.9). The topic of signaling is further discussed in Sections 3.13 and 3.9.

Attack Mitigation

Attack or threat mitigation plays a role once an ongoing attack has been detected, and allows to protect services such that they can continue their operation [8]. Common attack mitigation techniques that can be employed in a *centralized* SDN controller have been reviewed by [8], and a selection is presented here:

- **Drop packets:** Using rules, attacks are mitigated by dropping the packets.
- **Block port:** Traffic is blocked according to the attacking port.
- **Deep Packet Inspection:** Deep packet inspection looks at the header and data of packets. This allows for the detection of an attack, and its mitigation.
- **Redirection:** To achieve mitigation, traffic is differentiated and the *benign* traffic is *redirected* to a new IP address.
- **Traffic isolation:** The *illegitimate* traffic is isolated (*i.e., quarantined*).
- **Control bandwith:** This technique limits transmission flow rates.

However, these mitigation techniques also have issues. For instance, the fast and simple solutions (*i*) dropping packets, or (*ii*) blocking ports could, in practice not only stop an attack, but also *legitimate* traffic [8].

2.2 Heavy Hitters

Heavy hitters (HH) can be defined as traffic flows with high volumes, *i.e.,* a small proportion of the overall network flows is responsible for most network traffic [31]. As such, HH can also be seen as *elephant flows* or *mega flows* [31, 51]. This concept is not particular to DDoS attacks, but can be seen as a more general topic from network sciences and is relevant to applications such as DoS detection, traffic engineering, network anomaly or load balancing [74, 31].

The identification of such HH is paramount to enable a high QoS and load-balancing in the network [31]. Depending on the application, the identification of HH enables a special treatment, for instance, they allow for *priority queuing* or *packet dropping* [51]. Traditionally, the detection of HH is done using sampling metrics of network traffic using set intervals [31].

2.3 Publisher-Subscriber Pattern

The Publish-Subscribe (Pub/Sub) paradigm is a messaging pattern that allows publishers to send messages, and subscribers to receive messages according to topics that they

subscribe to [88]. This pattern is closely related to the "Observer Design Pattern" (*cf.* [85]), and it is designed to solve the problem of tight coupling between subjects and observers that can create various issues, such as scalability or flexibility [85, 38]. This is accomplished by introducing **Message Brokers**, allowing the asynchronous communication between services that do not need to be aware of each other, thus, publishers and subscribers are not coupled together, and the services can be operated independently from each other [27, 38]. Therefore, a dynamic and flexible network topology can be realized [88], and integrations and interactions between systems, applications, or platforms become simpler, whilst achieving a separation of concerns [38]. Additionally, this pattern offers the great advantage of high scalability via parallel operations, message caching, or tree-based/network-based routing [88].

However, this pattern comes with the disadvantage that the structure of the published data must be known, thus, there is less flexibility in *modifications* to the structure of the published data [88]. Furthermore, this pattern is *unidirectional* from the publishers to the subscribers, *i.e.,* implying that subscribers cannot communicate back to the publishers or give acknowledgements via the channels of this pattern [38].

The architecture of the Pub/Sub pattern includes three main components (*cf.* Figure 2.3):

- **Producers/Publishers:** Are senders of messages [38]. They are not aware of specific subscribers, or the potential lack thereof [88], though the format of the messages is known [38]. The messages are sent via the input channel to the message broker [38].
- **Message Broker:** Is an intermediary that is responsible for transferring the messages that arrive through the input channel to the subscribers, via the output channel, that have interest in this message [38]. Thus, the broker filters messages and forwards them to the right subscribers, and is also able to queue them based on priorities before routing [88].
- **Consumers/Subscribers:** The subscribers subscribe to a topic of interest [88], and for each consumer of messages there exists an output channel from the broker [38]. Similarly, subscribers are not aware of specific publishers, or the lack thereof [88].

The Pub/Sub pattern can be achieved with a single message broker (*cf.* Figure 2.3) or with multiple brokers, in a distributed fashion (*cf.* Figure 2.4).

2.3.1 Pub/Sub Frameworks

In this section a selection of Pub/Sub frameworks are presented and discussed. They have been selected due to their popularity.

Apache Kafka

Apache Kafka is a highly popular open-source distributed event streaming platform, with over 22′000 stars in Github [3, 4]. The platform supports the publishing and subscription

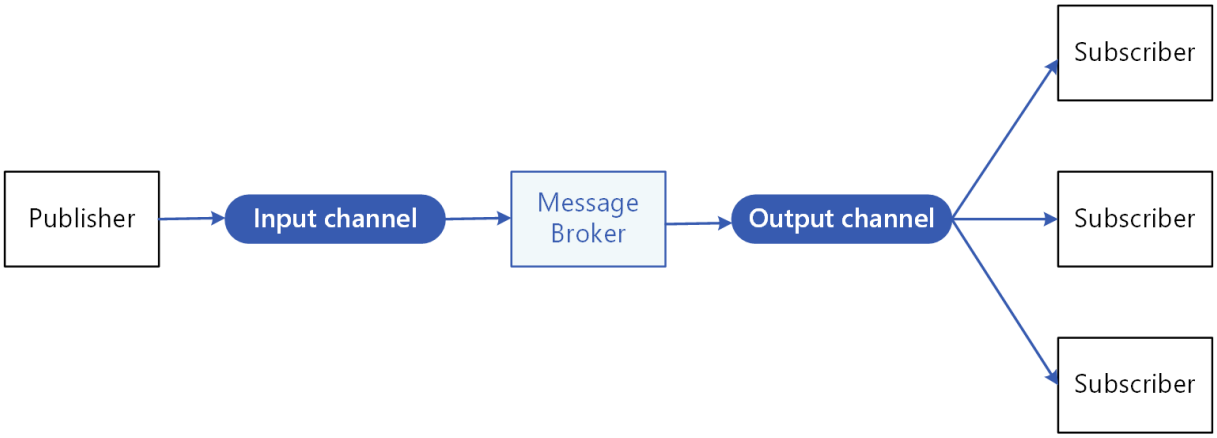


Figure 2.3: Components of the Pub/Sub Pattern [38]

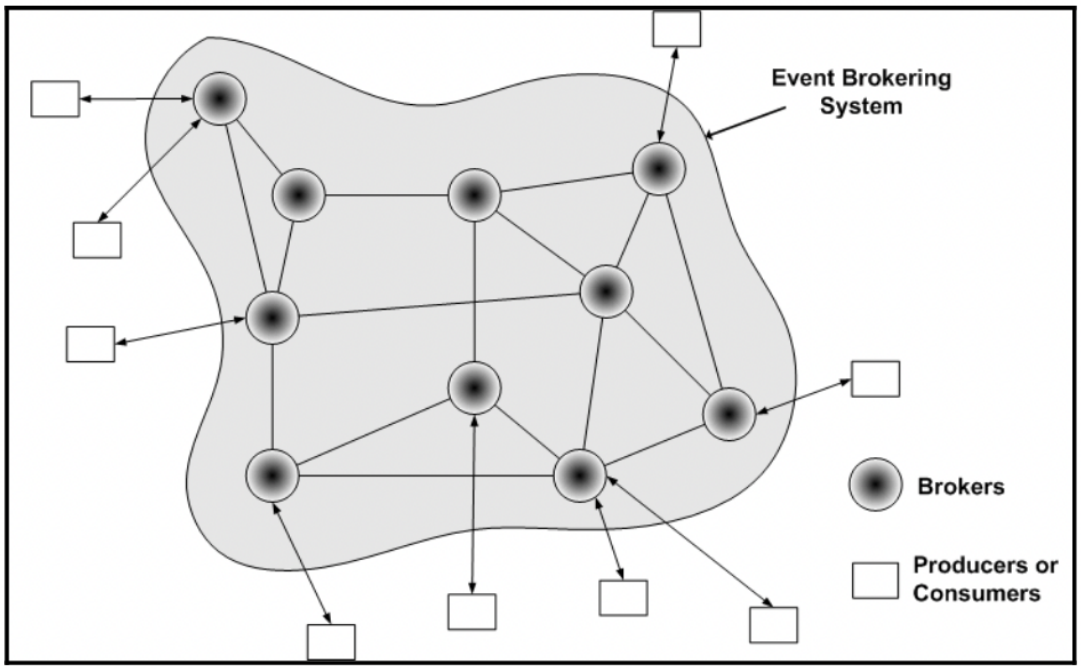


Figure 2.4: Distributed Pub/Sub Architecture [77]

of events, and also their storage [4]. An example of Apache Kafka deployed in a cluster is displayed in Figure 2.5, where its architecture relies on a Apache ZooKeeper (*cf.* [5]) cluster, a centralized service to provide synchronization for distributed applications, to store metadata and configuration of topics and locations of partitions [5, 36]. However, since the ZooKeeper dependency limits the scalability of Kafka, future releases will remove this dependency [4, 36].

Kafka can be deployed in a Kubernetes cluster (*cf.* [34]) [30] or Docker (*cf.* [18]) [7], and clients supporting many programming languages such as Java, Scala, C/C++ or Python are available [4].

Conceptually, in Kafka the smallest unit of data is a message, and each message belongs to a specific topic [72]. Additionally, Kafka uses the concept of partitioning the topics,

meaning that each topic has multiple partitions that can be distributed over servers to achieve high scalability, and the partitions themselves can be replicated as well, to handle potential failures [72].

So that consumers can keep track of the messages, each message includes a continuously increasing number, which is unique to a partition, called the offset, which results in consumers being able to begin and stop reading from a specific message onwards [72].

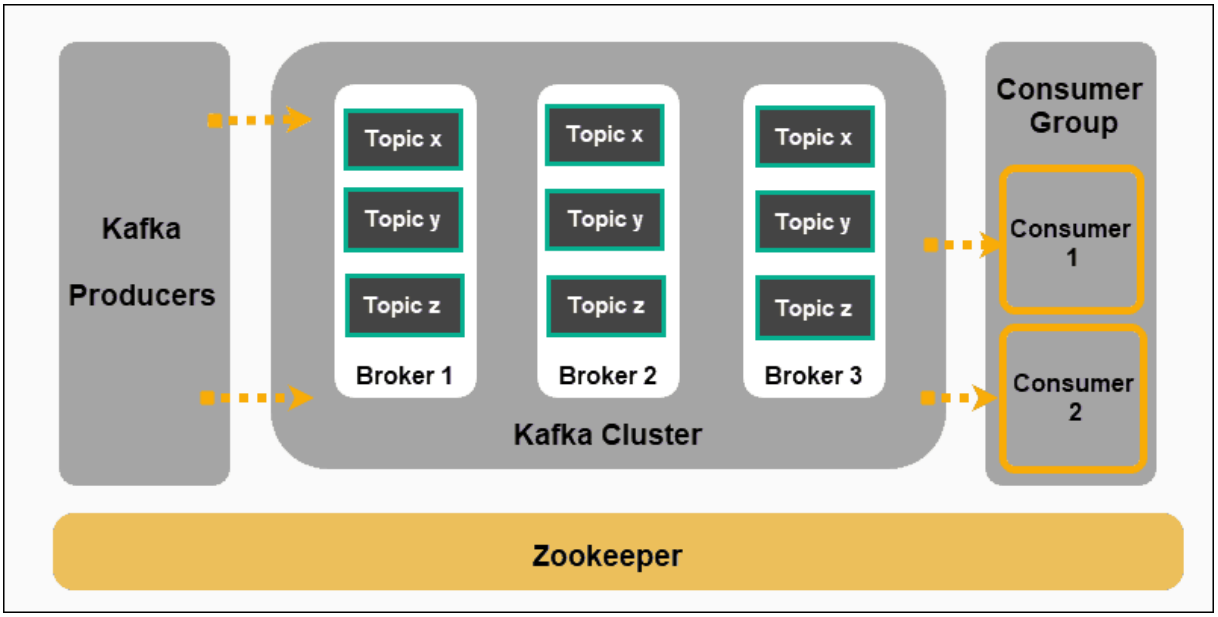


Figure 2.5: Apache Kafka using a Kubernetes Cluster [30]

Emitter

Emitter is an open source distributed Pub/Sub framework that supports more than 3′000′000 messages per second for a single broker [21]. Possible configurations that are supported by Emitter are shown in Figure 2.6. This framework supports various clients in a multitude of languages, such as JavaScript, TypeScript, Python, C# or Java [22]. For convenience, Emitter supports Docker (*cf.* [18]) and Kubernetes (*cf.* [34]) deployments [21]. At the time of writing, this project's latest release was in December 2021, and the project has 3300 stars in Github [21].

2.4 Bloom Filter

Bloom Filters were introduced in 1970 by Burton Howard Bloom, and are a probabilistic data structure [24, 91]. Their main use is the test of set membership, *i.e.,* whether a member is part of a set or not [91, 24, 70]. However, due to their probabilistic nature, the result of a query indicates that a member is possibly in a set or not part of the set [91, 24, 70]. These query results imply that no false negatives are possible, though false positive matches can occur, *i.e.,* the query result *possibly in a set* can be wrong [91].

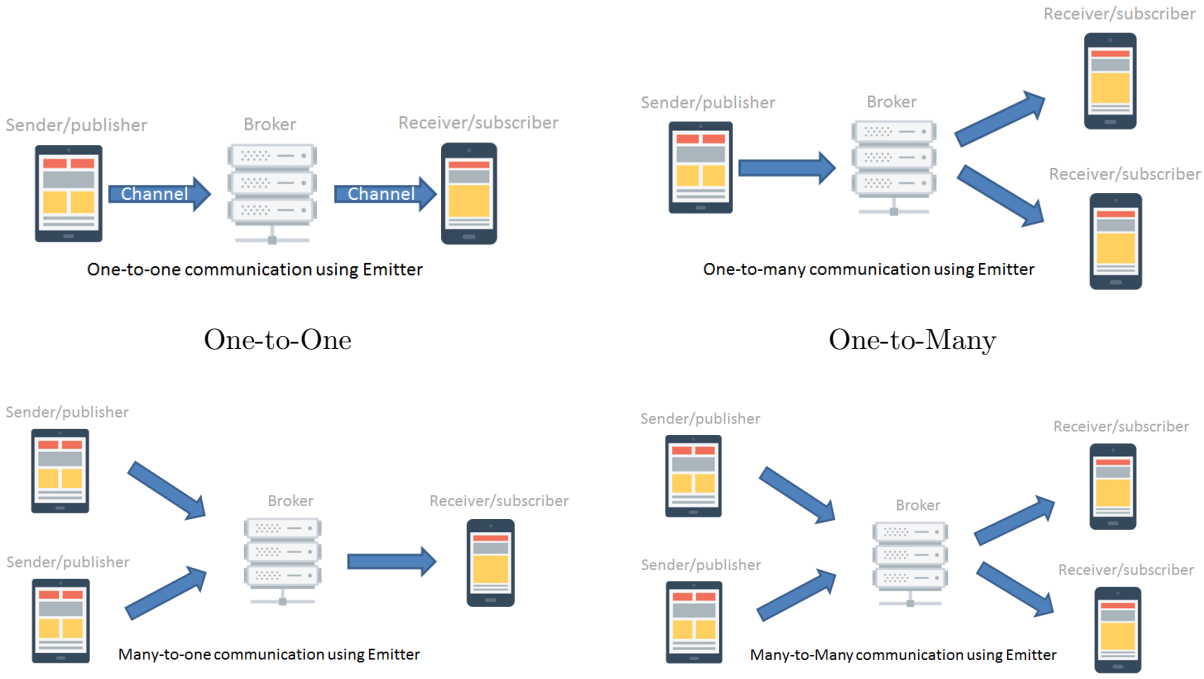


Figure 2.6: Possible Emitter Communication Configurations [22]

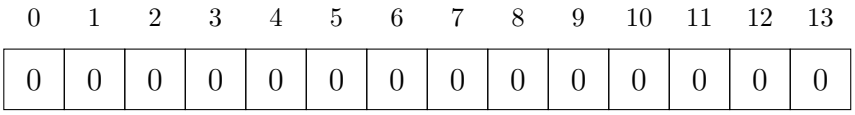
However, the false positives occur with low probability [70, 24], though a high number of items in a set leads to a higher probability of false positives [91]. In particular, it has been shown that the probability of false positives fp is:

$$fp = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k$$

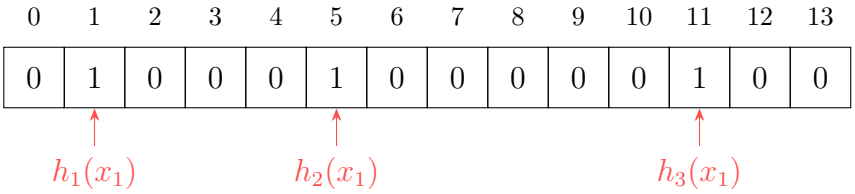
where k stands for the number of independent hash functions that are used, n the amount of added elements, and m the bits in the array that is used as a bloom filter [24, 91].

While at first glance this data structure might appear contrived, it is highly space-efficient due to not requiring storing the elements themselves and features a constant lookup time of $\mathcal{O}(k)$, with k denoting the number of distinct hash functions used, to check whether an element is in the set, which are both highly valuable properties for many applications [24, 91].

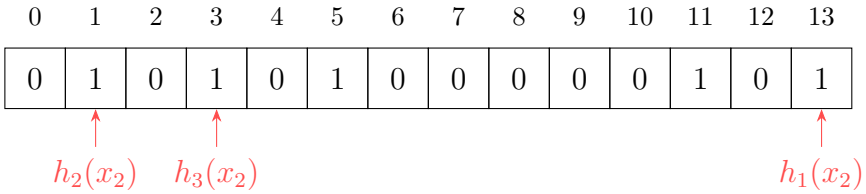
In Figure 2.7 an example of a bloom filter and its operations is given. The bloom filter vector is initially empty, *i.e.,* all bits are set to 0. The elements x_1, x_2, x_3 are each hashed by $k = 3$ hash functions h, and inserted into the vector A. Subsequently, the elements y_1, y_2 are queried.



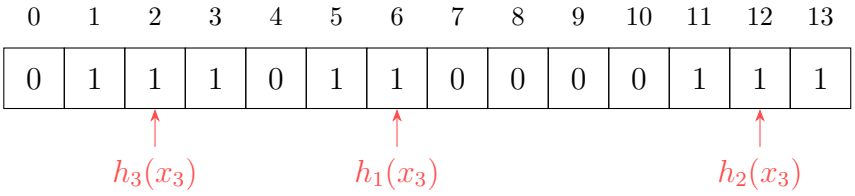
(a) Empty bloom filter example with a vector A using $m = 14$ bits.



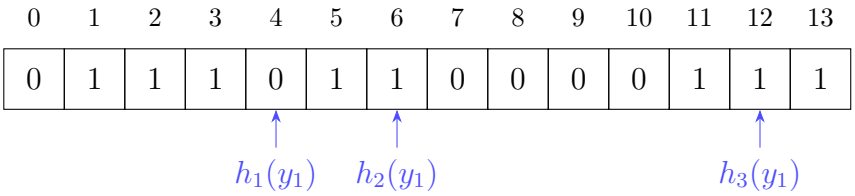
(b) Inserting the element x_1 using 3 hash functions h_i



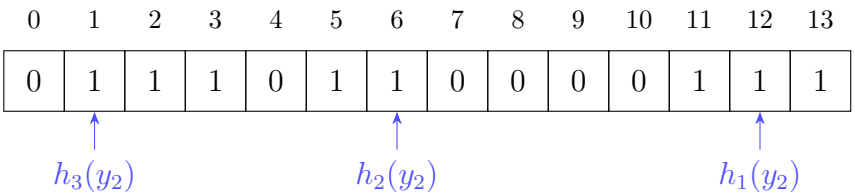
(c) Inserting the element x_2 using 3 hash functions h_i. In this example, $A[1]$ has a collision since $h_2(x_2)$ and $h_1(x_1)$ both map to $A[1]$.



(d) Inserting the element x_3 using 3 hash functions h_i



(e) Querying the element y_1 using 3 hash functions h_i. Since the query $\forall i \in \{1, \ldots, k\} : A[h_i(y_1)] = 1$ does not hold, y_1 is definitely not in the set.



(f) Querying the element y_2 using 3 hash functions h_i. Since $\forall i \in \{1, \ldots, k\} : A[h_i(y_2)] = 1$ holds, y_2 is probably in the set. However, this is a false positive here since no inserted element x had the configuration ($A[1]$, $A[6]$, $A[12]$).

Figure 2.7: Example of a bloom filter and its operations. Source: The Author, based on an example by [24].

Chapter 3

Related Work

Given the timeliness and importance of DDoS attacks (*cf.* Section 2.1), it is no surprise that many existing solutions and proposals exist. However, the solutions to how detection or mitigation are approached are very diverse, though in some cases also include overlaps. Thus, in this chapter, related projects and systems in the realm of DDoS defenses are presented and subsequently discussed and compared.

3.1 MULTOPS

MUlti-Level Tree for Online Packet Statistics (MULTOPS) is a data structure that was proposed by [25]. The core idea of the work is to detect ongoing attacks using heuristics, *i.e.,* by comparing the proportionality of the packet flow direction [25]. Thus, this operates under the assumption that the normal traffic flow behavior is symmetric, *i.e.,* there exists a proportionality in the flow from one direction to the other direction, implying that a disproportional flow is seen as malicious [25]. However, the authors note that under this assumption, MULTOPS fails to detect attacks, *e.g.,* attacks based on HTTP or FTP connections, that make use of proportional flows [25].

To cover the IPv4 address space, its data structure has been designed as a 4-level 256-ary tree, where each node consists of 256 records [25]. Further, each record includes the *from-rate*, *to-rate* and a *pointer* to the next tree level [25]. Under this scheme, the longer IP prefixes are contained in the deeper levels of the tree (*cf.* Figure 3.1), thus, the aggregate rates for a given IP address or prefix are easily read [25]. Consequentially, attacks are detected at the high levels of the tree [25]. Furthermore, the scheme includes *expansion* and *contraction* of the tree, to *(a)* create new nodes and *(b)* to avoid running out of memory [25].

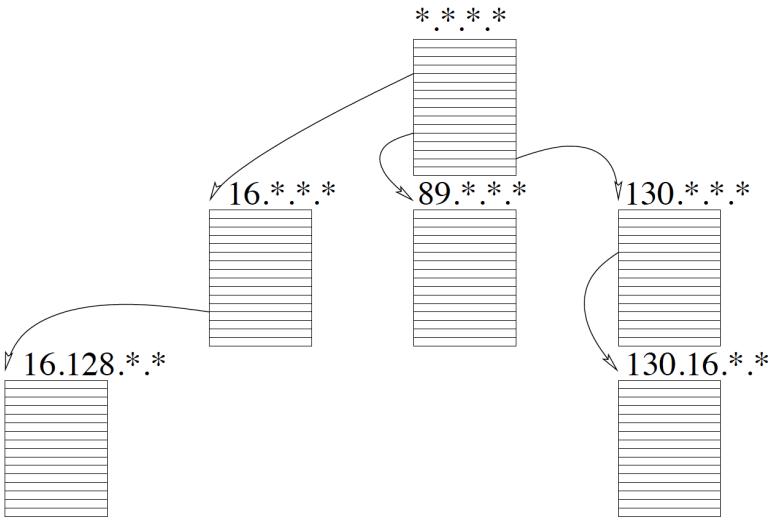


Figure 3.1: MULTOPS [25]

3.2 D-WARD

The work by [39] tries to tackle DDoS attacks at the source, *i.e.,* to prevent an attack from being put into practice. Hence, outgoing traffic from a router is monitored and compared to normal traffic models, and traffic that is discerned as *non-complying* gets dynamically rate-limited [39].

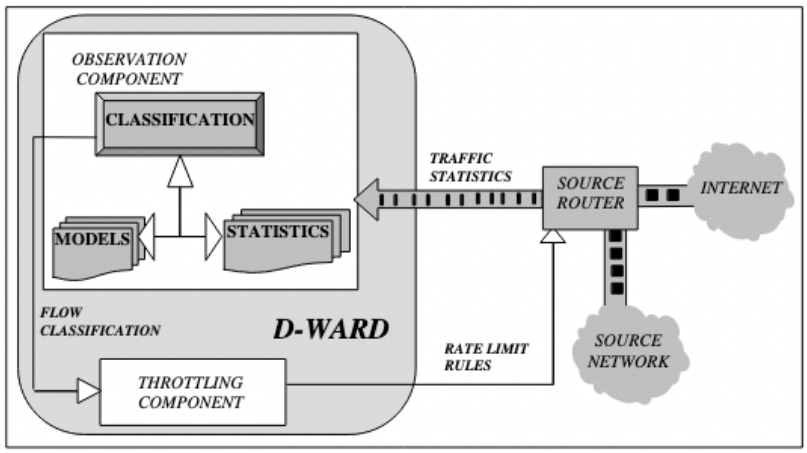


Figure 3.2: D-WARD [39]

In Figure 3.2 the architecture of D-WARD is displayed. Here, the D-WARD component is not part of the router but instead communicates with the router [39]. The D-WARD component itself consists of two components [39]:

- **Observation Component:** This module is responsible for monitoring the traffic that passes through the router and using models, traffic statistics are computed to compare against a baseline of traffic that is deemed as normal. Subsequently, a classifier is run to check whether the flow is an attack flow.

- **Throttling Component:** This component receives the statistics of the observation component and is in charge for upholding the traffic limits of the router and adjusting them accordingly.

Due to its deployment at the router, the authors note that this work lacks an incentive program, since the perceived benefits are targeted at a potential attack victim, and a deployer does not personally see any benefit *per se* [39]. However, by being at *the source*, this work has the advantage that the traceback of the attack is easier accomplished, and the traffic is stopped before it is mixed with other traffic flows, which could possibly obfuscate the source [39].

3.3 AITF

Active Internet Traffic Filtering (AITF) is a filtering protocol designed to mitigate DDoS attacks by filtering traffic [6]. This work uses a route record scheme, meaning that a collaborating router adds to each packet that it forwards its own IP address to the header (*cf.* Figure 3.3) [6]. Additionally, using hashes of the packet destination of each router as random values that are added to the route leads to spoofing protection, if no malicious nodes are present in the path [6].

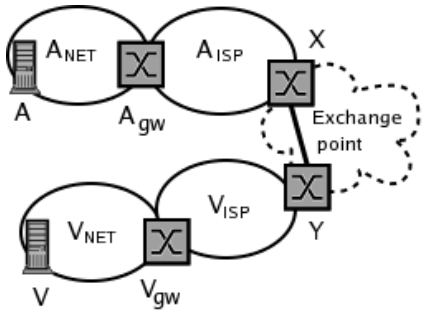


Figure 3.3: Recorded route $\{A_{gw}\ X\ Y\ V_{gw}\}$ between the hosts A and V. In this example A_{gw} is the attacker's gateway, *i.e.,* it is the router closest to the attacker A, and V is the victim [6]

In the case that illegitimate traffic is detected, the victim temporarily blocks the traffic and sends a request to the router that is the closest to the attacker [6]. Subsequently, in the good case, an agreement between the routers is reached to not forward specific packets, and the filter is removed [6]. On the other hand, if no agreement is reached, the work also includes escalation to the next closest router [6].

Their evaluation showed that one million illegitimate traffic flows could effectively be filtered, by using only 10′000 filters for each participating router using AITF, which the authors deem as a small amount [6].

3.4 DefCOM

[52] propose DefCOM, a collaborative and distributed Peer-to-Peer (P2P) DDoS defense system. Although a P2P overlay network is used, only DefCOM control messages are exchanged over this network, hence, data packets use the underlying IP network [52]. Due to each node storing a limited amount of state, this solution is designed to scale with the number of peers [52]. This work adds to existing router or defense nodes one, or multiple of the functionalities offered by DefCOM [52]:

- **Classifier:** This functionality allows nodes to differentiate between legitimate and attack traffic and marks legitimate traffic as such.
- **Rate Limiter:** This functionality algorithmically limits the traffic that is sent to a victim. This functionality must be offered by all router nodes in the DefCOM network, to ensure that the rate limit is maintained.
- **Alert Generator:** Using the overlay network, nodes with the alert generator functionality are responsible for detecting an attack and propagating the attack alert to other nodes. In comparison to the other nodes, these nodes are always active, since they require monitoring the traffic to detect attacks. However, this functionality itself is not able to detect an attack, the node must provide this functionality through another service.

In the case when an attack is detected, the alert generator floods the overlay network with alarm messages notifying all nodes [52]. Subsequently, the rate limiter and classifier nodes become active and start limiting the traffic and classifying between legitimate and attack traffic [52].

This work uses certificates, either issued manually or by a global certification authority, to prevent malicious nodes from joining DefCOM, however, the authors note that various attacks from existing nodes that are maliciously acting are possible [52].

3.5 Defense and Offense Wall (DOW)

[95] propose DOW to defend against Application layer DDoS attacks, that they categorize into *(i)* session flooding attacks, *(ii)* request flooding attacks and *(iii)* asymmetric attacks. The authors note that application layer attacks behave differently than network layer attacks, *i.e.,* they are more subtle and appear and behave as normal traffic, although employing the asymmetric computation differences between the client and server to conduct an attack [95]. Further, sites such as e-government or e-commerce are primary targets for application layer attacks, and [95] give as an example the scenario where a request requires computationally expensive queries on the server side, which can then in turn be exploited for an attack.

Their work contribution is two-fold, it uses (*i*) an anomaly detection model and (*ii*), an encouragement model [95]. The anomaly-detection is achieved using K-means clustering,

and suspicious sessions are dropped [95]. On the other hand, the encouragement model encourages legitimate sessions using a currency model, *i.e.,* the client's session rate [96, 95]. Consequentially, the authors note that the combination of both models leads to legitimate sessions achieving a higher service rate and lower delays [95, 96].

3.6 Zhang et al. (2008)

In [97] the authors propose a predictive blocklisting system to block attackers *individually* to each contributor by making use of a link analysis strategy that is similar to PageRank (*cf.* [92]). This architecture is built upon DShield (*cf.* [2]), a system that allows users to share their firewall logs automatically with DShield, which compiles and publishes a daily blocklist based on the reportings [97]. The architecture is depicted in Figure 3.4 and includes the following steps [97]:

- **Prefiltering**: The DShield logs are pre-processed to achieve a higher quality of data by removing and reducing noise and erroneous data [97].
- **Relevance Ranking and Severity Assessment**: These systems work in parallel: The *severity assessment* computes the maliciousness of the sources to generate a score. Additionally, the *relevance ranking* calculates the closeness between an attacker and a log contributor since contributors might share attackers and these correlations can be observed [97].
- **Blocklist Production**: This step combines the *relevance ranking* and the *severity assessment* to compile a blocklist that is individual to each contributor of logs [97].

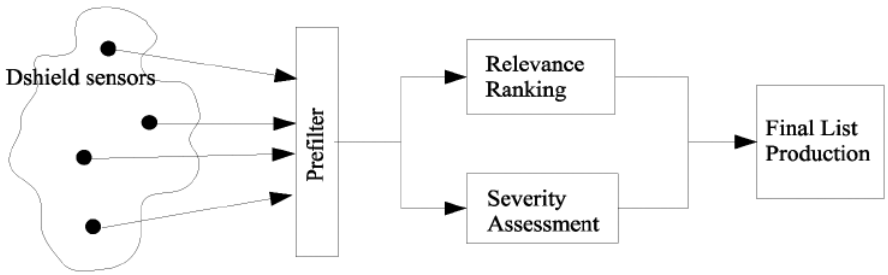


Figure 3.4: Architecture of the blocklisting system [97]

Their evaluation results show that with their approach they can compile blocklists with higher attacker hit counts and better predictions of attackers can be computed [97]. However, the authors also note that due to the nature of the volunteer-contributed data, *i.e.,* the logs, it is also possible for malicious contributors to poison the data to gain the system [97].

3.7 Velauthapillai et al. (2010)

[81] propose a distributed DDoS detection solution using cooperative overlay networks and a gossip-based information exchange protocol to detect DDoS attacks globally. To achieve this goal, this work adds to the participating routers a defense node that is part of the overlay network [81]. The authors note that traditional centralized client-server model approaches suffer from potential performance bottlenecks and are a single point of failure, thus, they reason for designing a distributed solution to mitigate against these shortcomings [81].

In this work, each participating node of the overlay network measures the traffic that is being sent to a victim locally and computes a distributed average using time-dependent averaging and router topology properties [81]. Subsequently, each node verifies whether the estimated average exceeds the capacity of the victim (*i.e.,* a fixed threshold), and if this is the case a DDoS attack is flagged [81]. However, the authors note that this work does not contain a solution to differentiate benign traffic from malign traffic, and also does not include a mitigation solution [81].

3.8 MISP

The work by [82] introduces **MISP**, a Malware Information Sharing Platform. This platform exploits collaboration between users to share information about threats and incidents, with the goal that the community of the platform can take preventive actions and set up counter measures [82]. The platform also supports the sharing of DDoS attack information using its own data model [42]. To restrict access to sensitive information, the platform supports various degrees of sharing levels, allowing its users to share an event within their own organization only, within a community, in connected communities, or to all MISP communities [82]. An example is displayed in Figure 3.5, where an event *e* is shared to the *Community A* and *e'* as *Connected Communities* [41].

Newly added threat events can be synchronized and shared between the MISP instances in various ways [82]:

- **Pull:** Using distribution rules and filters, events are pulled from the connected MISP instances.
- **Push:** This method pushes threat events to other MISP instances directly. Sharing in this manner can be triggered automatically by publishing a new event, or by pushing events manually.
- **Cherry Picking:** This feature allows users who can see events from the connected MISP instances to pick specific, deemed relevant events to pull to their own instance.
- **Feed System:** This synchronization scenario is relevant for MISP instances that are not directly connected to each other, or for providers that want to share data to all communities. With this feature, users are able to pull specific events only or all events from remote servers.

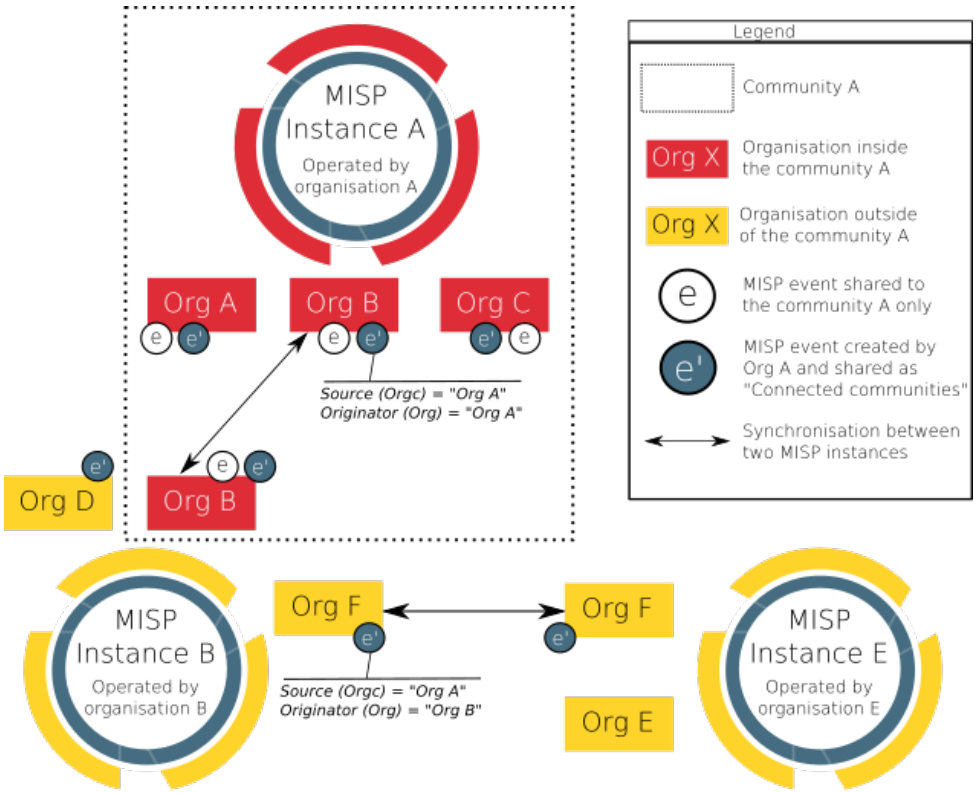


Figure 3.5: Example of event distribution levels and various community configurations [41]

Additionally, the platform supports Pub/Sub using ZeroMQ (*cf.* [76]), enabling the integration and automation of events from a MISP instance in custom software [41, 10]. Hence, real-time integrations are possible and supported [41]. However, the real-time information available for Pub/Sub has limited to the realm of visibility that the particular MISP instance is subject to [10].

3.9 BloSS

The Blockchain Signaling System (BloSS) makes use of consortium-based Blockchain (BC) technology and Smart Contracts (SC) to collaboratively protect against large-scale DDoS attacks since existing centralized defense systems often do not feature the required hardware resources or software capabilities to detect and mitigate large-scale DDoS attacks [68, 69].

The use of a consortium-based BC leads to an increase in security and a high trust environment since participants are known and the information access is limited [68]. The architecture in a Software-Defined Networking (SDN) environment is depicted in Figure 3.6, though the authors note, that the decentralized Application (dAPP) is not limited to SDN-based environments only [68].

The architecture's three layers are the following [68]:

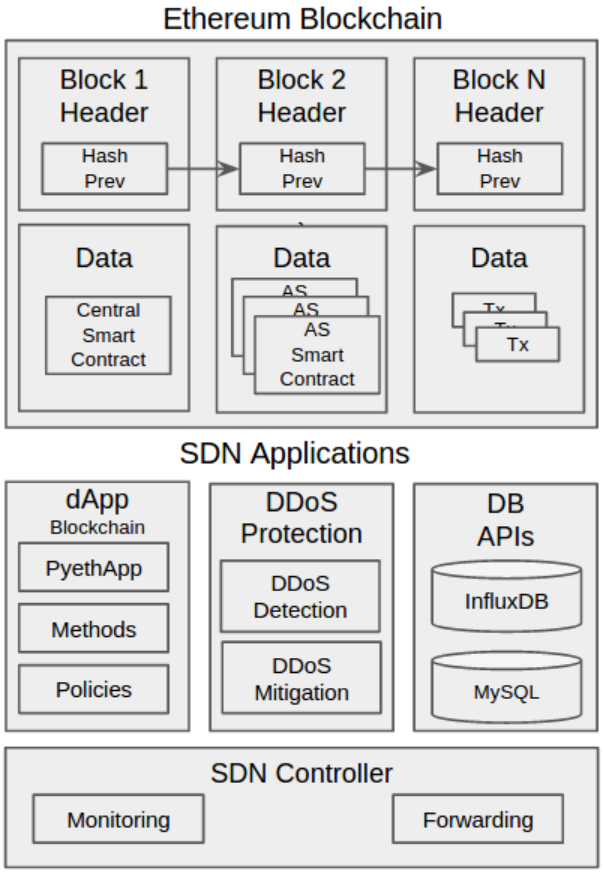


Figure 3.6: Architecture of BloSS [68]

- **SCs**, which are deployed in the Ethereum BC.
- The **dApp** that serves as an interface to the BC is used to report and retrieve BC addresses.
- A **SDN controller**, which is responsible to monitor and enforce rules in the switches.

The architecture makes use of a multitude of Autonomous Systems (AS), whereas each AS operates an IP network and stores its network addresses in its own SC [68]. Furthermore, a *central* SC stores for each AS its responsible IP network addresses, and can also update and configure these, and individual AS query the central SC to maintain a lookup table [68].

In the case of attack detection, the affected AS looks up the SC's address of the AS that operates the attacker's IP address and requests cooperative defense via a transaction on the SC, which for instance can be the request to block the IP addresses of the attackers [68]. After the BC block mining process, the requested ASs receive and individually handle this request, based on their own configuration and policies, and can for instance honor the IP address blocking request or reject it [68].

3.10 Doshi et al. (2018)

[20] propose a DDoS detection system using Machine Learning (ML) for Internet of Things (IoT) devices. Their work focuses on IoT devices since these devices are often insecure, thus, frequently seen as participants in botnets, and they note that these devices have their own idiosyncrasies, *e.g.,* repetitive network traffic and a small set of network endpoints, which makes their traffic distinguishable from other network traffic [20]. Hence, the traffic classification and identification of devices that participate in botnets are run in network middleboxes [20]. However, since the middleboxes can in this instance also include home gateway routers, there are several performance limitations that must be factored in, *e.g.,* memory and processing performance, thus, constraining the ML algorithms that are suitable for the classification task [20].

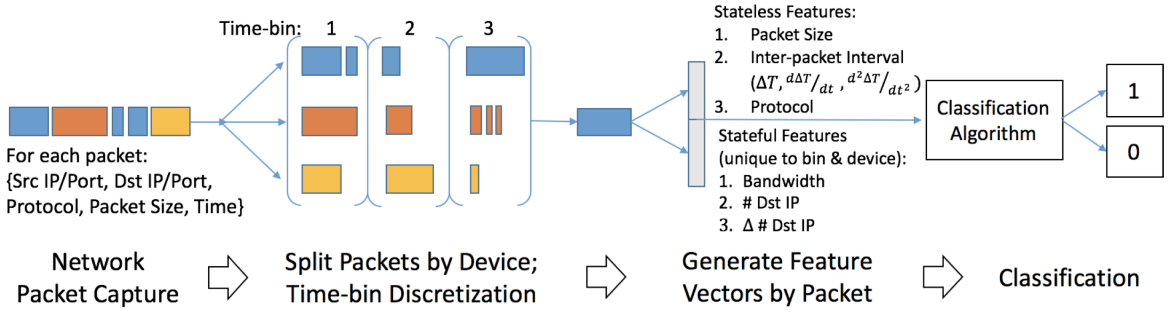


Figure 3.7: DDoS Classification Pipeline [20]

In Figure 3.7 the proposed DDoS detection for IoT devices, which includes four steps, is shown [20]:

1. **Traffic Capture:** In this step the source and destination *IP addresses* and *ports* are captured, along with *sizes* and *timestamps* of packets.

2. **Grouping of Packets by Device and Time:** The packets are grouped by source devices and into non-overlapping time windows.

3. **Feature Extraction:** Using domain knowledge of IoT devices, stateless and stateful features are generated for each packet. For instance, due to memory limitations, *stateful features* include aggregations of flow information over time windows, and *stateless features* include mostly packet header fields.

4. **Binary Classification:** Using a ML classifier, the DDoS attack traffic is distinguished from normal traffic.

Their evaluation showed that various ML classification algorithms were able to differentiate the traffic with very high accuracy, *e.g.,* Decision Trees and Neural Networks, and they expect that the neural network classifier's performance would scale well with more training data [20].

3.11 Wagner et al. (2021)

[83] propose the work "United We Stand: Collaborative Detection and Mitigation of Amplification DDoS Attacks at Scale", a central hub that allows the information exchange of amplification DDoS attacks. Amplification attacks make use of services such as Domain Name System (DNS) or Network Time Protocol (NTP) as *reflectors* to generate traffic that is up to 50′000 higher than normal [83]. Amplification attacks are further explored in Section 2.1.1. The idea is that attack mitigation platforms, operated in Internet Exchange Points (IXP), collaborate together by detecting attacks locally and sharing this information such that each IXP can drop the traffic, thereby effectively weakening the attack significantly [83].

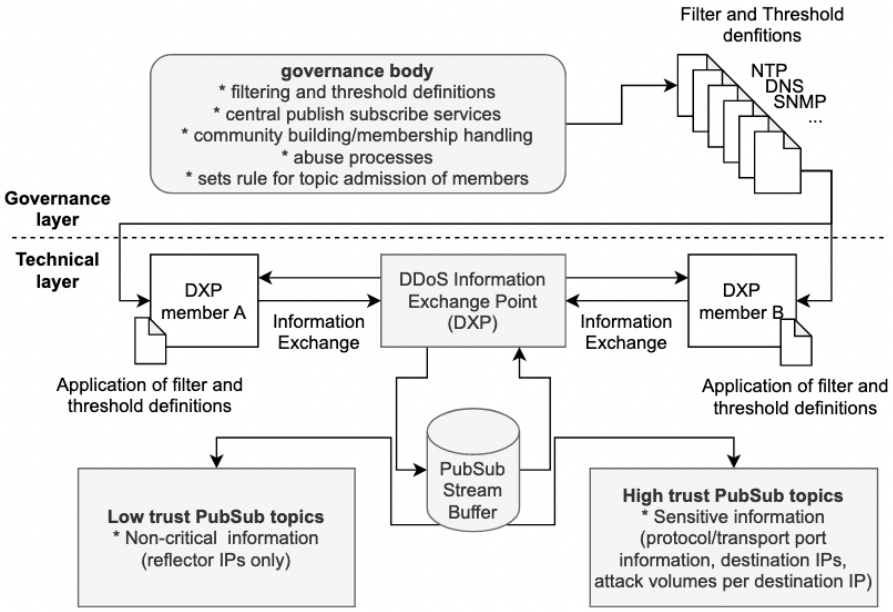


Figure 3.8: Concept of the DDoS Information Exchange Point [83]

Their concept is displayed in Figure 3.8, and their work uses a publish-subscribe system to share information between the participants [83]. Moreover, they differentiate between two different models that are used for PubSub topics [83]:

- **Low trust environment**: This topic includes less privacy-sensitive information, *i.e.,* the reflector's IP addresses and peak traffic values only since the reflector's IP addresses can be seen as public information, as the server's IP addresses can be found in research papers. Subscribers can based on this topic limit the rate or block reflectors systems entirely.

- **High trust environment**: This topic adds to the low trust environment topic information, the victim's IP addresses, and the respective attack volume per source port. This information exchange is performed in the high trust environment only, since the exchanged information is sensitive according to regulations such as GDPR, and their system assumes a membership with Non-Disclosure Agreements.

Their results show that collaboration via the DDoS Information Exchange Points (DXP) in the case of low trust environments, and also high trust environments, offer significant benefits in terms of DDoS attack detection and mitigation [83].

3.12 DDoS Clearing House

The DDoS Clearing House is part of the CONCORDIA project [80], which stands for "**C**yber security c**O**mpete**NC**e f**O**r **R**esearch an**D** **I**nnov**A**tion" [16]. This system essentially captures DDoS attack information based on the traffic and creates a fingerprint based on those characteristics [9]. These fingerprints are automatically shared within coalitions formed from various organizations, allowing its members to take proactive measures against this specific DDoS attack [80].

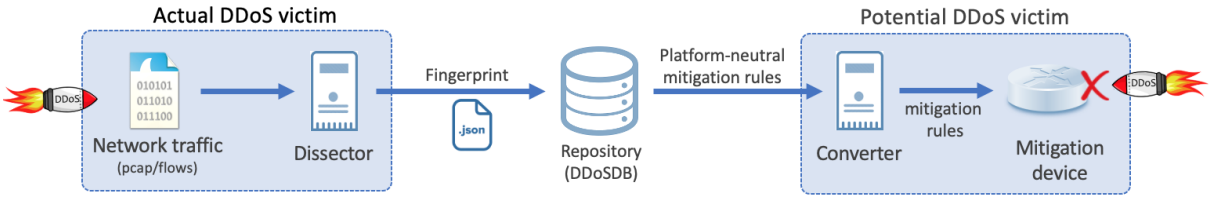


Figure 3.9: Architecture of the DDoS Clearing House [17]

In Figure 3.9 the overall flow and the system's architecture is displayed, highlighting the system's three core components [80, 17]:

- The **Dissector** is responsible for analyzing the DDoS traffic and creating its fingerprint. It captures information such as protocol types, packet lengths, and source IP addresses in the fingerprint, and due to its metadata nature, it can be shared with other members without inadvertently sharing sensitive information.
- **DDoS-DB**, is a centralized database operated by the coalition members and is in charge of storing and sharing the fingerprints.
- The **Converter** is run by each member and creates attack mitigation rules based on the fingerprints it receives. For instance, a possible mitigation strategy is to block IP addresses.

3.13 DDoS Open Threat Signaling (DOTS)

Internet Engineering Task Force (IETF) proposes DDoS Open Threat Signaling (DOTS), a standardization attempt for the real-time signaling of DDoS attack information, mitigation requests, and attack classification [28]. DOTS makes use of a client-server architecture (*cf.* Figure 3.10) and assumes the presence of other, unspecified attack mitigation or attack detection services that are outside the scope of DOTS [45].

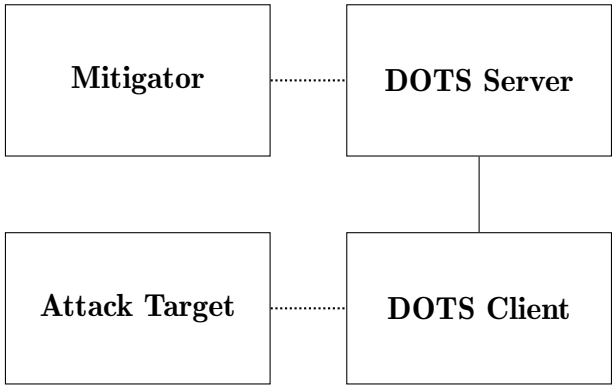


Figure 3.10: DOTS Architecture, based on [45]

- **DOTS Client:** A DOTS agent, which requests mitigation from the DOTS server [45].
- **DOTS Server:** Responds to DOTS clients, and delegates mitigation requests to mitigation services [45, 46].

In the case an attack is detected, the DOTS client requests mitigation from the DOTS server [45]. The DOTS server in turn invokes mitigator services which are responsible for the actual mitigation of the attack [45]. A DOTS client is connected to one or multiple DOTS servers via a *(i)* a signal channel, and *(ii)* an optional data channel [45]. The *signal channel* is used for the client to request mitigation help from the DOTS server, also potentially including telemetry information about the attack, whereas, in turn, the server uses this channel to respond to the client whether the mitigation request is accepted or denied [45]. To ensure the health of the channel, *i.e.,* to distinguish between a disconnected session and an idle signal, a bi-directional periodic heartbeat message is sent over the signal channel as well [46, 45]. The authors also note that this channel is required to be operable even while the connecting links are under attacks [45]. On the other hand, the *data channel* is used for configuration management and policy exchange between the DOTS client and DOTS servers [45]. However, this data exchange supposes normal conditions and as such, while possible, the channel should not be used while an attack is undergoing [45].

Furthermore, due to its standardization approach, this work does not impose restrictions regarding the collaboration of parties, as such, communication can be realized in a *inter-domain* or in a *intra-domain* fashion [28].

3.14 Comparison and Discussion

In Table 3.1 the related work in the realm of DDoS defenses is compared. The column **Cooperation** denotes whether the work includes a degree of cooperation or collaboration between various parties, *e.g.,* for collaborative detection or attack signaling. Further, the column **Type of Data Exchange** indicates the main architectural component responsible for the exchange of information relating to the signaling or detection in the case that collaboration between various parties has been designed as part of the work. In **DDoS Scope** the solution's DDoS defense scope is stated, including the categories:

- **Detection:** Mechanisms to detect DDoS attacks are the core idea of the work.
- **Signaling:** Denotes whether the project's scope includes the signaling of a DDoS attack or the signaling of attack information in a cooperative manner [69].
- **Mitigation:** Regards mechanisms to take mitigating measures, for instance, rate limiting the traffic [69].

Moreover, based on [69], the column **Behavior** denotes whether the work behaves as *reactive* or *proactive*. These categories have been defined in this work as follows:

- **Proactive:** Implies that the work intends to tackle DDoS attacks through preventive measures that are in play before an attack occurs. For example, based on previously determined attack signatures and thresholds.
- **Reactive:** Denotes the behavior of the mechanism of a system that reacts to an occurring attack and responds to it by starting signaling or mitigation steps.

[25] detect DDoS attacks locally by comparing the proportionality of traffic flow from one direction to the other direction. However, it does not employ collaboration between nodes. While this work also considers both directions in the detection, the perspectives are split between the participants. Furthermore, the idea of considering the proportionality of traffic flow has also been added to CH^{2}TF.

Compared to the other works, the work by **[39]** does not include a collaboration aspect, *i.e.,* D-WARD modules act autonomously and in autarky to reach defense against DDoS attacks. However, compared to the works discussed here, this work attempts to mitigate DDoS attacks at the source directly, which relates to this work's scheme that includes a hybrid approach and also considers the source.

[6] includes a unique scheme to combat spoofing in DDoS attacks and introduces collaboration between routers to do so, *i.e.,* signaling. Furthermore, it also encompasses a strategy to combat malicious behavior. Due to its *a priori* route recording mechanism that occurs before attacks arise, this work has been considered as *proactive*.

DefCOM **[52]**, uniquely uses a P2P network for the message exchange to achieve high scalability. However, [69] notes that this can also be seen as a detriment due to the highly involved message exchange. Although the cooperative aspect idea of DefCOM is comparable to this work, not only is Pub/Sub used rather than a P2P network, the main idea of their work can be seen as collaborative mitigation, not detection itself.

[95] uses an anomaly detection model and an encouragement model to achieve higher service rates for legitimate users. Regarding DDoS detection, they focus on application layer attacks, while this work targets heavy hitters. Furthermore, anomaly detection is based on K-means clustering, while this work uses threshold-based detection methods. However, [96] note that the scheme proposed by their work is too resource-heavy and not implementable.

Table 3.1: Comparison of Related Work in the realm of DDoS defense

Solution	Year	Cooperation	Type of Data Exchange	DDoS Scope	Behavior
MULTOPS [25]	2001	✗	-	Detection Mitigation	Reactive
D-WARD [39]	2002	✗	-	Detection Mitigation	Reactive
AITF [6]	2005	✓	Router-Router Communication	Signaling Mitigation	Proactive
DefCOM [52]	2006	✓	P2P Network	Signaling Mitigation	Reactive
DOW [95]	2007	✓	Client-Server	Detection Mitigation	Reactive
[97]	2008	✓	Centralized Database & Blocklists	Detection	Proactive
[81]	2010	✓	Overlay Network	Detection	Reactive
MISP [82]	2016	✓	Information Sharing Platform	Signaling	Proactive Reactive
BloSS [68]	2017	✓	Blockchain & Smart Contracts	Signaling	Reactive
[20]	2018	✗	-	Detection	-
[83]	2021	✓	PubSub	Detection Signaling	Reactive
[17]	2022	✓	Centralized Database	Signaling Mitigation	Reactive
DOTS [28]	2022	✓	Client-Server Communication	Signaling	Reactive
This work	2022	✓	PubSub	Detection	Reactive

[97] uses a centralized, predictive blocklist that is updated daily to defend against DDoS attacks. This differs greatly from this work since users must pull their updated list daily. No immediate information exchange between parties is possible, implying that information from an ongoing attack can not be shared with others to defend against it, *i.e.,* implying a *proactive* DDoS defense for parties that have not been hit for these specific attacks.

[81] uses a cooperative overlay network to detect DDoS attacks. This cooperation approach is similar to this work's core idea, though instead of an overlay network, Pub/Sub is used to communicate between the participating ASes. Additionally, the idea of detecting an attack locally and then asking the participants for cooperation is also present in this thesis. However compared with this thesis, CH^{2}TF not only detects an ongoing DDoS attack but can also detect the attackers.

[82] propose a generic threat and incident sharing platform that can also be used for shar-

ing DDoS incidents. While their work also uses Pub/Sub, their topics are not comparable since they are more generic and not specific to DDoS. Their use of Pub/Sub is restricted to the information available to the local instance only. Thus, information exchange between different parties using Pub/Sub is not supported, which differs from this work, which employs it as the main information exchange channel. This work has not been considered here as a *real-time* system due to potentially manually entered events being added and a signaling flow specific to DDoS attacks requiring custom out-of-the-box scripting. Under the assumption that such options are considered, this work could be used reactively against an attack and proactively against unaffected DDoS attacks.

Further, the work by **[68, 69]**, uses Blockchains & Smart Contracts to achieve a decentralized and cooperative approach to DDoS mitigation. In comparison, their work also includes a simple mitigation solution by allowing the direct requesting IP addresses to block to defend against DDoS attacks. Nonetheless, the main focus of the work lies in the signaling aspect. Most importantly, this work shares the core idea of deployments at ASes directly and the interaction and cooperation between the ASes.

[20] concerns itself with the detection of consumer IoT attack traffic using ML techniques. This work is more general and not specific to IoT devices, though future work could include ML-based detection techniques. Furthermore, this work is based on collaborative aspects.

Similarly to this work, United We Stand **[83]** employs the Pub/Sub framework to exchange DDoS-related information. However, Pub/Sub was used in their work differently, in particular, their chosen Pub/Sub topics differed. While they differentiated their Pub/-Sub topics between *High trust environments* and *Low trust environments*, this work chose Pub/Sub topics relating to the confidence of an attack. Additionally, their work focuses on the detection and mitigation of amplification attacks, contrasting the more general heavy hitter detection that is the focus of this work. Moreover, their module is deployed at IXPs, while this work assumes a deployment at ASes directly.

The work by **[17]** is about sharing information regarding DDoS attacks and does not include the detection itself. Furthermore, attack information is shared and stored via a *centralized* database, not Pub/Sub.

IETF proposes **[28]**, a DDoS signaling standard, and makes use of a client-server model. Its standardization approach does not impose specific technologies and delegates DDoS detection and mitigation to specific services. This is different from this work, which uses Pub/Sub and has detection as its main focus.

In summary, the novel idea of using Pub/Sub for data exchange can be traced back to [83], which also motivates this work's usage of Pub/Sub. Nonetheless, the chosen Pub/Sub topics are different, *i.e., Trust Environments* vs *Attack Confidence*. Furthermore, the idea of a local threshold-based detection has been applied multiple times, *i.e.,* [39, 83, 25, 81], though in a cooperative/collaborative manner only in [81, 83]. This work differentiates itself from these solutions by considering ASes at the core of the solution, similarly to [68], and not IXPs ([83]) or routers ([81]). This is motivated by allowing the mitigation to be done as close as possible to the source, *i.e.,* the attacker's AS, results in not only the victim's AS but also transit ASes being substantially less affected regarding their traffic load [73].

Chapter 4

Design

This chapter documents and explores the design of CH^{2}TF in an abstract level and describes the design considerations. First, the requirements that CH^{2}TF's design is subject to are recorded in Section 4.1. Subsequently, capabilities and features of CH^{2}TF are described in Section 4.2. Further, CH^{2}TF's architecture is explained in Section 4.3 and provides an overview of the modules that CH^{2}TF incorporates. Thereafter, the communication protocol and the Pub/Sub topics employed are discussed in Sections 4.4 and 4.5. Next, the policies to detect attacks and their attackers are documented in Sections 4.6 and 4.7. Finally, Section 4.8 explores the topic of privacy and in Section 4.9 a reputation-based scheme is presented.

4.1 Requirements

Based on the goals of the thesis that were outlined in Section 1.1, the following requirements have been derived:

- **R1:** The system shall detect HH in a time-efficient and data-oriented manner.
- **R2:** The system shall share the detected HH with the other participating instances.
- **R3:** The signaling system employed by the system shall support multiple Pub/Sub topics.

4.2 Capabilities and Features

In this section, CH^{2}TF's capabilities are listed. While these were derived from the requirements in Section 4.1, CH^{2}TF also includes additional features and capabilities.

- **DDoS detection:** CH^{2}TF includes targeted policies to detect DDoS attacks and signal this information to the collaborating ASes. The policies are explored further in Section 4.6.

- **Heavy Hitter detection:** Additionally to the attack detection, CH^{2}TF is also capable of detecting the HH that form the attack. This is also achieved using policies (*cf.* Section 4.7).

- **Privacy protection:** Since IP addresses can be considered as sensitive information, CH^{2}TF operates on hashes to conceal this information (*cf.* Section 4.8).

- **Cornerstone for DDoS mitigation:** Since CH^{2}TF focuses on DDoS detection, and DDoS mitigation is not part of this scope, this work does not include a complete DDoS mitigation scheme. However, a few foundations for a DDoS mitigation extension have been provided where applicable. This is primarily motivated by the use of Pub/Sub already providing a performant choice for the communication protocol between the ASes in CH^{2}TF, and integration of a mitigation solution would be the next logical step. This idea is further explored in Section 7.3.

4.3 Architecture

The architecture of CH^{2}TF is depicted in Figure 4.1. This architecture is technology-agnostic, *i.e.,* it does not impose any restrictions on the components regarding the use of specific technologies or programming languages. Furthermore, the scope of CH^{2}TF's design has been intended to exchange DDoS information between ASes (*cf.* [26]), *i.e.,* the module shall be installed on the AS level to communicate with other participating ASes. Thus, on a high-level, it includes the following modules:

- **Pub/Sub Framework:** A *generic* Pub/Sub framework responsible for the communication between the collaborating ASes. This is further explored in Section 4.5.

- **CH^{2}TF Module:** This module includes the analysis components for the detection of attacks and HH. Also, it is responsible for the flow of information in the system and is connected to the Pub/Sub framework.

- **Traffic Module:** To analyze the traffic, CH^{2}TF must obtain the packets from somewhere. This is this module's responsibility. *I.e.,* to sniff the traffic that passes through an AS and forward it to CH^{2}TF. However, this is outside of CH^{2}TF's scope.

- **Mitigation Module:** As described in Section 4.2 this work also includes a foundation where mitigation can be added. However, this is not part of CH^{2}TF's scope.

Furthermore, as displayed in Figure 4.1, the CH^{2}TF module can be decomposed into the following components:

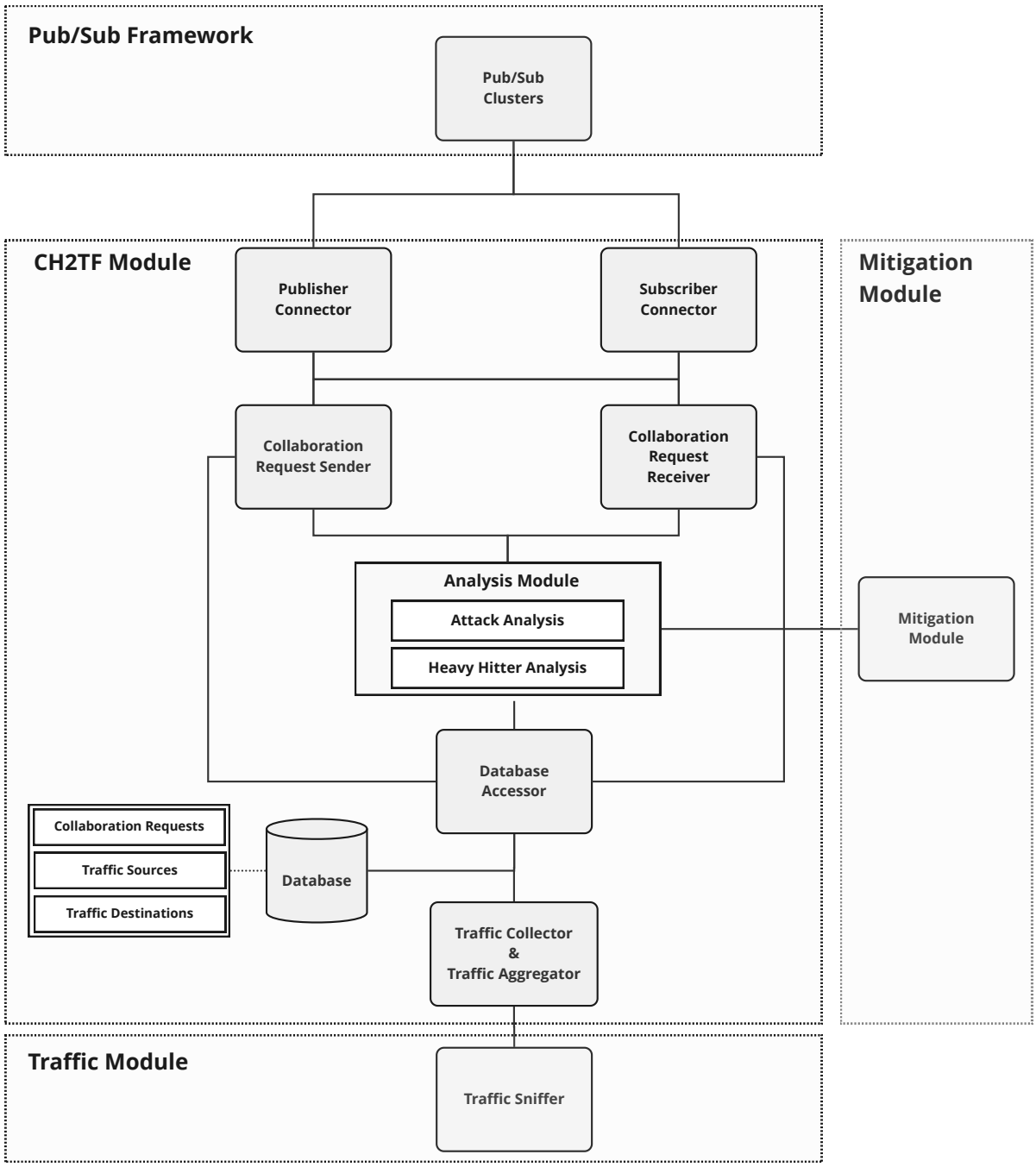


Figure 4.1: Architecture of CH²TF. While the *Traffic Module* and *Mitigation Module* are displayed here, they are not part of the project's scope.

- **Publisher Connector:** This component is responsible for connecting to the Pub/Sub framework as a publisher, *i.e.,* to publish messages in a certain topic.
- **Subscriber Connector:** Similarly to the **Publisher Connector**, this component connects to the Pub/Sub framework as a consumer to be able to receive messages.
- **Collaboration Request Sender:** This component allows the sending of collaborating

requests to the participating ASes.

- **Collaboration Request Receiver:** Similarly to the **Collaboration Request Sender**, this component is responsible for receiving messages from participating ASes.
- **Analysis Module:** The responsibility of this module is two-fold, (*i*) it performs the analysis to detect an attack, and (*ii*) it performs the analysis to detect the HH of an attack.
- **Traffic Collector & Aggregator:** This component performs the sampling of the packets and is connected to the **Database Accessor** to store them in the **Database**.
- **Database Accessor:** This component is responsible to connect to the **Database**.
- **Database:** Responsible for storing relevant information regarding the traffic senders and destinations, as well as keeping track of collaborating requests that have been previously sent and storing this information for later use (*e.g.,* analytics or logging).

4.4 Communication Protocol

In this section, the proposed communication protocol is discussed. The protocol is split into two parts, namely **Request Sending** and **Request Receiving**:

- **Request Sending:** This flow is depicted in Figure 4.2. Every AS monitors the traffic continuously, and upon detecting a DDoS attack, it acts as a request sender. This means that it sends a collaboration request to the participating ASes. This request includes a list of the potential attackers and the victim's IP address. At this point, the attackers are not certain yet; thus, the attackers can only be considered *potential* attackers. The request is sent by publishing a message to the Pub/Sub framework.
- **Request Receiving:** This flow is displayed in Figure 4.3. An AS acts as a request receiver upon receiving a collaboration request. Thus, it can be considered that an attack has happened or an attack is ongoing. Based on the list of potential attackers that was included in the request, the request receiver will perform an analysis to determine the HH from IP addresses that it manages as an AS. Based on the analysis findings, the AS publishes a message with the list of attackers it has found to the other ASes. Nota bene, this flow implies that HH are only searched for a specific attack that has been triggered. This is ultimately a design choice, though in CH^{2}TF this restriction has been imposed since policies regarding the attacker detection are more elaborate (*cf.* Section 4.7), and this restriction removes some of the module's processing load.

The aforementioned request sending and receiving flows are depicted in Figures 4.2 and 4.3 respectively, while the composition of both flows is displayed in Figure 4.4. Furthermore, the AS that detects an attack can either receive its request through the Pub/Sub framework or perform this sequentially in the program flow more efficiently. Since this

chapter discusses the design considerations on an abstract and generic level, no restrictions are imposed here. However, for the diagram in Figure 4.4 the sequential option was used.

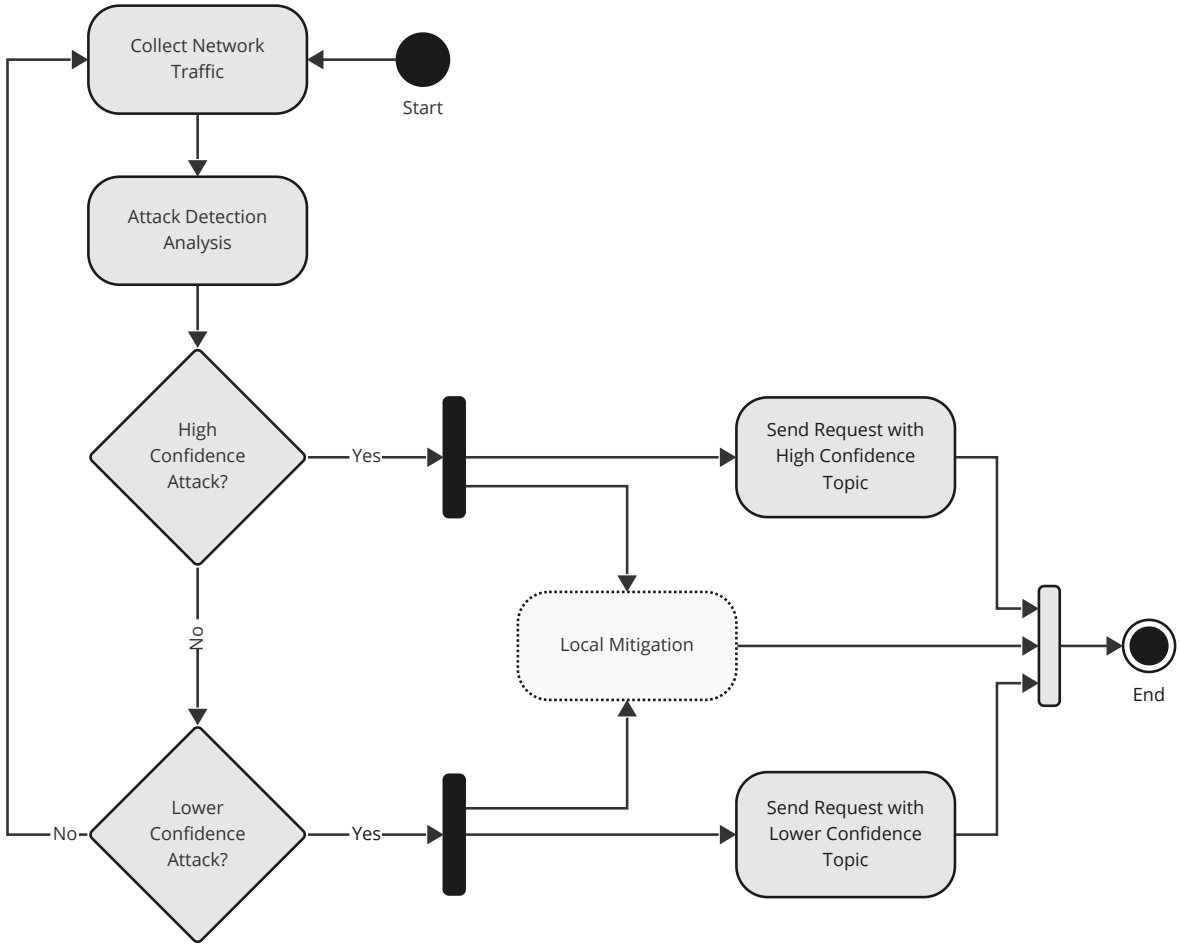


Figure 4.2: Activity diagram: **Request Sender** perspective.

Nota bene, in Figure 4.4 a specific order is displayed between the flow of processing, sending requests, and receiving responses. However, since the ASes can be seen as distributed systems, these events are ultimately asynchronous. The order of receiving the responses does not influence the other flows, *i.e.,* whether *AS100* or *AS200* receives the respective response first is arbitrary and only causally ordered by the fact that the sending of the respective response had to have occurred first before the receiving of the response occurs. Furthermore, if a *AS300* were to be included in this scheme, the order between *AS200* and *AS300* would not be related. Moreover, if a performant Pub/Sub framework is used, it is to be assumed that the receiving of messages would be almost instantly, and as such, in Figure 4.4 the 'GET Response (AS100)' would occur almost immediately after 'PUT Response (AS100)' comes up.

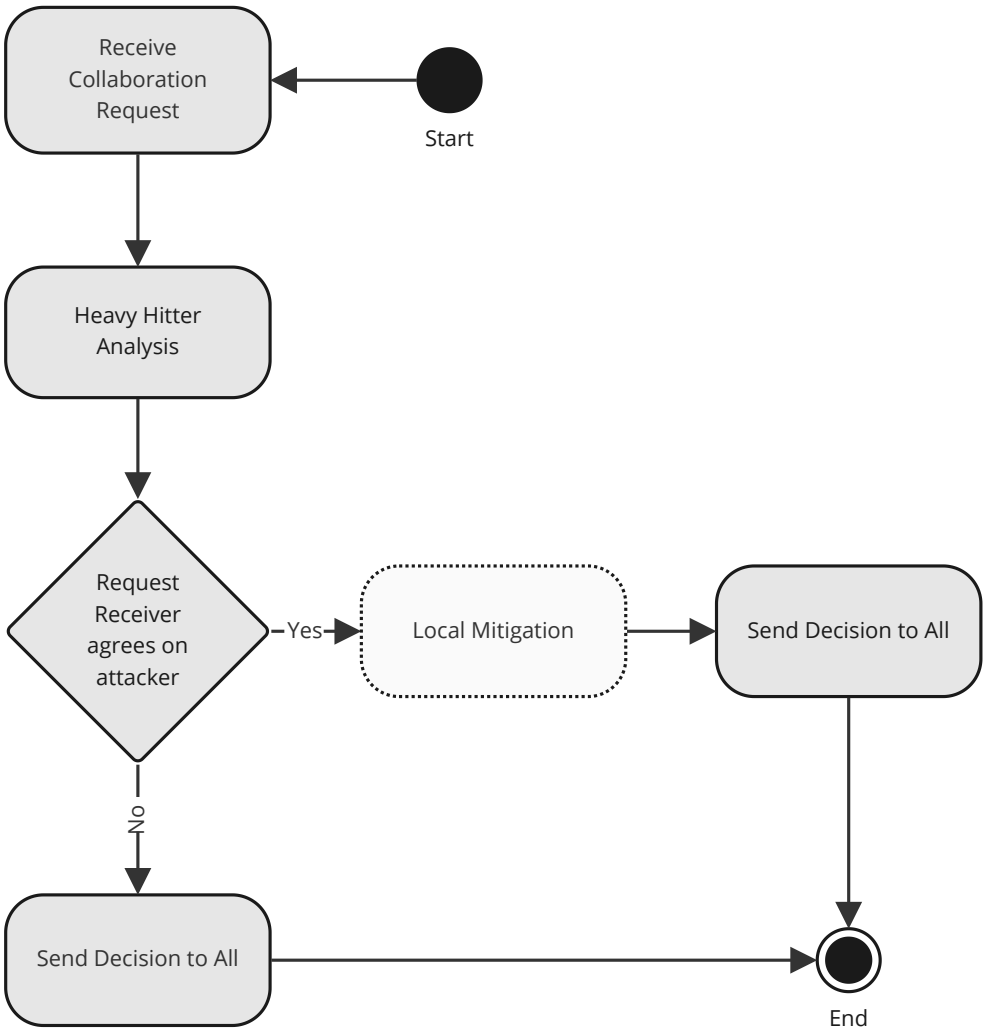


Figure 4.3: Activity diagram: **Request Receiver** perspective.

4.5 Pub/Sub Framework and Topics

As hinted in the previous sections, this work employs a Pub/Sub framework to enable the communication between the participating ASes. Pub/Sub has been chosen due to its various advantages over other messaging patterns and that it can be considered a popular and tested messaging pattern (*cf.* Section 2.3). Importantly, Pub/Sub allows the distinction between topics; as such, topic-specific communities can be formed. For instance, regional or international *trust-based* communities can be added, similarly to [83].

CH^{2}TF proposes the following Pub/Sub topics for the *core* functionality, however, the design is intended to be flexible, such that it lets each AS choose to which *additional* topics it wants to subscribe to via configuration:

- **Lower Confidence Topic Request**
- **Lower Confidence Topic Response**

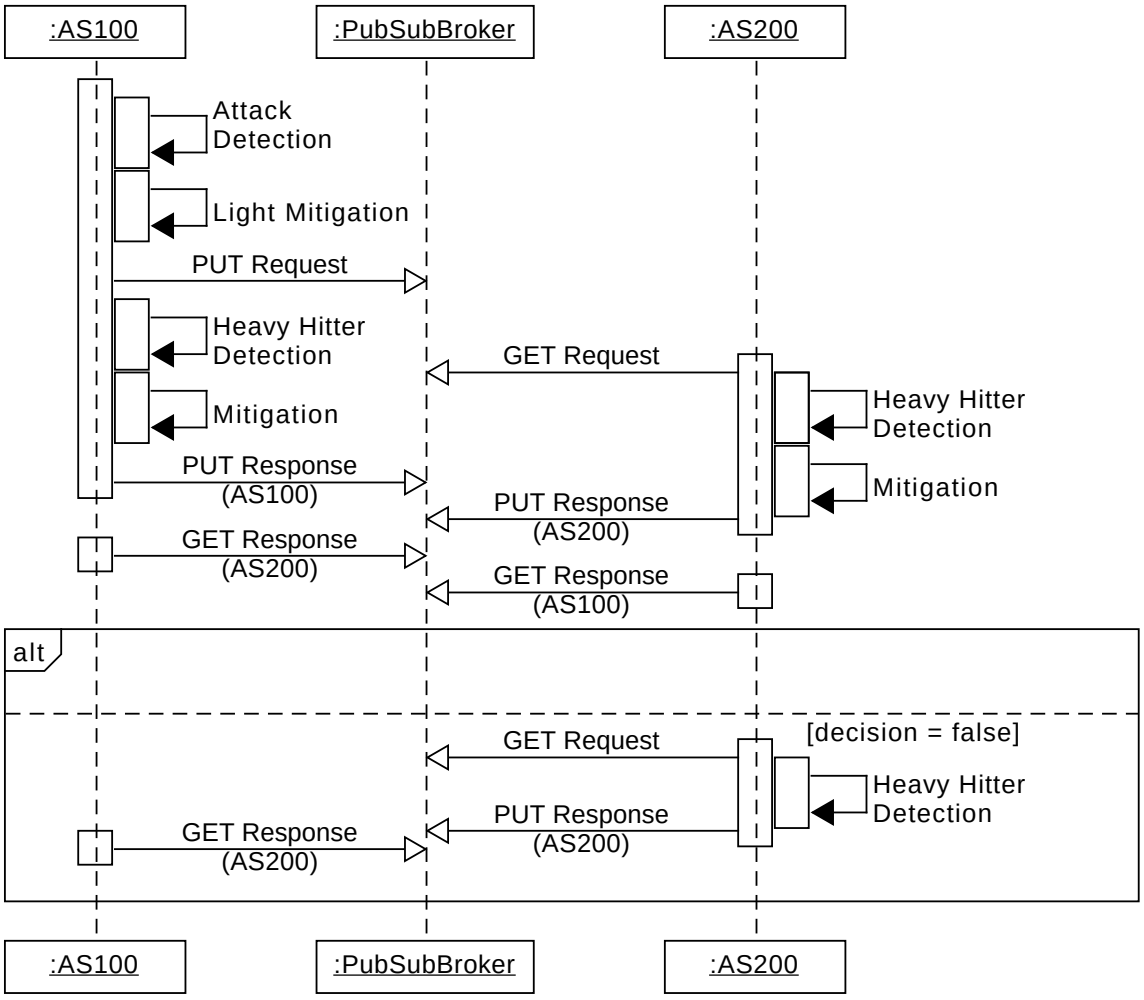


Figure 4.4: Sequence Diagram of CH²TF's communication protocol. In this example, two ASes are used. No HH were found in the alternative path, and $AS200$ responds accordingly.

- **Higher Confidence Topic Request**
- **Higher Confidence Topic Response**

The distinction between *higher* and *lower* confidence is regarding the confidence of an attack, and plays a role in the reputation-based scheme (*cf.* Section 4.9). The topic (*i.e.,* lower or higher confidence request) is chosen based on the policies' thresholds.

Furthermore, as has been described in Section 2.3, the messaging pattern is *unilateral*, *i.e.,* a subscriber is not able to reply to a publisher. Since the employed communication protocol of CH²TF requires the communication to be *bilateral* (*cf.* Section 4.4), two *response* related topics were added (*i.e.,* **Lower Confidence Topic Response** and **Higher Confidence Topic Response**). However, while they have been denominated with 'response', they are not only related as a response to the request sender but to all participating ASes.

4.6 Policies: Attack Detection

To detect an attack, CH^{2}TF uses policies based on thresholds. The proposed policies were selected based on the related work review. However, the goal of these policies is to detect an attack, as such, while this work proposes a selection of policies, they are ultimately not related to the proposed communication protocol.

The perspective of the attack detection is from the *victim's* perspective. *I.e.,* the analysis is not run for the source (*attacker*) IP addresses but looks for a potential victim to trigger an attack. The *raison d'être* of this design choice is that with a *destination-perspective* (*cf.* [69]), the amount of traffic volume that reaches a destination is clear. Also, [69] notes that *destination-based* are usually weak in differentiating between *legitimate* and *illegitimate* traffic. Hence, the idea of taking the perspective of the destination to only trigger an attack, but not yet detecting the attackers, is used in CH^{2}TF.

- **Case 1 - Amount of packets arriving at the destination is above threshold:** This policy is the simplest one, it only checks if, during the analysis interval, the destination is subject to a high volume of traffic. Although it can be seen as a simple criterion to detect an attack, it can be considered as highly effective in detecting an attack that is based on HH.

- **Case 2 - Increase in traffic above threshold:** This policy checks whether there has been a significant (*i.e.,* above a certain threshold) increase in traffic targeted to a destination. *I.e.,* if the previous analysis run had a relatively low amount of traffic, and in the subsequent run a high increase of traffic (that is over the threshold) is observed, this policy flags an attack.

The selected policies have been purposefully kept simple. They are designed to be performant since the analysis is run for millions of IP addresses each analysis cycle. Nevertheless, the second policy has memory requirements, since it needs to store the aggregated results of the previous analysis run. Since only the previous analysis run (and the ongoing) need to be stored, this means that the memory depends on number of destination IP addresses, and as such, scales linearly.

Furthermore, while [8] notes that using *thresholds* for attack detection is *inflexible* in regards to customizability, CH^{2}TF lets each participant choose their own thresholds, thus, solving this problem.

4.7 Policies: Heavy Hitter Detection

Similarly to Section 4.6, the HH detection is also based on policies that use thresholds. Nevertheless, since the goal of this analysis is to detect the *attackers* of a given *attack*, the chosen policies are evidently different.

- **Case 1 - Source sends too many packets to specific victim:** The idea of this policy is simple, it checks if the attacker is specifically targeting a specific victim by sending a high number of packets to the victim.
- **Case 2 - Source sends too many packets to many destinations:** This policy is similar to **Case 1**, however, it is more general, and not specific to a victim. It checks if the overall traffic that is being sent from the source IP is too high (*i.e.,* over a certain threshold).
- **Case 3 - Source sends packets only to the victim:** While this policy sounds similar to **Case 1**, it is more subtle. It is intended to catch attacks where a low, but constant amount of traffic is used to target the victim, but otherwise does not send a high number of traffic.
- **Case 4 - Traffic direction proportionality:** This policy checks the direction of traffic, *i.e.,* is the destination also replying back to the (potential) attacker?

This approach to detect HH can essentially be seen as a *source-based* approach. Advantages of a such an approach include that the effort to analyze traffic is low, and the attack can easily be traced [69]. However, the typical problem of *source-based* defense mechanisms (*cf.* [96, 69]) that the highly distributed nature of a DDoS attack makes source-based mechanisms less effective since the attack is distributed and not visible at the source is countered in CH^{2}TF. The idea to detect the *attackers* at the source is to make use of the *a priori* knowledge that an attack is ongoing. As such, the participant does not have to analyze every possible source, but only the reported, potential few attackers. Also, using the *a priori* knowledge of an attack and its victim, the analysis is better able to differentiate between *legitimate* and *illegitimate* traffic, which is otherwise hard (*cf.* [69, 96]).

While CH^{2}TF proposes and also uses these policies for the prototype implementation (*cf.* Chapter 5), future work can include more intricate solutions. The design of CH^{2}TF is, in this regard, generic so that other analyses can easily be added.

4.8 Privacy

CH^{2}TF assumes that the participating AS are trusted since the policies and their thresholds should be worked out together between the ASes to agree on their definitions and settings to maximize the system's potential in detecting HH. However, the idea behind the design philosophy of CH^{2}TF is to not overshare information between participating ASes. Ultimately, a trust-based relationship does not imply that trusted parties are willing to share everything with each other. Multiple scenarios can be imagined where competing ASes are willing to collaborate together to achieve the common goal of DDoS defense, though are less willing to share details and information regarding their systems. To appease these scenarios and make CH^{2}TF viable, these circumstances have been considered.

Nevertheless, and despite the aforementioned argument, privacy regulating frameworks such as GDPR consider IP addresses as sensitive [83]. As such, to comply with privacy

regulations, CH^{2}TF proposes to hash the IP addresses and only operate on those hashes, similarly to [83].

Nonetheless, it can be noted that this adds additional complexity for future work related mitigation approaches since, for traffic-forwarding ASes, it would not be immediately clear which IP addresses should be mitigated, *e.g.,* rate limited or blocked. However, the goal of CH^{2}TF is not to propose a mitigation solution, as such, this disadvantage and the discussion thereof is related to future work. Nonetheless, this trade-off must be accepted if regulations require such shortcomings.

4.9 Reputation

It is fathomable that participating ASes take advantage of CH^{2}TF scheme to mitigate detected HH, though are themselves unwilling to mitigate attackers from their own set of managed IP addresses. Such a scenario could be envisaged where an AS that is partaking in a cyberwarfare (*cf.* [67]) actively would use the DDoS defense offered by CH^{2}TF, though never honor any requests to detect attackers from its own set of managed IP addresses. Essentially, this is the famous "Free-rider problem" (*cf.* [32]) that results in a system (or market) failure.

To counter such cases, CH^{2}TF also proposes a simple reputation-based scheme to keep track of ASes who never acknowledge an attack. This is so far important since the detection of HH is coupled with the detection of an attack. Thus, if an AS sets their thresholds too high, it will never acknowledge an attack, implying that it is acting adverse to the system, even if the intention is not *per se* malicious. *I.e.,* such an AS would always return 'Not Acknowledging' in the response to a request.

This problem stems from the fact that CH^{2}TF does not impose *global* thresholds for the policies in use. Since various ASes have different sizes and capabilities to handle traffic, they can not directly be compared. Consequentially, normal traffic for a very large AS would ring the alarm-bells for a local, small-sized AS.

To amend the aforementioned problems, CH^{2}TF proposes a variable that is part of the configuration that is related to the *size* of a given AS. This variable is used when sending requests to the other ASes, though is not disclosed to the other ASes. This is shown in Section 5.5. The goal is to make attack detections and thresholds comparable between the various ASes, no matter their size. As such, if a request receiver does not acknowledge an attack, it means that its own thresholds are set too high, resulting in a penalty. Conversely, it could also mean that the request sender's thresholds were set too *low*. However, given the goal to detect HH and the requirement of CH^{2}TF's design that an attack has to have been triggered before the HH are searched for, it implies that setting the thresholds too *low*, is not necessarily a bad thing, and should not be penalized. CH^{2}TF proposes that requests from participating parties with high reputation should immediately be tried to be mitigated to incentivize setting appropriate thresholds.

Nevertheless, the author notes that while this scheme solves an aspect of the problem, it is also not perfect and could be exploited to spam the system with unnecessary requests

that would stem from setting the thresholds too low and redirecting a full-fledged solution to future work. Ultimately, without imposing restrictions on the trust assumptions or a global, governing body that is responsible for the system, it is fathomable that no perfect solution exists to solve all potential shortcomings.

Chapter 5

Prototype Implementation

In this chapter, a technical overview of CH^{2}TF based on the design guidelines examined in Chapter 4 is documented. In Section 5.1, an architectural overview of the prototype implementation of CH^{2}TF is presented. Further, Section 5.2 explores the Pub/Sub implementation details. Moreover, Sections 5.3, 5.4 and 5.8 are about the respective *threads* of CH^{2}TF. Next, Sections 5.5 and 5.7 go into the details about the details of sending and receiving of requests. Further, Section 5.8 documents the attack and HH analyses. Lastly, Section 5.9 presents the use of Bloom filters in CH^{2}TF.

5.1 Components

The prototype implementation of CH^{2}TF has been written in Python (*cf.* [58]) and is containerized as a Docker container (*cf.* [18]). The components are depicted in Figure 5.1. Python and Docker are popular tools allowing for fast prototyping and deployment of applications and, thus, are suitable for CH^{2}TF's goals.

Furthermore, as a Pub/Sub framework Apache Kafka (*cf.* [4]) with its dependency Apache ZooKeeper (*cf.* [5]) has been selected. Thus, the components of this architecture are the following:

- **ZooKeeper:** Has been added due to being a necessary dependency to run Kafka.
- **Kafka:** The chosen Pub/Sub framework. Responsible for the message exchange between the ASes.
- **CH^{2}TF:** Includes the components relating to the Kafka connection, reading the forwarded traffic and processing the attack and HH analysis.

Regarding the database that was mentioned in Section 4.3, the prototype implementation of CH^{2}TF uses built-in Python dictionaries, which can be seen as key-value stores (*cf.* [59]). Although more intricate solutions such as Redis (*cf.* [66]), MongoDB (*cf.*

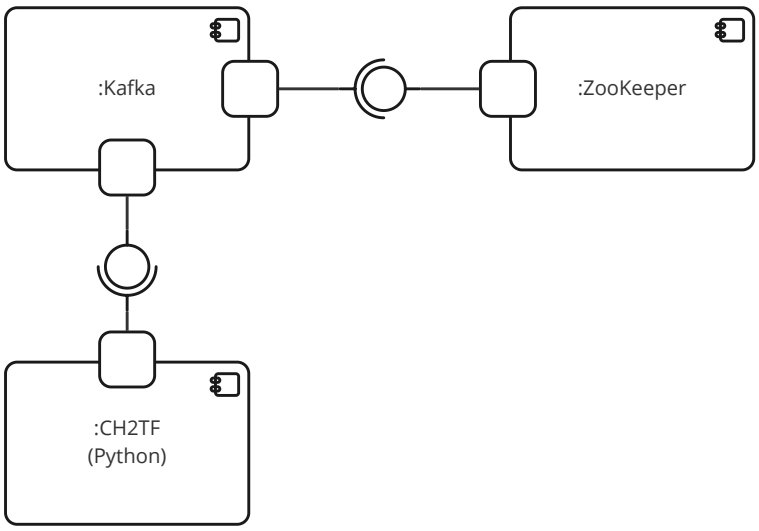


Figure 5.1: The component diagram of CH^{2}TF

[44]), UltraDict (*cf.* [79]) or SQLite (*cf.* [63]) were explored and considered, the built-in dictionary solution has been selected due to the performance advantage that it has (*cf.* [79]). However, the author notes that while this choice considers the performance based on read and write speed, it does not consider memory constraints. *I.e.,* a large attack could potentially lead to CH^{2}TF running out of RAM. Nevertheless, this potential issue has not been further considered for prototyping purposes. Furthermore, CH^{2}TF does not require the traffic-monitoring data to be stored indefinitely, *i.e.,* to save memory, only two traffic-monitoring data sets are stored, which are required for the analyses. As such, built-in dictionaries are sufficient for the prototype implementation. Consequently, the database component outlined in Section 4.3 has not been included in Figure 5.1.

CH^{2}TF itself runs in multiple threads to make use of the shared memory (*cf.* [11]):

- **Collection of packages:** This thread is discussed in Section 5.3. It is responsible is to collect the packages that it receives and store them.
- **Listening to requests and responses:** This thread listens to the messages that it receives through Kafka. Details are found in Section 5.4.
- **Analysis:** This thread is responsible to run the attack detection analysis and send a collaboration request in the case an attack is detected. This is further discussed in Section 5.5.

5.2 Pub/Sub: Apache Kafka

As hinted in Section 5.1, Apache Kafka has been chosen as a Pub/Sub framework. This choice has been made consciously by the fact that Kafka is a tried and tested platform supporting high throughput (*cf.* Section 2.3.1), which is of importance to CH^{2}TF.

Regarding the Kafka deployment, for the prototype implementation, only a single broker has been chosen and the replication factor has been set to 1 (*cf.* Listing C). Of course, for a real-world deployment scenario, these values would need to be adjusted, however, for a prototype these settings have been considered as sufficient.

To connect CH²TF to Kafka, the Python library "kafka-python" (*cf.* [29]) has been chosen. In Listing 5.1 the Kafka Consumer is shown while Listing 5.2 shows the Kafka Producer. In line 2 of each Listing the 'bootstrap _servers' keyword argument is used, this is part of the configuration from CH²TF and is used to connect to Kafka.

```
1 consumer = KafkaConsumer(
2     bootstrap_servers=[KAFKA],
3     api_version=(0, 10, 0),
4     value_deserializer=json_deserializer,
5     auto_offset_reset="latest",
6     enable_auto_commit=False
7 )
```

Code Listing 5.1: Kafka Consumer

```
1 self.producer = KafkaProducer(
2     bootstrap_servers=[KAFKA],
3     api_version=(0, 10, 0),
4     value_serializer=json_serializer
5 )
```

Code Listing 5.2: Kafka Producer

5.3 Package Collecting

The package collecting thread is responsible for the storing of data that it receives through a '(multiprocessing) queue' (*cf.* [61, 60]). A queue is chosen due to the multiprocessing nature of CH²TF, corresponding to the idea that an (external) process forwards the traffic data to CH²TF. The multiprocessing aspect has been chosen to use multi-core architectures to run processes fully in parallel, which does not hold for threads in the selected Python implementation (*cf.* [64, 60, 47]).

Listing 5.4 shows the `collect_packages` method, where in line 8 an element of the queue is removed and returned, and in line 9 forwarded to the `_store_data` method (*cf.* Listing 5.6). Further, the `collect_packages` method assumes that a CH²TF external method adds the `PacketData` (*cf.* Listing 5.3) elements to the queue. Nevertheless, for illustrative purposes and as a reference, a traffic `Sniffer` class has also been added, though it is not in use in CH²TF. The idea is that the traffic sniffer captures the live traffic that is passing through a network interface using the Python library "pyshark" (*cf.* [55]), which makes use of Wireshark's network analyzer "tshark" (*cf.* [93]). This `Sniffer` reference class is shown in Listing A.1, in Appendix A.

```
@dataclass
class PacketData:
    src: str
    dst: str
    srcport: str
    dstport: str
    timestamp: datetime
    transport_layer: str
```

Code Listing 5.3: The `PacketData` class

```
def collect_packages(self) -> None:
    """
    Collect traffic packages that the method receives through the queue.

    :return: None
    """
    while True:
        received: PacketData = self.queue.get()
        self._store_data(received, self.dest_dict, self.src_dict)
```

Code Listing 5.4: The `collect_packages` method

Listing 5.6 depicts the `_store_data` method. In lines 12 – 13, the *sampling* is shown. The idea here is that to improve performance, not all packets are considered. As such, if the method `is_sampling_skip` returns *true*, this particular packet is not stored and will lead to it not being considered in the analyses. The `is_sampling_skip` method is listed in Listing 5.5. If a random number is larger than the *sampling_rate*, then this method returns *true*, *i.e.*, an entry is skipped (line 14). Thus, if the *sampling_rate* is ≥ 1, no elements are skipped, since the interval of `random.random()` is $[0, 1)$ [62].

```
def is_sampling_skip(sampling_rate: float, rand: float = None) -> bool:
    """
    Returns whether the element should be skipped.

    :param sampling_rate: pos number >= 0
    :type sampling_rate: float
    :param rand: random number in the interval [0,1)
    :type rand: float
    :return: True if entry should be skipped
    :rtype: bool
    """
    if rand is None:
        rand = random.random()
    return rand > sampling_rate
```

Code Listing 5.5: The `is_sampling_skip` method

Further, in lines 15 – 17 of Listing 5.6, it is checked whether the *source* address of the packet is managed by this particular AS (*cf.* Listing 5.21), leading to only storing the

source perspective of the packet in the case that the AS manages this IP address. This is done since, in the prototype implementation of CH^{2}TF, only the AS managing the source IP address takes the source perspective for the HH analysis.

```
1  def _store_data(self, received: PacketData, dest_dict: defaultdict,
                                        src_dict: defaultdict) -> None:
2      """
3      Aggregates packages.
4      :param received: A received packet
5      :type received: PacketData
6      :param dest_dict: Destination perspective dict
7      :type dest_dict: defaultdict(Counter)
8      :param src_dict: Source perspective dict
9      :type src_dict: defaultdict(Counter)
10     :return: None
11     """
12     if is_sampling_skip(SAMPLING_RATE):
13         return
14     dest_dict[received.dst][received.src] += 1
15     if not self.check_if_is_managed(received.src):
16         return
17     src_dict[received.src][received.dst] += 1
```

Code Listing 5.6: The `_store_data` method

5.4 Listener

The listener thread connects to Kafka and listens to requests and responses as a Pub/Sub consumer. As such, it takes the role of the **Subscriber Connector** and **Collaboration Request Receiver** in Figure 4.1.

The `listener` method (*cf.* Listing 5.7) is executed in this thread. Based on the message's topic it receives, it delegates the message to the appropriate method (*cf.* Lines 23 – 30). The handling of collaboration requests (*cf.* Lines 25 – 28) is documented in Section 5.6. In Lines 16 – 17, the subscription topics are selected, which are made up of the *additional* (*cf.* Section 4.5), and *low* and *high* priority topics. Thus, the implementation is flexible concerning the topics an AS wants to subscribe to. The implementation does not force an AS to use the *default* topics, *i.e., low* and *high* confidence. Nevertheless, while the prototype implementation supports this feature, whether the use and creation of sub-communities are useful or detrimental is discussed in Section 7.2.

```
1  def listen(self) -> None:
2      """
3      listens as a consumer to the topics and delegates the function call
                                              according to topic
4      :return:
5      """
6
7      log.info("listening")
8      consumer = KafkaConsumer(
9          bootstrap_servers=[KAFKA],
10         api_version=(0, 10, 0),
11         value_deserializer=json_deserializer,
12         auto_offset_reset="latest",
13         enable_auto_commit=False,
14     )
15
16     topics = self._init_consumer_topics(TOPICS, [TOPIC_HIGH, TOPIC_LOW])
17     consumer.subscribe(topics)
18     log.info(consumer.topics())
19
20     message: ConsumerRecord
21     for message in consumer:
22         topic = message.topic
23         if "REQ" in topic:
24             if TOPIC_HIGH in topic:
25                 self.handle_collab_req(message, high_prio=True, topic=
                                                          topic)
26             else:
27                 # low prio or non-standard topics
28                 self.handle_collab_req(message, topic=topic)
29         elif "RES" in topic:
30             self.handle_collab_res(message, topic=topic)
```

Code Listing 5.7: The `listener` method

5.5 Sending Requests

In this section, the sending of collaboration requests is documented. Listing 5.8 shows part of the `run_analysis` method. This method runs in a separate thread and detects an attack and sends the collaboration request(s). Moreover, it also handles the analysis periods and clears the data of previous analyses. Thus, for every analysis cycle, it tries to find an attack by checking whether a *destination* IP address is a potential victim of an attack (*cf.* Section 4.6). Line 4 depicts the return value of the attack analysis (*cf.* Listing 5.15), and in the case that the analysis returns *false*, the loop breaks, and the next IP address is checked (*cf.* Lines $2, 5-6$). Next, the topic of the Pub/Sub message is determined (*cf.* Lines $8-14$). In the case that the *higher* threshold is violated, the *high confidence* topic is chosen, unless the AS wants to use *additional* topics (*cf.* Lines $13-14$), *e.g.,* if it is part of a sub-community. Since multiple *additional topics* can be used, the message is sent for each topic (*cf.* Lines $29-35$). Additionally, CH^{2}TF uses an environment variable for the *message length* since the list of potential attackers is included

in the request. It is fathomable that this list could get lengthy, as such, this list is split if it is larger than the threshold, and multiple requests are sent with the limited "sublists" (*cf.* Lines 18 – 21).

Moreover, Lines 26 – 27 show part of the *reputation* based-scheme (*cf.* Section 4.9). Thus, the collaboration request includes which detection case was triggered and by how much. To achieve this, the request adds the *requests_relative_to_size* variable, which includes *size* information of the AS. This is further explored in Section 5.6.

```
1  ...
2  for dest_ip, src_ips in dest_dict.items():
3      ...
4      detected, detection_case, ratio = self.attack_analysis.run_analysis
                                                 (...)
5      if not detected:
6          continue
7      ...
8      topic = TOPIC_LOW
9      if num_packets_for_this_destination > THRESHOLD_VICTIM_HI:
10         topic = TOPIC_HIGH
11     publish_topics = [topic]
12     # if true, will skip 'default' topics and send to each additional
13     if TOPICS_USE_ADDITIONAL:
14         publish_topics = TOPICS
15     potential_attacker_ips = list(dest_dict[dest_ip])
16
17     # split list into more manageable list of MSG_LENGTH
18     splitted_potential_attacker_ips = [
19         potential_attacker_ips[x : x + MSG_LENGTH]
20         for x in range(0, len(potential_attacker_ips), MSG_LENGTH)
21     ]
22     for subli in splitted_potential_attacker_ips:
23         request = DefenseCollaborationRequestData(
24             potential_attacker_ips=subli,
25             potential_victim=dest_ip,
26             request_detection=detection_case,
27             requests_relative_to_size=ratio / AS_SIZE,
28         )
29         for top in publish_topics:
30             topic = top + ".REQ"
31             self.producer.send(
32                 topic=topic,
33                 value=request.to_json(),  # type: ignore
34                 key=str.encode(request.request_id),
35             )
36
37         req_dict[str(request.request_id)] = request
38         # go directly to analysis, do not need to go through kafka
39         self.handle_collab_req(
40             def_collab_req=request, topics=publish_topics
41         )
42     ...
```

Code Listing 5.8: Part of the `run_analysis` method

Furthermore, it is also noteworthy that no *pre-processing* is done on the potential attacker's list, they are sent *as-is* in the collaboration request. Future work could try to limit this set to (*i*), remove the amounts of requests that need to be sent, and (*ii*), to reduce the amount of analyses that the collaboration request receivers must process.

5.6 Receiving Requests

The `handle_collab_req` method (*cf.* Listings 5.9 and 5.10) is called by the listener thread (*cf.* Lines 24 – 28 in Listing 5.7), or by the `run_analysis` method (*cf.* Lines 39 – 41 in Listing 5.8) in case the request comes from the same AS. This is also shown in Lines 8 – 9 in Listing 5.9 where the own request is ignored that an AS receives through Kafka. Further, the relative threshold is computed (*cf.* Lines 14 – 16 in Listing 5.9). The *raison d' être* of this is procedure is documented in the discussion of Listing 5.11.

```
1  def handle_collab_req(...):
2      def_collab_req = (
3          DefenseCollaborationRequestData.from_json(message.value)
4          if message
5          else def_collab_req
6      )
7      # ignore own request that receives through kafka consumer
8      if def_collab_req.request_originator == AS_NAME and message is not
                                            None:
9          return
10     # reputation
11     if high_prio and self.reputation_dict[def_collab_req.
                                            request_originator] > 0.5:
12         self.mitigation.filter_ips(def_collab_req.potential_attacker_ips
                                               )
13     # check relative threshold
14     is_larger_than_threshold: bool = self._is_larger_than_own_threshold(
15         def_collab_req
16     )
17     if not is_larger_than_own_threshold:
18         decision = DecisionEnum.NOT_ACK
19     else:
20         for potential_attacker in def_collab_req.potential_attacker_ips:
21             # check if ip of a potential attacker is managed by this AS.
22             if self.check_if_is_managed(potential_attacker):
23                 if self.attacker_analysis.run_analysis(...):
24                     list_ack_attacker.append(potential_attacker)
25                 else:
26                     list_not_attacker.append(potential_attacker)
27         decision = (
28             DecisionEnum.UNDER_THRS
29             if len(list_ack_attacker) == 0
30             else DecisionEnum.FOUND
31         )
32         ...
```

Code Listing 5.9: The `handle_collab_req` method (part i)

If the request receiver does not agree on an attack, no HH analysis is done, and this decision is published as a message (*cf.* Lines 17 – 18 in Listing 5.9). However, if an attack is *acknowledged*, the collaboration request is processed further (*cf.* Line 19 in Listing 5.9). As such, for each *potential* attacker IP that was provided in the Pub/Sub message, it is checked if this IP address is managed by the corresponding AS (*cf.* Line 22 in Listing 5.9). *I.e.,* this implies that only the AS that manages an IP address can check if this IP is an attacker. This is further discussed in Sections 5.3 and 5.9. Furthermore, for each IP address that *is* managed, the HH analysis is run (*cf.* Line 23 in Listing 5.9 and Section 5.8.2).

Finally, the *decision* (*cf.* Listing B.3), *i.e.,* whether the attack is not acknowledged, attackers were not found, or attackers were found is added to the response message, which is in turn published (*cf.* Lines 47 – 51 in Listing 5.10). Further, the topic is chosen based on the topic that it received. Additionally, suppose this method is handling the request from itself. In that case, it is possible that the request has been published to multiple subscribers, as such a list of topics is passed, and the response is published for each topic (*cf.* Lines 45 – 51 in Listing 5.10). Conversely, if the request message stems from Kafka, it only belongs to one topic, and for simplicity, it is handled in the same way (*cf.* Line 43 – 44 in Listing 5.10).

```
33      ...
34      def_collab_res: DefenseCollaborationResponseData = (
35          DefenseCollaborationResponseData(
36              request_id=def_collab_req.request_id,
37              ack_potential_attacker_ips=list_ack_attacker,
38              decision=decision,
39              request_originator=def_collab_req.request_originator,
40              as_name=os.getenv("AS_NAME", default=""),
41          )
42      )
43      if topics is None:  # if received through kafka topics is None
44          topics = [topic]
45      for topic in topics:
46          topic = topic + ".RES"
47          self.producer.send(
48              topic=topic,
49              value=def_collab_res.to_json(),
50              key=str.encode(def_collab_res.request_id),
51          )
```

Code Listing 5.10: The `handle_collab_req` method (part ii)

Listing 5.11 shows the `_is_larger_than_own_threshold` method, which computes if a request that an AS receives would also violate its thresholds. *I.e.,* the collaboration request from another AS is adjusted to the relative sizes. Since every AS is likely to support a different amount of traffic, this mechanism plays a role in leveling the field to make the thresholds more comparable. As such, for each received request, the *own* AS size is multiplied by the number that was passed in a request (*cf.* Line 15 in Listing 5.11 and Line 27 in Listing 5.8). This is then compared to the own thresholds, based on which threshold was violated by the request originator (*cf.* Lines 16 – 26).

```
1  def _is_larger_than_own_threshold(
2      self, def_collab_req: DefenseCollaborationRequestData
3  ) -> bool:
4      """
5      adjust the collaboration request to the relative sizes
6      => is a request originating from an AS
7      also above my threshold when adjusted to the relative sizes and
8      thresholds? Essentially, would these flows also trigger my threshold
                                              ?
9      :param def_collab_req:
10     :type def_collab_req: DefenseCollaborationRequestData
11     :return: whether this AS agrees that an attack has been detected
12     :rtype: bool
13     """
14
15     requests_relative_to_size = AS_SIZE * def_collab_req.
                                             requests_relative_to_size
16     match def_collab_req.request_detection:
17         case DetectionEnum.THRESHOLD:
18             is_larger_than_own_threshold = (
19                 requests_relative_to_size > THRESHOLD_VIC_LO
20             )
21         case DetectionEnum.TRAFFIC_INCREASE:
22             is_larger_than_own_threshold = (
23                 requests_relative_to_size > THRESHOLD_VIC_T_PERC
24             )
25         case _:
26             is_larger_than_own_threshold = False
27     return is_larger_than_own_threshold
```

Code Listing 5.11: The `_is_larger_than_own_threshold` method

5.7 Receiving Responses

Listing 5.12 shows the handling of collaboration responses. This method is called by the `listener` method (*cf.* Line 30 in Listing 5.7). Since a collaboration response includes the *decision* (*cf.* Listings B.2 and B.3), the flow depends on the decision that was included in the message (*cf.* Lines 9 – 31 in Listing 5.12). If an attack is found (*cf.* Lines 10 – 20), this method adds the acknowledged attackers to the *HH table* (*cf.* Lines 18 – 20 and also Section 5.9). Additionally, this would essentially start the mitigation process, which is not part of this work. Further, it is possible that the request receiver does not manage any potential IP addresses, in this case, nothing can be done (*cf.* Lines 29 – 31). However, future work could examine whether building a knowledge base of which IP addresses are managed by which AS is a suitable choice.

Additionally, reporting that attackers were found is seen as *positive* in the reputation scheme, as such, the reputation of the responding AS increases (*cf.* Line 16 – 17). Similarly, if an AS does not acknowledge an attack (*cf.* Lines 21 – 24, it implies a mismatch between the thresholds. Nota bene, not acknowledging an attack implies that the request receiving AS skips the HH analysis process, which could be exploited akin to the

"Free-rider problem" (*cf.* Section 4.9). The reputation-based scheme tries to counter this behavior by decreasing the reputation of the AS that responded to the attack. In the case that an attack is acknowledged, but no particular IP address is flagged as an attacker, the third case is triggered (*cf.* Lines 25 – 28). In this case, the reputation scheme does not play a role since it is perfectly possible that no IP address from this AS is partaking in the attack. Thus, without a statistical approach to estimate the likelihood of such an occurrence, it would not be correct to decrease the reputation of this AS *as-is*.

```
1  def handle_collab_res(self, message, topic):
2      ...
3      collab_res: DefenseCollaborationResponseData = (
4          DefenseCollaborationResponseData.from_json(message.value)
5      )
6      # store response
7      responses[collab_res.request_id][collab_res.as_name] = collab_res
8      given_decision: DecisionEnum = collab_res.decision
9      match given_decision:
10         case DecisionEnum.FOUND:
11             log.info("Found Attackers")
12             # mitigation starts here, but not part of this work
13             self.mitigation.filter_ips(
14                 collab_res.ack_potential_attacker_ips
15             )
16             if collab_res.request_originator == AS_NAME:
17                 self.reputation_dict[collab_res.as_name] += 0.1
18             self.h_h_table = add_to_bloom_filter(
19                 self.h_h_table, collab_res.ack_potential_attacker_ips
20             ) # add to heavy hitter table
21         case DecisionEnum.NOT_ACK:
22             # originates from this AS, build reputation scheme
23             if collab_res.request_originator == AS_NAME:
24                 self.reputation_dict[collab_res.as_name] -= 0.1
25         case DecisionEnum.UNDER_THRS:
26             # ack attack and has manages some ips
27             # claims no attackers from this AS
28             pass
29         case DecisionEnum.NOT_MANAGED:
30             # no managed ip addresses
31             pass
32     ...
```

Code Listing 5.12: The `handle_collab_res` method

5.8 Attack and Heavy Hitters Analyses

For the attack and HH analyses (*cf.* Sections 5.8.1 and 5.8.2), the prototype implementation of CH²TF makes use of an abstract base class `Analysis` (*cf.* Listing 5.13). This class serves as an interface for the `run_analysis` method, which needs to be overridden by the subclasses. Furthermore, CH²TF employs the *strategy design pattern* (*cf.* [90]) to

make the behavior generic and the extension and selection of further analyses as simple as possible.

```
1  class Analysis(ABC):
2      @abstractmethod
3      def run_analysis(
4          self,
5          attacker_ip: str,
6          victim_ip: str,
7          src_dict: defaultdict,
8          dst_dict: defaultdict,
9          *args,
10         **kwargs,
11     ) -> Any:
12         raise NotImplementedError
```

Code Listing 5.13: The abstract `Analysis` class

5.8.1 Attack Analysis

As discussed in Section 4.6, CH^{2}TF employs policies based on thresholds to detect an attack. In particular, the two policies, *i.e.,* (*i*) **Amount of packets arriving at the destination is above threshold** and (*ii*) **Increase in traffic above threshold** that were designed were implemented here. Listing 5.14 shows the subclass `AttackAnalysis`, which inherits from the `Analysis` class. The main difference between the two is that the return type of `AttackAnalysis` is more specific (*i.e., covariant*), and as such follows the Liskov Substitution Principle (*cf.* [84]).

```
1  class AttackAnalysis(Analysis):
2      @abstractmethod
3      def run_analysis(
4          self,
5          attacker_ip: str,
6          victim_ip: str,
7          src_dict: defaultdict,
8          dst_dict: defaultdict,
9          *args,
10         **kwargs,
11     ) -> Tuple[bool, DetectionEnum, float]:
12         raise NotImplementedError
```

Code Listing 5.14: The `AttackAnalysis` class

Listing 5.15 depicts the `DDoSAttackAnalysis` class, which is responsible for detecting DDoS attacks. Lines 21 – 22 represent the detection of case (*i*), *i.e.,* whether a potential IP address can be considered a *victim*, *i.e.,* is under attack or suffering from heavy load. Lines 25 – 27 are about case (*ii*), the detection of DDoS attacks by seeing a sudden high increase in traffic. As such, method `check_timed_difference` (*cf.* Lines 3 – 15 in Listing 5.15) returns whether a significant difference in traffic between the previous and the current

analysis interval has been experienced. Additionally, to be more robust, this method verifies whether the percentual traffic increase is above a given threshold and whether minimum traffic has been experienced (Lines 12 and 15). Finally, the `run_analysis` method not only returns whether an attack was detected but also the detection case and by how much the thresholds were deviated from (*cf.* Lines 22 and 28). If no attack is detected for this IP address, the method returns *False* (*cf.* Line 28).

```
1  class DDoSAttackAnalysis(AttackAnalysis):
2      @staticmethod
3      def check_timed_difference(
4          victim_ip: str, dest_dict: defaultdict, dest_dict_aggregated:
                                                    defaultdict
5      ) -> tuple[bool, float]:
6          new = dest_dict
7          old = dest_dict_aggregated
8
9          num_old = old.get(victim_ip, 0)
10         num_new = sum(new[victim_ip].values())
11
12         if num_old == 0 or num_new < THRESHOLD_VICTIM_TIME_MIN:
13             return False, 0.0
14         difference = float(num_new / num_old)
15         return difference > THRESHOLD_VICTIM_TIME_PERCENTAGE, difference
16
17     @stopwatch(name="AttackAnalysis")
18     def run_analysis(...) -> Tuple[bool, DetectionEnum, float]:
19         # case 1: amount of packets arriving is above threshold
20         ...
21         if num_packets_destination > THRESHOLD_VICTIM_LO:
22             return True, DetectionEnum.THRESHOLD,
                                                  num_packets_destination
23         # case 2: increase in traffic above threshold
24         ...
25         rel_new_requests, ratio = self.check_timed_difference(...)
26         if rel_new_requests:
27             return True, DetectionEnum.TRAFFIC_INCREASE, ratio
28         return False, DetectionEnum.NONE, 0
```

Code Listing 5.15: The `DDoSAttackAnalysis` class

5.8.2 Heavy Hitters Analysis

The HH analysis is constructed similarly to the Attack Analysis (*cf.* Section 5.8.1). Listing 5.16 shows the base `AttackerAnalysis` class that extends the `Analysis` class (*cf.* Listing 5.13). Similarly, this class serves as an *interface* for (potential) analyses.

```python
class AttackerAnalysis(Analysis):
    @abstractmethod
    def run_analysis(
        self,
        attacker_ip: str,
        victim_ip: str,
        src_dict: defaultdict,
        dst_dict: defaultdict,
        *args,
        **kwargs,
    ) -> bool:
        raise NotImplementedError
```

Code Listing 5.16: The `AttackerAnalysis` class

Listings 5.17 and 5.18 show the `HeavyHitterAnalysis` class that extends the `AttackerAnalysis` class (*cf.* Listing 5.16).

The method `_is_traffic_direction_proportional` of the class `AttackerAnalysis` is shown in Listing 5.17. This method was inspired by related work and considers whether both sides are sending traffic in similar proportions. This means that if an attacker overwhelms a victim with traffic and the victim never sends a request back, this will be flagged as suspicious. In the case that one side never sends traffic to the other, this is treated as more likelier to be suspicious (*cf.* Lines 16 – 17 in Listing 5.17).

```python
class HeavyHitterAnalysis(AttackerAnalysis):
    @staticmethod
    def is_traffic_direction_proportional(
        atk_ip: str,
        vic_ip: str,
        num_packets_from_src_to_victim_only: int,
        dst_dict: dict,
    ):
        """
        Proportionality in flow between src and destination.
        """
        num_to_vic = num_packets_from_src_to_victim_only
        num_from_vic = dst_dict[atk_ip].get(vic_ip, 0)
        # case where attacker has not received any traffic from victim
        # these are considered as likelier attackers here => weighted
                                                    10x more
        if num_from_vic == 0:
            num_from_vic = 1e-1

        # only consider the case atk_to_vic > atk_from_vic.
        # not concerned here with the victim being an attacker
        if THRESHOLD_TRAFFIC_PROPORTIONALITY <= num_to_vic /
                                                    num_from_vic:
            return False
        return True
```

Code Listing 5.17: The `HeavyHitterAnalysis` class (part i)

The four attack detection cases discussed in Section 4.7 are shown in Listing 5.18. Nota bene, the attack cases use different threshold variables and values since they are not directly comparable, and one value would not be able to distinguish between the cases. Case (*i*) **Source sends too many packets to specific victim** is shown in Lines 30 – 31. *I.e.,* if one source sends *too* many packets to the reported victim it gets flagged as an attacker. Lines 34 – 35 depict case (*ii*): **Source sends too many packets to many destinations**. This means that the source is not sending traffic to a specific victim only, but to *many*, and can thus be considered a HH. In lines 37 – 41, case (*iii*) **Source sends packets only to the victim** is shown. As discussed in Section 4.7 this is about detecting sources that send constant but small traffic to the victim, but not to others. Case (*iv*) **Traffic direction proportionality** is depicted in Lines 43 – 46, which calls the method `_is_traffic_direction_proportional` that is depicted in Listing 5.17. Lastly, if none of the cases detected an attack, the method returns *false* (*cf.* Line 47).

```
26      ...
27      def run_analysis(...) -> bool:
28          ...
29          # case 1: source sends too many packets to victim
30          if num_src_to_victim_only > THRESHOLD_SRC_1:
31              return True
32          # case 2: source sends many packets to many victims
33          ...
34          if num_from_this_src > THRESHOLD_SRC_2:
35              return True
36          # case 3: source is sends packets ONLY to the victim
37          if num_packets_from_this_src > THRESHOLD_SRC_3_MIN and (
38              (ratio := (num_src_to_victim_only / num_from_this_src))
39              >= THRESHOLD_SRC_3
40          ):
41              return True
42          # case 4: traffic direction proportionality
43          if not self.is_traffic_direction_proportional(
44              attacker_ip, victim_ip, num_src_to_victim_only, dst_dict
45          ):
46              return True
47          return False
```

Code Listing 5.18: The `HeavyHitterAnalysis` class (part ii)

However, in comparison to the attack detection analysis (*cf.* Section 5.8.1), the HH detection only returns *true or false*, *i.e.,* whether a source address can be considered an attacker, and does not compute any confidence value that it has for this result. Such an aspect would be interesting to add to *mitigation* schemes, and as such, has been directed to future work. The author also notes that attack detection case (*iv*) can be treated as a *generic* attack detection scheme but is not necessarily a *HH* detection scheme.

5.9 Bloom Filters

The prototype implementation of CH²TF uses Bloom filters for storing and handling IP addresses and the storage of *HH*. To achieve this, the Python library "python-bloomfilter" has been selected (*cf.* [57]). A Bloom filter has been chosen to reduce memory usage and the constant lookup time (*cf.* Section 2.4).

Listing 5.19 depicts the initialization of a new Bloom filter instance via the method `init_bloom_filter`. As a default value a capacity of 10′000 is given (*cf.* Line 1). This method is used to initialize the Bloom filters for the managed IP addresses and also the HH table.

```
1 def init_bloom_filter(capacity: int = 10_000):
2     return pybloom_live.BloomFilter(capacity=capacity)
```

Code Listing 5.19: The `init_bloom_filter` and methods

5.9.1 Managed IP Addresses

The prototype of CH²TF assumes that each AS keeps track of the *managed* IP addresses in a text file. As such, the Bloom filter is initialized by reading the entries of this very file (*cf.* Lines 16 – 18 in Listing 5.20). Additionally, if hashes are to be used to preserve privacy, each entry is *hashed* first (*cf.* Lines 19 – 20) before being added to the Bloom filter (*cf.* Line 21).

```
1  def init_managed_ips(
2      managed_ip_path: str, is_use_hash: bool, capacity: int = 100_000
3  ) -> pybloom_live.BloomFilter:
4      """
5      Initializes the bloom filter by adding the managed ip addresses
6      :param is_use_hash: whether the managed ips should be hashed
7      :type is_use_hash: bool
8      :param managed_ip_path: path of textfile containing ips
9      :type managed_ip_path: str
10     :param capacity: capacity of the bloom filter
11     :type capacity: int
12     :return: populated bloom filter
13     :rtype: pybloom_live.BloomFilter
14     """
15     bloom_filter = init_bloom_filter(capacity)
16     with open(managed_ip_path, mode="r", encoding="utf-8") as f:
17         for line in f:
18             entry = line.rstrip("\n")
19             if is_use_hash:
20                 entry = sha3_hash(entry)
21             bloom_filter.add(entry)
22     return bloom_filter
```

Code Listing 5.20: The `init_managed_ips` method

The method `_check_if_is_managed` (*cf.* Listing 5.21) checks whether a given element (*i.e.,* IP address) is contained in the bloom filter. However, as outlined in Section 2.4, *false positives* are possible. However, *false positives* are not an issue, since the analysis return would treat a non-managed IP address as a non-attacker.

```
1  def _check_if_is_managed(self, ip_address: str) -> bool:
2      """
3      Checks whether a given ip_address is probably managed by the AS.
4      This is done by verifying if it is contained in the bloom filter.
5
6      Due to the bloom filter's working, false positives are possible.
7
8      :param ip_address: ip address to check if is in bloom filter
9      :type ip_address: str
10     :return: whether ip_address is in the bloom filter
11     """
12     return ip_address in self.managed_ips
```

Code Listing 5.21: The `_check_if_is_managed` method

5.9.2 Heavy Hitter Tables

Furthermore, each AS keeps track of the reported *HH* from the collaborating network. Similarly to the managed IP addresses, a Bloom filter data structure was chosen as an implementation detail to store the *HH*. The addition to the table is depicted in Lines $18 - 20$ in Listing 5.12. While the HH tables enable the DDoS mitigation process, this has been directed to future work.

Chapter 6

Evaluation

In this chapter, the prototype is evaluated based on various aspects. First, in Section 6.1, multiple attack cases are explored. Subsequently, in Section 6.2 a performance evaluation is performed. Finally, the evaluation findings are discussed in Section 6.3. The evaluations described in Sections 6.1 and 6.2 have been conducted on a MacBook Pro (Apple M1 Max, 64GB system memory), and three ASes have been modeled using separate Docker Containers.

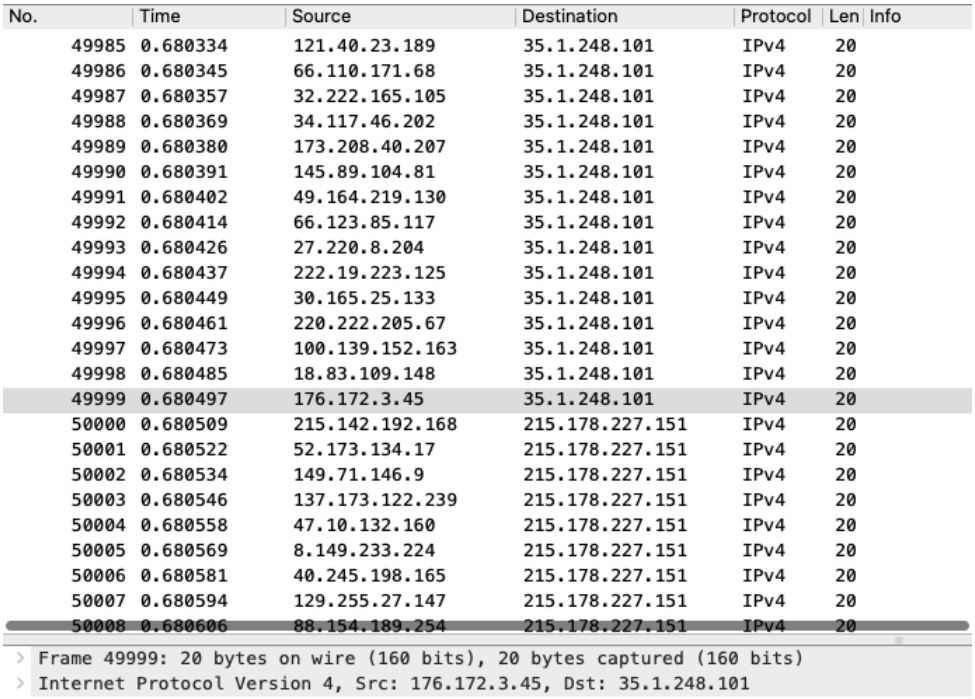
6.1 Attack Cases

This section describes and simulates the DDoS attack scenarios to evaluate CH^{2}TF's effectiveness in detecting an attack and the HH. The chosen attack types were inspired by the articles [12, 13].

To conduct the evaluation experiments, for every modeled AS, the *managed* IP addresses were selected and written into a text file passed to each ASes' Docker Container. The number of managed IP addresses was arbitrarily chosen at 10′000 for each AS, and it was ensured that no duplicates between the IP addresses exist. Furthermore, this process was only done once, since there was no need to vary the IP addresses between the experiments.

Subsequently, for every attack scenario and each modeled AS PCAP (*cf.* [86]), files are generated using the Python library *Scapy* (*cf.* [71]), one for *legitimate traffic* and one for *illegitimate traffic*. Moreover, if a packet goes from an AS to another AS, it requires that the PCAP files for both ASes include this traffic. PCAP files, which include the packet capture data [86], are sequentially read from a traffic generator module and passed to the CH^{2}TF module for analysis. Thus, specific attack scenarios can be replayed for evaluation purposes using packet captures. An example of such a PCAP file used for the scenario in Section 6.1.2 is displayed in Figure 6.1. These traffic flows were computed using the aforementioned *managed IP addresses* files.

The evaluation architecture that is used in the attack case scenarios is depicted in Figure 6.2. Three different ASes have been modeled using Docker Containers. Furthermore,



No.	Time	Source	Destination	Protocol	Len	Info
49985	0.680334	121.40.23.189	35.1.248.101	IPv4	20	
49986	0.680345	66.110.171.68	35.1.248.101	IPv4	20	
49987	0.680357	32.222.165.105	35.1.248.101	IPv4	20	
49988	0.680369	34.117.46.202	35.1.248.101	IPv4	20	
49989	0.680380	173.208.40.207	35.1.248.101	IPv4	20	
49990	0.680391	145.89.104.81	35.1.248.101	IPv4	20	
49991	0.680402	49.164.219.130	35.1.248.101	IPv4	20	
49992	0.680414	66.123.85.117	35.1.248.101	IPv4	20	
49993	0.680426	27.220.8.204	35.1.248.101	IPv4	20	
49994	0.680437	222.19.223.125	35.1.248.101	IPv4	20	
49995	0.680449	30.165.25.133	35.1.248.101	IPv4	20	
49996	0.680461	220.222.205.67	35.1.248.101	IPv4	20	
49997	0.680473	100.139.152.163	35.1.248.101	IPv4	20	
49998	0.680485	18.83.109.148	35.1.248.101	IPv4	20	
49999	0.680497	176.172.3.45	35.1.248.101	IPv4	20	
50000	0.680509	215.142.192.168	215.178.227.151	IPv4	20	
50001	0.680522	52.173.134.17	215.178.227.151	IPv4	20	
50002	0.680534	149.71.146.9	215.178.227.151	IPv4	20	
50003	0.680546	137.173.122.239	215.178.227.151	IPv4	20	
50004	0.680558	47.10.132.160	215.178.227.151	IPv4	20	
50005	0.680569	8.149.233.224	215.178.227.151	IPv4	20	
50006	0.680581	40.245.198.165	215.178.227.151	IPv4	20	
50007	0.680594	129.255.27.147	215.178.227.151	IPv4	20	
50008	0.680606	88.154.189.254	215.178.227.151	IPv4	20	

Figure 6.1: Excerpt from the evaluation attack PCAP file inside Wireshark (*cf.* [94]) that was used for the attack case in Section 6.1.2.

Kafka and ZooKeeper were also deployed using Docker Containers and connected in a Docker Compose Network (*cf.* [19]). In addition to the CH^{2}TF architecture, an evaluation module was added. Its purpose as an additional consumer is to capture the requests and responses sent back and forth, aggregate the evaluation results, and generate the traffic PCAP files and the resulting figures. Due to the use of Pub/Sub and Kafka, the evaluation module has readily been added as an additional Kafka Consumer without requiring any changes to the modeled ASes.

No packet sampling has been used in these scenarios to remove randomness from the evaluation. Furthermore, the attack rate and thresholds have been kept constant between the attack scenarios to make the attack case results comparable. Also, for all attack cases 5 traffic files were each generated, and the results were aggregated over 5 different experiment runs. The Docker Compose YAML file is depicted in Listing C.1 in Appendix C.

6.1.1 Volumetric Attack

This attack scenario aims to overwhelm a specific victim by the sheer amount of traffic [48]. As such, for this attack case, 1000 attackers from each AS were selected to target a single victim during the entirety of the attack duration and written into the traffic files. The results of this evaluation case are displayed in Figure 6.3 and in Table 6.2. For this experiment, $n = 30$ requests and their responses from the collaborating ASes were listened to and aggregated.

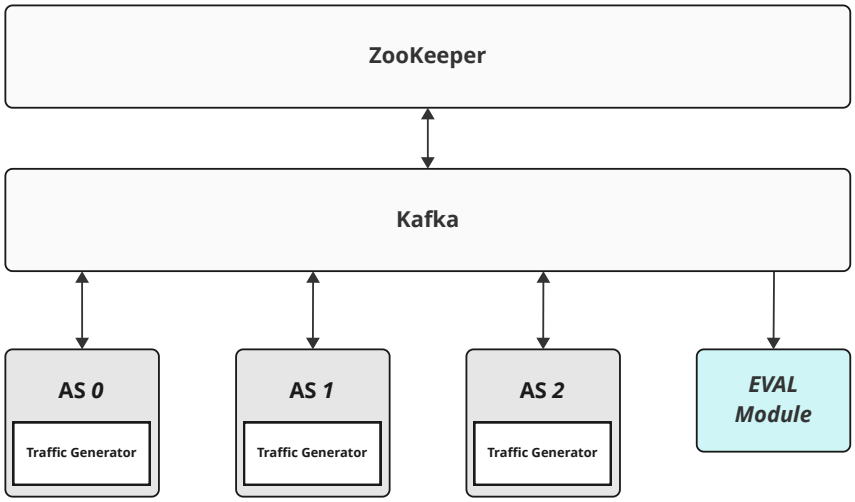


Figure 6.2: Schema of the evaluation architecture used for the attack scenarios. The evaluation module has been added here as an additional Kafka consumer. Also, every AS includes a Traffic Generator that reads the prepared PCAP files to simulate traffic.

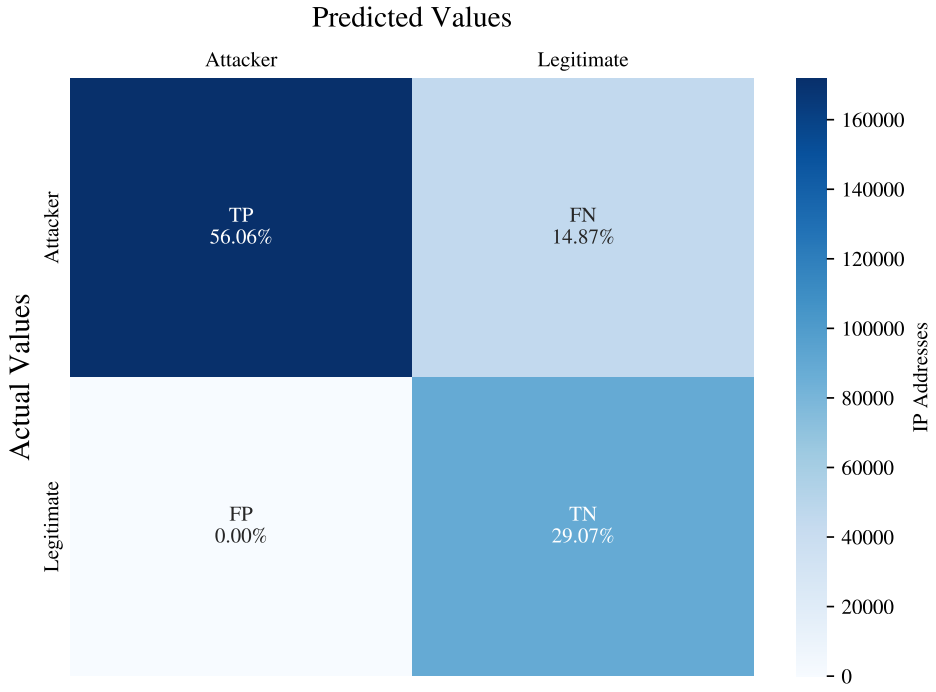


Figure 6.3: Confusion matrix for the volumetric attack case.

6.1.2 Burst / Pulse Wave Attack

This type of attack can be seen as a high-volume volumetric attack, though differs by being for only a short duration of time [1, 50]. Furthermore, they exploit the existing problem of state of the art DDoS defenses, namely their reaction time to these kind of attacks [1]. Thus, the simulated *attack* traffic has been specifically created to model this

scenario, and four victims were selected for this evaluation, and 1000 attackers switched to the next victim cyclically after a specific time. The selected cycle was 5000 packets, meaning that after an AS sent this amount of traffic to the victim, it switched to the next victim.

For this experiment, $n = 30$ requests and their responses from the collaborating ASes were listened to and aggregated. The results of this attack case are displayed in Figure 6.4 and in Table 6.2, where it is compared against the other attack case results.

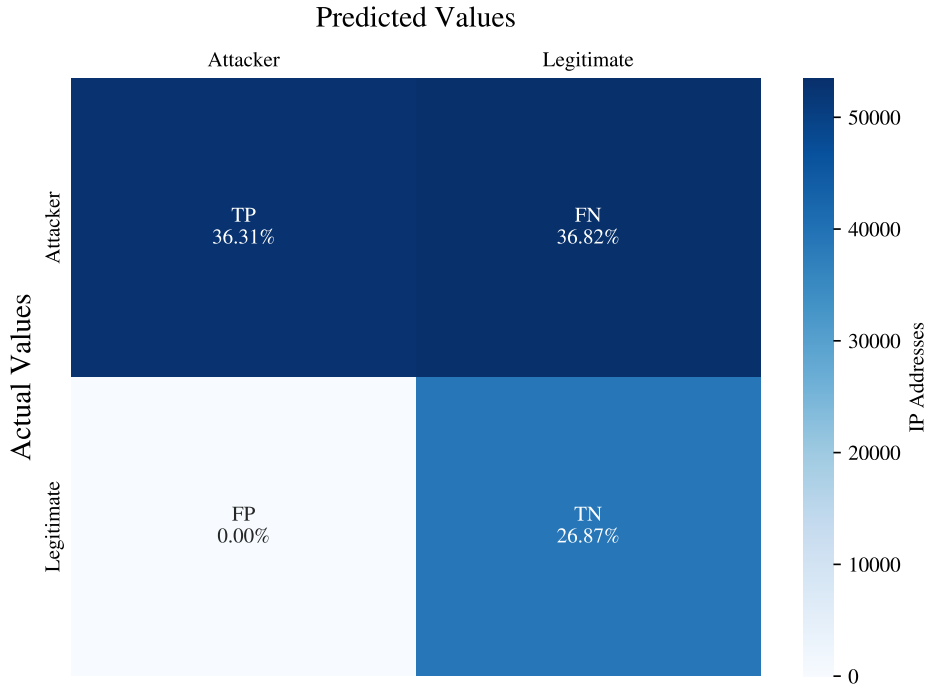


Figure 6.4: Confusion matrix for the burst attack case.

6.1.3 Botnets

This attack case models a botnet attack, *e.g.,* using insecure IoT devices that form part of a larger botnet as an attack vector [15]. The idea behind this attack scenario evaluation experiment is that each attacker sends traffic only in a sporadic amount of time to hide against detection by not appearing suspicious. This is in tune with IoT devices being low-bandwith [78]. Thus, 5000 attackers were selected from each AS to target a specific victim, and for this experiment, $n = 30$ requests and their responses from the collaborating ASes were listened to and aggregated. The results of the botnet attack case are depicted in Figure 6.5 and Table 6.2, comparing it against the other attack cases.

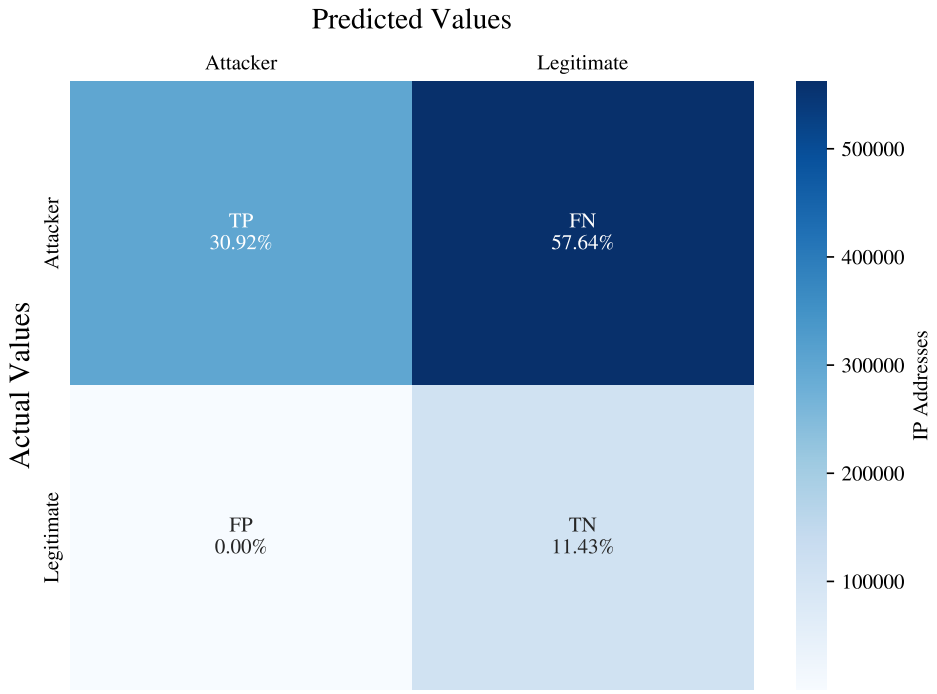


Figure 6.5: Confusion matrix for the botnet attack case.

6.2 Performance

In this section, the performance of CH^{2}TF is conducted. Since Kafka's performance has repeatedly been evaluated (*cf.* [33, 35]), this section focuses on two main aspects that are critical for CH^{2}TF's performance and not on the distributed Pub/Sub aspects. In particular, the main aspects to examine are the **DDoS Detection Analysis** and the **Heavy Hitter Detection Analysis** due to their importance in the system. Each analysis focuses on a specific IP address: a victim of an attack or an attacker. Thus, this evaluation aims to measure how long each analysis takes and whether the results indicate that CH^{2}TF would scale to millions or more packets per second.

Hence, an attack was simulated for this evaluation experiment, similar to those conducted in Section 6.1. The analysis functions were timed to conduct the measurements, and the result of $n = 200$ analyses for each detection was read from an ASes log file. The results of the evaluation experiments are displayed in Figures 6.6 and 6.7, respectively. In Table 6.1 the mean and standard deviation of the results is depicted.

Table 6.1: Mean (μ) and standard deviation (σ) of the analysis measurement times. Units in μs.

Analysis	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$
Attack Detection	2.34	1.11
Heavy Hitter Detection	28.13	6.18

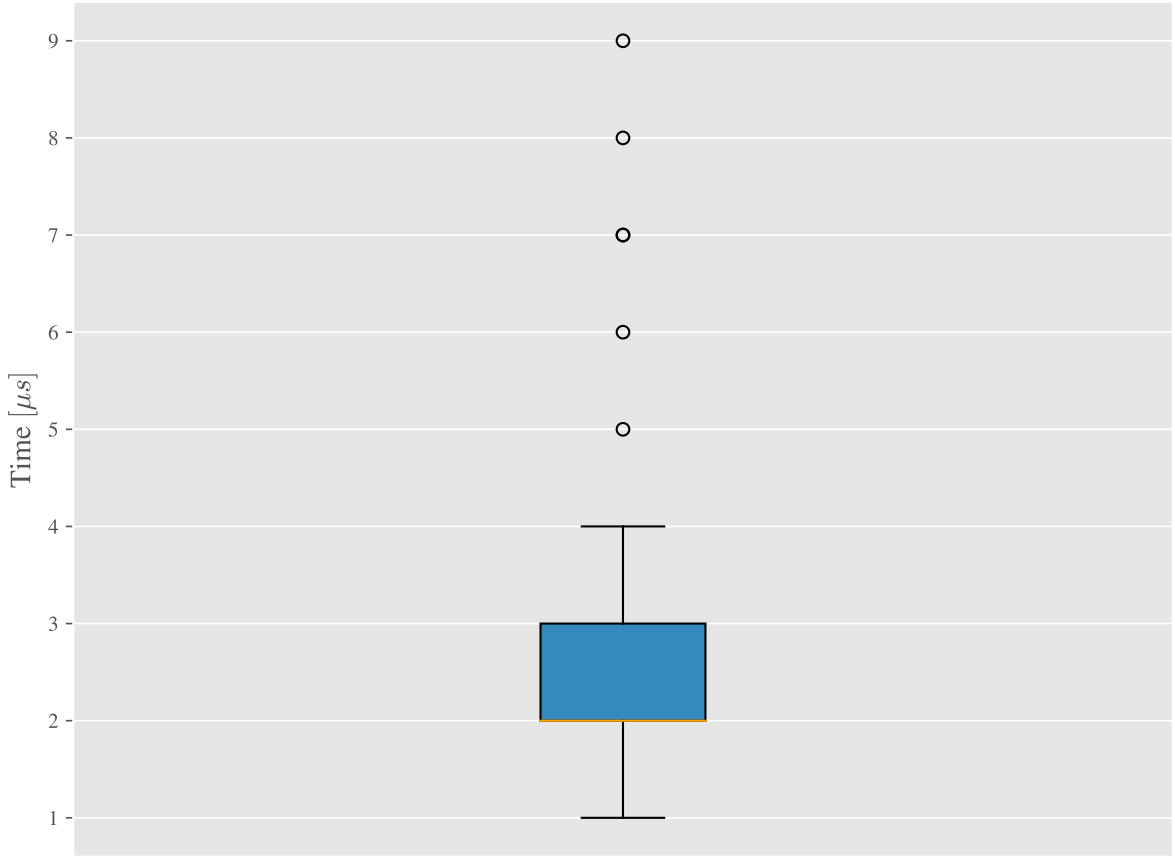


Figure 6.6: Boxplot for the Attack Detection analysis. Time in μs.

6.3 Discussion

The performance evaluation results (*cf.* Figures 6.6 and 6.7) showed a noticeable discrepancy between the Attack Detection analysis and the Heavy Hitter Detection analysis. This is not unsurprising since the latter has specifically been designed to be more "in-depth" than the former and, thus, is more complex in the computation steps that are performed during the analysis. It can be seen that the Attack Detection Analysis has a low median, though in rare cases, higher outliers are possible. Nevertheless, the low measurement times indicate that this analysis can be performant. This is important and has also been specifically designed to achieve this goal since, in real-world usage, many IP addresses would need to be sequentially analyzed to detect an attack. Conversely, the Heavy Hitter Detection Analysis is comparatively slower. Since this analysis only gets triggered once a potential attack has been flagged, it is not running continuously, unlike the Attack Detection Analysis. This implies that the effect of this analysis being slow would only relate to how fast the attack can be mitigated. The results indicate a potential bottleneck in the prototype regarding this analysis in case a high number of potential attackers would need to be analyzed. This can be seen as an inherent trade-off aspect between performance (and, as such, latency) and effectiveness. *I.e.,* a highly complex analysis can be seen as more likely to catch every attacker but would require more computation power. This discussion is further explored in Chapter 7.3 regarding botnet (IoT) attacks.

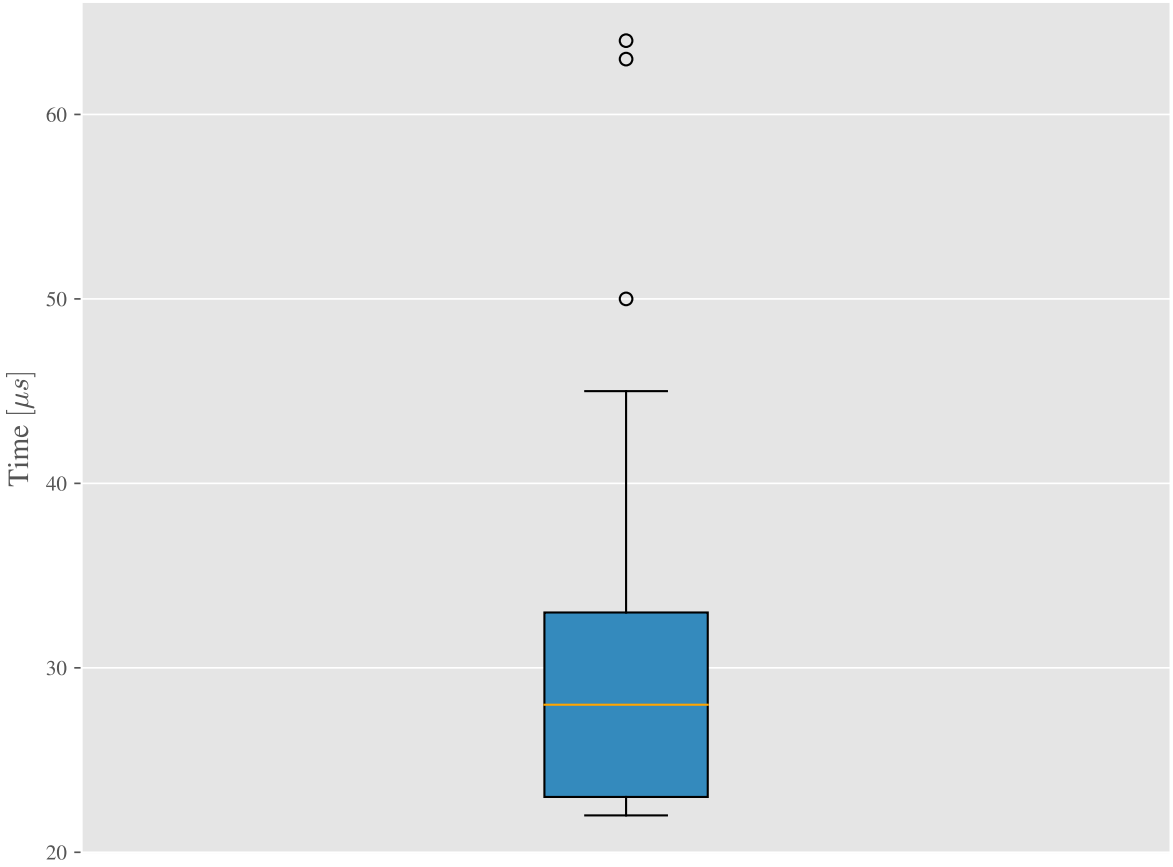


Figure 6.7: Boxplot for the Heavy Hitter Detection analysis. Time in μs.

Nevertheless, the performance evaluation results are only indicative of the performance of the prototype that has been written in Python, a high-level language [89], and CH^{2}TF's prototype has not been optimized to be as performant as possible, but rather serves as a demonstration of the underlying scheme and protocol. As such, a real-world deployment should require a highly optimized implementation. Furthermore, these evaluation results are also highly affected by the traffic-generating module that reads the PCAP files to simulate the traffic and send the traffic to CH^{2}TF. This traffic generation requires comparatively higher CPU usage (*cf.* Figures A.1 and A.2 in Appendix A). Since the analysis can not be performed without traffic, it is impossible to evaluate the analyses' executions in a vacuum.

The attack case evaluation results showed that while HH can successfully be detected, *i.e.,* the largest flows, this work struggles in detecting attackers that employ low attack rates each, similar to botnets.

This observation is shown in Figure 6.8, where it is noticed that if each attacker keeps their attack rates low, the attack is noticed, and collaborators respond to the request, though no specific attackers are to be found since each of them only contributes for a small part in the overall flows. This conclusion is also depicted in Table 6.2, where the volumetric attack case obtained the highest results, while the attackers in the botnets attack scenario went mostly undetected, with a high number of False Negatives (FN) and a low number of

Table 6.2: MCC and Accuracy (ACC) comparison for the attack cases. Computed using Equations 6.1 and 6.2. For each attack case 30 requests and responses were listened to, and averaged over 5 different experiment runs. MCC uses the range $[-1, 1]$ (*cf.* [87]), and ACC uses the range $[0, 1]$ (*cf.* Equation 6.2).

Case	MCC	ACC
Volumetric	0.72	0.85
Bursts	0.46	0.63
Botnets	0.24	0.42

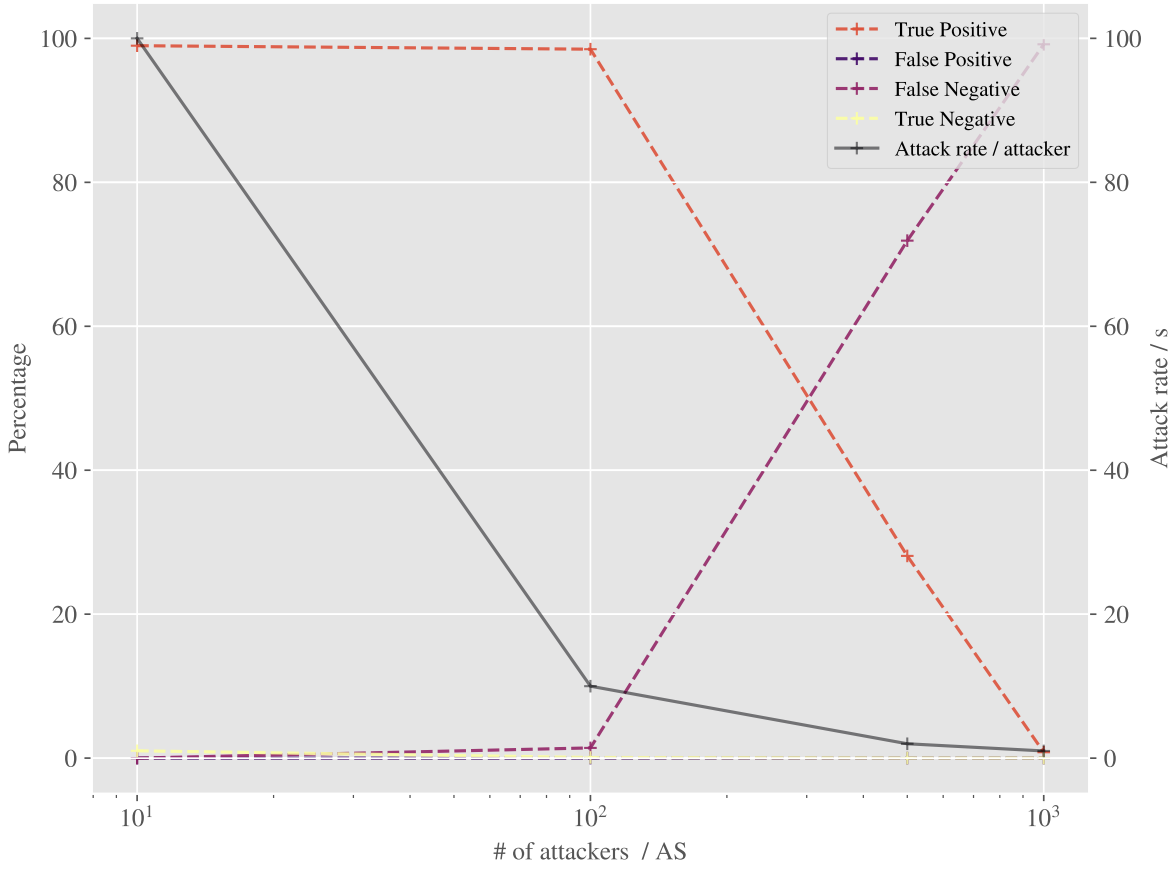


Figure 6.8: Effect of increasing the number of attackers while keeping the overall attack rate constant in the system. Further, policies' thresholds were kept constant between the experiments. For each number of attackers, $n = 100$ different requests and their responses were listened to for the evaluation.

True Positives (TP) (*cf.* Figure 6.5). This implies that the chosen policies are not precise enough to successfully differentiate between legitimate and illegitimate traffic since both take on the same form, *i.e.,* have the same behavior from the policies' point of view.

Furthermore, Figure 6.8 also shows that when the percentage of TP drops off, and inversely the percentage of FN rises, the percentage of False Positives (FP) is constantly kept at 0, *i.e.,* the *legitimate* traffic has never been classified as *illegitimate* attack traffic.

Consequently, this also highlights a potential issue in the choice of the thresholds' values for the detection of attackers, and opens the discussion on whether having *legitimate* traffic being classified as *illegitimate* (*i.e.,* a FP), is inherently a problem in such a system. Nevertheless, this topic of discussion can be seen as more appropriate in the context of the discussion of the *mitigation* of DDoS attacks, and depending on the mitigation scheme used, completely blocking off legitimate traffic might be the wrong choice. Accordingly, it can be argued that it is not the task of the detection framework to decide on such matters, and having high confidence in the choice of attackers is the more apt choice.

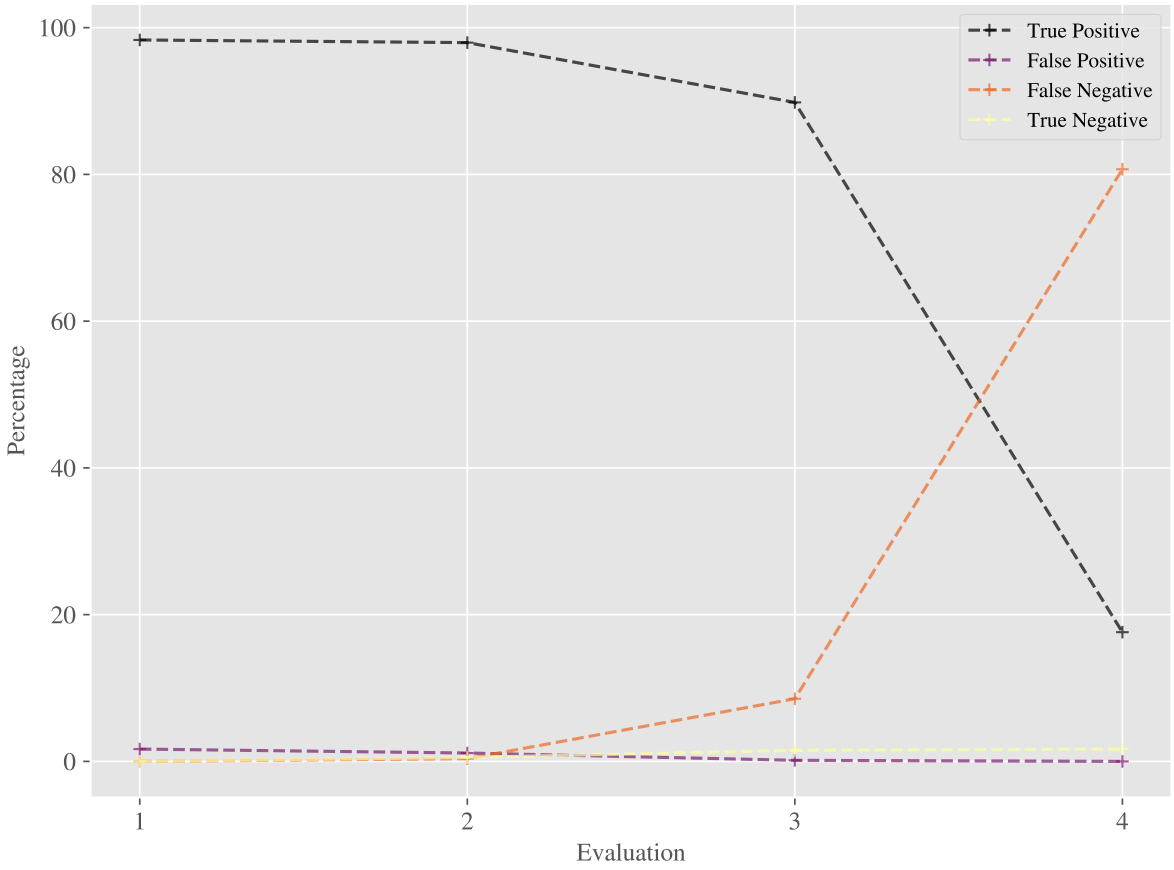


Figure 6.9: The effect of scaling the policies' thresholds, *ceteris paribus*, over four experiments, with each $n = 100$ requests and responses. *I.e.,* experiment 1 had the lowest settings, while experiment 4 used the most stringent thresholds.

Additionally, to highlight the inherent challenge of setting appropriate policies' thresholds, Figure 6.9 depicts the effect of using different thresholds (shared by all ASes) over four different experiments. The thresholds have been scaled, while all other experiment settings have been kept constant over experiments. This figure shows that a range of effective thresholds and a range of ineffective threshold settings exists. It can be observed, that when FN rises, inversely TP drops. This is clear, since both are related to each other by the threshold baseline that acts as the binary classifier. Besides, though a bit more subtle in difference, the same observation can be made about TN and FP. Therefore, if the thresholds are too low, *legitimate* traffic is classified as *illegitimate*, and vice-versa, if the thresholds are too high, *illegitimate* traffic is classified as *legitimate*.

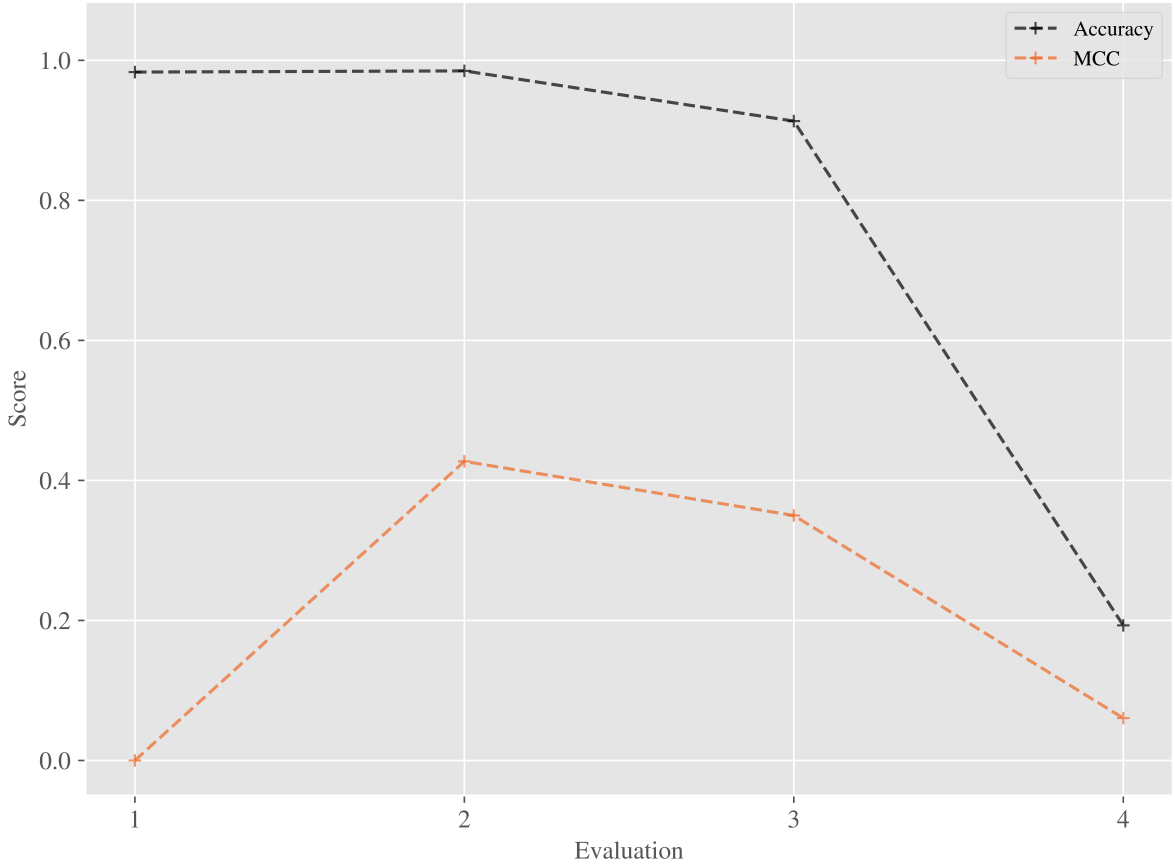


Figure 6.10: Accuracy and MCC comparison by varying the threshold levels over four experiments (*cf.* Figure 6.9). *I.e.,* experiment 1 had the lowest settings, while experiment 4 used the most stringent thresholds.

Furthermore, in Figure 6.10, Accuracy and Matthews Correlation Coefficient (MCC) are plotted for the very experiments shown in Figure 6.9. These were computed using the Equations 6.1 and 6.2, [87]:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{6.1}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.2}$$

While the accuracy is the highest for the lowest threshold settings, it results in 0.0 for the MCC, implying that it is no better than a random prediction (*cf.* [87]). This can be explained by the fact that in this experiment, no True Negatives (TN) were obtained (*cf.* Figure 6.9) since every *legitimate* traffic was (incorrectly) predicted as *illegitimate* traffic, *i.e.,* a FP.

Also, a class imbalance is obtained due to the much larger number of attack traffic vs. normal traffic in the experiments, making the high accuracy result misleading [87]. Thus, in this case MCC is more informative since it considers both the positive and negative cases [87], explicitly showing that having the lowest possible threshold settings is not a good solution for the system and leads to a negative impact. Unsurprisingly, the highest threshold values also do not lead to good results due to the high number of FN that impact the scores negatively (*cf.* Equations 6.1 and 6.2). Nonetheless, the scores displayed in Figures 6.9 and 6.10 are not to be seen as the CH^{2}TF's maximum capability, but rather serve as display purposes for the underlying challenges at hand of choosing the right thresholds. Thus, the baseline of the thresholds that were then scaled between the experiments was arbitrarily chosen and not fine-tuned to their potential.

The Burst attack scenario's result lies between the other cases (*cf.* Table 6.2). Nonetheless, this result must be considered carefully. Since CH^{2}TF allows each AS to control the analysis period, cases could artificially be constructed where attacks would go unnoticed. This implies that if attackers knew which analysis periods the individual ASes use, attacks could be formulated and designed to avoid detection if *a priori* knowledge that CH^{2}TF is in use is present. Taking this argument further, the same point could be made for the policies' thresholds.

Consequently, the evaluation has shown that for an effective system, the thresholds must be set appropriately and consider the rates each individual AS can handle. Thus, collaborating parties must agree on the definitions of the thresholds. Unsurprisingly, this can be seen as a detrimental shortcoming since deploying such a collaborative system would require meticulous fine-tuning of the parameters and various evaluation rounds to ensure it reaches its highest potential.

In summary, the evaluation showed that the requirements outlined in Section 4.1 were fulfilled. *I.e.,* **R1** required a time-efficient and data-oriented algorithm to find the HH, which was fulfilled by the results from experiment 1 (*cf.* Section 6.2). Further, **R2**, which required the HH to be shared with the other collaborating instances, is implicitly fulfilled by the evaluation experiment setup used for the experiments in Section 6.1, where the HH were shared. **R3** requires support for multiple Pub/Sub topics. This has not explicitly been evaluated, but is also supported by CH^{2}TF (*cf.* Section 5.5).

Chapter 7

Final Considerations

In this final chapter, the thesis is drawn to a close by introspecting the objectives, contributions, and considerations that have been regarded. As such, a summary of the thesis and its contributions are presented in Section 7.1. Further, the considerations are discussed in Section 7.2. Finally, possible future work and extensions of the proposed work are described in Section 7.3.

7.1 Summary

In summary, CH^{2}TF explores the topic of *collaboration* between parties in the realm of DDoS defenses in multiple dimensions to enable the filtering of traffic. Thus, in this thesis a collaborative HH detection protocol has been proposed, implemented, and finally, evaluated. To achieve this goal, a related work review has been conducted in the scope of DDoS detections, collaboration, and cooperation schemes in the realm of DDoS defenses and signaling of DDoS information to understand and build upon the state of the art of these topics.

Thus, the major contributions of CH^{2}TF are threefold:

(i) **Collaboration Protocol** to facilitate DDoS defense

(ii) **Detection** of *attacks* and *heavy hitters*

(iii) **Open Source** prototype

Contribution (*i*) includes the **Collaboration Protocol** that employs Pub/Sub to enable and facilitate the collaboration between the participating parties (*cf.* Section 4.4), (*ii*) proposes analysis policies for the **Detection** of attacks and HH that are based on threshold values (*cf.* Sections 4.6 and 4.7), and (*iii*) is the **Open Source** publication of the prototype. Additionally, the design and prototype implementation took privacy aspects in mind and also allows the collaborating parties to form collaborating sub-communities.

Furthermore, the evaluation experiments included three parties that shared attack information collaboratively, and the results of the prototype have shown that the HH of specific attack cases (*i.e., volumetric attacks*) can successfully be detected with sufficiently high accuracy (0.85), though the prototype does not fare that well in other specific attack detection scenarios (*i.e., botnets*, 0.42). Moreover, the evaluation results showed that the analyses are performant and expected to scale well.

7.2 Considerations

It is to be considered, that in CH^{2}TF ASes have been selected as collaborating parties, similarly to related work. By taking the source perspective of the traffic flow in the network, it facilitates the detection of HH of an attack that are managed by each AS. However, it is to be expected that if a system such as CH^{2}TF is *not* deployed at the edge of a network and multiple traffic paths exist, the detection of HH is more complex and not as straightforward. In this case, aggregation would be required and the communication protocol would need to be extended to handle such cases.

Furthermore, the evaluation showed that the analysis policies based on *thresholds* are not without issues. In fact, it is not *trivial* to come up with threshold values that *(a)* capture all kinds of attack scenarios, and *(b)* are shareable between the collaborating parties. Thus, *global* threshold values are not to be expected to produce satisfactory results if the parties themselves are not *homogeneous* and are able to handle various amounts of traffic. As such, these policies are up to discussion to the parties, and an agreement needs to be reached on their definitions. Nota bene, CH^{2}TF lets every party define their own threshold values, as such *global* thresholds are not in use. Nevertheless, each collaborating party that would want to join would need to know or find out which values are *optimal* for its own capabilities. This would require discussion with the other parties or a *global* body that is responsible for this task and manages such discussions. Nevertheless, these problems related to thresholds are not a problem of the prototype and CH^{2}TF *per se*, but an inherent problem of using threshold values. However, it was also noticeable that threshold-based analyses are performant and thus expected to scale well.

Additionally, *privacy*, *trust*, and *regulations* play a role in collaborating environments. While CH^{2}TF and (some) related work have a solution for these issues, they can not be perfectly solved *per se*. For instance, it is fathomable that new regulation changes (*e.g.,* politically motivated) would be able to restrict information sharing between collaborating parties, leading to a weakening of the effectiveness of systems such as CH^{2}TF. Also, the *trust* factor between the collaborating parties must be considered. CH^{2}TF assumes a certain level of trust between the ASes, and includes a reputation-based scheme to prevent the "Free-riding Problem" in certain dimensions, but is unfortunately not unexploitable against the sharing of false information and malicious behavior.

Moreover, while this work supports the flexibility of additional, non CH^{2}TF-standard Pub/Sub topics that would lead to the forming of sub-communities, it is not immediately clear whether this feature is detrimental to the system. Due to CH^{2}TF proposed heavy hitter detection scheme, a large body of collaborating parties would be more effective

than various disjoint sub-groups that do not communicate and collaborate with the other groups. Indeed, without communication efforts, they would not be aware of the *global* view of the network but only see part of it. Nevertheless, this feature implies that if parties trust each other, but do not want to partake in the (standard) collaborating network, they would still be able to benefit from CH^{2}TF's DDoS defense.

7.3 Future Work

Regarding future work, a few selected points are to be considered, which are discussed below:

(i) **Real-world deployment**

(ii) **ML-based detections and alternative analyses**

(iii) **Dynamic threshold values**

(iv) **Mitigation extension**

CH^{2}TF has been evaluated using *simulated* DDoS attacks and has been deployed in Docker containers. Future work could look into the deployment in a *real-world* system. Also, the prototype has been implemented in a high-level programming language, it is fathomable that for a *real-world* deployment it would need to be highly optimized.

Furthermore, the analysis for the attack and attackers is based on thresholds. Related work has shown that for instance for IoT-based botnet attacks the detection of attackers can be performed accurately via ML. However, it is not clear whether ML-based solutions are able to perform the analyses in *real-time* which would be required for an effective mitigation. Nevertheless, future work could investigate on additional policies or approaches for attack or attacker analyses that conform with CH^{2}TF.

Besides, since thresholds mismatches are possible and detrimental to the system, future work could extend CH^{2}TF's proposed protocol to keep track of individual ASes thresholds and essentially introduce *dynamic* thresholds that are adjusted based on the responses of each individual collaborating party.

As has been hinted in CH^{2}TF, the mitigation aspect of DDoS attacks has been directed to future work. The collaboration protocol proposed in CH^{2}TF could be extended to perform and start such a scheme once the HH are known (or acknowledged) by the collaborating parties.

Bibliography

[1] Albert Gran Alcoz, Martin Strohmeier, Vincent Lenders, and Laurent Vanbever. Aggregate-based congestion control for pulse-wave DDoS defense. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 693–706, 2022.

[2] ANS Internet Storm Center. DShield, 2022. `https://www.dshield.org`, Last visit July 12, 2022.

[3] Apache Kafka Github Contributors. Apache Kafka Github, 2022. `https://github.com/apache/kafka`, Last visit July 31, 2022.

[4] Apache Software Foundation. Apache Kafka, 2022. `https://kafka.apache.org/`, Last visit July 31, 2022.

[5] Apache Software Foundation. Apache ZooKeeper, 2022. `https://zookeeper.apache.org/`, Last visit July 31, 2022.

[6] Katerina J. Argyraki and David R. Cheriton. Active internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX annual technical conference, general track*, volume 38, 2005.

[7] Tapan Avasthi. Guide to Setting Up Apache Kafka Using Docker, 2021. `https://www.baeldung.com/ops/kafka-docker-setup`, Last visit July 31, 2022.

[8] Narmeen Zakaria Bawany, Jawwad A. Shamsi, and Khaled Salah. DDoS attack detection and mitigation using SDN: methods, practices, and solutions. *Arabian Journal for Science and Engineering*, 42(2):425–441, 2017.

[9] João Ceron, Pim van Stam, Gerald Schaapman, and Cristian Hesselman. New DDoS classifiers for the DDoS Clearing House, 2022. `https://www.concordia-h2020.eu/blog-post/new-ddos-classifiers-for-the-ddos-clearing-house/`, Last visit July 2, 2022.

[10] Niklas Christensen. MISP Engine to Assess and Evaluate Threat Events Based on Data Quality. Master's thesis, Aalborg University Copenhagen, Copenhagen, June 2021.

[11] Molly Clancy. What's the Diff: Programs, Processes, and Threads, 2022. `https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/`, Last visit November 26, 2022.

[12] Cloudflare. DDoS attack trends for 2022 Q1, 2022. https://blog.cloudflare.com/ddos-attack-trends-for-2022-q1/, Last visit November 1, 2022.

[13] Cloudflare. DDoS attack trends for 2022 Q2, 2022. https://blog.cloudflare.com/ddos-attack-trends-for-2022-q2/, Last visit November 1, 2022.

[14] Cloudflare. Smurf DDoS attack, 2022. https://www.cloudflare.com/learning/ddos/smurf-ddos-attack/, Last visit July 22, 2022.

[15] Cloudflare. What is a DDoS botnet?, 2022. https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/, Last visit November 16, 2022.

[16] Concordia Consortium. Concordia, 2022. https://www.concordia-h2020.eu/, Last visit July 2, 2022.

[17] DDoS Clearing House Github Contributors. DDoS Clearing House, 2022. https://github.com/ddos-clearing-house, Last visit July 2, 2022.

[18] Docker Inc. Docker, 2022. https://www.docker.com/, Last visit July 25, 2022.

[19] Docker Inc. Networking in Compose, 2022. https://docs.docker.com/compose/networking/, Last visit November 21, 2022.

[20] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine Learning DDoS Detection for Consumer Internet of Things Devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. IEEE, 2018.

[21] Emitter Github Contributors. Emitter Github, 2021. https://github.com/emitter-io/emitter, Last visit July 25, 2022.

[22] Emitter Studios B.V. Emitter, 2018. https://emitter.io, Last visit July 25, 2022.

[23] Laura Feinstein, Dan Schnackenberg, Ravindra Balupari, and Darrell Kindred. Statistical approaches to DDoS attack detection and response. In *Proceedings DARPA information survivability conference and exposition*, volume 1, pages 303–314. IEEE, 2003.

[24] Shahabeddin Geravand and Mahmood Ahmadi. Bloom filter applications in network security: A state-of-the-art survey. *Computer Networks*, 57(18):4047–4064, 2013.

[25] Thomer M. Gil and Massimiliano Poletto. MULTOPS: A Data-Structure for bandwidth attack detection. In *10th USENIX Security Symposium (USENIX Security 01)*, Washington, D.C., August 2001. USENIX Association.

[26] John Hawkinson and Tony Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930, 1996.

[27] IBM Cloud Education. Message Brokers, 2020. https://www.ibm.com/cloud/learn/message-brokers, Last visit July 24, 2022.

[28] IETF. DDoS Open Threat Signaling (dots), 2022. https://datatracker.ietf.org/wg/dots/about/, Last visit August 8, 2022.

[29] Kafka-Python Github Contributors. Kafka-Python, 2022. `https://github.com/dpkp/kafka-python`, Last visit November 26, 2022.

[30] Vladimir Kaplarevic. How to Set Up and Run Kafka on Kubernetes, 2020. `https://phoenixnap.com/kb/kafka-on-kubernetes`, Last visit July 31, 2022.

[31] Xin Zhe Khooi, Levente Csikor, Jialin Li, Min Suk Kang, and Dinil Mon Divakara. Revisiting Heavy-Hitter Detection on Commodity Programmable Switches. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pages 79–87. IEEE, 2021.

[32] Oliver Kim and Mark Walker. The free rider problem: Experimental evidence. *Public choice*, 43(1):3–24, 1984.

[33] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A Distributed Messaging System for Log Processing. In *Proceedings of the NetDB*, volume 11, pages 1–7, 2011.

[34] Kubernetes Authors. Kubernetes, 2022. `https://kubernetes.io/`, Last visit July 25, 2022.

[35] Paul Le Noac'H, Alexandru Costan, and Luc Bougé. A performance evaluation of Apache Kafka in support of big data streaming applications. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4803–4806. IEEE, 2017.

[36] Colin McCabe. Apache Kafka Needs No Keeper: Removing the Apache ZooKeeper Dependency, 2020. `https://www.confluent.io/blog/removing-zookeeper-dependency-in-kafka/`, Last visit July 31, 2022.

[37] Damian Menscher. Exponential growth in DDoS attack volumes, 2020. `https://cloud.google.com/blog/products/identity-security/identifying-and-protecting-against-the-largest-ddos-attacks`, Last visit July 18, 2022.

[38] Microsoft. Publisher-Subscriber pattern, 2022. `https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber`, Last visit July 24, 2022.

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

[40] Jelena Mirkovic and Peter Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[41] MISP Contributors. MISP - User Guide, 2022. `https://www.circl.lu/doc/misp/book.pdf`, Last visit August 4, 2022.

[42] MISP project. MISP Threat Sharing project, 2022. `https://www.misp-project.org/datamodels/`, Last visit August 4, 2022.

[43] Robin Moffatt. Kafka Listeners - Explained, 2018. https://rmoff.net/2018/08/02/kafka-listeners-explained/, Last visit November 21, 2022.

[44] MongoDB Inc. MongoDB, 2022. https://www.mongodb.com/languages/python, Last visit November 25, 2022.

[45] Andrew Mortensen, Tirumaleswar Reddy.K, Flemming Andreasen, Nik Teague, and Rich Compton. DDoS Open Threat Signaling (DOTS) Architecture. RFC 8811, August 2020.

[46] Andrew Mortensen, Tirumaleswar Reddy.K, and Robert Moskowitz. DDoS Open Threat Signaling (DOTS) Requirements. RFC 8612, May 2019.

[47] Giorgos Myrianthous. Multi-threading and Multi-processing in Python, 2022. https://towardsdatascience.com/multithreading-multiprocessing-python-180d0975ab29, Last visit November 26, 2022.

[48] Netscout. Volumetric DDoS Attacks, 2022. https://www.netscout.com/what-is-ddos/volumetric-attacks, Last visit November 16, 2022.

[49] Netscout. What is a Reflection Amplification DDoS Attack?, 2022. https://www.netscout.com/what-is-ddos/what-is-reflection-amplification-attack, Last visit July 22, 2022.

[50] Sean Newman. Bursts, Waves and DDoS: What You Need to Know, 2018. https://www.corero.com/blog/bursts-waves-and-ddos-what-you-need-to-know/, Last visit November 1, 2022.

[51] Georgios Nikolaidis, Jeongkeun Lee, and Changhoon Kim. Data plane with heavy hitter detector, February 23 2021. U.S. Patent 10,931,547.

[52] George Oikonomou, Jelena Mirkovic, Peter Reiher, and Max Robinson. A Framework for A Collaborative DDoS Defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 33–42. IEEE, 2006.

[53] Opeyemi Osanaiye, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework. *Journal of Network and Computer Applications*, 67:147–165, 2016.

[54] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Computing Surveys (CSUR)*, 39(1):3–es, 2007.

[55] Pyshark Github Contributors. pyshark, 2022. https://github.com/KimiNewt/pyshark, Last visit November 28, 2022.

[56] Pyshark Github Contributors. PyShark Documentation, 2022. http://kiminewt.github.io/pyshark/, Last visit December 09, 2022.

[57] Python Bloom Filter Github Contributors. Python Bloom Filter, 2022. `https://github.com/joseph-fox/python-bloomfilter`, Last visit November 26, 2022.

[58] Python Software Foundation. Python, 2022. `https://www.python.org/`, Last visit November 25, 2022.

[59] Python Software Foundation. Python Dictionaries, 2022. `https://docs.python.org/3/tutorial/datastructures.html#dictionaries`, Last visit November 25, 2022.

[60] Python Software Foundation. Python Multiprocessing, 2022. `https://docs.python.org/3/library/multiprocessing.html`, Last visit November 26, 2022.

[61] Python Software Foundation. Python Queue, 2022. `https://docs.python.org/3/library/queue.html`, Last visit November 26, 2022.

[62] Python Software Foundation. Python Random, 2022. `https://docs.python.org/3/library/random.html`, Last visit November 26, 2022.

[63] Python Software Foundation. Python sqlite3, 2022. `https://docs.python.org/3/library/sqlite3.html`, Last visit November 25, 2022.

[64] Python Software Foundation. Python Threading, 2022. `https://docs.python.org/3/library/threading.html`, Last visit November 26, 2022.

[65] Python Software Foundation. What's New In Python 3.10, 2022. `https://docs.python.org/3/whatsnew/3.10.html`, Last visit December 09, 2022.

[66] Redis-py Github Contributors. Redis-py, 2022. `https://github.com/redis/redis-py`, Last visit November 25, 2022.

[67] Michael Robinson, Kevin Jones, and Helge Janicke. Cyber warfare: Issues and challenges. *Computers & security*, 49:70–94, 2015.

[68] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. Enabling a Cooperative, Multi-Domain DDoS Defense by a Blockchain Signaling System (BloSS). *Semantic Scholar*, 2017.

[69] Bruno Bastos Rodrigues. *Blockchain signaling system (BloSS)*. PhD thesis, University of Zurich, 2021.

[70] Ori Rottenstreich and Isaac Keslassy. The Bloom Paradox: When not to Use a Bloom Filter. *IEEE/ACM Transactions on Networking*, 23(3):703–716, 2014.

[71] Scapy Github Contributors. Scapy, 2022. `https://github.com/secdev/scapy`, Last visit November 12, 2022.

[72] Gwen Shapira, Todd Palino, Rajini Sivaram, and Krit Petty. *Kafka: The Definitive Guide*. O'Reilly Media, Inc., Sebastopol, CA, USA, 2021.

[73] Lumin Shi. PathFinder: Collecting Traffic Footprint at Autonomous System Level. Directed research project, College of Arts and Sciences, University of Oregon, 5 2017. Available at `https://www.cs.uoregon.edu/Reports/DRP-201705-Shi.pdf`.

[74] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176, 2017.

[75] Stephen Specht and Ruby Lee. Taxonomies of Distributed Denial of Service Networks, Attacks, Tools and Countermeasures. *CEL2003-03, Princeton University, Princeton, NJ, USA*, 2003.

[76] The ZeroMQ authors. ZeroMQ, 2022. `https://zeromq.org/`, Last visit August 4, 2022.

[77] Peter Triantafillou and Andreas Economides. Subscription Summarization: A New Paradigm for Efficient Publish/Subscribe Systems. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 562–571. IEEE, 2004.

[78] Liam Tung. A tiny botnet launched the largest DDoS attack on record, 2022. `https://www.zdnet.com/article/a-tiny-botnet-launched-the-largest-ddos-attack-on-record/`, Last visit November 16, 2022.

[79] UltraDict Github Contributors. UltraDict, 2022. `https://github.com/ronny-rentner/UltraDict`, Last visit November 25, 2022.

[80] Thijs van den Hout, Remco Poortinga-van Wijnen, Cristian Hesselman, Christos Papachristos, and Karin Vink. Developing and running a testbed for the DDoS Clearing House, 2021. `https://www.sidnlabs.nl/en/news-and-blogs/developing-and-running-a-testbed-for-the-ddos-clearing-house`, Last visit July 2, 2022.

[81] Thaneswaran Velauthapillai, Aaron Harwood, and Shanika Karunasekera. Global Detection of Flooding-Based DDoS Attacks Using a Cooperative Overlay Network. In *2010 Fourth International Conference on Network and System Security*, pages 357–364, 2010.

[82] Cynthia Wagner, Alexandre Dulaunoy, Gérard Wagener, and Andras Iklody. MISP: The Design and Implementation of a Collaborative Threat Intelligence Sharing Platform. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*, pages 49–56, 2016.

[83] Daniel Wagner, Daniel Kopp, Matthias Wichtlhuber, Christoph Dietzel, Oliver Hohlfeld, Georgios Smaragdakis, and Anja Feldmann. United We Stand: Collaborative Detection and Mitigation of Amplification DDoS Attacks at Scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 970–987, 2021.

[84] Wikipedia Contributors. Covariant Return Type, 2022. `https://en.wikipedia.org/wiki/Covariant_return_type`, Last visit November 30, 2022.

[85] Wikipedia Contributors. Observer pattern, 2022. `https://en.wikipedia.org/wiki/Observer_pattern`, Last visit July 24, 2022.

[86] Wikipedia Contributors. pcap, 2022. `https://en.wikipedia.org/wiki/Pcap`, Last visit November 12, 2022.

[87] Wikipedia Contributors. Phi Coefficient, 2022. `https://en.wikipedia.org/wiki/Phi_coefficient`, Last visit November 06, 2022.

[88] Wikipedia Contributors. Publish-subscribe pattern, 2022. `https://en.wikipedia.org/wiki/Publish-subscribe_pattern`, Last visit July 24, 2022.

[89] Wikipedia Contributors. Python, 2022. `https://en.wikipedia.org/wiki/Python_(programming_language)`, Last visit November 16, 2022.

[90] Wikipedia Contributors. Strategy Pattern, 2022. `https://en.wikipedia.org/wiki/Strategy_pattern`, Last visit November 30, 2022.

[91] Wikipedia Contributors. Wikipedia: Bloom Filter, 2022. `https://en.wikipedia.org/wiki/Bloom_filter`, Last visit August 17, 2022.

[92] Wikipedia Contributors. Wikipedia: PageRank, 2022. `https://en.wikipedia.org/wiki/PageRank`, Last visit July 12, 2022.

[93] Wireshark. tshark Manual Page, 2022. `https://www.wireshark.org/docs/man-pages/tshark.html`, Last visit November 28, 2022.

[94] Wireshark. Wireshark, 2022. `https://www.wireshark.org/`, Last visit November 12, 2022.

[95] Jie Yu, Zhoujun Li, Huowang Chen, and Xiaoming Chen. A Detection and Offense Mechanism to Defend Against Application Layer DDoS Attacks. In *International Conference on Networking and Services (ICNS '07)*, pages 54–54, 2007.

[96] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.

[97] Jian Zhang, Phillip A Porras, and Johannes Ullrich. Highly predictive blacklisting. In *USENIX security symposium*, pages 107–122, 2008.

Abbreviations

ACC	Accuracy
AS	Autonomous System
BC	Blockchain
CPU	Central Processing Unit
dAPP	Decentralized Application
DDoS	Distributed Denial of Service
DoS	Denial of Service
DNS	Domain Name System
DXP	DDoS Information Exchange Point
FN	False Negative
FP	False Positive
HH	Heavy Hitters
IoT	Internet of Things
IXP	Internet Exchange Point
MCC	Matthews Correlation Coefficient
ML	Machine Learning
NTP	Network Time Protocol
P2P	Peer-to-Peer
QoS	Quality-of-Service
SC	Smart Contract
SDN	Software Defined Networking
SIP	Session Initiation Protocol
SOM	Self-organizing Maps
TN	True Negative
TP	True Positive

List of Figures

List of Tables

Code Listings

Appendix A

Traffic Generation and Sniffing

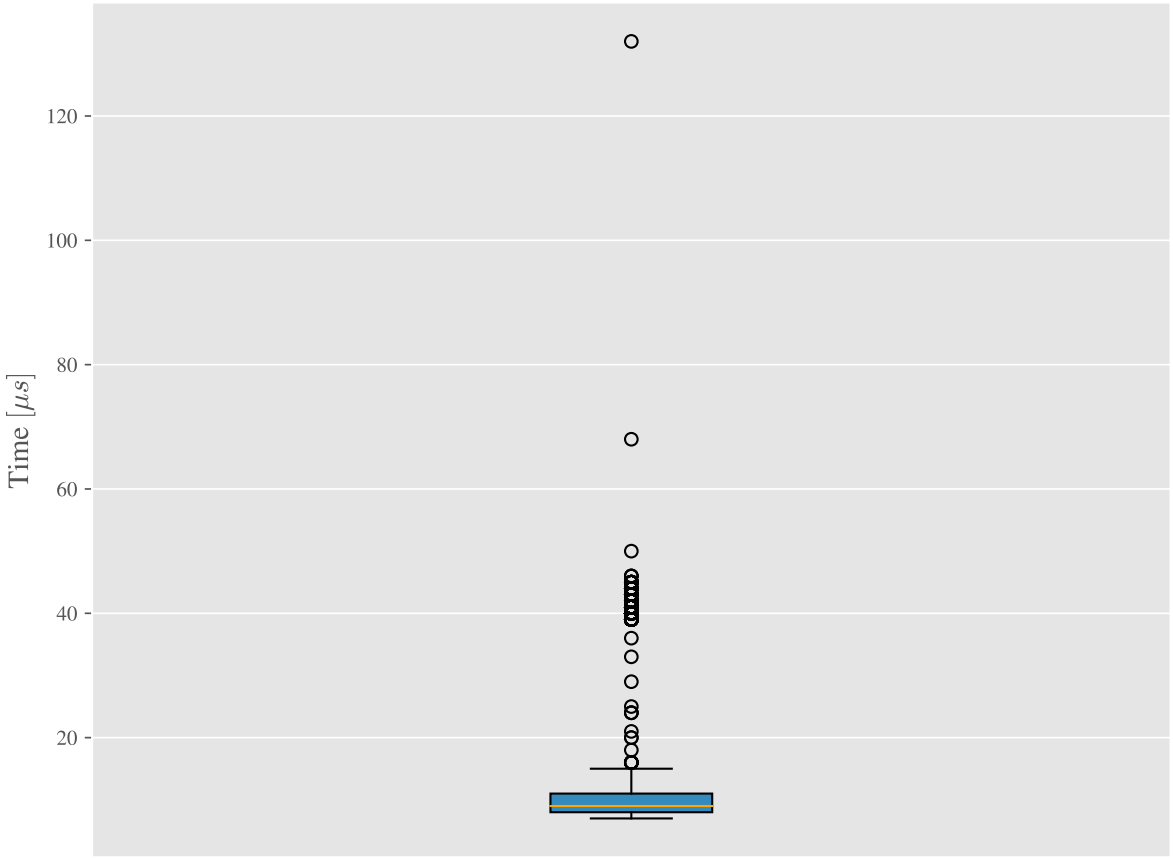


Figure A.1: Boxplot for the Traffic Generation simulation. Time in μs, and $n = 2305$ data points were collected. The indicated time is the result of reading a PCAP traffic file and sending it to CH^{2}TF for analysis (*cf.* Section 6.1), without considering any target attack rate. $\mu = 15.11$, $\sigma = 12.92$.

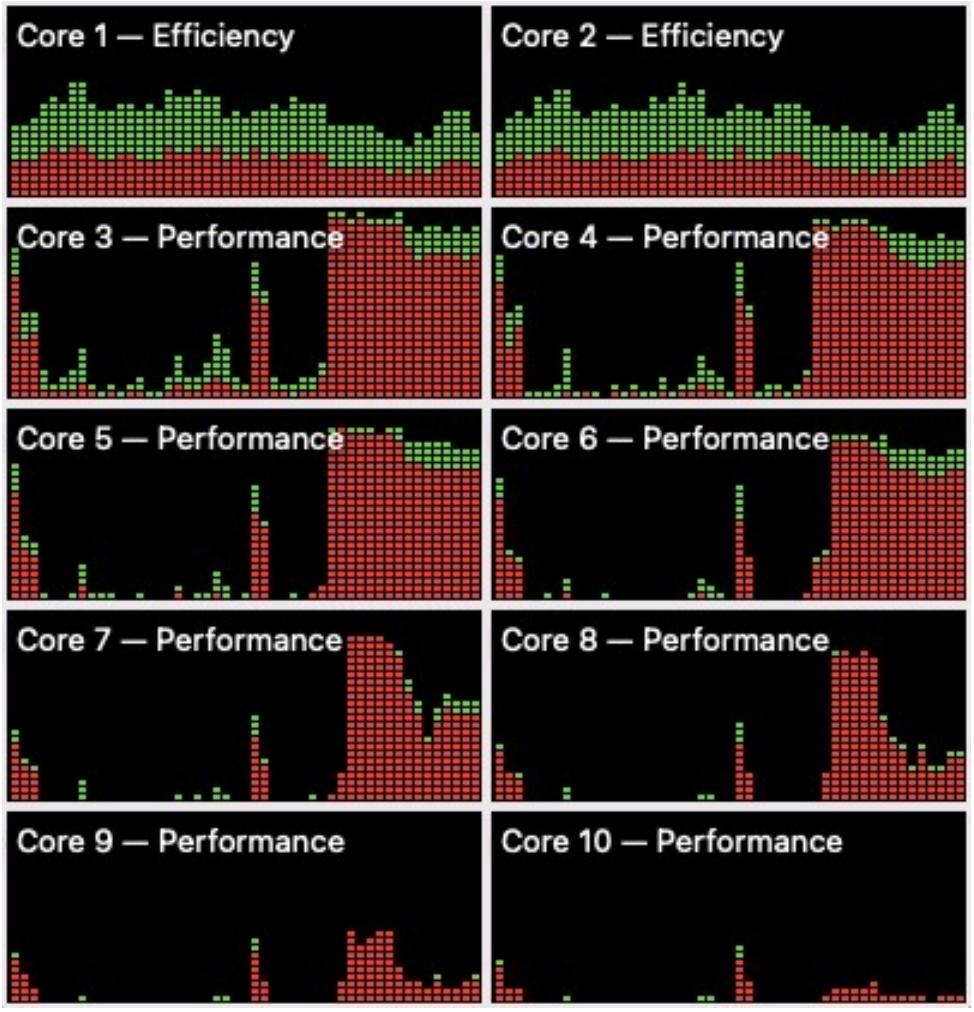


Figure A.2: Screenshot of macOS's 'Activity Monitor' displaying the CPU usage history using three ASes in Docker Containers, with analyses turned off. *I.e.,* only traffic generation is active. The peak in the middle indicates when the Docker Containers started (*cf.* cores 3-10), and the load at the right side of each core indicates when the traffic generation started. Timeline 1 *min*. This figure also displays that the first two (efficiency) cores were not used by the Docker Containers.

```python
class Sniffer:
    """
    Reference for a traffic sniffer class.
    """

    transport_layers = ["UDP", "TCP"]

    def __init__(self, queue: Queue, iface_name: str = "en0"):
        self.queue: Queue = queue
        self.iface_name: str = iface_name

    def get_packet_information(self, packet: Packet):
        transport_layer = packet.transport_layer
        if transport_layer not in self.transport_layers:
            return
        ip = packet.ipv6 if hasattr(packet, "ipv6") else packet.ip
        timestamp = packet.sniff_time.isoformat()

        packet_data: PacketData = PacketData(
            src=ip.src,
            dst=ip.dst,
            srcport=packet[transport_layer].srcport,
            dstport=packet[transport_layer].dstport,
            timestamp=timestamp,
            transport_layer=transport_layer,
        )
        self.queue.put(packet_data)

    def start_sniffing(self):
        capture = pyshark.LiveCapture(interface=self.iface_name)
        print(capture.interfaces)
        capture.apply_on_packets(self.get_packet_information)
```

Code Listing A.1: Traffic `Sniffer` class. This is for reference purposes only, and is considered outside of the scope of CH^{2}TF. Inspired by [56].

Appendix B

Additional Implementation Classes

```python
@dataclass_json
@dataclass
class DefenseCollaborationRequestData:
    """
    Request for a defense collaboration
    """
    potential_attacker_ips: List[
        str
    ]  # the ips to be checked by the receivers
    potential_victim: str
    requests_relative_to_size: float
    request_detection: DetectionEnum
    request_id: str = field(default_factory=lambda: str(uuid.uuid4()))
    request_originator: str = os.getenv("AS_NAME", default="")
```

Code Listing B.1: The `DefenseCollaborationRequestData` class. An instance of this class is published via Kafka to the subscribers.

```python
@dataclass_json
@dataclass
class DefenseCollaborationResponseData:
    """
    Response for a defense collaboration
    """
    # potential attackers that the AS acknowledges.
    # subset of the potential attackers that is sent in the request
    ack_potential_attacker_ips: List[str]
    decision: DecisionEnum
    as_name: str
    request_id: str  # use same id as original request
    request_originator: str  # from which AS the request came from
```

Code Listing B.2: The `DefenseCollaborationResponseData` class. An instance of this class is published via Kafka to the subscribers.

```
class DecisionEnum(Enum):
    NOT_MANAGED = "AS does not manage any of the ip"
    NOT_ACK = "AS does not acknowledge this as an attack"
    UNDER_THRS = "No potential attacker pass the thresholds"
    FOUND = "Attacker(s) found"
```

Code Listing B.3: The `DecisionEnum` class.

Appendix C

Docker Compose YAML

```yaml
version: '3.5'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    networks:
      - kafka_network
    volumes:
      - ./zoo/data:/var/lib/zookeeper/data
      - ./zoo/log:/var/lib/zookeeper/log

  kafka:
    image: confluentinc/cp-kafka:latest
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - 9092:9092
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST
          ://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,
          PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "true"
    networks:
      - kafka_network

  as0:
    build: .
    depends_on:
      - kafka
```

```
37      networks:
38        - kafka_network
39      restart: on-failure
40      container_name: as0
41      environment:
42        AS_SIZE: 100
43        KAFKA_HOST: kafka
44        KAFKA_PORT: 29092
45        AS_NAME: as0
46        EVAL_SIMULATED_TRAFFIC_PATH: ./eval_data/traffic_files/burst/
              AS_0_traffic-5.pcap
47        EVAL_SIMULATED_ATK_TRAFFIC_PATH: ./eval_data/traffic_files/burst/
              AS_0_attack_traffic-5.pcap
48        MANAGED_IPS_PATH: ./eval_data/managed_ips/AS_0_managed_ip_10000.
              txt
49
50    as1:
51      build: .
52      depends_on:
53        - kafka
54      networks:
55        - kafka_network
56      restart: on-failure
57      container_name: as1
58      environment:
59        AS_SIZE: 100
60        AS_NAME: as1
61        KAFKA_HOST: kafka
62        KAFKA_PORT: 29092
63        EVAL_SIMULATED_TRAFFIC_PATH: ./eval_data/traffic_files/burst/
              AS_1_traffic-5.pcap
64        EVAL_SIMULATED_ATK_TRAFFIC_PATH: ./eval_data/traffic_files/burst/
              AS_1_attack_traffic-5.pcap
65        MANAGED_IPS_PATH: ./eval_data/managed_ips/AS_1_managed_ip_10000.
              txt
66
67    as2:
68      build: .
69      depends_on:
70        - kafka
71      networks:
72        - kafka_network
73      restart: on-failure
74      container_name: as2
75      environment:
76        AS_SIZE: 100
77        AS_NAME: as2
78        KAFKA_HOST: kafka
79        KAFKA_PORT: 29092
80        EVAL_SIMULATED_TRAFFIC_PATH: ./eval_data/traffic_files/burst/
              AS_2_traffic-5.pcap
81        EVAL_SIMULATED_ATK_TRAFFIC_PATH: ./eval_data/traffic_files/burst/
              AS_2_attack_traffic-5.pcap
82        MANAGED_IPS_PATH: ./eval_data/managed_ips/AS_2_managed_ip_10000.
              txt
83
```

```
84
85 networks:
86   kafka_network:
87     name: kafka_network
```

Code Listing C.1: Docker Compose File for the evaluation in Section 6.1.2. Inspired by [43, 7]. It includes Kafka, ZooKeeper and 3 ASes that use separate Docker containers.

Appendix D

Installation Guidelines

CH^{2}TF has been made public under `https://github.com/ch2tf/ch2tf`. The full installation instructions, guidelines, and alternatives are included in the `readme.md` file.

This project assumes either a Docker installation, *e.g.,* for evaluation purposes or to test the interaction between multiple ASes, or a "local" installation. In any case, `ZooKeeper` and `Kafka` need to be running. This documentation assumes that they are running in Docker containers, though other installations would also be possible.

However, in any case, the ports need to match (*cf.* C.1). This means, that within Docker containers `Kafka` runs on `kafka:29092` (*i.e.,* Section D.1). And to connect to from outside of Docker containers to `Kafka` (*i.e.,* Section D.2), it requires `localhost:9092`.

Assuming that `ZooKeeper` and `Kafka` should be run in a Docker container, the following commands need to be entered in the main directory of CH^{2}TF:

1. `docker compose build`
2. `docker compose up zookeeper kafka`

This will start these two services, and let the developer the freedom whether CH^{2}TF should be run in a Docker container (*cf.* Section D.1) or locally (*cf.* Section D.2).

Furthermore, depending on whether this is a 'fresh' installation, and the Pub/Sub topics do not exist yet, they must be created first (*cf.* `readme.md`).

D.1 Docker Installation

- Requirements: Docker

To run CH^{2}TF the following commands need to be entered in the main directory of CH^{2}TF:

1. `docker compose build`
2. `docker compose up`

Nota bene, using these commands the required dependencies `ZooKeeper` and `Kafka` will also automatically start.

D.2 Local Development

This project has been tested with Python versions 3.10 and 3.11. Furthermore, due to the use of 'structural pattern matching' statements, Python versions < 3.10 are not supported (*cf.* [65]).

- Requirements: `ZooKeeper`, `Kafka`, `Python`

To run CH^{2}TF the following commands need to be entered in the main directory of CH^{2}TF:

1. `python --version` $\Rightarrow$ *should return >= 3.10*
2. `pip3 install -r requirements.txt`
3. `export PYTHONPATH="${PYTHONPATH}:/src"`
4. `python3 src/main.py`

I.e., the "working directory" (*e.g.,* for IDEs) should be `.../src`. This can be seen in the file `docker_entrypoint.sh` that can also be used to start the application.

D.3 Configurations

The configuration is done directly in the `ch2tf/.env`. This allows to set *global* configurations (if needed), which can then be individually overwritten for any AS in the docker compose file (`ch2tf/docker_compose.yml`) directly (*cf.* C.1).

The important parts of the configuration are the following points:

- `KAFKA_HOST`: The host where `Kafka` is running (*e.g.,* `kafka` or `localhost`).
- `KAFKA_PORT`: The port where `Kafka` is running, (*e.g.,* 29092 or 9092).
- `AS_NAME`: Name of this AS. Should not be set *globally.* This will be included in the requests and responses to identify the AS.

- `AS_SIZE`: Size variable for how much traffic this AS can handle. Should also not be set *globally*.
- `TOPIC_LOW`: Topic for *lower* confidence. Should be set *globally*.
- `TOPIC_HIGH`: Topic for *higher* confidence. Should be set *globally*.
- `TOPICS_USE_ADDITIONAL`: Whether this AS wants to use additional topics. *I.e.,* `False`. Should not be set *globally*.
- `AS_TOPICS`: The additional (comma separated) topics that an AS wants to use. `TOPICS_USE_ADDITIONAL` must be `True` if those want to be used. Overwrites the default topics.
- `ANALYSIS_PERIOD`: Analysis period. Could be set *globally*, but not required.
- `MSG_LENGTH`: The length for potential attackers that should be used while publishing a Pub/Sub message.
- `SAMPLING_RATE`: Rate of sampling that should be used for a specific AS. A value of 1.0 implies that every packet is considered.
- `USE_HASH`: Whether *hashed* IP addresses are used. Should be set *globally*.
- `MANAGED_IPS_PATH`: Path to a text file that contains the managed IP addresses of a specific AS. Trivially, this should not be set *globally*.

Furthermore, there are the analysis related configuration values which were discussed in Section 5.8. Naturally, these should be set for each AS individually.

- `THRESHOLD_VICTIM_LO`
- `THRESHOLD_VICTIM_HI`
- `THRESHOLD_VICTIM_TIME_PERCENTAGE`
- `THRESHOLD_VICTIM_TIME_MIN`
- `THRESHOLD_SRC_1`
- `THRESHOLD_SRC_2`
- `THRESHOLD_SRC_3`
- `THRESHOLD_SRC_3_MIN`
- `THRESHOLD_TRAFFIC_PROPORTIONALITY`

Appendix E

Contents of the Submission

This thesis has been submitted digitally and contains the following files and directories:

- **MA_Thesis_Fabian_Kueffer.pdf:** This thesis in PDF format.
- **MA_Thesis_Fabian_Kueffer.zip:** Source code of this thesis.
- **CH2TF.zip:** Source code of the prototype and evaluation scripts & data.
- **Figures:** Directory containing all figures that were used in the report.
- **Additional:** Additional material.
- **abstract_de.txt:** German abstract.
- **abstract_en.txt:** English abstract.