Machine Learning-Guided Aggregate Filtering to Counteract Volumetric DDoS Attacks

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Hauke Heseding

Tag der mündlichen Prüfung: 05. Mai 2025

1. Referentin: Prof. Dr. Martina Zitterbart
   Karlsruher Institut für Technologie (KIT)
2. Referent: Prof. Dr. Holger Karl
   Hasso-Plattner-Institut

Acknowledgments

This dissertation has been a journey of intellectual exploration and personal growth. It was often challenging, but always rewarding. I am deeply grateful to the many individuals who have contributed to this journey, whether in profound or subtle ways.

First, I express my sincere gratitude to my supervisor, Prof. Dr. Martina Zitterbart, for her invaluable guidance, patience, and steadfast support throughout this research. She allowed me to explore new ideas while ensuring that I remained rooted in academic precision. I am fortunate to have had such a dedicated mentor. I am also very grateful to Prof. Dr. Holger Karl for his thoughtful suggestions and for generously taking the time to review my dissertation, despite his many responsibilities.

My heartfelt appreciation goes to my colleagues and fellow researchers at the Institute of Telematics. Their insightful discussions, moral support, and the collaborative environment they fostered were indispensable to my work. I feel especially fortunate to have worked closely with Samuel Kopmann, whose camaraderie and indomitable spirit made a deep and lasting impression on me. I would also like to thank Robert Bauer, Matthias Flittner, Sebastian Friebe, Frank Winter, and Roland Bless for their mentorship, friendship, and the many memorable moments we shared. To my students, thank you for your numerous contributions that continually inspired me to grow as a researcher and educator.

I extend my gratitude to the faculty and staff of the KIT Department of Informatics and the KASTEL Security Research Labs for providing an enriching academic environment and access to essential research resources. I am equally grateful to the Helmholtz Association for their financial support (through POF structure 46.23.01: Methods for Engineering Secure Systems), which made this research possible.

I am especially indebted to my dear friend Ulrike Vogl, whose help throughout my studies was both generous and invaluable.

Finally, to my family and all my friends, thank you for laying the foundation upon which this work was built. Your love, patience, and steadfast belief in me sustained me through every challenge. Your kindness and support were the constant source of strength that carried me to the finish line.

Karlsruhe, May 2025

Contents

Chapter 1

Introduction

The Internet has become a cornerstone of modern society. With tens of billions of connected devices, it permeates numerous aspects of life, including business, arts, education, entertainment, sports, science, healthcare, government, politics, transportation, and personal relationships. In 2016, NATO recognized cyberspace as a domain of operations in addition to the old battlefields of land, sea, and air [Wie23], underscoring the strategic importance of a globally connected IT infrastructure.

As the importance of the Internet continues to grow, so does the need to protect it. Still, elaborate attacks that target software vulnerabilities such as Stuxnet [FMC+11], WannaCry [MP17; Gha+17], or SolarWinds [Alk+21] can have far-reaching, long-lasting consequences and continue to evolve. A simpler yet effective approach is to disrupt the flow of information. So-called **Distributed Denial of Service (DDoS)** attacks direct multiple systems against a common target to render its services unavailable. Despite their simplicity, these attacks can have regional or global impact, like the attack on the country of Estonia [Les07] or the attacks on Domain Name System (DNS) providers in 2016 that affected Internet service availability worldwide [ARN19].

Unfortunately, DDoS attacks of varying intensity are now commonplace. Tens of thousands of attacks take place every day [NET24]. A large portion of them not only affects the intended attack target, but also have a collateral impact on the network infrastructure and thereby other connected services. The basic principle of these **volumetric DDoS attacks** is quite simple: To overload the network with so much attack traffic that legitimate traffic can no longer be forwarded.

Despite ongoing research efforts, the threat of volumetric DDoS attacks persists for decades since they first became public around the year 2000 [Pax01b; Cha02]. For once, the basic concept still works. But in addition, these attacks are constantly evolving as new attack vectors emerge, the number of systems participating in attacks increases steadily, and attack patterns change over time. This leaves not only the intended attack targets vulnerable but also the network infrastructure.

1.1 Problem Statement

Volumetric DDoS attacks sent attack traffic from many sources to a common target to render it unavailable. Individual attacking systems do not need to send a high traffic volume, but as the attack traffic is forwarded to the target, it becomes increasingly aggregated and can congest the network infrastructure. The main effect of a volumetric DDoS attack manifests itself in the bottleneck links upstream from a target. To protect not only the target but also the network infrastructure, traffic should be filtered to remove attack traffic before its aggregate volume reaches critical levels.

Early attack traffic removal. Removing the attack traffic before it reaches bottleneck links is challenging. The distribution of attack traffic across multiple sources means that upstream systems can only monitor a portion of the traffic, so that the characteristically high volume of a volumetric DDoS attack is not evident. This makes it difficult to distinguish between attack traffic and legitimate traffic. Essentially, there is a gap between classification precision and the effectiveness of attack traffic removal, as higher precision is more likely to be achieved in a target's proximity, while filtering traffic upstream preserves more infrastructure resources. To bridge the gap, monitoring data should be collected downstream (near a target) and provided to upstream systems that can filter out the attack traffic.

Efficient traffic processing. In order to deal with volumetric DDoS attacks, traffic filtering and monitoring need to process traffic in line speed to avoid back pressure. This requires fast memory to make forwarding decisions and to store acquired monitoring data. However, network devices are limited in the capacity of memory that is fast enough. This means that only a limited amount of monitoring data can be stored and only limited information about attacking systems can be processed upstream. In a highly distributed attack, such memory restrictions can become a bottleneck when individual traffic sources are being monitored. Instead, traffic aggregates (e.g., at the level of subnets) can be monitored to reduce memory requirements. This makes it more challenging to distinguish between attack and legitimate traffic sources.

Continuous monitoring. Changes in the distribution of traffic sources and traffic volume can cause monitoring information to become outdated. This can lead to attackers being able to circumvent established defenses or the inadvertent removal of legitimate traffic. Consequently, monitoring information must be updated regularly to maintain the effectiveness of traffic filtering. This must be consistent with the constraints of efficient traffic processing.

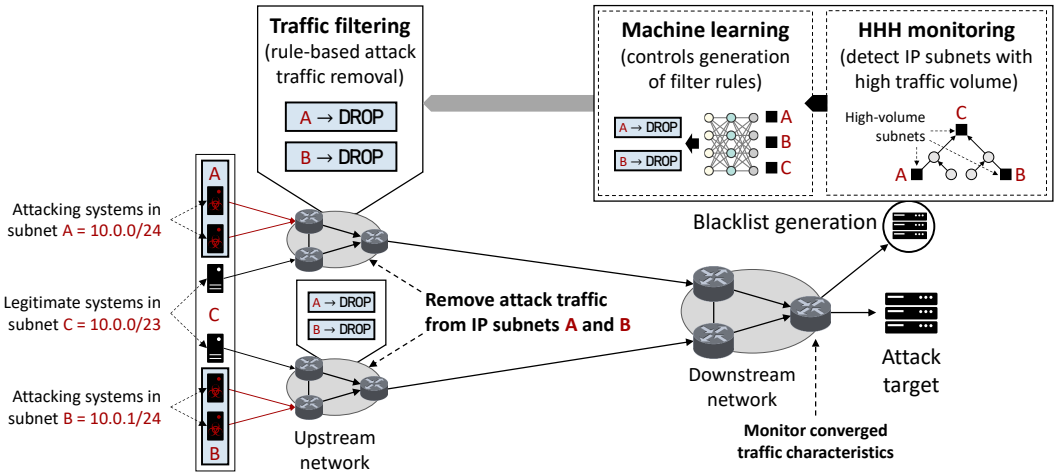


Figure 1.1: Filter rule generation for upstream networks based on monitoring with Hierarchical Heavy Hitter (HHH) and machine learning.

1.2 Basic Approach

The basic idea behind this thesis is to combine two key technologies to counteract volumetric DDoS attacks before they achieve their intended effect:

- **Hierarchical Heavy Hitter (HHH) algorithms** aggregate traffic over the IP prefix hierarchy to detect subnets that send high-volume traffic. Monitoring traffic aggregates at various levels of the IP prefix hierarchy (rather than just individual sources) serves to detect individually strong attack traffic sources, as well as widely distributed sources that send low-volume attack traffic. Efficient HHH algorithms offer the capability to approximate traffic volumes in line speed while also maintaining low memory requirements.
- **Machine learning techniques** that use the information provided by HHH algorithms to generate prefix-based filter rules for upstream systems. Applying machine learning techniques serves two purposes: First, to learn to identify subnets that mainly send attack traffic based on aggregated information. Second, to learn to control the number of generated filter rules to utilize the memory of upstream network devices efficiently.

Figure 1.1 provides a high-level overview of this concept. The HHH algorithms monitor the converged traffic characteristics near the target to detect subnets sending high traffic volumes, which indicates the potential presence of attacking systems.

This information is then refined into filter rules with the help of machine learning. Network devices in upstream networks apply the filter rules to remove ingress traffic in order to protect the network infrastructure and the attack target.

Using IP prefixes instead of individual source addresses can reduce the number of filter rules, since multiple addresses can be handled by a single rule. This particularly serves to mitigate attacks that employ a large number of attacking systems, where monitoring each individual address would exhaust the memory of upstream network devices. The downside is that prefix-based filtering can also affect legitimate systems that are close to attacking systems in the IP address space. This leads to a three-way trade-off between the number and granularity of filter rules (in terms of subnet size), the removed attack traffic volume, and the volume of preserved legitimate traffic. For example, removing traffic from the entire subnet C shown in Figure 1.1 would require only a single rule (instead of two separate rules for subnets A and B) but would affect the legitimate traffic sources in that address space region as well. This trade-off should be considered across the entire IP prefix hierarchy. Furthermore, it must be re-assessed periodically to account for changes in the distribution of attack and legitimate traffic over the IP address space. Otherwise, outdated filter rules can inadvertently affect legitimate traffic.

1.3 Research Questions

To realize machine learning-controlled filter rule generation based on HHH algorithms, this thesis addresses three research questions:

(1) How to estimate the volume of attack traffic sent from individual IP subnets with deep learning while balancing the low memory requirements of HHH algorithms against estimation errors?

(2) How to translate the information on the attack traffic volume of IP prefixes into accurate information on the location of attacking systems in the IP address space that can be used to define filter rules?

(3) How to use deep Reinforcement Learning (RL) to control trade-offs between the number of required filter rules, the volume of removed attack traffic, and the impact on legitimate traffic when the attack traffic and the legitimate traffic change over time?

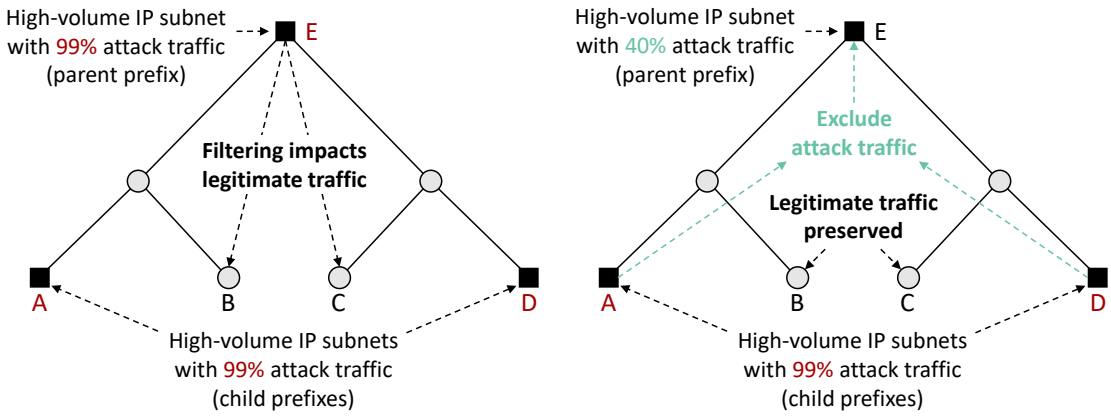
1.4 Main Contributions

This thesis presents Selective Aggregate Filtering — a mitigation system against volumetric DDoS attacks that generates filter rules for upstream systems. Selective Aggregate Filtering builds on the concept of HHH algorithms and complements it with deep learning to identify regions in the IP address space where attacking systems are located. It monitors the distribution and characteristics of traffic over time to periodically generate and adapt IP prefix-based filter rules. This serves to maintain effective mitigation trade-offs. To enable the concept of Selective Aggregate Filtering, this thesis makes three main contributions that build on each other:

Traffic Purity Estimation. A key characteristic of volumetric DDoS attacks is a (combined) high traffic volume. HHH algorithms aggregate traffic to detect IP subnets sending high-volume traffic. However, distinguishing between attack and legitimate traffic based on traffic volume alone is likely to fail in the presence of high-volume legitimate traffic. To identify subnets that mainly send attack traffic, Traffic Purity Estimation [HKZ23] was introduced. The main contribution is a novel concept based on deep learning that enables estimating the ratio of attack traffic sent from IP subnets of various sizes. Relying on deep learning enables training on labeled datasets to automatically learn characteristics of attack vectors. The underlying monitoring is based on HHH algorithms, which Traffic Purity Estimation extends to monitor additional traffic features over IP subnets besides volume. The combination of deep learning with HHH algorithms serves to locate attack traffic sources and to assess the potential impact a filter rule has on attack and legitimate traffic.

To balance increased memory requirements for additional features against estimation errors, highly relevant features of aggregated traffic are identified through a method called permutation feature importance. The ability to achieve low estimation errors is evaluated for several feature selections based on legitimate real-world traffic traces overlaid with synthesized attack traffic that uses multiple attack vectors. The results show that deep learning can maintain low mean estimation errors ($\sim 1\,\%$) when volumetric DDoS attacks use previously unseen attack traffic compositions. This requires three to seven times more memory to monitor aggregated traffic features (depending on estimation error).

Filter Rule Refinement. Traffic Purity Estimation determines the ratio of attack traffic for each subnet independently. A challenge arises from considering hierarchical dependencies between (shorter) parent prefixes and (longer) child prefixes. Parent prefixes require fewer filter rules because they can eliminate the need for multiple



(a) Disregarding hierarchical dependencies **(b)** Considering hierarchical dependencies

Figure 1.2: Example of filter rule selection when disregarding or considering hierarchical dependencies between subnets. Attack traffic ratios are estimated using Traffic Purity Estimation.

child prefixes. However, parent prefixes can have an unnecessary impact on legitimate traffic when child prefixes would achieve more precise filtering. The challenge resides in quantifying the *additional* attack and legitimate traffic that a filter rule with an parent prefix removes over time.

For example, consider the hierarchy of subnets depicted in Figure 1.2a. When Traffic Purity Estimation estimates an attack traffic ratio of 99 % for the high-volume IP subnets A, D, and E removing traffic from subnet E would also remove legitimate traffic from subnets B and C. However, most of the attack traffic of subnet E may be located in subnets A and D. Excluding the attack traffic of subnets A and D would reduce the attack traffic ratio of subnet E, for example, to 40 % as shown in Figure 1.2b. As a result, the legitimate traffic in subnets B and C can be preserved.

This challenge of quantifying the impact of parent prefixes on attack and legitimate traffic was addressed by Filter Rule Refinement [Hes+23]. The main contribution is a method to discount the attack traffic that is removed by child prefixes when estimating the attack traffic ratio of parent prefixes. Furthermore, Filter Rule Refinement is designed to compensate for changes in attack and legitimate traffic volumes. When a subnet temporarily stops sending high-volume traffic, its attack traffic is no longer discounted. In this case, the attack traffic ratio of an parent prefix can increase and become a valid prefix for a filter rule. This effect can cascade across the IP prefix hierarchy and cause filter rule instability. To compensate, Filter Rule Refinement introduces the concept of trackers that smooth the estimated attack traffic ratios of

subnets over time. The findings show that Filter Rule Refinement can consistently avoid the removal of legitimate traffic from large portions of the address space that occurs when disregarding hierarchical prefix dependencies.

RL-based Control. The number of attack traffic sources that are simultaneously active can increase or decrease over time. When a high number of attack traffic sources are active, more rules than can be stored in upstream network devices may be required to accurately remove attack traffic. This raises the question whether certain rules should simply be omitted or if parent prefixes should replace multiple child prefixes (as redundant child prefixes can be omitted). The exclusion of prefixes reduces the volume of removed attack traffic, while preferring parent prefixes incurs a higher risk of removing legitimate traffic. In dynamic traffic situations, the removal of attack and legitimate traffic may need to be constantly balanced against the number of generated filter rules (to address memory constraints). To enable control over the three-way trade-off between the removed legitimate and attack traffic, as well as the number of filter rules, a deep RL-based system was introduced [Hes22; HZ22]. The main contribution is the design of an RL agent based on Deep Q-Networks and the definition of a parameterizable reward function that can convey desired trade-offs.

The agent utilizes the monitoring information provided by HHH algorithms to observe changes in the distribution of traffic sources over the IP address space. The reward function defines the importance of preserving legitimate traffic in comparison to reducing the number of filter rules and removing attack traffic. The task of the agent is to choose parameters that control the generation of filter rules. For example, the agent controls how large the attack traffic ratio in a subnet must be in order for the corresponding prefix to be used in a filter rule. In the example in Figure 1.2b, it is the agent's responsibility to choose a threshold higher than 40 % attack traffic ratio to prevent the removal of traffic from subnet E.

The agent learns the relationship between monitoring information, parameter choices, and reward through training on dynamic traffic scenarios. The evaluation uses authentic traffic traces in combination with a mixture of different attack vectors that change over time to train and test the agent. The findings show that the realized trade-offs can be adjusted by tuning reward function parameters. The trade-offs can be maintained despite changes in traffic patterns, which indicates that the agent effectively counteracts the volumetric DDoS attacks.

1.5 Outline

The thesis is structured into seven chapters with the main contributions being found in chapters 4 to 6. The contents of each chapter are as follows:

- Chapter 2 provides background on volumetric DDoS attacks, as well as the most important aspects of aggregate-based monitoring, deep learning, and RL that are used throughout this thesis.
- Chapter 3 defines the attacker model and further elaborates overarching design goals of Selective Aggregate Filtering. The chapter presents the architecture, the workflow, and the flow of information between architectural components.
- Chapter 4 presents the concepts behind the deep learning-based approach of Traffic Purity Estimation as well as feature selection in the context of volumetric DDoS traffic monitoring. This addresses the first research question. The chapter also introduces the basic structure of evaluation scenarios that are used throughout the thesis.
- Chapter 5 addresses the second research question by introducing the design of Filter Rule Refinement. Special attention is given to the implications of hierarchical dependencies between prefixes. Specifically, how attack traffic can be attributed to parent and child prefixes as well as the effect of changing monitoring information on filter rule stability.
- Chapter 6 elaborates how deep RL can be used to control trade-offs while mitigating volumetric DDoS attacks. The approach builds upon Deep Q-Networks to learn the relationship between monitoring information, parameter choices, and realized trade-offs. The focus is on addressing the third research question. In particular, the design of an RL agent and the ideas behind a reward function that can convey mitigation goals are presented. The evaluation focuses on the agent's ability to distinguish between different attack situations (with few and many attacking systems) to effectively counteract volumetric DDoS attacks.
- Chapter 7 concludes this thesis and offers perspectives for future research.

Appendix A provides supplemental information.

Chapter 2

Background

2.1 Volumetric Distributed Denial of Service Attacks

Volumetric DDoS attacks seek to reduce the availability of targeted systems. The basic idea is to congest the network infrastructure in the vicinity of a target with unsolicited high-volume traffic from multiple attack traffic sources. Using multiple attack traffic sources serves to increase the attack intensity and also makes it more difficult to identify individual attacking systems. The general principle of such volumetric DDoS attacks is system-agnostic, unlike attacks that exploit specific system vulnerabilities.

A key aspect of volumetric DDoS attacks is that their primary impact manifests in the upstream network infrastructure of the actual target. This can negatively impact neighboring systems. Furthermore, it requires an attack target to coordinate its defense with upstream systems in order to remove the attack traffic before it can coalesce to a critical volume.

To conduct a volumetric DDoS attack, an attacker can execute a **direct-path attacks** using attack traffic sources under its direct control or execute an **amplification attack** leveraging publicly accessible services to generate the attack traffic. The operating principles of these attacks are outlined in the following sections.

2.1.1 Direct-Path Attacks

In a direct-path attack, an attacker uses systems that are under his control (so-called **bots**) to flood an attack target over a direct path with high traffic volumes. Bots are often compromised through mass-infection and integrated into a botnet, for example, with the Mirai malware [Ant+17]. Reports on the size of botnets can differ, but are in the order of up to several hundred thousand to millions of bots [Cha+15; Wan+18; Wan+20; Abu+20]. Botnets comprised of a large number of bots can

launch high-volume attacks against a common attack target, e.g., by flooding the attack target with unsolicited traffic. Multiple attack vectors such as ICMP flooding or UDP flooding can be used for this. When the attack traffic is forwarded downstream towards the target, the combined traffic volume increases until it congests a bottleneck link on the path to the target. This results in packet loss at the bottleneck link and makes the attack target unavailable for legitimate client systems that use the same bottleneck link. The distribution of bots over the Internet topology and the IP address space can make it challenging to accurately remove the attack traffic.

2.1.2 Amplification Attacks

An amplification attack (also called reflection attack) uses publicly accessible servers to intensify a volumetric DDoS attack. The concepts of amplification attacks and their impact have been extensively studied in the literature [Ros14; Gri+21; Naw+21; Too+21; ALK22]. The key idea is to exploit the fact that several network services do not validate the origin of a request. When such a service responds to small queries with large responses, it acts as an **amplifier**. An amplifier is not required to be under direct control of an attacker. Still, attacking systems can send requests to one or more amplifiers while pretending to be the attack target. A reflector then generates a larger response (in terms of transmitted bytes) and sends it to the intended attack target, effectively multiplying the traffic volume generated by attacker-controlled systems with a high **bandwidth amplification factor (BAF)** [Ros14]. As a result, attackers carry out high-volume DDoS attacks while investing only a fraction of the data rate that reaches an attack target.

The majority of amplification attack vectors utilize services that rely on the User Datagram Protocol (UDP) since it is connectionless and does not validate IP source addresses. Unless a network service applies additional countermeasures, an attacking system can simply spoof the IP source address of a UDP datagram to make it appear that a request originated from an attack target. In return, a UDP-based service sends its response to the targeted system. This offers the additional advantage that an attacker does not have to reveal the IP addresses of systems under its direct control as the attack target can only observe the traffic generated by the amplifiers.

DNS amplification attacks. A commonly used service for UDP-based amplification attacks is the Domain Name System (DNS) [Moc87]. While DNS amplification attacks have been known for several decades [Pax01a] they are still popular due to widespread availability of publicly accessible amplifiers that achieve high BAFs [Jon+17;

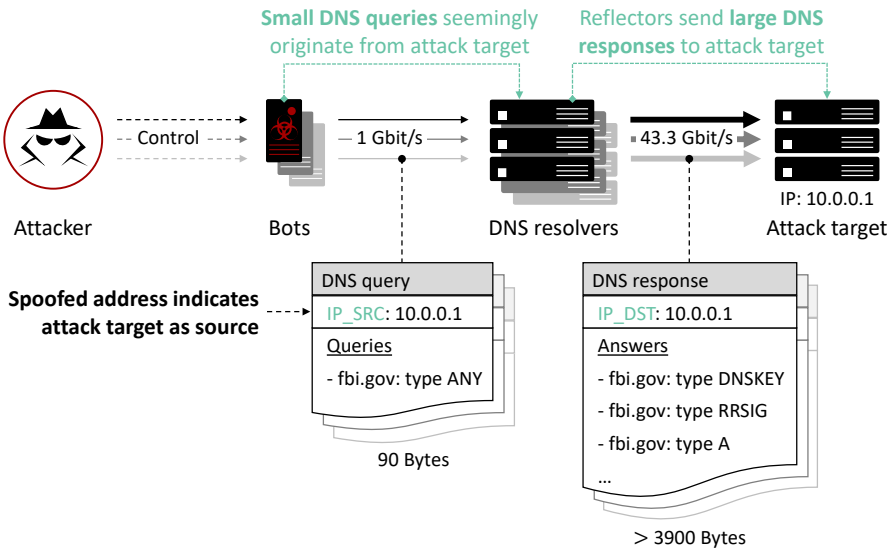


Figure 2.1: Example DNS amplification attack that multiplies bot-generated traffic of 1 Gbit/s by a factor of 43.4 using DNS queries to ANY-type resource records. DNS resolvers sent their responses directly to the attack target due to the IP source address of requests being spoofed.

KDH21; Naw+21]. Through DNS, systems can retrieve information associated with domain names by sending DNS queries to DNS resolvers. DNS resolvers provide information for their corresponding domain (acting as authoritative name servers) or retrieve information on arbitrary domains on behalf of any system on the Internet (so-called open resolvers). The requested information is encoded as resource records of a DNS response, such as a type A record that is commonly used to obtain the IPv4 address for a host name. By requesting specific resource records for selected domain names, attackers can cause DNS resolvers to generate large responses of several thousand bytes for each individual query. For example, such responses can be generated by requesting resource records of type ANY (a pseudo-record used to retrieve all records of a domain name that are known to a DNS server), DNSKEY (used to distribute public keys), or RRSIG (used to sign other resource records). The large response size became possible with the introduction of the Extension Mechanism for DNS (EDNS(0)) [SGV13] that allows for UDP payloads of more than 512 bytes in DNS. This was in part motivated by a need to prevent DNS response forgeries through the inclusion of digital signatures.

Example 2.1.1. Figure 2.1 illustrates how an attacker can amplify the attack traffic by querying specific domain information from open DNS resolvers. Initially, the

attacker instructs bots under its direct control to attack a system with IP address `10.0.0.1`.[1] In turn, the bots generate DNS queries to multiple open resolvers for ANY-type resource records of the domain `fbi.gov`. Domains under the `.gov` top-level domain are often preferred choices due to being subject to a DNSSEC mandate [BB16; Naw+21], which requires the inclusion of cryptographic material (such as keys and signatures) in DNS responses and enlarges response sizes [RSP14]. If the requested information is not directly available to the open resolver, it will retrieve it from an authoritative name server and cache it for later use (this step is omitted in the illustration). Any bot-generated query is then immediately answered from the open resolvers cache so that the authoritative name server remains unaware of any ongoing attack. While each query can be encoded into an IP datagram using 90 Bytes, the DNS response consists of several resource records (such as A, DNSKEY and RRSIG records) with a combined size of over 3900 Bytes. This yields an effective BAF above 43.3 and typically fragments the DNS response into multiple IP datagrams, thereby also increasing the number of packets that must be processed and reassembled.

Open resolvers are sometimes misconfigured to provide large DNS responses via UDP even if an authoritative name server provides them only via TCP. An attacker can exploit this to redirect the DNS responses to the attack target. To this end, the attacker-controlled bots spoof the IP source address in the DNS query's IP header to match the target's address `10.0.0.1`. Failing to validate the origin of the requests, the open resolver sends its responses directly to the target instead of returning them the bots. Consequently, a bot-generated traffic of 1 Gbit/s results in significantly stronger attack traffic of 43.3 Gbit/s at the attack target.

Distributing the bot-generated queries across multiple reflectors offers some key advantages to the attacker. For once, the attack traffic is no longer limited by the data rate of a single resolver. Alternatively, an attacker can seek to avoid detection by lowering the traffic volume generated from queries and responses at each resolver. Through this, it becomes more difficult to notice ongoing attacks from the view point of a resolver as they blend in with normal operations.

While Example 2.1.1 outlines core concepts of DNS amplification attacks, more effective amplification attacks can be launched through DNS. Stronger amplification can be achieved by carefully selecting queried information and resolvers. According to [Yaz+22], about 6.5% of open resolvers achieve an amplification factor of 125 or more.

[1] IP addresses under the IPv4 prefix `10/8` belong to a private address range that is not routed on the Internet. Addresses from this range are used in examples throughout this thesis instead of globally routable addresses to avoid confusion with real-world systems.

Service	Average BAF	Top BAF	Top X%	Sources
CharGen	358.8	n/a	n/a	[Ros14]
DNS	n/a	≥ 125.0	(6.5%)	[Yaz+22]
Memcached	n/a	$\geq 51,000.0$	(n/a)	[Cyb19; NIS18]
NTP	556.9	4,670.0	(10%)	[Ros14]
SNMP v2	6.3	11.3	(10%)	[Ros14]
SSDP	30.8	75.9	(10%)	[Ros14]

Table 2.1: Average BAF of selected protocols used in amplification attacks. The table also includes the top BAF reached on average by the top X percent of all amplifiers.

Alternatively, an attacker can control the DNS response size by querying information from a name server under its own control, e.g., by supplying a large number of resource records in ANY-type resource records for queries to a self-registered domain. Since DNS resolvers typically cache the responses, the attacker's own authoritative name server experiences only negligible load during an attack.

Alternative attack vectors. Table 2.1 provides a (non-exhaustive) list of alternative attack vectors for amplification attacks. The table also lists the average BAF and the highest BAF that is reached on average by the listed percentage of available amplifiers (when these values are available).

UDP-based protocols such as the Network Time Protocol (NTP) [Mar+10] and the Character Generator Protocol (CharGen) [Pos83] can achieve higher BAFs than DNS (358.8 and 556.9 on average compared to a top BAF of 125.0 for DNS) [Ros14]. For NTP, at least 1,405,186 amplifiers were available at the beginning of 2014 [Czy+14], underlining the potential for highly distributed amplification attacks. Despite offering a lower BAF, the Simple Network Management Protocol Version 2 (SNMP v2) [Pre02] and Simple Service Discovery Protocol (SSDP) [Alb+99] can contribute to volumetric DDoS attacks by offering additional attack vectors with different characteristics. Emerging attack vectors, such as OpenVPN-based amplification attacks, further increase the attack vector diversity [KDH21]. While DNS and NTP are bound to specific UDP port numbers (53 and 123), the SSDP protocol offers an attacker the advantage to use randomized port numbers [ALK22]. This can be used to evade port-based filter mechanism by launching amplification attacks from unpredictable ports.

Despite being connection-oriented, the TCP protocol can also be leveraged for amplification attacks with high BAF. Misconfigured instances of the memcached system (used for distributed in-memory caching) were reported to achieve BAFs of at least 51,000 [NIS18; SS18; Cyb19]. The initial queries to a memcached service can be sent via TCP or UDP and result in UDP-based response traffic. Using specific query patterns can raise the BAF of memcached to more than 140,000 [Kon+22] although it is unknown if these patterns are used in real-world attacks. Furthermore, middle boxes that parse HTTP queries sent via TCP can be manipulated into sending large responses to an attack target [Boc+21]. Automatically tuning the HTTP queries with genetic algorithms can achieve a BAF of up to 7455 by exploiting middle boxes designed to censor Internet traffic.

2.2 Network Abstractions

Networks cover a variety of complex use cases such as routing, traffic engineering, load balancing, monitoring, and many others. The concept for attack traffic removal in this thesis focuses on generating and establishing filter rules. Network paradigms for softwarized networks such as Software Defined Networking (SDN) [Jar+14; Kre+15] provide convenient abstractions for this use case. Complex decision making processes can be implemented in the control plane, while decisions can be enforced in the data plane by simple flow rules.

2.2.1 Flow Rules

Flow rules encode decisions about how traffic should be processed by a switch. For example, a flow rule can specify that IP packets addressed to a particular subnet should be discarded or forwarded through a specific switch port. To encode these decisions, each flow rule consists of a match, an action, and a priority that determine how a packet is processed.

- **Matches**. The matches $M = \langle \mathtt{m}_i \rangle_{i=1,2,\ldots}$ are a finite sequence of key-value pairs $\mathtt{m} = (\mathtt{k}, \mathtt{v})$. A key $\mathtt{k}$ references a single header field of a packet or meta data acquired during packet processing (such as the ingress port on which a packet is received). This data is compared to the value $\mathtt{v}$ to determine if a filter rule applies to a packet. The value $\mathtt{v}$ can specify a single numeric value or a bitmask that represents a range of values (such as an IP prefix representing a subnet).

Flow rule	Matches M	Actions A	Priority
$\mathcal{F}_a$	$\langle\, \mathtt{m}_1 = (\mathtt{k}_1 = \mathtt{IP\_PROTO}, \mathtt{v}_1 = \mathtt{IPv4}),$ $\mathtt{m}_2 = (\mathtt{k}_2 = \mathtt{IP\_SRC}, \mathtt{v}_2 = \mathtt{10.0.0/24}) \,\rangle$	$\langle \mathtt{a}_1 = \mathtt{DROP} \rangle$	$\mathtt{p} = 3$
$\mathcal{F}_b$	$\langle\, \mathtt{m}_1 = (\mathtt{k}_1 = \mathtt{IP\_PROTO}, \mathtt{v}_1 = \mathtt{IPv4}),$ $\mathtt{m}_2 = (\mathtt{k}_2 = \mathtt{IP\_SRC}, \mathtt{v}_2 = \mathtt{10.0/16}) \,\rangle$	$\langle \mathtt{a}_1 = \mathtt{OUTPUT:1} \rangle$	$\mathtt{p} = 2$
$\mathcal{F}_c$	$\langle\, \mathtt{m}_1 = (\mathtt{k}_1 = \mathtt{IP\_PROTO}, \mathtt{v}_1 = \mathtt{IPv4}),$ $\mathtt{m}_2 = (\mathtt{k}_2 = \mathtt{IP\_SRC}, \mathtt{v}_2 = \mathtt{10/8}) \,\rangle$	$\langle \mathtt{a}_1 = \mathtt{DROP} \rangle$	$\mathtt{p} = 1$

Table 2.2: Three example flow rules $\mathcal{F}_a, \mathcal{F}_b$, and $\mathcal{F}_c$ that match IPv4 packets originating from nested subnets of the IPv4 address space.

- **Actions**. The actions $A = \langle \mathtt{a}_i \rangle_{i=1,2,\ldots}$ are a finite sequence of actions $\mathtt{a}$ that are applied to a packet. The actions determine how a packet is processed and executed in the specified order. For example, an action $\mathtt{a} = \mathtt{DROP}$ can discard a packet, while an action $\mathtt{a} = \mathtt{OUTPUT:1}$ forwards it to the first port of a switch.

- **Priority**. The priority $\mathtt{p} \in \mathbb{N}$ determines the order in which a flow rule is evaluated relative to other rules. Flow rules with higher priority take precedence over rules with lower priority. Since matches of different flow rules do not have to be mutually exclusive, priorities serve as a tiebreaker to resolve conflicts when multiple flow rules match a packet.

Example 2.2.1. Table 2.2 lists three example flow rules $\mathcal{F}_a, \mathcal{F}_b$, and $\mathcal{F}_c$. The flow rules match the IPv4 source address of packets against different IPv4 subnets (specified in the matches $\mathtt{m}_2$) and they apply different actions $\mathbf{a}_1$. $\mathcal{F}_a$ and $\mathcal{F}_c$ discard matching packets, while $\mathcal{F}_b$ forwards packets on the first output port of a switch. For packets from the IPv4 subnet `10.0.0/24`, all three rules would have a positive match, so the matches alone are not enough to decide which rule to apply. This conflict is resolved by the different priorities $\mathtt{p}$, resulting in $\mathcal{F}_a$ being applied due to having the highest priority. In effect, $\mathcal{F}_a$ constitutes an exception for packets from the subnet `10.0.0/24` to the forwarding of packets from the subnet `10.0/16` performed by $\mathcal{F}_b$. Similarly, $\mathcal{F}_b$ exempts some packets from being discarded by $\mathcal{F}_c$, which matches a larger subnet.

Flow rules are often assumed to be programmed into a switch by a network controller. Through this, complex decision making can be delegated to a control plane, while switches perform fast traffic processing in the data plane of a network.

2.2.2 Flow Tables

Flow tables constitute a central component of a switch. They store and evaluate flow rules during packet processing. Switches often support multiple flow tables (either logically or through direct hardware support) to organize flow rules that perform different tasks during packet processing. For example, a flow table can be dedicated to enforcing access control decisions or to control the forwarding of packets.

Flow tables that allow IP prefix-based matching have a specific requirement, namely, to ignore a part of a packet's source or destination IP address in order to compare only the relevant bits of an IP prefix. This requires hardware support if matching must be performed at line speed. Therefore, flow tables that allow prefix-based matches are often realized as Ternary Content-Addressable Memory (TCAM). This memory technology enables parallel evaluation of flow rule matches that are based on IP prefixes. Through this, all matches of all flow rules in a flow table can be compared to packet header field simultaneously, which facilitates fast packet processing. However, this memory technology involves high monetary costs and high energy consumption, which leads to capacity constraints for flow tables [AÖ19; AMA19; Isy+20]. Several research efforts focus on addressing capacity constraints of switches arising from limited flow table capacities, for example, through software offloading [SC16; YXC18], delegation of flows to other switches [Bau20], or through flow table compression [Zha+20; Zho+22a].

2.3 Monitoring High-Volume Traffic

To separate volumetric DDoS attack traffic from legitimate traffic, the combined traffic must be monitored and analyzed. This involves the challenge of monitoring high-volume traffic at line speed. Otherwise, monitoring overhead can result in additional forwarding delays, which can further exacerbate the congestion caused by volumetric DDoS attacks. If a monitoring system becomes a bottleneck, upstream network systems are forced to buffer packets to the point where they themselves become congested. As a result, the impact of a volumetric DDoS attack then spreads further upstream, affecting additional parts of the network infrastructure.

A characteristic feature of volumetric DDoS attacks is, by definition, the high combined attack traffic volume. This feature can be used as a starting point to identify individual attack traffic sources or subnets from which attack traffic originates. For this purpose,

a so-called stream summary can be computed from monitoring information in order to track the most frequently occurring elements in a data stream. In the context of network monitoring, such a stream summary tracks the traffic volume (in terms of packets or bytes) originating from a single source or a subnet. In order to remain efficient, a stream summary should address the following key requirements:

- **Fast packet processing**. To avoid introducing additional monitoring overhead, each packet should be processed within a limited time budget (determined by the maximum data rate of a link). This imposes a constant upper bound on the number of compute operations that can be performed for each packet.
- **Memory efficiency**. Network traffic should be monitored directly in the data plane to ensure time constraints on packet processing. As a consequence, the size of a data structure that is necessary to compute a stream summary is limited by the memory available in the data plane of network devices. Since the available memory is constrained, it imposes a constant upper bound on the memory requirement of a stream summary.
- **Online processing**. A consequence of limiting memory requirements is that each packet should only be processed once. If a packet needs to be stored to be processed multiple times, the memory requirements of a stream summary can grow with the number packets of in a data stream. This would violate the constant upper bound on the memory requirement.

In particular, the requirement of memory efficiency prevents the computation of exact stream summaries for streams of arbitrary length. Such a data stream could consist of entirely different elements. In this case, the size of the data structure that tracks the exact traffic volume of each element would necessarily have to grow with the (arbitrary) length of the stream. To maintain memory efficiency, network monitoring typically relies on approximate stream summaries. The following sections describe algorithmic designs for efficiently computing stream summaries of network traffic.

2.3.1 Heavy Hitter Algorithms

A first building block in identifying attack traffic sources is to find individual sources that send at least a certain fraction of the total traffic volume. This is captured in the notion of Heavy Hitters (cf. [CCF02; Cor+03; MAE05]). In the context of attack traffic filtering, these are defined as follows:

Definition 2.1: Heavy Hitter

Let $\mathcal{S}$ denote a finite data stream of length $n \in \mathbb{N}$ comprised of a sequence of packets $\langle \mathcal{P}_i \rangle_{i=1,\ldots,n}$, where each packet $\mathcal{P}_i = (\mathsf{p}_i, \mathsf{v}_i)$ has a fully qualified prefix p_i (its source address) and an associated volume v_i (the number of bytes in the packet). Further, let $\mathcal{V} = \sum_{(\mathsf{p}_i,\mathsf{v}_i)\in\mathcal{S}} \mathsf{v}_i$ denote the total traffic volume of $\mathcal{S}$ and $\phi \in [0,1]$ a chosen threshold. Then, a **Heavy Hitter** is a prefix p that sends at least a fraction ϕ of the total traffic volume $\mathcal{V}$, i.e.,

$$\phi\mathcal{V} \leq V_{\mathsf{p}} \stackrel{\text{def}}{=} \sum_{\{(\mathsf{p}_i,\mathsf{v}_i)\in\mathcal{S} \mid \mathsf{p}_i=\mathsf{p}\}} \mathsf{v}_i \tag{2.1}$$

Heavy Hitters can be used directly to identify *individual* traffic sources that send high-volume traffic. For example, a Heavy Hitter can represent a single amplifier with a high bandwidth amplification factor.

Definition 2.1 provides a notion of *exact* Heavy Hitters. In general, the computation of exact Heavy Hitters is not feasible when monitoring high-volume traffic, since the exact computation does not meet the efficiency requirements outlined above. Several optimized algorithm designs have been proposed to realize more efficient monitoring. The following outlines the operating principles of two well-studied algorithms.

Space Saving. The Space Saving algorithm [MAE05] uses a fixed number of counters to perform approximate monitoring of most frequently occurring elements in a data stream. More precisely, limiting the number of counters maintains a constant memory requirement but allows a small error in the estimation of the traffic volume. Essentially, Space Saving ensures a maximum relative error $\varepsilon \in (0,1]$ in the estimated volume V_{p} of a prefix p when monitoring at most $\frac{1}{\varepsilon}$ prefixes (cf. Theorem 3 in [MAE05]). In addition, Space Saving guarantees that any prefix with a true volume greater than $\phi\mathcal{V}$ is reported as a Heavy Hitter.

To limit the memory required by its data structures, Space Saving restricts the number of monitored prefixes to a maximum of $m \in \mathbb{N}$. As long as this maximum is not reached, a new entry is created in the data structure for every currently unmonitored prefix. The traffic volume corresponding to each prefix is stored in an ordered linked list of counters c_i ($i \leq m$). When the maximum number of prefixes is reached and a currently unmonitored prefix is processed, the prefix with the lowest counter value is replaced by the new prefix and the counter is increased. Figure 2.2 illustrates this update strategy for an Space Saving algorithm that monitors at most two prefixes. Initially, the data structure tracks the traffic volume of two prefixes p_1 and p_2 with

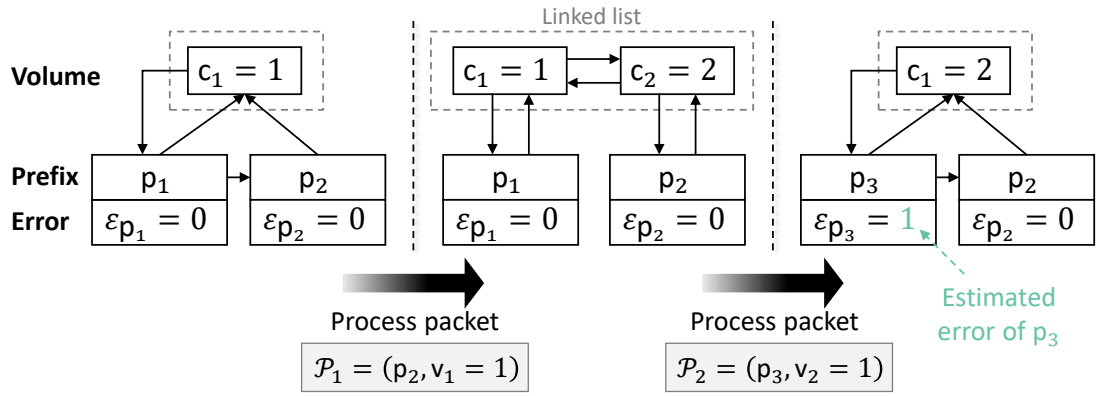


Figure 2.2: Update of a Space Saving data structure that monitors at most two prefixes. The data structure initially monitors two prefixes p_1 and p_2 using the counter c_1. Two additional packets $\mathcal{P}_1$ and $\mathcal{P}_2$ with identical volume $v_1 = v_2 = 1$ but different prefixes p_2 and p_3 are processed. This results in an estimated error $\varepsilon = 1$ for the monitored prefix p_3 (cf. [MAE05]).

a counter c_1. When processing the packet $\mathcal{P}_1$ with prefix p_2 and volume $v_1 = 1$, a counter $c_2 = c_1 + v_1 = 2$ is inserted into the linked list. At this point, the maximum number of prefixes m is reached. When the packet $\mathcal{P}_2$ with a new prefix p_3 and volume $v_2 = 1$ is processed, the prefix p_3 replaces prefix p_1, which has the lowest counter value. In addition, the prefix p_3 tracks the previous counter value of p_1 as an error ε_{p_3}. When reporting Heavy Hitters, the Space Saving algorithm checks if $\phi\mathcal{V} \leq c - \varepsilon$ for monitored prefixes with counter value c and error ε. Considering the error ε ensures that reported elements are in fact Heavy Hitters.

Sketch-based algorithms. Similar to the Space Saving algorithm, a Count-Min Sketch algorithm [CM05] monitors traffic volume with a fixed number of counters to achieve memory efficiency. Instead of maintaining a list, a Count-Min Sketch uses d pairwise-independent hash functions in combination with a two-dimensional array of width w and depth d. Each array entry constitutes a counter $c_{i,j}$ $(i = 1, \ldots, d; j = 1, \ldots, h)$ that monitors the traffic volume of one or more prefixes. Specifically, when a new packet $\mathcal{P}_i = (p_i, v_i)$ is processed, the counters $c_{1,h_1(p_i)}, \ldots, c_{d,h_d(p_i)}$ that need to be updated are determined by applying the hash functions h_i to the packets address p_i. The current counter values are then incremented by the packets volume v_i. This update procedure is shown in Figure 2.3.

When multiple packets are processed sequentially by a Count-Min Sketch, hash collisions can occur among different prefixes as the width of the sketch data structure is limited. This results in counters storing an over-estimated traffic volume for one

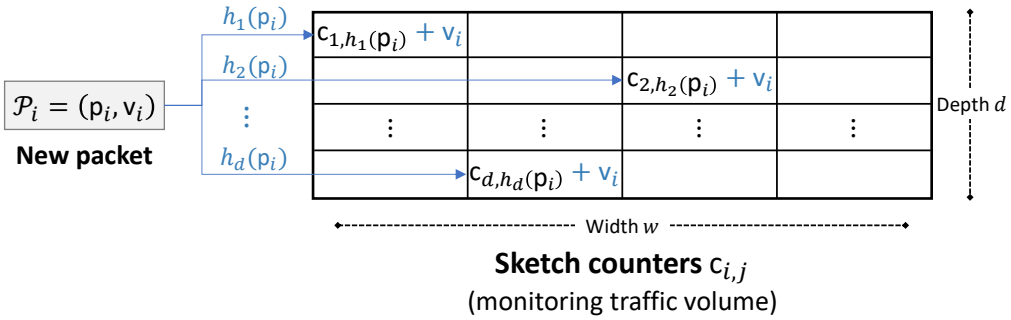


Figure 2.3: Update of a Count-Min Sketch with height h and depth d upon arrival of a new packet $\mathcal{P}_i = (p_i, v_i)$ (with source address p_i and volume v_i). Counters that must be updated are indexed by d pairwise-independent hash functions $h_1, \ldots, h_d$. The counters $c_{1,h_1(p_i)}, \ldots, c_{d,h_d(p_i)}$ are incremented by the packet's volume v_i.

or more prefixes. To reduce the error, a Count-Min Sketch reports the minimum value of the counters $c_{1,h_1(p)}, \ldots, c_{d,h_d(p)}$ when queried for the traffic volume of a prefix p. Through this, the maximum over-estimation is at most $\varepsilon\mathcal{V}$ with probability at least $1 - \delta$ when the Count-Min Sketch data structure has width $w = \lceil \frac{e}{\varepsilon} \rceil$ and depth $d = \ln(1/\delta)$ (see Theorem 1 in [CM05]). As a result, a Count-Min Sketch can efficiently determine if a given prefix is a Heavy Hitter (with probabilistically bounded error). However, a Count-Min Sketch cannot report all Heavy Hitters, as the data structure does not store the monitored prefixes. For this, a different update strategy and data structure is required, as outlined below.

Recently introduced sketch-based algorithms such as Nitrosketch [Liu+19], the work of [Ben+20], ARMHH [HWH21], and CocoSketch [Zha+21] focus on hardware-optimized designs. This facilitates direct integration into the data plane to achieve traffic processing at line speed. CocoSketch uses a two-dimensional array and pairwise-independent hash functions h_i like the Count-Min Sketch. However, the array of CocoSketch stores a prefix $p_{i,j}$ alongside each counter $c_{i,j}$ and the algorithm employs a different update strategy.

When updating the counters of a prefix p, the algorithm distinguishes the following cases. If a counter $c_{i,h_i(p)}$ has no associated prefix $p_{i,j}$, the algorithm directly increases the counter value by a packet's volume v and updates the associated prefix. Otherwise, the prefix $p_{i,j}$ is replaced by the newly monitored prefix p with probability $v/c_{i,h_i(p)}$, where v denotes the volume by which a counter is increased. The probabilistic prefix update ensures that the error in traffic volume estimations of prefixes satisfies

an upper bound, depending on the choice of the width w and the depth d of the two-dimensional array (see Theorem 3 in [Zha+21]).

2.3.2 Hierarchical Heavy Hitter Algorithms

Heavy Hitter algorithms monitor the traffic volume sent by individual traffic sources that are represented by fully qualified prefixes. However, in a volumetric DDoS scenario, individual attacking systems do not need to generate high traffic volume on their own. Only the *combined* attack traffic volume needs to be high enough to reduce service availability. This makes it difficult to locate attack traffic sources when the attack traffic is spread across a large number of sources.

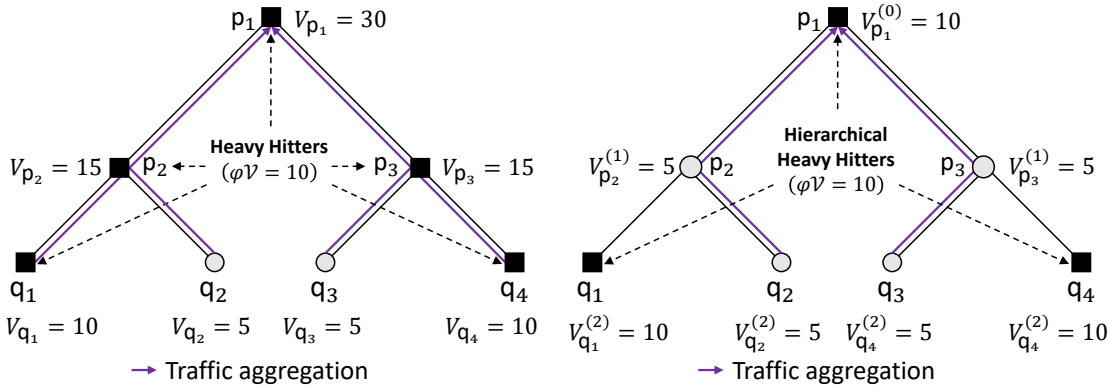
Instead of monitoring individual traffic sources, traffic volume can be aggregated over the IP prefix hierarchy to identify subnets (instead of individual traffic sources) that send high traffic volumes. For example, the traffic volume originating from IP addresses `10.0.0.0` through `10.0.0.255` can be summarized and attributed to the common parent prefix `10.0.0/24`.[2] If the combined traffic volume of the prefix `10.0.0/24` is exceptionally high, this can serve as an indicator for a potential presence of attack traffic sources in the corresponding IP address range.

For any two prefixes p and q, the relationship that p is an parent prefix of q is denoted as $\mathsf{q} \prec \mathsf{p}$. Conversely, q is said to be **generalizable** to p. For a set $\mathcal{Q} = \{\mathsf{q}_i\}_{i=1,\dots,n}$ of fully qualified prefixes with traffic volumes V_{q_i} and common parent prefix p, the aggregated traffic volume of p can be calculated as

$$V_{\mathsf{p}} = \sum_{\mathsf{q}_i \in \mathcal{Q}} V_{\mathsf{q}_i} \,. \tag{2.2}$$

Given the aggregated traffic volume V_{p} of all prefixes p, Heavy Hitters can be detected with respect to a threshold $\phi\mathcal{V}$ on each level of the IP prefix hierarchy (as in Definition 2.1). However, this unconditional aggregation results in every parent prefix of a Heavy Hitter also becoming a Heavy Hitter, as shown in the following example.

[2] By convention, the first address in a subnet is reserved to identify a subnet while the last address serves as broadcast address. Nevertheless, these addresses are still part of the IP prefix hierarchy and therefore included in the address range of example IP prefixes.



(a) Heavy Hitters. **(b)** Hierarchical Heavy Hitters.

Figure 2.4: Example aggregation of traffic volume over a hierarchy with four fully qualified prefixes $q_1, \ldots, q_4$ and three parent prefixes $p_1, \ldots, p_3$. The traffic volumes associated with q_1 and q_4 are 10 while those of q_2 and q_3 are 5.

Example 2.3.1. Figure 2.4a illustrates the aggregation of traffic volume of four fully qualified prefixes q_1, ..., q_4 to their parent prefixes p_1, p_2, and p_3. The numerical example uses traffic volumes of 10 for q_1 and q_4 as well as volumes of 5 for q_2 and q_3. A prefix p is classified as a Heavy Hitter if the aggregated traffic volume V_p reaches the threshold $\phi\mathcal{V} = 10$. Since q_1 and q_4 are Heavy Hitters, all parent prefixes automatically become Heavy Hitters because they aggregate the traffic of q_1 and q_4 and the traffic volume increases monotonically as the aggregation progresses.

The automatic classification of parent prefixes as Heavy Hitters effectively reduces insight into the distribution of the traffic volume over the IP address space. Instead of aggregating traffic volume unconditionally, the aggregation can be stopped when a prefix becomes a Heavy Hitter. In this case, parent prefixes become Heavy Hitters only if the *remaining* traffic volume (that is not part of other Heavy Hitters) is sufficient to reach a chosen threshold. The following definition formalizes this idea.

Definition 2.2: Hierarchical Heavy Hitter (HHH)

Let $\mathcal{S}$ denote a data stream of length $n \in \mathbb{N}$ comprised of a sequence of packets $\langle \mathcal{P}_i = (\mathsf{p}_i, \mathsf{v}_i) \rangle_{i=1,\dots,n}$ as in Definition 2.1. Further, let $\mathcal{V} = \sum_{(\mathsf{p}_i,\mathsf{v}_i)\in\mathcal{S}} \mathsf{v}_i$ denote the total traffic volume of $\mathcal{S}$ and $\phi \in [0,1]$ a chosen threshold. Then, the set of **Hierarchical Heavy Hitters (HHHs)** $\mathcal{H}$ is the set $\mathcal{H}_0$, which is defined recursively over a hierarchy of prefixes p that have a maximum length L (cf. [Cor+08]):

- $\mathcal{H}_L$ denotes the set of fully qualified prefixes that are Heavy Hitters of $\mathcal{S}$.
- $\mathcal{H}_l \stackrel{\text{def}}{=} \mathcal{H}_{l+1} \cup \left\{ \mathsf{p} \mid \mathsf{length}(\mathsf{p}) = l \wedge V_{\mathsf{p}}^{(l)} \geq \phi\mathcal{V} \right\}$

Here, $\mathsf{length}(\mathsf{p})$ denotes the length of a prefix p and the **conditional traffic volume** $V_{\mathsf{p}}^{(l)}$ is defined as follows:

$$V_{\mathsf{p}}^{(l)} = \sum_{\{(\mathsf{p}_i,\mathsf{v}_i)\in\mathcal{S} \mid \mathsf{p}_i \prec \mathsf{p} \,\wedge\, \nexists \mathsf{q}\in\mathcal{H}_l : \mathsf{p}_i \prec \mathsf{q}\}} \mathsf{v}_i \tag{2.3}$$

A prefix $\mathsf{p} \in \mathcal{H}$ is referred to as **HHH prefix**, while ϕ and $\phi\mathcal{V}$ are called the **relative** and **absolute HHH detection thresholds**.

Example 2.3.2. Figure 2.4b illustrates the aggregation of traffic volume for the prefixes from Example 2.3.1 when using HHHs instead of Heavy Hitters. Only the prefixes $\mathsf{p}_1 \in \mathcal{H}_0$ and $\mathsf{q}_1, \mathsf{q}_4 \in \mathcal{H}_2$ are classified as an HHH prefixes. The conditional traffic volumes $V_{\mathsf{p}_2}^{(1)}$ and $V_{\mathsf{p}_3}^{(1)}$ of prefixes p_2 and p_3 do not reach the threshold $\phi\mathcal{V}$. This is because the conditional summation in Equation 2.3 excludes the volume of prefixes that are generalizable to longer HHH prefixes (in this case q_1 and q_4). Since p_2 and p_3 are omitted from the set of HHHs, these prefixes no longer directly indicate that q_2 and q_3 are sources of high-volume traffic (unlike in Example 2.3.1). Only when the traffic of q_2 and q_3 converges at the top of the hierarchy, the additional HHH prefix p_1 is detected. By considering that q_1 and q_4 are HHH prefixes, the origin of the traffic aggregated by p_1 can be traced back directly to q_2 and q_3.

2.4 Deep Learning

The concept of (artificial) neural networks was inspired by the structure of organic brains [MP43]. The core idea is to use artificial neurons (like perceptrons [Ros58]),

which provide simple computational models of biological neurons, and combine them in a computational graph. This enables the composition of neural networks representing more complex computational models, analogous to a cluster of biological neurons connected by synapses. With a neural network of sufficient size, any measurable function can be approximated arbitrarily well [HSW89; Cyb89; Fun89; Hor91; Lu+17]. In particular, two basic tasks in machine learning can be addressed using neural networks:

- **Regression analysis**. In regression analysis (or **regression** for short), function approximation serves to learn a model of the form $\mathsf{y}_i \approx f(\mathsf{x}_i, \theta)$, $(i = 1, 2, \ldots)$. That is, a neural network learns the parameters θ of a function f to closely resemble the relationship between (error-free) independent variables x_i and dependent variables y_i. Dependent variables may contain an error $\mathsf{y}_i^{\mathsf{ERROR}}$, so that $\mathsf{y}_i = \mathsf{y}_i' + \mathsf{y}_i^{\mathsf{ERROR}}$ (where y_i' is error-free). Such an error can arise, for example, from random noise in measurements. The variables $\mathsf{x}_i, \mathsf{y}_i$, and θ can be scalars, vectors, or tensors with multi-dimensional shapes. In the context of machine learning, the independent variables x_i and the dependent variables y_i are also referred to as **features** and **targets**, while the actual output $\widehat{\mathsf{y}} = f(\mathsf{x}_i, \theta)$ of a neural network is called **prediction**.

- **Classification**. In classification, the coefficients of y_i and y_i' are restricted to binary values. That is, a classifier assigns each input feature x_i one or more classes that are represented by non-zero coefficients of y_i. Optionally, variables y_i with only zero coefficients can be interpreted as a separate class. The variables y_i are also referred to as **labels** in machine learning.

The function f is determined by the structure of a chosen neural network. However, the parameters θ are typically learned in an automatic fashion through **training** on datasets, which comprise **samples** consisting of features and targets or labels. Once a neural network is trained, it can be used to perform **inference**, where it outputs an (approximated) function value without changing its learned parameters.

The computational effort of training and inference depends on the structure of the computational graph. The graph of a neural network is typically organized into **layers**, where the outputs of neurons in one layer constitute the inputs for neurons in subsequent layers. Using shallow neural networks with one layer may require a large number of parameters to approximate a function sufficiently well [Bar94]. The large number of parameters must be learned during training and evaluated during inference, resulting in high computational complexity.

In contrast, **deep learning** uses neural networks with more layers, which can often reduce the number of parameters required [Mon14; LTR17]. The idea behind using **deep neural networks** is that earlier layers can be used to detect regularities in the input data. This information can then be reused by subsequent layers. For example, early layers may be used to detect lines and circles in an image, while subsequent layers can use this information to recognize more complex objects (such as specific traffic signs). Reusing already available information at multiple points throughout the computational graph of a neural network can result in fewer parameters being required. This reduces not only the memory requirements of a neural networks, but also the computational effort involved in training and performing inferences.

2.4.1 Building Blocks of Neural Networks

One of the essential building blocks of a neural network is the neuron. While a single neuron provides only rudimentary functionality, neural networks can combine multiple neurons into a computational graph to perform complex tasks. The set of neurons is usually organized into layers that determine the flow of data between groups of neurons. Instead of specifying the entire computational graph, a neural network is often described in terms of its layers and how they are connected. The essential concepts behind neurons and different types of layers are elaborated below (for further details see [GBC16; Agg18]).

Neurons. Each neuron of a neural network calculates a weighted sum $\mathsf{x} \cdot \mathsf{w}^\top$ of the coefficients of a feature vector $\mathsf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and a weight vector $\mathsf{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$, $(n \in \mathbb{N})$. Optionally, a neuron can add a **bias** $b \in \mathbb{R}$ to calculate $a = b + \mathsf{x} \cdot \mathsf{w}^\top$. The value $a \in \mathbb{R}$ constitutes the input of an **activation function** $\mathfrak{f} : \mathbb{R} \to \mathbb{R}$. Activation functions can introduce non-linearity, which is required for a neural network to learn a non-linear relationship between features and targets or labels. In short, each neuron calculates an output value $y \in \mathbb{R}$ as follows:

$$y = \mathfrak{f}(a) = \mathfrak{f}\left(b + \mathsf{x} \cdot \mathsf{w}^\top\right) = \mathfrak{f}\left(b + \sum_{i=1}^{n} x_i \cdot w_i\right) \tag{2.4}$$

Figure 2.5 illustrates the structure and data flow of a single neuron with five features and weights, a bias, and an activation function. The weights $w_1, \ldots, w_n$ and the

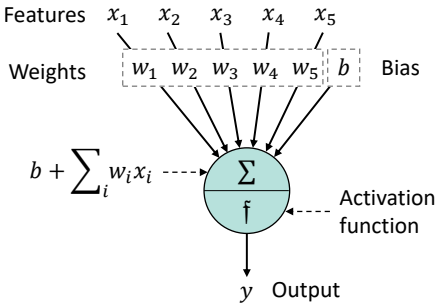


Figure 2.5: A neuron with activation function ƒ, weights $w_1, \ldots, w_5$, and bias b.

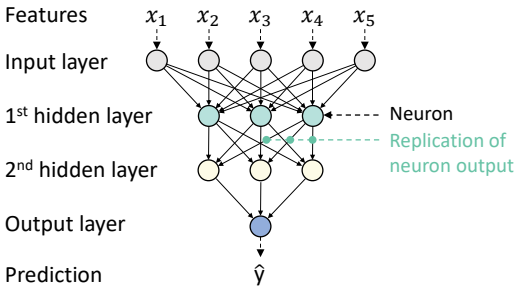


Figure 2.6: Example computational graph of a feed-forward neural network.

bias b of a neuron are determined during training. To introduce non-linearity, several different activation functions ƒ can be selected (as described below).

Layers. Each layer of a neural network processes the output of one or more previous layers. An example computational graph of a neural network with multiple layers is shown in Figure 2.6. Layers between the input and output layer are referred to as **hidden layers**. The input layer does not perform any computations, but simply replicates the features $x_1, \ldots, x_n$ to each neuron in the first hidden layer. Each neuron in the first hidden layer then applies its weights, bias, and activation function to the replicated features. Processing continues for each subsequent layer until the output layer produces the network's prediction.

The specific architecture of a neural network that feeds the output of one layer directly into all neurons of the subsequent layer is referred to as a **feed-forward neural network**. This kind of neural network is fully defined by specifying the number of neurons in each layer and the neuron's activation function. More complex neural networks can be designed by explicitly controlling the flow of data between layers and by using more complex layer structures. The following describes different types of layers used in the design of neural networks throughout this thesis.

- **Fully-connected layers**. A fully-connected layer connects all of its neurons to all neurons in the preceding layer. That is, if L is a fully-connected layer with n neurons and its preceding layer L' has m neurons, then the weights that the layer L applies to the (replicated) outputs $\mathsf{x} = (x_1, \ldots, x_m)$ of L' can be represented in a matrix $\mathsf{W} \in \mathbb{R}^{n \times m}$. Consequently, the application of L to its input x can be expressed with the following equation:

$$\mathsf{y} = \mathfrak{f}\left(\mathsf{b} + \mathsf{x} \cdot \mathsf{W}^{\top}\right) \tag{2.5}$$

Here, $\mathfrak{f}$ denotes the activation function of L, $\mathsf{b} \in \mathbb{R}^n$ is the vector of bias values of the neurons in L and $\mathsf{y} \in \mathbb{R}^n$ is the output of the fully-connected layer.

- **Convolutional layers**. A convolutional layer repeatedly calculates a dot product between the coefficients of an input tensor X and a smaller tensor K (called **kernel**) while passing the kernel K over the tensor X (at regular intervals). Neural networks that employ convolutional layers are referred to as **Convolutional Neural Networks (CNNs)** [LeC89; LBH15] and are commonly used to process inputs with a spatial structure, such as images. For example, a tensor $\mathsf{X} \in \mathbb{R}^{m_1 \times m_2}$ can represent a (two-dimensional) gray-scale image, to which a kernel $\mathsf{K} \in \mathbb{R}^{n_1 \times n_2}$ is applied. The kernel K is advanced in fixed-size steps s_1 and s_2 (called **strides**) along the first and second dimensions of the image. Using larger strides reduces the size of the output image, so that the meaningful information contained in high-dimensional input data can be encoded as lower-dimensional representations. This produces an image $\mathsf{Y} \in \mathbb{R}^{((m_1-n_1)/s_1+1)\times((m_2-n_2)/s_2+1)}$ as layer output.[3] When $x_{i,j}$ and $\kappa_{i,j}$ denote the coefficients of X and K, the coefficients $y_{i,j}$ of Y are given by the following equation (cf. [GBC16]):

$$y_{i,j} = \sum_{u=0}^{n_1} \sum_{v=0}^{n_2} \kappa_{u,v} \cdot x_{i \cdot s_1 + u, j \cdot s_2 + v} \tag{2.6}$$

In addition to calculating the dot product in Equation 2.6, an activation function $\mathfrak{f}$ can be applied to each coefficient $y_{i,j}$ to introduce non-linearity. Furthermore, multiple kernels $\mathsf{K}_1, \ldots, \mathsf{K}_d$ can be used to process the same input tensor. In this case, each kernel calculates dot products with different coefficients and acts as a **filter** that extracts different information from the same part of the input tensor. The shape of the output then gains an additional dimension of size d (referred to as **depth**) to store outputs from different filters.

[3] The shapes of the tensor X and the kernel K may need to be adjusted so that the strides s_1 and s_2 become integer divisors of the dimensions $m_1 - n_1$ and $m_2 - n_2$. This is commonly accomplished by adding additional coefficients to the tensor X via so-called padding.

Convolutional layers can also be generalized to accept three-dimensional input data. This includes images with multiple channels to represent different colors or output of previous convolutional layers that has a depth greater than one. In this case, the input $\mathsf{X} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$, the d kernels $\mathsf{K}^{(l)} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ $(l = 0, \ldots, d-1)$, and the output $\mathsf{Y} \in \mathbb{R}^{((m_1-n_1)/s_1+1) \times ((m_2-n_2)/s_2+1) \times d}$ have a three-dimensional shape and coefficients $x_{i,j,k}, \kappa^{(l)}_{i,j,k}, y_{i,j,k}$. The input X and the kernels $\mathsf{K}^{(l)}$ are required to have an identical depth $n_3 = m_3$ in order to calculate the dot product. The coefficients of Y are then given as follows (cf. [Agg18]):

$$y_{i,j,l} = \sum_{u=0}^{n_1} \sum_{v=0}^{n_2} \sum_{w=0}^{n_3} \kappa^{(l)}_{u,v,w} \cdot x_{i \cdot s_1 + u, j \cdot s_2 + v, w} \tag{2.7}$$

Given the shape (m_1, m_2, m_3) of the input data, a convolutional layer is fully defined by the kernel's shape (n_1, n_2, n_3), the strides (s_1, s_2) in each dimension, the number of filters d, and the activation function $\mathfrak{f}$. The depth n_3 of the kernel's shape can be omitted, since it is determined the input's shape.

The coefficients of a kernel K are learned during training, which eliminates the need to specify them explicitly. Instead, convolutional layers can be (automatically) trained to extract different features from the spatial structure of the input for processing by subsequent layers.

- **Max-Pooling layers**. Similar to a convolutional layer, a max-pooling layer passes over a tensor X, but it determines the maximum value among a range of coefficients of X instead of calculating a dot product. The range of coefficients is determined by the shape of the max-pooling layer's kernel K (a rectangular shape), and the position at which the kernel is applied. Specifically, given a tensor $\mathsf{X} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$ with coefficients $x_{i,j,k}$, a kernel of size $n_1 \times n_2$, and strides s_1, s_2, the coefficients $y_{i,j,l}$ of the output tensor Y are as follows:

$$y_{i,j,k} = \max \left\{ x_{i \cdot s_1 + u, j \cdot s_2 + v, k} \mid u = 0, \ldots, n_1, v = 0, \ldots, n_2 \right\} \tag{2.8}$$

The last dimension of the shape of Y has size m_3 (like X) because the kernel K is applied independently to each sub-tensor with two-dimensional shape across the depth of X. Consequently, a max-pooling layer is fully defined by the kernel's size and the strides when the shape of the input data is known. The

strides of a max-pooling layer are commonly chosen to be grater than one, so that a max-pooling layer reduces the dimension of the input. Combining multiple convolutional layers and max-pooling layers often serves to train a neural network to increasingly focus on lower-dimensional representations of the most relevant information in the input data.

- **Dropout layers**. A dropout layer [Hin+12] is designed to reduce the risk of overfitting on training data. Generally, averaging over the outputs of an ensemble of different neural networks can serve to reduce errors. The idea behind dropout layers is to simulate using an ensemble of neural networks by randomly disabling the input to neurons of a single neural network. As a result, a neural network's architecture effectively changes with each training step, simulating the training of different neural networks. Each dropout layer disables the input of neurons of a specific layer with a fixed probability during training. By introducing multiple dropout layers into a network architecture, neurons in different layers can be disabled at random. Through this, large portions of the network can change repeatedly during training.

Activation functions. Each neuron applies an activation function $\mathfrak{f}$ as in Equation 2.4. In the context of deep learning, a variety of activation functions has been proposed in the past [DQZ18; Api+21; DSC22]. Common choices include the following functions, which are applied to an input $x \in \mathbb{R}$:

$$\mathsf{id} : x \mapsto x \tag{2.9}$$

$$\mathsf{ReLU} : x \mapsto \max(0, x) \tag{2.10}$$

$$\mathsf{sigmoid} : x \mapsto \frac{1}{1 + e^{-x}} \tag{2.11}$$

$$\tanh : x \mapsto \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.12}$$

The identity id does not change the output of a neuron. This function is commonly used for the activation in the last layer of a neural network that performs a regression, so that the range of predicted target values remains unrestricted. In contrast, the ReLU and sigmoid functions as well as the hyperbolic tangent tanh restrict the range of possible output values. These functions primarily serve to introduce non-linearity into a neural network architecture. In comparison to sigmoid and tanh, the ReLU function is easier to compute while yielding comparative results.

2.4.2 Training neural networks

Training neural networks is an iterative optimization process that seeks to gradually reduce the error between a neural network's prediction $\hat{\mathsf{y}}$ and the correct target or label y. The error is reduced by repeatedly adjusting the networks weights, which can be accomplished through gradient descent.

Stochastic gradient descent. To adjust a neural networks weights, gradient descent alternates between two passes through the network's layers (cf. [Agg18]):

- **Forward pass**. A forward pass chooses one or more training **samples** from a training dataset, which consist of features x and corresponding targets or labels y. The training algorithm then computes a prediction $\hat{\mathsf{y}}$ for each input x using the current weights of the neural network. Next, y and $\hat{\mathsf{y}}$ are compared using a differentiable **loss function** $\mathcal{L}$ to obtain a **loss** $l = \mathcal{L}(\mathsf{y}, \hat{\mathsf{y}})$. The loss indicates how far the neural network's output deviates from the correct output.

- **Backward pass**. Given the loss l from the forward pass, **gradients** of the loss function with respect to the weights of a neural network can be calculated. The gradients serve as indicators of how the weights of a neural network need to be adjusted in order to reduce the loss. Gradients are determined through **back-propagation**, which repeatedly applies the chain rule of differential calculus starting at the output layer of a neural network. This allows the gradients of weights in a layer L_i to be determined after the gradients up to the next layer L_{i+1} have been calculated in the backward direction. In a neural network with n layers, the gradients g_{w_i} given by the partial derivative $\frac{\partial \mathcal{L}}{\partial w_{\mathsf{L}_i \to \mathsf{L}_{i+1}}}$ for weights $w_{\mathsf{L}_i \to \mathsf{L}_{i+1}}$ on connections $\mathsf{L}_i \to \mathsf{L}_{i+1}$ can be calculated recursively (see [Agg18]):

$$g_{w_i} \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial w_{\mathsf{L}_i \to \mathsf{L}_{i+1}}} = \delta(\mathsf{L}_{i+1}, \mathsf{L}_n) \cdot \mathsf{L}_i \,, \tag{2.13}$$

$$\text{with} \quad \delta(\mathsf{L}_{i+1}, \mathsf{L}_n) \overset{\text{def}}{=} \mathsf{f}'(a_{i+1}) \cdot \sum_{\{\mathsf{L} \mid \mathsf{L}_{i+1} \to \mathsf{L}\}} w_{\mathsf{L}_{i+1} \to \mathsf{L}} \cdot \delta(\mathsf{L}, \mathsf{L}_n) \,, \tag{2.14}$$

$$\text{and} \quad \delta(\mathsf{L}_n, \mathsf{L}_n) \overset{\text{def}}{=} \mathsf{f}'(a_n) \cdot \frac{\partial \mathcal{L}}{\partial \hat{\mathsf{y}}} \,. \tag{2.15}$$

Here, $\mathsf{f}'(a_i)$ denotes the derivative of the activation function of layer L_i being applied to its input a_i. The term $\delta(\mathsf{L}_{i+1}, \mathsf{L}_n)$ is recursively evaluated and initial-

ized by $\delta(\mathsf{L}_n, \mathsf{L}_n)$ in a backward pass that starts at the output layer L_n. Efficient computation of gradients can be achieved through dynamic programming.

Multiple samples can be processed in parallel as a **batch** to speed up the training process. In this case, the calculation of gradients averages across all losses of the batch. The processing of all samples in a training dataset is referred to as an **epoch**. A batch is usually sampled randomly from the training data to introduce a stochastic process that breaks correlations between consecutive samples in a dataset.

While gradients indicate how the weights of a neural network need to be adjusted, using gradients directly for weight updates comes with a drawback. Large gradients calculated for one batch can significantly reduce the losses for that batch, but the weight updates may increase the losses for other batches. In this case, updates are said to overshoot, which can prevent training from minimizing losses across an entire dataset. Instead, more gradual updates can be performed over multiple epochs by multiplying gradients with a **learning rate** $\lambda \in \mathbb{R}_{>0}$. A new value w' for a weight w and a gradient g_w is then calculated as follows:

$$w' \quad = \quad w - \lambda g_w \tag{2.16}$$

A high learning λ can accelerate the training of a neural network but may fail to minimize losses. Choosing the learning rate too low may result in unnecessarily slow training, and the training may get stuck in local loss minima. This motivates the use of a **learning rate schedule** that controls the learning rate over multiple epochs. For example, a learning rate schedule can start with a high learning rate to speed up loss convergence and reduce the learning rate over time to further reduce losses.

Weight updates can also be controlled with an **optimizer** such as adam [KB17]. The adam optimizer tracks exponential moving averages of the gradient and the squared gradient for each individual weight, where two parameters (β_1 and β_2) control the moving averages. These parameters can be used to adjust gradients in a way that can reduce training time and improve overall training results. The adam optimizer can also be used in conjunction with a learning rate schedule. In this case, a schedule can ensure that weight updates continuously decrease in magnitude.

Loss functions. A variety of loss functions can be used to train models for purposes of classification and regression [Jad20; Wan+22]. In this work, the **absolute error (AE)**, the **squared error (SE)**, and the **Huber loss** [Hub64] are of particular interest.

These functions calculate the loss on the residuals $r_i = y_i - \widehat{y}_i$ for targets or labels $\mathsf{y} = (y_i)_{i=1,\dots,n}$ and predictions $\widehat{\mathsf{y}} = (\widehat{y}_i)_{i=1,\dots,n}$ as follows:

$$\mathsf{AE}: \quad r_i \quad \mapsto \quad |r_i|\,, \tag{2.17}$$

$$\mathsf{SE}: \quad r_i \quad \mapsto \quad r_i^2, \tag{2.18}$$

$$\mathsf{huber\_loss}_\delta: \quad r_i \quad \mapsto \quad \begin{cases} \frac{1}{2} r_i^2 & \text{if } |r_i| \leq \delta, \\ \delta\left(|r_i| - \frac{1}{2}\delta\right) & \text{otherwise.} \end{cases} \tag{2.19}$$

The parameter $\delta \in \mathbb{R}_{>0}$ of the Huber loss in Equation 2.19 can be chosen arbitrarily. The loss of multi-dimensional targets, labels, and predictions can be reduced to a single scalar value by calculating the mean or sum over the losses of all residuals.

The gradients of the absolute error in Equation 2.17 are constant for all residuals $r_i \neq 0$. This may cause gradients to overshoot when residuals are particularly small, whereas the gradients of the squared error in Equation 2.18 diminish continuously as the residuals approach zero. However, the squared error is more sensitive to residuals with large absolute values because the loss increases quadratically. For these residuals, the gradients increase linearly, which can cause training to be strongly influenced by outliers in the dataset. As a result, the training may fail to minimize losses across an entire dataset. The Huber loss addresses the drawbacks of the absolute and the squared error by ensuring that gradients decrease when the residuals approach zero and that they remain constant for residuals $|r_i| > \delta$.

Normalization. Normalization addresses several challenges that arise in model training. First, features can occur on different scales. Through this, a subset of the features that constitute the input of a neural network can have a strong impact on losses. This does not necessarily reflect their importance for a learning task. Scaling features to a comparable value range reduces this effect. In addition to neural network inputs, normalization can be applied to the output of layers within a neural network. By ensuring that output remains within a certain range, the technique reduces the risk of vanishing or exploding gradients that can prevent a model from learning. The following describes some commonly employed normalization techniques.

- **Z-score normalization**. The values x of a single feature are normalized by applying the following transformation:

$$x \mapsto \frac{x-\mu}{\sqrt{\nu}}. \tag{2.20}$$

  The variables μ and ν denote the (arithmetic) mean and variance of the feature's values in the training data. After applying z-score normalization, the features of the training data have mean $\mu = 0$ and standard deviation $\sigma = 1$. The values of μ and ν are determined before training a model. During inference, the sample distribution can deviate from the training data, so that a mean of zero and a standard deviation is no longer ensured.

- **Batch normalization**. Adjusting feature values with batch normalization [IS15] follows a similar principle as z-score normalization. However, batch normalization operates on the input batches of a neural network layer and tracks the mean μ and the variance ν during training across a batch. Batch normalization can be applied between consecutive layers in neural networks to normalize the outputs of layers throughout the neural network. This stabilizes the calculation of gradients and can achieve more efficient training.

- **Layer normalization**. Layer normalization [BKH16] operates similarly to batch normalization. However, it normalizes over each layer, instead of normalizing across a batch. Layer normalization can be applied throughout the architectures of a neural network in order to repeatedly adjust the input of neural network layers to remain within reasonable value ranges. This can improve overall model training.

2.5 Deep Reinforcement Learning

In RL, an **agent** learns to perform a task through interactions with an **environment**. Unlike training on a static dataset (as described in Section 2.4.2), the environment can change continuously, and an agent's decisions can impact the outcome of future interactions. Consequently, the distribution of training samples becomes non-stationary and the agent itself can influence the learning process.

Markov decision process (MDP). The concept of learning through interaction with an environment can be described with an MDP. Each MDP describes the environment with the following elements (see [SB18]):

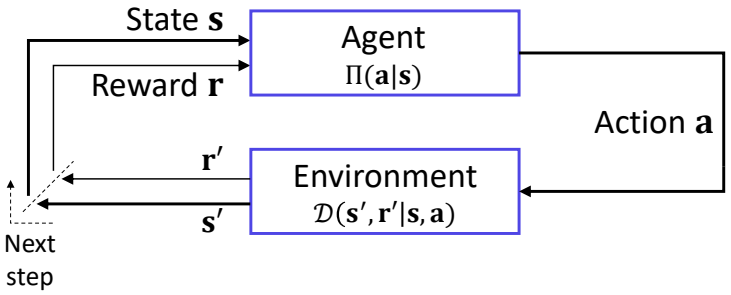


Figure 2.7: Interaction between an agent using a policy Π and an environment with dynamics function $\mathcal{D}$ during a single step of a MDP.

- A set of states $\mathbf{S}$ called the **state space**. The elements $\mathbf{s} \in \mathbf{S}$ describe the **state** of an environment that can be observed by the agent.
- A set of actions $\mathbf{A}$ called the **action space**, which comprises all **actions** $\mathbf{a} \in \mathbf{A}$ an agent can take to control an environment.
- A set of rewards $\mathbf{R} \subset \mathbb{R}$ comprising all **rewards** $\mathbf{r} \in \mathbf{R}$ an environment can generate. Rewards provide a numerical measure of how well a task was performed.
- A **dynamics function** $\mathcal{D} : \mathbf{S} \times \mathbf{R} \times \mathbf{S} \times \mathbf{A} \to [0, 1]$ denoting the probability $\mathcal{D}(\mathbf{s}', \mathbf{r}' \mid \mathbf{s}, \mathbf{a})$ that taking action $\mathbf{a}$ in state $\mathbf{s}$ results in state $\mathbf{s}'$ and reward $\mathbf{r}$.

The function $\mathcal{D}$ fully describes the dynamics of an environment in a stochastic manner. To describe state transitions and reward generation independently, two separate functions can be derived from the dynamics function:

- A **state transition probabilities function** $\mathcal{T} : \mathbf{S} \times \mathbf{S} \times \mathbf{A} \to [0, 1]$ that denotes the probability $\mathcal{T}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ of an environment transitioning to a state $\mathbf{s}'$ from state $\mathbf{s}$ after taking an action $\mathbf{a}$.
- A **reward function** $\mathcal{R} : \mathbf{S} \times \mathbf{A} \to \mathbf{R}$ that calculates the reward $\mathbf{r}' = \mathcal{R}(\mathbf{s}, \mathbf{a})$ for taking an action $\mathbf{a}$ when the environment is in state $\mathbf{s}$.

In addition to modelling the environment as outlined above, an agent's decision making can also be expressed in a stochastic manner:

- A **policy** $\Pi : \mathbf{S} \times \mathbf{A} \to [0, 1]$ denotes the probability $\Pi(\mathbf{s} \mid \mathbf{a})$ of an agent choosing an action $\mathbf{a}$ when the environment is in a state $\mathbf{s}$.

Figure 2.7 illustrates a single interaction (referred to as a **step**) between an agent and an environment, where the agent follows a policy Π and chooses an action $\mathbf{a}$. The action is applied by the environment, which updates its state $\mathbf{s}$ to a new state $\mathbf{s}'$ and generates a new reward $\mathbf{r}'$ for the agent. In general, an MDP progresses through a sequence of multiple steps. In each step at a time index t, an agent observes an environment state $\mathbf{s}_t$ and chooses an action $\mathbf{a}_t$ according to its policy. The agent then receives a reward $\mathbf{r}_{t+1} = \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t)$ and observes a new state $\mathbf{s}_{t+1}$ at the following time index $t+1$. To model repetitive tasks, such as playing multiple rounds of a game, the sequence of steps can be subdivided into **episodes**. In this case, each episode begins by resetting the environment to a (possibly random) initial state.

When an agent learns from interacting with an environment, its policy Π can change from step to step. Considering an MDP at time index t, the goal of training is to optimize an agent's policy so that it maximizes the expected **discounted return**

$$\mathbb{E}\left[\sum_{i=0}^{n} \gamma^i \mathbf{r}_{t+i}\right]. \tag{2.21}$$

The **discount rate** $\gamma \in (0,1)$ in Equation 2.21 determines to what extent future rewards are considered when optimizing the policy. Restricting the reward discount factor to $\gamma < 1$ ensures that the discounted return converges for infinite time horizons with $n = \infty$. Future rewards can also be disregarded by choosing $\gamma = 0$, for example, when there is no dependency between consecutive steps. In this case, training the agent serves to maximize the immediate reward $\mathbf{r}_t$.

Deep Q-Networks (DQNs). DQNs [Mni+13] address the optimization objective in Equation 2.21 by combining Q-Learning [WD92] with deep neural networks. In Q-Learning the function $\mathcal{Q}^*(\mathbf{s}, \mathbf{a}) \stackrel{\text{def}}{=} \max_\Pi \mathbb{E}\left[\sum_{i=0}^{n} \gamma^i \mathbf{r}_{t+i} \mid \mathbf{s} = \mathbf{s}_t, \mathbf{a} = \mathbf{a}_t, \Pi\right]$ determines the maximum expected discounted return that can be achieved by following any policy Π after taking an action $\mathbf{a}$ in a state $\mathbf{s}$. Knowing $\mathcal{Q}^*$ allows it to realize an optimal policy by choosing the action $\mathbf{a}^* = \arg\max_{\mathbf{a}} \mathcal{Q}^*(\mathbf{s}, \mathbf{a})$ that maximizes the value of $\mathcal{Q}^*$ in each state $\mathbf{s}$. However, calculating $\mathcal{Q}^*$ is often infeasible. Instead, a deep neural network (called **value network**) can be used to approximate $\mathcal{Q}^*$. A value network outputs an approximated value of $\mathcal{Q}^*$ for each action $\mathbf{a} \in \mathbf{A}$ in a finite action space $\mathbf{A}$ when receiving a state $\mathbf{s}$ as input.

To train a DQN for this purpose, the value network is updated after each interaction with an environment using a technique known as experience replay [Lin92]. Specifi-

cally, experience replay collects a sequence of **transitions** $\tau = (\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_{t+1}, \mathbf{s}_{t+1})$ that comprise a state $\mathbf{s}_t$ at time index t, the chosen action $\mathbf{a}_t$, the resulting reward $\mathbf{r}_{t+1}$, and the next state $\mathbf{s}_{t+1}$. These transitions are stored in a fixed-size **replay buffer**, with new entries replacing the oldest entry when the buffer is full. After interacting with the environment a **mini batch** (i.e., a small subset of the stored transitions) is sampled at random from the replay buffer. This mini batch constitutes the dataset used for training the value network (as described in Section 2.4.2).

Keeping transitions in the replay buffer increases the efficiency of sample collection as each transition can be used multiple times to update a value network. In addition, sampling mini batches at random from a replay buffer breaks correlations between consecutive transitions that can cause unwanted feedback loops. If an agent learns only from transitions with high reward that were derived from its most recent policy, it may become biased towards its recently chosen actions. The agent may then fail to learn that different actions must be chosen in subsequent steps to maintain high reward. This can cause agent training to fail entirely.

Despite random sampling of mini batches, feedback loops can still occur when transitions in a replay buffer are derived solely from an agents policy. In this case, an agent learns only from its previously chosen actions and may become increasingly biased toward a subset of the action space as training progresses. In DQN training, this is addressed by realizing a trade-off between **exploration** and **exploitation**. In exploitation, an agent uses its learned policy to choose actions that are expected to generate high reward, whereas exploration involves having an agent trying out different actions. DQN training uses an ϵ-greedy strategy, where an agent chooses an action according to its policy with probability $1-\epsilon$ and samples a random action from the action space with probability ϵ. The parameter ϵ is referred to as the **exploration rate** and can be adapted with an **exploration rate schedule** as the training progresses over multiple steps. For example, a schedule can reduce the exploration rate over time, allowing an agent to first explore the action space and then increasingly focus on actions known to generate high rewards.

An additional challenge in training DQNs is that updating the policy while simultaneously using it to choose actions can lead to oscillation or divergence of the policy. This can be addressed by using a separate **target network** in addition to the **value network** [Mni+15]. The value network is trained to approximate the function $\mathcal{Q}^*$ and is updated in every step as outlined above. The target network is used to realize the agent's policy by choosing actions in each step. However, updates of the target network occur less frequently than updates of the value network (e.g., only after

every fixed number of steps) and copy weight information from the value network to the target network. The less frequent updates effectively keep the policy stable while the value network continues to learn. By decoupling the policy evaluation performed by the target network from the training of the value network, the risk of fluctuations or deviations can be reduced.

Chapter 3

Selective Aggregate Filtering

This chapter introduces the concepts of Selective Aggregate Filtering — a machine learning-driven mitigation system that is designed to counteract volumetric DDoS attacks. Its purpose is to protect network infrastructures from being overloaded with attack traffic. The basic approach is to conduct early traffic filtering, i.e., to enable differentiated packet processing in upstream networks in order to selectively remove attack traffic before it can build up to a critical volume near an attack target. To this end, Selective Aggregate Filtering generates flow rules that can leverage the fast traffic processing capabilities of data plane systems (such as SDN switches) to establish control over ingress traffic. A key idea behind Selective Aggregate Filtering is to combine HHH algorithms that perform efficient monitoring of high-volume traffic with RL. This serves to maintain effective trade-offs between attack traffic removal, preservation of legitimate traffic, and the memory required to perform traffic filtering in upstream systems.

To outline the scope under which the mitigation system is designed to operate, Section 3.1 presents an attacker model that describes the objectives, capabilities, and restrictions of attackers under consideration. The following section specifies the design goals of Selective Aggregate Filtering and provides quantitative metrics for determining the effectiveness of traffic filtering. The core concepts of Selective Aggregate Filtering are introduced in Section 3.3, which provides an overview of the mitigation system's architecture and the workflow between its components. Section 3.4 summarizes the main aspects of the proposed mitigation system.

3.1 Attacker Model

The following attacker model builds the necessary foundation for later considerations regarding design decisions and evaluations of developed mechanisms. In general,

attackers execute volumetric DDoS attacks directed at specific targets. They either use systems under their direct control or leverage reflectors. The attackers under consideration are assumed to be resource-constrained in terms of available time, computational power, and network resources.

The following assumptions of the attacker model describe the objectives and capabilities of attackers in greater detail. In addition, several restrictions are formulated that are necessary to allow the protection of an attack target through traffic filtering.

Objectives. The objectives determine what an attacker generally seeks to achieve (regardless of attack techniques and available systems).

(A_1) **Reducing the availability of the attack target**. The primary goal of an attacker executing a volumetric DDoS attack is to significantly reduce the availability of the attack target.

(A_2) **Resource efficiency**. Since attackers have limited resources, they seek to maximize their gain, i.e., the expected impact of a volumetric DDoS attack relative to the required resource investment.

Capabilities. The capabilities of an attacker determine how and to what extent a volumetric DDoS attack can be conducted.

(A_3) **Access to distributed systems**. An attacker can use a potentially large number of distributed systems to generate attack traffic. These can be the attacker's own systems, compromised third-party systems directly under the attacker's control, or third-party reflectors. It can be assumed that an attacker acquires knowledge of potential reflectors and compromises systems before an attack occurs. All systems participating in an attack are assumed to be capable of sending traffic to an attack target.

Because attackers are assumed to have limited resources, the total number of systems that can contribute to an attack is potentially large but still limited (and significantly smaller than the total number of systems on the Internet). This is due to the necessary computational and administrative effort involved in finding reflectors, compromising remote systems, and maintaining the attacker's own systems.

(A_4) **Access to multiple attack vectors**. An attacker can utilize multiple different attack vectors during an ongoing attack, and several attack vectors can be combined at the same time to diversify attack traffic characteristics. The

available attack vectors depend on the systems to which an attacker has access. In particular, the choice of reflectors determines which amplification attack vectors can be used, since each reflector typically supports only a single attack vector.

(A_5) **Time-dynamic behavior**. An attacker can control the activity of attacking systems over time. In particular, an attacker can change attack traffic characteristics in a number of ways during an ongoing attack.

a) An attacker can change the selection of systems that actively participate in an attack at any given time. This allows an attacker to add or remove attack traffic sources and enables multiple attacking systems to contribute to an attack simultaneously.

b) An attacker can change attack vectors at any time. The attack vectors that can be used depend on the systems that actively contribute to an attack at a certain point in time.

c) An attacker can control the amount of attack traffic generated by each attacking system.

Despite the ability to control attacking systems over time, attackers are not assumed to have the ability to learn and reliably predict a mitigation system's behavior in order to circumvent its response to an attack.

Restrictions. The following restrictions outline prerequisites for protecting an attack target through upstream traffic filtering.

(A_6) **Location of attack traffic sources**. The location of attack traffic sources in the network topology is restricted in such a way that traffic filtering can process the majority of the attack traffic. The attack traffic that can circumvent traffic filtering entirely must remain unable to reduce the availability of the attack target's services.

(A_7) **Mitigation system integrity**. An attacker cannot gain direct control over a mitigation system or interfere with its operation. In particular, this assumption ensures that an attacker cannot manipulate a mitigation system's communication, its training data, or its control flow.

(A_8) **Attack target integrity**. An attacker cannot gain direct control over the attack target. Specifically, an attacker cannot directly terminate system

operation of an attack target, and a target retains the ability to inform a mitigation system that it is under attack.

3.2 Design Goals

The primary objective of a DDoS mitigation system is to preserve the availability of attack targets. To defend against volumetric DDoS attacks, Selective Aggregate Filtering translates this objective to removing attack traffic before it can overload a network infrastructure. The basic approach is to generate flow rules for upstream networks that can be directly applied in the data plane to selectively discard packets. The main goals of this approach can be summarized as follows:

- Ensure efficient traffic filtering.
- Remove the attack traffic while preserving the legitimate traffic.

Section 3.2.1 and 3.2.2 elaborate these goals in more detail and provide metrics that can be used to quantify the extent to which individual goals are achieved.

Additionally, Selective Aggregate Filtering is designed for use in network environments that exhibit dynamic traffic patterns (in both attack and legitimate traffic). Therefore, Section 3.2.3 and 3.2.4 describe two complementary design goals:

- Adaptation of traffic filtering to changing traffic.
- Maintaining effective trade-offs between different goals over time.

3.2.1 Efficient Traffic Filtering

In general, it is desirable to avoid delaying packets during traffic filtering to keep service response times low. More importantly, a mitigation system should not become a bottleneck during an attack. If traffic filtering requires more processing time than forwarding, it incurs the risk that volumetric attack traffic overloads the network infrastructure at the filter. This can lead to a situation where the mitigation system itself increases the effectiveness of an attack.

To avoid incurring higher processing delays than packet forwarding, an important goal of Selective Aggregate Filtering is to enable efficient traffic filtering directly in the data plane. More precisely, the goal is to utilize flow rules that leverage the packet

processing capabilities of upstream switches to discard packets selectively. Relying on flow rules incurs no additional processing delay compared to forwarding. Flow rules specifically used for the purpose to discard packets are referred to as filter rules throughout this thesis.

Definition 3.1: Filter Rule

Given a set of matches M, a **filter rule** $\mathcal{F}$ is a three-tuple consisting of priority `p`, a match `m`, and an action `a`:

$$\mathcal{F} \stackrel{\text{def}}{=} (\texttt{p},\ \texttt{m},\ \texttt{a}) \in \mathbb{N} \times M^* \times \{\texttt{DROP}\} \tag{3.1}$$

Multiple filter rules are considered to be **conflict-free** if they have different priorities or if they have pairwise disjoint sets of matched packets. Prefixes used in filter rules are referred to as **filter rule prefixes**.

Instead of supporting arbitrary actions, filter rules use only `DROP`-actions to immediately discard matching packets. The use of specific matches enables differentiated packet processing, e.g., to remove traffic that originates from a single IP subnet while leaving other traffic unaffected. This can be accomplished by matching IP source addresses against a specific prefix. In general, matches can be selected from the range of matching criteria. When support for a common set of matching criteria is ensured, filter rules can be applied by any switch of an upstream network to enable early traffic filtering directly in the data plane.

In many situations, a single filter rule is insufficient for traffic filtering. Instead, several filter rules are often required to match attack traffic packets with different characteristics. For example, several IP prefixes are necessary in order to remove traffic originating from multiple non-contiguous subnets. In the context of this thesis, a combination of multiple filter rules is referred to as blacklist.

Definition 3.2: Blacklist

A **blacklist** $\mathcal{B}$ is a finite, conflict-free set of filter rules.

To determine if a packet has to be discarded or not, it must be matched against *all* filter rules in a blacklist. If at least one filter rule matches, the packet should be discarded. However, evaluation of multiple filter rules is challenging, since the number of matches that must be evaluated increases linearly with the number of filter rules. A strictly sequential evaluation of a large number of filter rules would incur a prohibitive computational effort that can easily lead to a high processing

delay. Instead, filter rules can be stored in TCAM. This enables evaluation of filter rules in parallel. Traffic filtering can then be performed without incurring additional processing delay. However, TCAM capacity is limited due to high monetary costs and high power consumption.

In general, the number of filter rules in a blacklist should be kept low address memory constraints of network devices while enabling efficient traffic processing. To evaluate the extent to which the goal of efficient traffic filtering is achieved, the size of a blacklist is used as a performance metric.

Definition 3.3: Blacklist Size

The **blacklist size (**BSIZE**)** denotes the cardinality $|\mathcal{B}|$ of a blacklist $\mathcal{B}$:

$$\text{BSIZE} \stackrel{\text{def}}{=} |\mathcal{B}| \tag{3.2}$$

3.2.2 Filter Rule Effectiveness

The notion of filter rule effectiveness summarizes to what extent the two goals of attack traffic removal and preservation of legitimate traffic are achieved. Ideally, a blacklist removes the entire attack traffic, while leaving the legitimate traffic unaffected. In complex network environments, it is often not possible to achieve both goals simultaneously because the characteristics of legitimate and attack traffic are not fully known, or they change over time. This makes it difficult to select matches for filter rules. To quantitatively assess the effectiveness of filter rules, statistical measures are used to quantify the extent to which the goals of removing attack traffic and preserving legitimate traffic are achieved.

To assess filter rule effectiveness through statistical measures, the bytes that are processed during traffic filtering are viewed as a set of samples. Each sample is either *positive*, indicating that a byte is part of the attack traffic, or *negative* if a byte is part of the legitimate traffic. The correct distinction between positive and negative samples constitutes the ground truth. The application of filter rules classifies packets, and therefore bytes, in absence of knowledge of the ground truth. This results in the four possible classification results summarized in Table 3.1, depending on whether the classification is correct or not. Samples are either correctly classified as **true positives** and **true negatives**, or they are wrongly classified as **false positives** and **false negatives**. With this distinction, the design goals of attack traffic removal and preservation of legitimate traffic can be formalized.

	Classified as positive	Classified as negative
Positive	True positive (TP)	False negative (FN)
Negative	False positive (FP)	True negative (TN)

Table 3.1: Possible classification results of filter rule application.

Preservation of legitimate traffic. Traffic filtering should avoid the removal of legitimate traffic. If a mitigation system discards packets of legitimate traffic, the system itself reduces the availability of an attack target. Perfect preservation of legitimate traffic is often not achievable due to the difficulty of distinguishing between attack and legitimate traffic. Given the classification results from Table 3.1, the extent to which this goal is fulfilled can be quantified by the false positive rate.

Definition 3.4: False Positive Rate

Let x_{TN} denote the number of true negatives and x_{FP} denote the number of false positives resulting from the application of filter rules. Then, the **false positive rate (FPR)** of traffic filtering is given as:

$$\mathsf{FPR} \stackrel{\text{def}}{=} \frac{x_{\mathsf{FP}}}{x_{\mathsf{FP}} + x_{\mathsf{TN}}} \tag{3.3}$$

Maintaining low false positive rates is an important goal of traffic filtering to preserve as much legitimate traffic as possible.

Attack traffic removal. Filter rules serve to render volumetric DDoS attacks ineffective by discarding attack traffic before it can overload a network infrastructure. It is not necessary to remove the entire attack traffic as long as the availability of an attack target can be sustained. Still, the more attack traffic is removed, the more network infrastructure resources are preserved. Consequently, one goal of attack traffic removal is to maintain a low false negative rate:

Definition 3.5: False Negative Rate

Let x_{FN} denote the number of false negatives and x_{TP} denote the number of true positives resulting from the application of filter rules. Then, the **false negative rate (FNR)** achieved during traffic filtering is given by:

$$\mathsf{FNR} \stackrel{\text{def}}{=} \frac{x_{\mathsf{FN}}}{x_{\mathsf{FN}} + x_{\mathsf{TP}}} \tag{3.4}$$

3.2.3 Adaptation to Changing Traffic

In a network environment, traffic characteristics can change over time, and traffic sources can appear or vanish. These changes can occur in the attack traffic, but also in the legitimate traffic. Common examples for traffic changes include the following:

- A volumetric DDoS attack starts or ends. This causes new attack traffic sources to join or leave an ongoing attack, which increases or decreases the number of attack traffic sources within a subnet and potentially the attack traffic volume.
- Attackers transition between attack vectors to circumvent established defenses. This changes traffic characteristics and can also change the number of attack traffic sources across multiple subnets. For example, transitioning from one amplification attack vector to another may require different reflectors that have other locations in the IP address space.
- Legitimate traffic sources appear and disappear. This shifts the distribution of legitimate traffic across the IP address space and alters traffic characteristics. Services that many systems access for only a short period of time can experience a particularly high fluctuation of legitimate traffic sources.
- The behavior of individual legitimate systems can change over time. This may be due to changing user behavior or because applications and operating systems react to certain events, such as a TCP connection experiencing congestion. Hence, characteristics such as traffic volume may vary.

To adapt to changing traffic, filter rules should be updated (either regularly or on demand) to match evolving traffic patterns in order to maintain filter rule effectiveness. Otherwise, one of two adverse effects can occur:

- Outdated filter rules that affect a subnet where the attack traffic decreases cause the inadvertent removal of legitimate traffic. This leads to an unnecessarily high false positive rate.
- An outdated blacklist fails to remove new attack traffic from previously unaffected address space regions. As a result, the false negative rate increases.

Failing to adapt to changing attack traffic incurs the risk that attackers with time-dynamic behavior (assumption (A_5) of the attacker model) bypass traffic filtering.

Since failure to perform timely updates of filter rules directly correlates with false positive and false negative rates, no additional metric is required to explicitly quantify the ability to adapt to changing traffic.

3.2.4 Maintaining Effective Trade-Offs

In most situations, it is not possible to achieve all three design goals of a low false positive rate, a low false negative rate, and a low blacklist size simultaneously. In this case, certain trade-offs must be made to balance the different goals. The following example illustrates this in a simple scenario:

Example 3.2.1. Consider $n \in \{1, \ldots, 255\}$ distinct, adjacent IP subnets `10.0.1.0/29` to `10.0.`n`.0/29` as illustrated in Figure 3.1. Each subnet $i \in \{1, \ldots, n\}$ contains two attacking systems (with IP addresses `10.0.`i`.0` and `10.0.`i`.2`) and six legitimate systems. Matching filter rules on source IP addresses to remove the attack traffic essentially offers three options for each subnet i:

(1) Traffic filtering can use two different filter rules that match on the source IP addresses `10.0.`i`.0` and `10.0.`i`.2`. This removes the attack traffic without impacting any legitimate traffic.

(2) Traffic filtering can use a single filter rule that matches on the source IP prefix `10.0.`i`.0/30`. This removes the attack traffic but affects the legitimate traffic from the systems with IP addresses `10.0.`i`.1` and `10.0.`i`.3`.

(3) Traffic filtering can use no filter rules. Obviously, this does not remove traffic.

Choosing option (3) results in a maximum false negative rate $\mathsf{FNR} = 1$, while option (2) results in a false positive rate $\mathsf{FPR} > 0$. Option (1) has minimal false positive and false negative rates $\mathsf{FPR} = \mathsf{FNR} = 0$ but requires twice as many filter rules as option (2).

The amount of legitimate traffic sent from the systems with IP addresses `10.0.`i`.1` and `10.0.`i`.3` can vary across the different subnets. When the legitimate traffic volume is low, it may be preferable to choose option (2) for some of the subnets to reduce the blacklist size. A similarly beneficial result can be achieved by generating no filter rules in subnets with a low combined attack traffic volume.

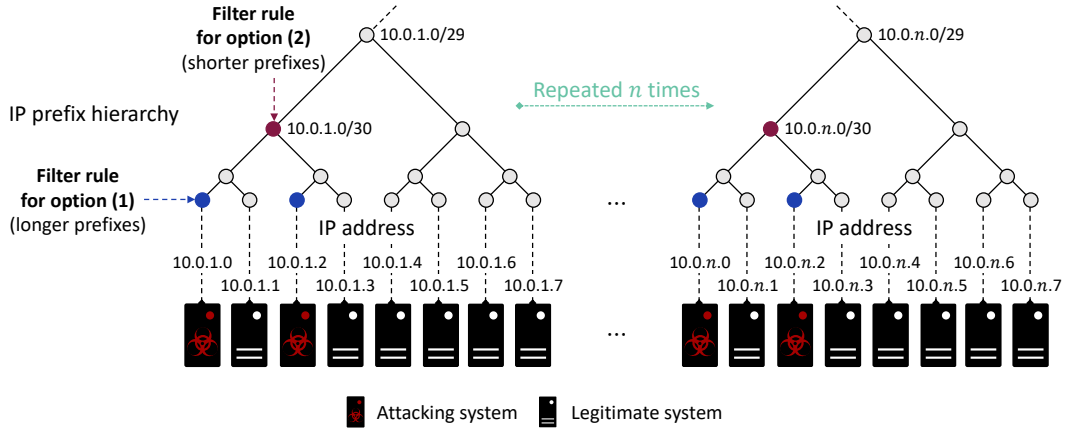


Figure 3.1: Example scenario with different choices for IP prefix-based filter rules. Different choices of filter rules lead to different trade-offs between the false positive rate, the false negative rate, the and blacklist size.

The previous example illustrates that different trade-offs can be realized by selecting filter rules. The example uses a simple scenario in which the distribution of traffic sources follows a regular pattern over a small region of the address space. In general, the distribution of attack and legitimate traffic sources can be highly diverse, making it challenging to select filter rules in order to balance different mitigation goals.

A specific design goal of Selective Aggregate Filtering is to balance low false positive rates, low false negative rates, and low blacklist sizes over time in order to maintain effective trade-offs. The realized trade-offs should be controllable to meet demands of the mitigation system and attack target operators. For example, it should be possible to specify an (approximate) increase in blacklist size that is tolerable when certain improvements in the false positive rate or the false negative rate can be achieved.

3.3 Architecture and Workflow

This section provides an overview of the architecture and workflow of Selective Aggregate Filtering. Throughout this section, all system components, their interactions, and the flow of data between the components are presented. In short, the components of Selective Aggregate Filtering perform two tasks:

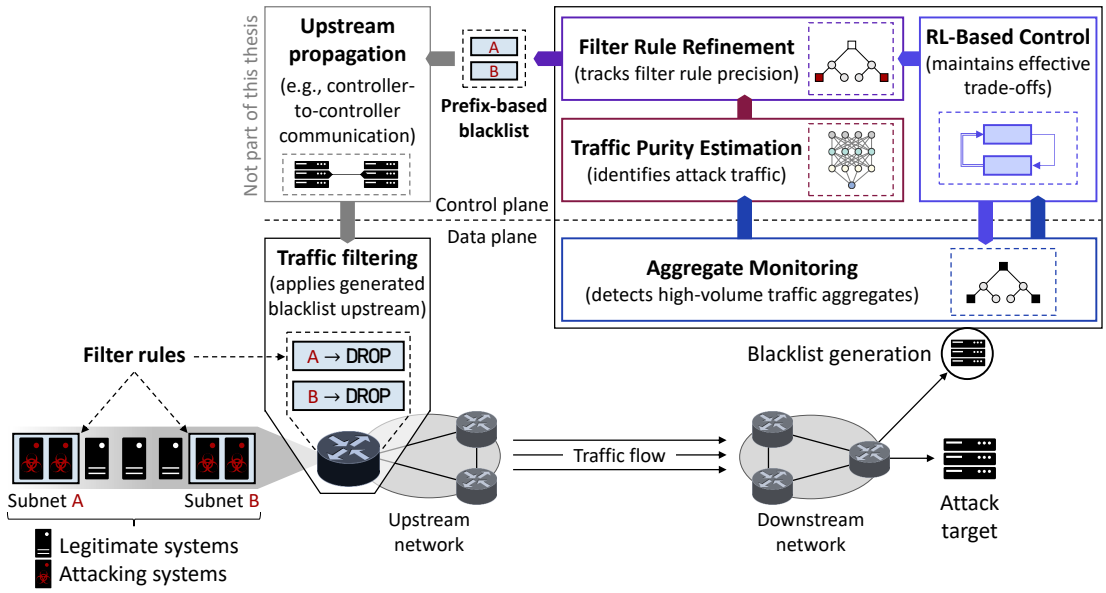


Figure 3.2: Architecture of the Selective Aggregate Filtering system.

(1) **Generating blacklists for upstream networks**. The blacklist generation process specifically addresses the design goals of efficient traffic filtering and filter rule effectiveness, as outlined in Section 3.2.2.

(2) **Adaptation to changing traffic**. The workflow of Selective Aggregate Filtering is designed to continuously monitor traffic in order to react to changes in attack and legitimate traffic. When traffic patterns change, system parameters are adjusted, and blacklists are updated to maintain effective trade-offs as described in Section 3.2.4.

3.3.1 System Overview

The operating principle of Selective Aggregate Filtering is to monitor ingress traffic with efficient HHH algorithms to compute HHH prefixes in order to locate potential attack traffic sources in the IP address space. This provides basic information to enable generating IP prefix-based filter rules that selectively discard traffic originating from specific IP address space regions. However, HHH algorithms cannot differentiate between high-volume attack and legitimate traffic. To reduce the risk of inadvertently removing legitimate traffic, the system performs additional steps to refine HHH prefixes into effective filter rules.

Figure 3.2 provides an overview of the system components involved in generating blacklists for a single attack target. The components interact by executing a continuous control loop to keep blacklists updated. In a nutshell, the main task of each component can be summarized as follows:

(1) **Traffic filtering** applies the blacklist generated by Selective Aggregate Filtering. In particular, traffic filtering is performed by an upstream switch that evaluates the matches of all flow rules to decide which packets must be discarded. Specifically, filtering is conducted based on IP prefixes to discard all packets with IP source addresses in a prefix's address range. The filtering switch is placed in an upstream network to enable early attack traffic removal in order to protect the network infrastructure from being overloaded.

(2) **Aggregate Monitoring** detects subnets sending high-volume traffic by computing HHH prefixes using an HHH algorithm (as described in Section 2.3.2). HHH prefixes indicate the potential location of attack traffic sources in the IP address space. However, this information is insufficient to generate effective filter rules. Therefore, the HHH algorithm also monitors the traffic composition of aggregates corresponding to HHH prefixes (e.g., the use of specific protocols commonly used by volumetric DDoS attack vectors). This information is later used to distinguish more accurately between legitimate and attack traffic.

(3) **Traffic Purity Estimation** serves to identify HHH prefixes that represent aggregates mainly composed of attack traffic. For this, Traffic Purity Estimation uses the information on traffic composition provided by Aggregate Monitoring. Its purpose is to reduce the risk of generating prefix-based filter rules with a strong impact on legitimate traffic.

(4) **Filter Rule Refinement** generates the blacklist for the traffic filter by selecting prefixes from the information provided by Traffic Purity Estimation. Its objective is to further reduce the risk of removing legitimate traffic. First, Filter Rule Refinement compensates for short-term traffic changes that can result in the application of ineffective filter rules. For this, it tracks the precision of filter rules over time. Second, it considers the hierarchical order of HHH prefixes to prioritize filter rules that use longer, more specific IP prefixes when shorter parent prefixes would remove too much legitimate traffic.

(5) **RL-based Control** serves to maintain effective mitigation trade-offs. When traffic changes over time, it adjusts system parameters of Aggregate Monitoring and Filter Rule Refinement to balance the blacklist size, the false positive rate,

and the false negative rate. For this, the component uses an RL agent that determines new system parameters based on the monitored traffic distribution and detected HHH prefixes. The necessary information is provided by the HHH algorithm used for Aggregate Monitoring.

The propagation of blacklists to upstream networks is outside the scope of this work. Still, several solutions exist to exchange blacklists between remote networks. For example, blacklists can be exchanged via controller-to-controller communication in SDN, BGP black-holing [Tur04; KM09; Kin+16], or DDoS Open Threat Signalling [MRM19].

Further details on each previously described system component are provided below. In addition, Traffic Purity Estimation, Filter Rule Refinement, and RL-based Control are discussed in detail throughout Chapter 4 to Chapter 6.

3.3.2 Traffic Filtering

Traffic filtering is conducted in upstream networks to prevent attack traffic from overloading the network infrastructure. To selectively discard packets, a switch matches every incoming packet against all filter rules in a previously generated blacklist. If a packet matches any rule, it is discarded. Otherwise, the packets are processed according to the remaining flow rules programmed into the switch. Evaluation of filter rules in switches does not incur additional processing delay in comparison to standard operations like packet forwarding (as discussed in Section 3.2.1).

Selective Aggregate Filtering is designed to build filter rules from detected HHH prefixes (including additional refinement steps as previously outlined). Therefore, traffic filtering focuses on removing traffic, which originates from IP subnets that mainly send attack traffic. To perform this task, every filter rule applies a pair of matches $\mathtt{m} = (\mathtt{m}_{\mathsf{DST}}, \mathtt{m}_{\mathsf{SRC}})$ to ensure the following:

- The match $\mathtt{m}_{\mathsf{DST}}$ ensures that filter rules do not inadvertently affect traffic destined to other systems by matching the destination addresses of IP packets against the address of an attack target. Under the assumptions (A_7) and (A_8) of the attacker model, an attack target retains the ability to inform a mitigation system of its own IP address. Hence, the value of the match $\mathtt{m}_{\mathsf{DST}}$ can be assumed to be known when generating blacklists and during filter rule application.

- The match $\mathtt{m}_{\mathtt{SRC}}$ specifies an arbitrary IP prefix. If the source address of an IP packet matches the prefix, the packet is discarded (through the `DROP` action of a filter rule as outlined in Section 3.2.1). Multiple filter rules effectively partition the IP address space into regions that are allowed to send traffic to the attack target and blacklisted regions that are prevented from doing so.

When considering a mitigation for a single attack target, the match $\mathtt{m}_{\mathtt{DST}}$ can be omitted. This is the default case throughout this thesis. Therefore, generating filter rules for traffic filtering focuses mainly on determining values for the $\mathtt{m}_{\mathtt{SRC}}$ matches.

Storing filter rules in a switch. When programming a blacklist into a switch, the filter rules reside in a flow table (e.g., they are stored in TCAM to enable efficient prefix-based matching). Ensuring that the filter rules comprising a blacklist are conflict-free can be achieved in a straightforward manner. The only conflict that can arise between filter rules of a blacklist is that the matches $\mathtt{m}_{\mathtt{SRC}}$ of two rules overlap on a common IP address range. This conflict can be resolved by choosing different priorities for flow rules. In particular, the priority of filter rules can be inferred from the length of the prefix used in the match $\mathtt{m}_{\mathtt{SRC}}$. Using higher priorities for longer prefixes resolves the conflict, since it effectively implements longest prefix matching.

When a switch that performs traffic filtering also stores flow rules of other network applications (like load balancers or access control systems), avoiding conflicts may require additional precautions. Such conflicts can be resolved in one of two ways:

(1) A straightforward solution is to use a dedicated flow table for filter rules when multiple flow tables are available. Any flow rule that takes precedence over a filter rule can be programmed into flow tables with higher priorities. For example, this can be used to define exceptions for critical flows that should not be processed by filter rules. Flow rules that should be evaluated after the application of filter rules reside in flow tables with lower priority.

(2) Automated conflict resolution methods (like [Kan+13; Jin+15; Zho+22b]) can be used to store filter rules in the same flow table as flow rules of other network applications. Such approaches introduce additional flow rules that process packets, which could be matched by two or more flow rules with identical priorities. In this case, the priorities must be arranged in such a way that the priority of network applications is maintained (e.g., to create exceptions to traffic filtering for critical flows). However, automated conflict resolution introduces additional computational effort and can also lead to an overall

higher number of flow rules. This provides an additional incentive for smaller blacklist sizes when resorting to automated conflict resolution methods.

The remainder of this thesis assumes that the first option is used. Nevertheless, automated conflict resolution usually operates in a transparent manner and could be used as an alternative.

Maintaining insight into traffic characteristics. Since traffic filtering is performed upstream, removing the entire traffic affected by a filter rule would prevent Aggregate Monitoring from monitoring the corresponding traffic characteristics. However, this information is necessary to monitor attack traffic characteristics and the impact of filter rules on legitimate traffic in order to maintain effective trade-offs. Therefore, traffic filtering samples packets probabilistically to exclude a fraction of the traffic from blacklist application. This fraction is determined by a **sampling rate** s. Throughout this thesis, a sampling rate $s = 0.1$ is assumed to retain insight into the traffic distribution over the IP address space and traffic characteristics.

3.3.3 Aggregate Monitoring

The task of Aggregate Monitoring is to continuously monitor traffic directed to an attack target to identify IP address space regions that send a combined high traffic volume. This serves to locate potential attack traffic sources in the IP address space to provide basic information for subsequent blacklist generation.

As described in Section 2.3, approximate HHH algorithms are well suited for this task. They can detect HHH prefixes (that indicate the location of potential attacking systems) in a fast, resource efficient, and reliable manner. Selective Aggregate Filtering utilizes this capability to generate prefix-based filter rules. Specifically, the Aggregate Monitoring component of the mitigation system utilizes one HHH algorithm instance to monitor the traffic that is directed to an attack target. The HHH algorithm is placed downstream at a location near the attacked system in order to monitor converged characteristics of traffic originating from distributed sources. The monitoring information (HHH prefixes and traffic volumes) enables the detection of subnets sending high traffic volumes. In a volumetric DDoS attack, high-volume subnets of different sizes can indicate widely distributed attack traffic sources with combined high traffic volume or individually strong attack traffic sources.

Example 3.3.1. Figure 3.3 illustrates a simplified example in which calculating HHH prefixes detects seven distributed attack traffic sources that send a traffic volume

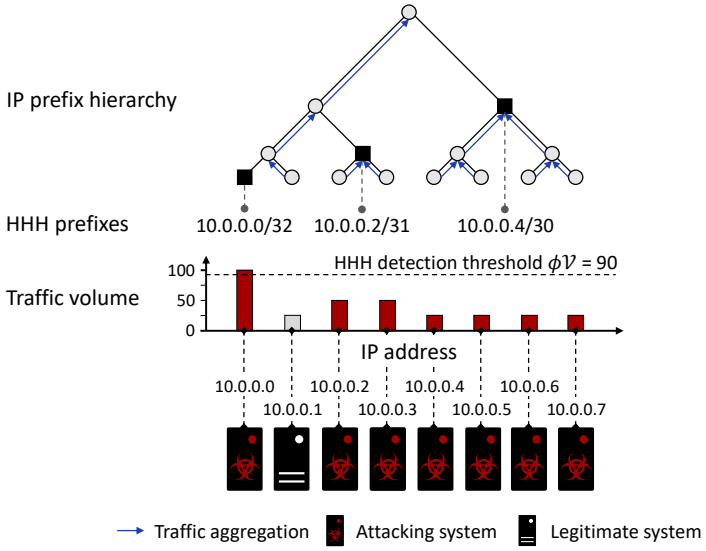


Figure 3.3: Detection of HHH prefixes that accurately identify attack traffic sources.

of 25, 50, or 100. The absolute HHH detection threshold $\phi\mathcal{V}$ is set to 90. Only the attacking system with IP address `10.0.0.0` sends sufficient traffic that directly results in the detection of an HHH prefix. The remaining attack traffic sources are still detected due to the aggregation of traffic volume during the calculation of HHH prefixes. In this simple example, the three detected HHH prefixes could be used directly to specify the match $\mathtt{m}_{\mathsf{SRC}}$ of filter rules. The resulting blacklist would remove the entire attack traffic without affecting the traffic of the single legitimate system.

Note that the HHH prefixes in the previous example yield the minimum number of filter rules required to discard the entire attack traffic without impacting the legitimate traffic. The HHH detection threshold ϕ can be adjusted to reduce or increase the traffic volume required for HHH detection (see Section 2.3.2). Since every filter rule is derived from an HHH prefix, the value of ϕ influences the size of the generated blacklist. Reducing ϕ results in longer, more specific HHH prefixes, while increasing ϕ generates shorter HHH prefixes that correspond to larger subnets. This can be used to reduce or increase the blacklist size, potentially at the cost of increased false positive rates or false negative rates. In Example 3.3.1, raising ϕ can combine the two HHH prefixes `10.0.0.0/32` and `10.0.0.2/31` into the prefix `10.0.0.0/30` at the cost of removing the legitimate traffic in that subnet.

While HHH algorithms can conduct resource efficient monitoring, it is not advisable to indiscriminately generate a filter rule from every detected HHH prefix. This can

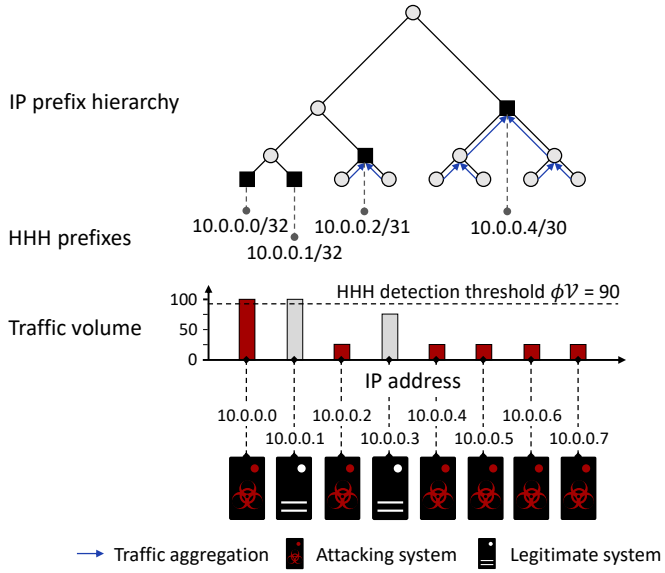


Figure 3.4: Detection of HHH prefixes that represent both attack and legitimate systems. Directly generating filter rules based on these HHH prefixes should be avoided, as removing the legitimate traffic results in a high false positive rate.

lead to several problems, which are discussed in detail in Chapter 4. Essentially, the aggregated traffic of an HHH prefix can contain significant *legitimate* traffic. Thus, filter rules derived from such prefixes can cause high false positive rates. This is illustrated in the following example.

Example 3.3.2. Figure 3.4 depicts a different scenario from the one used in the previous example. The traffic volume sent by the legitimate system with the IP address `10.0.0.1` now exceeds the absolute HHH detection threshold $\phi\mathcal{V}$. As a result, the HHH prefix `10.0.0.1/32` corresponds to purely legitimate traffic. A filter rule with this prefix would cause the removal of the high-volume legitimate traffic. In addition, the aggregated traffic of the HHH prefix `10.0.0.2/31` is now comprised of a mixture of mainly legitimate traffic along with some attack traffic. Both prefixes should be excluded from blacklist generation to avoid high false positive rates. Instead, only the prefixes `10.0.0.0/32` and `10.0.0.4/30` should be used to define filter rules.

To reduce the risk of removing legitimate traffic, a distinction between attack and legitimate traffic is required in addition to the purely volume-based detection of HHH prefixes. To enable this distinction, insight into the traffic composition of an aggregate is required. For example, if the attacking systems in Example 3.3.2 would perform a UDP flood attack while the legitimate systems send only TCP traffic, a clear

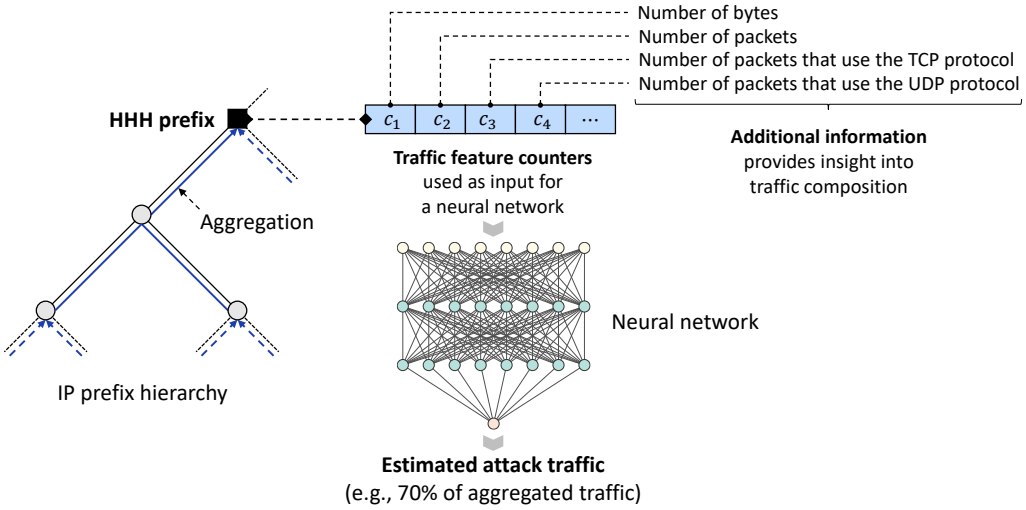


Figure 3.5: Operating principle of Traffic Purity Estimation. The attack traffic is estimated with a neural network from feature counters collected for each HHH prefix.

distinction between attack and legitimate traffic based on transport layer protocols would be possible. Chapter 4 describes how HHH algorithms are extended to count the occurrence of specific traffic features, e.g., the number of times a packet uses a certain transport layer protocol or how often its source port falls into a given range. Such feature counters can indicate the use of volumetric DDoS attack vectors. Their purpose is to provide the necessary input for subsequent steps in blacklist generation, specifically for Traffic Purity Estimation and Filter Rule Refinement.

3.3.4 Traffic Purity Estimation

Traffic Purity Estimation distinguishes between aggregates that consist primarily of attack traffic and aggregates that contain a significant legitimate traffic volume. Example 3.3.2 illustrated a scenario where an HHH prefix corresponds to purely legitimate traffic (the prefix `10.0.0.1/32`). Considering only prefixes whose traffic aggregates consist primarily of attack traffic (such as the prefixes `10.0.0.0/32` and `10.0.0.4/30`) reduces the risk of generating filter rules with strong impact on legitimate traffic. In contrast, selecting filter rule prefixes based on traffic volume alone can result in removing a significant amount of legitimate traffic.

To identify aggregates consisting primarily of attack traffic, Chapter 4 describes a data-driven approach to realize Traffic Purity Estimation. The approach uses the data

collected through Aggregate Monitoring, specifically, the feature counters of the HHH algorithm. These counters reflect aggregated characteristics of legitimate and attack traffic. Through supervised learning, a neural network can be trained to learn these characteristics in order to estimate the ratio of attack traffic within an aggregate.

Figure 3.5 illustrates the general operating principle of Traffic Purity Estimation. Initially, feature counters monitor the occurrence of specific traffic characteristics for all HHH prefixes during Aggregate Monitoring. These counters provide insight into the composition of traffic originating from a prefix's address range. The task of the neural network is to estimate the ratio of attack traffic for each HHH prefix from the corresponding feature counters. For example, a DNS amplification attack (as described in Example 2.1.1) would result in an increased count of UDP packets. The deviation from the UDP packet count of legitimate traffic (along with other features) can be used to estimate the ratio of attack traffic. To perform such estimations, the neural network is trained on previously monitored labeled traffic data. This training process conducts a regression analysis that reduces the estimation error. During the process, a neural network learns to differentiate between characteristics of legitimate traffic and attack vector characteristics. The choice of feature counters that lead to provide accurate estimations is further investigated in Chapter 4.

The information provided by Traffic Purity Estimation is later used to either accept a prefix (if its aggregate is primarily composed of attack traffic) or to reject a prefix (if its aggregate contains a significant legitimate traffic volume). The distinction between accepted and rejected prefixes is made by Filter Rule Refinement (see below), which imposes a lower bound on the required attack traffic ratio.

3.3.5 Filter Rule Refinement

Filter Rule Refinement uses the information provided by Traffic Purity Estimation to take the hierarchical relationship between prefixes into account and to track filter rule precision over time. In particular, this addresses two challenges that arise from generating filter rules from HHH prefixes:

(1) Short HHH prefixes that cover wide address space regions can result in filter rules that unnecessarily remove legitimate traffic. The risk of removing legitimate traffic can be reduced by considering the hierarchical order of prefixes as outlined below.

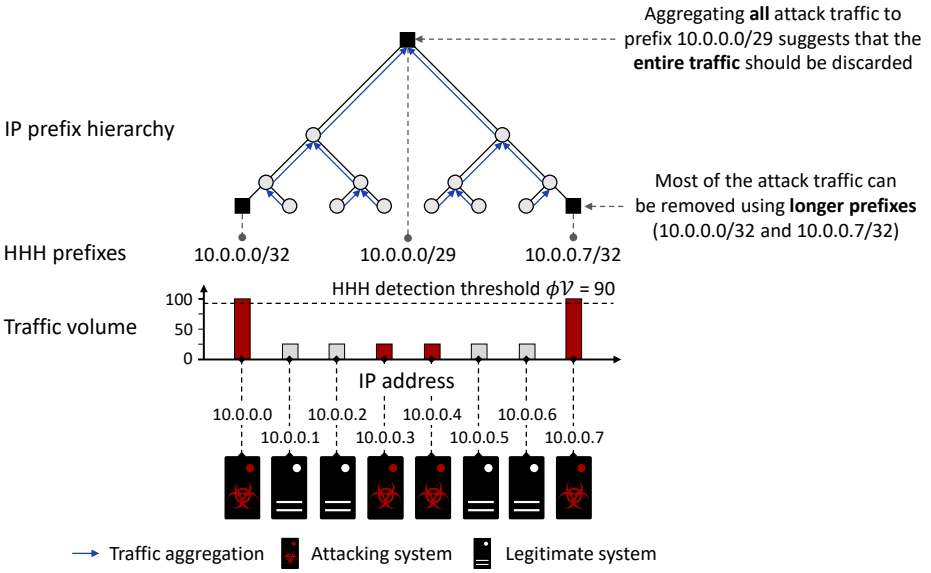


Figure 3.6: Attack traffic aggregation results in the aggregate of prefix `10.0.0.0/29` being primarily composed of attack traffic, suggesting to remove all traffic (including the legitimate traffic). This can be avoided by filtering longer prefixes only.

(2) Short-term traffic variations can lead to the application of ineffective filter rules. For example, momentarily high attack traffic volume can lead to a high estimated precision for a filter rule. When the attack traffic vanishes, the filter rule would exclusively affect legitimate traffic.

Both challenges are studied in greater detail throughout Chapter 5. The following outlines the key ideas to address these challenges.

Considering the hierarchical order of prefixes is essential to preserve legitimate traffic. In volumetric DDoS scenarios the attack traffic volume typically exceeds the legitimate traffic volume significantly. The more the attack traffic is aggregated into higher levels of the IP prefix hierarchy, the higher the likelihood that aggregates will have a high ratio of attack traffic. This would suggest to use shorter prefixes (with wide address space coverage) to generate filter rules. However, most of the attack traffic may already be removed by filter rules that use longer, more specific prefixes. Including the shorter prefixes in blacklists can thus lead to the unnecessary removal of legitimate traffic. The following example illustrates this effect.

Example 3.3.3. Figure 3.6 depicts a scenario where most of the attack traffic originates from the IP addresses `10.0.0.0` and `10.0.0.7`. The attack traffic volume sent by

the two attacking systems is sufficient to immediately result in HHH prefixes. Using the prefixes `10.0.0.0/32` and `10.0.0.7/32` to define filter rules removes most of the attack traffic (without affecting legitimate traffic). Only the attack traffic from the attacking systems with IP addresses `10.0.0.3` and `10.0.0.4` remains. However, aggregating all attack traffic into the aggregate of the HHH prefix `10.0.0.0/29` causes the aggregate to be mainly composed of attack traffic. Using this prefix to define a filter rule results in removing the entire legitimate traffic. This is unnecessary because the remaining attack traffic has low volume. Instead, the legitimate traffic can be preserved by using only the longer, more specific prefixes for blacklist generation.

To avoid generating filter rules from short prefixes that cause the unnecessary removal of legitimate traffic, the idea of Filter Rule Refinement is to reduce the traffic volumes of shorter prefixes by the attack traffic volume that is already removed by their child prefixes. Through this, only the *remaining* unfiltered traffic is considered when estimating the precision achieved by filter rules. As a consequence, filter rules with short prefixes can be rejected when their inclusion in a blacklist in addition to longer prefixes would primarily affect legitimate traffic.

Compensating for short-term traffic changes serves to avoid high false positive rates and high false negative rates that can result from variations in legitimate as well as attack traffic. If legitimate traffic from a subnet temporarily vanishes while the attack traffic from that subnet remains stable, the corresponding aggregate is entirely composed of attack traffic. This would suggest using the subnet's prefix to define a filter rule. When the legitimate traffic reappears, the filter rule would remove the traffic, resulting in a high false positive rate. Short-term increases in attack traffic can also lead to filter rules that mainly affect legitimate traffic. A temporarily high attack traffic volume within a subnet may result in an aggregate consisting mainly of attack traffic, suggesting to generate a filter rule with the corresponding prefix. If the attack traffic vanishes before the filter rule is applied, the result is an unnecessarily high false positive, since the filter rule affects legitimate traffic only.

The idea to address short-term traffic changes is to track the precision of filter rules over time to calculate a *smoothed* precision estimate from current and historical data. Filter rules are then rejected or accepted based on the smoothed precision estimate. By considering historical information, filter rules that had low precision in the past can be more easily rejected (to preserve legitimate traffic), while filter rules with previously high precision can be more readily accepted (to remove more attack traffic). In short, using smoothed precision estimates stabilizes the blacklist and reduces false positive and false negative rates.

After considering the hierarchical order of prefixes and calculating a smoothed precision estimate for all prefixes, Filter Rule Refinement generates the blacklist for upstream systems. For this, it enforces a filter rule precision threshold. Only prefixes whose smoothed precision reaches the threshold are accepted into the blacklist.

In addition to generating blacklists, Filter Rule Refinement manages the lifecycle of historical information. Specifically, the attack traffic ratios of detected HHH prefixes have to be stored over time to calculate smoothed precision estimates. When different HHH prefixes are detected over time, the stored data increases. To avoid a monotonic growth in the memory consumption, the data is pruned once a prefix was not classified as an HHH prefix over an extended period of time. To this end, Filter Rule Refinement tracks the lifetime of each detected HHH prefix.

3.3.6 RL-Based Control

RL-based Control adjusts runtime-configurable mitigation system parameters of the HHH algorithm (used by Aggregate Monitoring) and of Filter Rule Refinement to achieve the design goal of maintaining effective trade-offs between false positive rates, false negative rates, and blacklist sizes. While certain parameters may be effective in one situation, they can undermine one or more mitigation goals when traffic changes. For example, a highly distributed attack can result in an excessive amount of filter rules when a low HHH detection threshold was previously chosen. To adapt mitigation system parameters, an RL agent uses the monitoring information provided by Aggregate Monitoring to observe and detect changes in the traffic. The agent then determines a new HHH detection threshold (used during an HHH query) and a new filter rule precision threshold (which is applied by Filter Rule Refinement). The agent learns how to select parameters by interacting with a training environment.

A key challenge of the training process is to convey the importance of different mitigation goals in relation to each other to an RL agent. This is achieved by providing feedback in the form of a scalar reward to the agent. The value of the reward weighs the different quantitative goals of false positive rate, false negative rate, and blacklist size that are realized in each situation against each other. Thus, the agent can learn the relationship between traffic volume distribution, parameter choices, and the importance of different goals.

Chapter 6 presents the design of an agent that controls mitigation parameters using deep RL. In particular, the following aspects of agent training are addressed:

- **State representation**. The agent determines its parameter choices based on the information provided by Aggregate Monitoring. This requires representing the monitoring information in a way that the agent can process it. In particular, the distribution of traffic volume over the IP prefix hierarchy needs to be conveyed.
- **Action space modeling**. The agent chooses mitigation parameters by selecting an action from its action space that represents a specific combination of the HHH detection threshold ϕ and the filter rule precision threshold. Hence, the action space must be modeled in such a way that the selection of effective parameter combinations is possible.
- **Agent modeling and training process**. Deep RL uses neural networks that learn the relationship between an input state and the effect of choosing an action. The neural network architectures and hyperparameters of the training process must be chosen to enable an effective training process that guides an agent to maintain effective trade-offs between different mitigation goals.

3.3.7 Monitoring Intervals

Selective Aggregate Filtering generates new blacklists periodically to enable adaptation to changing traffic patterns. To enable this reaction, Selective Aggregate Filtering performs adaptations over a sequence of consecutive monitoring intervals that include the execution of all previously described components. More precisely, these monitoring intervals are defined as follows:

Definition 3.6: Monitoring Interval

Let the timestamps $t, t' \in \mathbb{R}_{>0}$ $(t' > t)$ denote the time at which two consecutive blacklists have been generated. Then, the **monitoring interval** $\mathcal{I}_t^{(\Delta t)} \subset \mathbb{R}$ is defined as the time span $\mathcal{I}_t^{(\Delta t)} \stackrel{\text{def}}{=} [t, t + \Delta t)$, where $\Delta t = t' - t$ is the **monitoring interval duration**.

The index Δt is omitted from $\mathcal{I}_t^{(\Delta t)}$ when Δt is constant across a sequence of multiple monitoring intervals or when it is clear from context.

A single monitoring interval constitutes one part of a continuously executed control loop that consists of multiple monitoring intervals with duration Δt. Updates of a blacklist can be performed at fixed times to make the duration independent of the execution time of individual components.

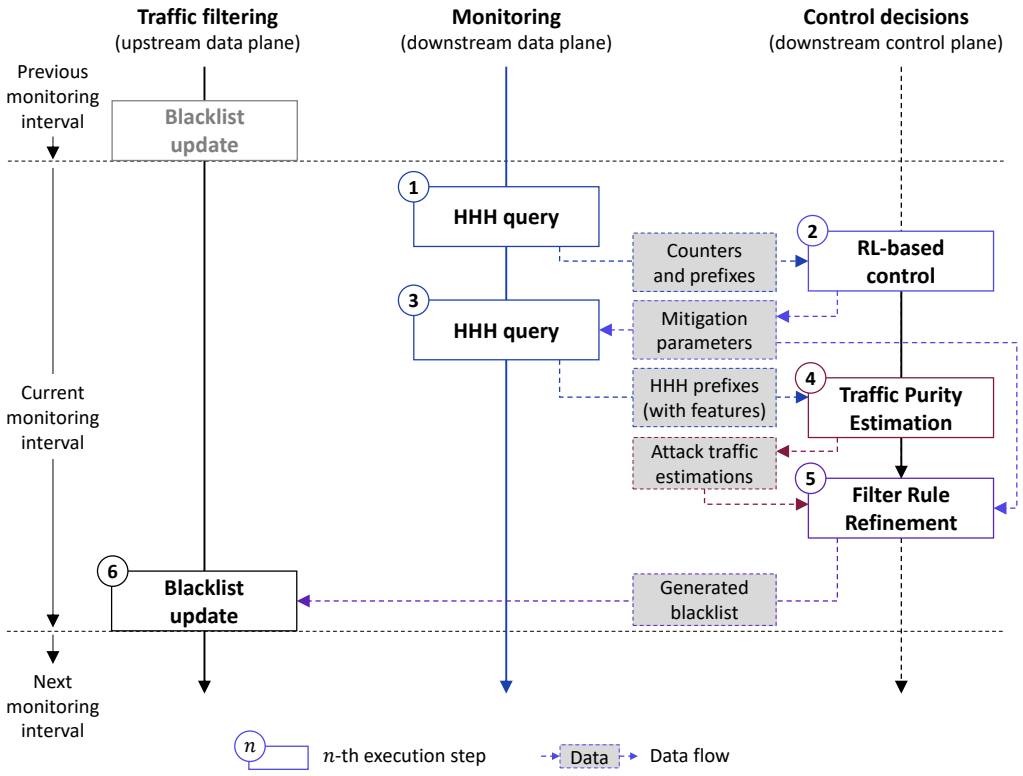


Figure 3.7: Control and data flow throughout a monitoring interval.

Figure 3.7 shows the control and data flow within a monitoring interval. A monitoring interval begins after a blacklist has been generated (in the previous interval) and executes the following steps in the indicated order:

① Initially, the HHH algorithm used for Aggregate Monitoring is queried to obtain monitoring information collected during the previous monitoring interval. At this time, the HHH detection threshold for the current monitoring interval has not yet been chosen. To provide insight into the traffic distribution that is necessary to perform RL-based Control, traffic volume information for all prefixes (not just HHH prefixes) that are currently monitored by the HHH algorithm is reported.

② The monitoring information obtained in step ① is used to determine new mitigation parameters for Aggregate Monitoring and Filter Rule Refinement. Specifically, RL-based Control determines the HHH detection threshold for the current monitoring interval and the threshold on required filter rule precision. The values of these parameters are determined based on the traffic volume

distribution monitored by the HHH algorithms, which constitutes the input for the RL agent. The parameter choices serve to maintain a balance between the three mitigation goals of low false positive rates, low false negative rates, and low blacklist size when traffic changes.

③ The HHH algorithm used for Aggregate Monitoring is queried again with the HHH detection threshold selected in step ②. This results in the computation of HHH prefixes in order to detect subnets with high traffic volumes. Furthermore, the HHH algorithm reports all feature counters corresponding to the HHH prefixes. Once the monitoring information is obtained, the HHH algorithm is reset to prevent outdated information from interfering with blacklist generation in subsequent monitoring intervals. After the reset, the HHH algorithm continues to monitor the traffic throughout the monitoring interval while further blacklist generation steps are executed.

④ Traffic Purity Estimation uses the monitoring information obtained in the previous step to identify aggregates that mainly contain attack traffic. For this, it performs inference on a trained neural network. As a result, Traffic Purity Estimation returns an estimate of the attack traffic ratio in each traffic aggregate of a detected HHH prefix.

⑤ Filter Rule Refinement selects filter rule prefixes based on the information provided by Traffic Purity Estimation and generates a new blacklist. It ensures the effectiveness of filter rules by enforcing a threshold on the minimum expected filtering precision of each filter rule. The threshold on required filter rule precision is provided by RL-based Control in step ②. To estimate filter rule precision, Filter Rule Refinement tracks historical filter rule precision over time. Once a blacklist has been generated, the historical information is updated based on the new data provided by Traffic Purity Estimation.

⑥ The newly generated blacklist (from step ⑤) is propagated to upstream systems. Programming the updated filter rules into switches is the responsibility of the upstream network's controller. After the new blacklist is programmed, traffic filtering continues with the updated filter rules. Incoming packets are matched against all filter rule prefixes until a new blacklist is programmed in the next monitoring interval.

Throughout this thesis, monitoring intervals are considered to have a common fixed duration Δt. Using a fixed duration serves to allow the HHH algorithm used for Aggregate Monitoring to acquire sufficient monitoring information. An alternative

would be to query and reset the HHH algorithm immediately once a new blacklist has been programmed in step ⑤. This can lead to short monitoring intervals of varying length with potentially adverse effects on the blacklist generation process:

- The accuracy of estimations of the attack traffic in aggregates provided by Traffic Purity Estimation may decrease. A reduction in information on aggregated traffic characteristics makes it more challenging for a machine learning model to provide accurate estimations. This is because traffic characteristics are less converged as each aggregate is comprised of fewer packets.
- Monitoring interval durations that depend on the time required to generate blacklists can negatively affect RL-based Control. The time required to detect HHH prefixes depends on the HHH detection threshold, as lower values result in more HHH prefixes being computed. The agent must then also learn how its own parameter choices influence the monitoring interval duration and thus monitoring information. This makes training of RL agents more challenging.
- Frequent updates of a blacklist can cause a significant computational effort for switches. For example, reordering of rules may become necessary to maintain the order of rule priorities when filter rules are stored in TCAM. In addition, small monitoring interval durations require frequent propagation of blacklists to upstream systems. This can result in a high communication overhead.

3.4 Summary

This chapter provided an overview of Selective Aggregate Filtering. The scope in which the system is intended to function was described by introducing an attacker model that includes objectives, capabilities, and restrictions of attackers. Furthermore, the design goals of Selective Aggregate Filtering were elaborated and a modular architecture for filter rule generation was presented. The specific task of each of the architecture's component was elaborated as well as the workflow between components. The chapter concluded by introducing monitoring intervals, which organize the control flow of Selective Aggregate Filtering into a continuously executed control loop that counteracts traffic changes.

Chapter 4

Traffic Purity Estimation

HHH algorithms provide an efficient method to detect IP subnets sending high-volume traffic. However, information about traffic volume alone is insufficient to generate effective filter rules. With successive aggregation or in the presence of traffic sources that send high-volume legitimate traffic on their own, HHH prefixes can represent aggregates that contain high volumes of legitimate traffic. Using such HHH prefixes indiscriminately to generate blacklists can lead to high false positive rates.

To make the efficient monitoring capabilities of HHH algorithms useful for traffic filtering, this chapter introduces Traffic Purity Estimation. This data-driven approach uses supervised learning to estimate the ratio of attack traffic in traffic aggregates. This is intended to reduce the risk of generating filter rules from HHH prefixes whose aggregates contain a significant legitimate traffic volume. For this, Traffic Purity Estimation extends HHH algorithms to monitor additional features that can indicate the presence of attack traffic in an aggregate. Through a regression analysis based on deep learning, the attack traffic ratio is estimated from the aggregated features.

Using additional features increases the memory requirements of HHH algorithms. To maintain a balance between memory requirements and estimation errors, Traffic Purity Estimation focuses on highly relevant features. For this, the importance of individual features is assessed by measuring their impact on the estimation.

Section 4.1 describes the challenges involved in Traffic Purity Estimation, while Section 4.2 elaborates its design goals in greater detail. Details on estimating attack traffic ratios through deep learning are provided in Section 4.3. Section 4.4 discusses balancing memory requirements of HHH algorithms against estimation errors by identifying important features. The feasibility of the approach is evaluated in Section 4.5. Finally, Section 4.6 compares Traffic Purity Estimation to related work.

4.1 Challenges

The goal of Traffic Purity Estimation is to distinguish between HHH prefixes that mainly represent attack traffic and those representing aggregates of legitimate or mixed traffic. Estimating the attack traffic ratio of an aggregate can be used to reject prefixes that would result in filter rules with strong impact on legitimate traffic.

4.1.1 Mixed and Legitimate Aggregates

A distinction between aggregates containing significant legitimate traffic and aggregates mainly composed of attack traffic is challenging with HHH algorithms. In particular, the aggregation of traffic during HHH computation frequently leads to mixed or legitimate aggregates, as described below.

Mixed aggregates. The attack traffic volume in a subnet may not be sufficient to reach the absolute HHH detection threshold $\phi\mathcal{V}$. Still, when combined with legitimate traffic from the same subnet, the total traffic volume can reach the threshold and result in an HHH prefix. Generating filter rules from HHH prefixes representing mixed aggregates with significant legitimate traffic can result in high false positive rates, since the legitimate traffic will be removed by traffic filtering.

Mixed aggregates often occur when using high HHH detection thresholds because more traffic needs to be aggregated to yield an HHH prefix. This increases the likelihood that an aggregate contains legitimate traffic. An attempt to avoid mixing attack and legitimate traffic during aggregation would be to use low HHH detection thresholds. In this case, the attack traffic in a subnet can directly yield an HHH prefix (without having to aggregate additional legitimate traffic). However, this is not a viable strategy, since low HHH detection thresholds increase the risk of generating HHH prefixes directly from legitimate traffic.

Legitimate aggregates. Aggregates of HHH prefixes can be composed entirely of legitimate traffic. This occurs when individual traffic sources send enough legitimate traffic to reach the HHH detection threshold, or when aggregation of purely legitimate traffic from multiple sources results in reaching that threshold. Filter rules using such HHH prefixes directly increase the false positive rate of traffic filtering.

To keep false positive rates low, only HHH prefixes representing aggregates primarily composed of attack traffic should be used to define filter rules. However, it is difficult to achieve a clear distinction between legitimate and attack traffic based on traffic

volume alone. Since the HHH detection threshold $\phi\mathcal{V}$ applies to all prefixes equally, strictly avoiding mixed and legitimate aggregates is no longer possible if one of the two following conditions holds for any subnet:

(C_1) Consider the combined traffic volume of all HHH prefixes within a subnet that aggregate attack traffic only. The absolute HHH detection threshold $\phi\mathcal{V}$ must be greater than the remaining combined legitimate and attack traffic volume within the subnet. Otherwise, at least one more HHH prefix occurs, whose aggregate is not entirely composed of attack traffic. Consequently, that aggregate must contain legitimate traffic and be either a mixed or legitimate aggregate.

(C_2) Consider the traffic of an HHH prefix without the traffic that is aggregated into its child HHH prefixes. The remaining traffic can be composed of (remaining) legitimate and (remaining) attack traffic. In the presence of remaining legitimate traffic, the absolute HHH detection threshold $\phi\mathcal{V}$ must not exceed the combined volume of the remaining attack traffic. Otherwise, part of the legitimate traffic must be aggregated into a mixed aggregate, or it must be fully aggregated into legitimate aggregates.

These conditions impose upper and lower bounds on the HHH detection threshold. Whether they can be met simultaneously depends on the distribution of attack and legitimate traffic volume across the IP address space. While they may be satisfied for some subnets, it can be impossible to avoid mixed and legitimate aggregates throughout the entire IP prefix hierarchy, as outlined in the following example.

Example 4.1.1. The scenario in Figure 4.1 shows HHH prefix detection on the left side that uses a higher HHH detection threshold ($\phi\mathcal{V} = 70$), while the threshold is lower on the right side ($\phi\mathcal{V} = 40$). The higher threshold ensures that conditions (C_1) and (C_2) hold for the prefix `10.0.0.0/30`, which has a combined attack traffic volume of 75 and a legitimate traffic volume of 50. However, choosing a threshold of at least 50 and below 75 violates condition (C_2) for subnet `10.0.0.4/30`. In this case, the threshold is above the attack traffic volume of 50, which causes the legitimate traffic from IP address `10.0.0.7` to be aggregated with the attack traffic from IP address `10.0.0.5` before yielding an HHH prefix. This results in a mixed aggregate, which persists for any higher choice (of at least 75) of the HHH detection threshold.

Still, a lower threshold that satisfies condition (C_2) for the subnet `10.0.0.4/30` does not remedy this situation (as indicated on the right side of Figure 4.1). Choosing a threshold below the attack traffic volume sent from IP address `10.0.0.5` (e.g.,

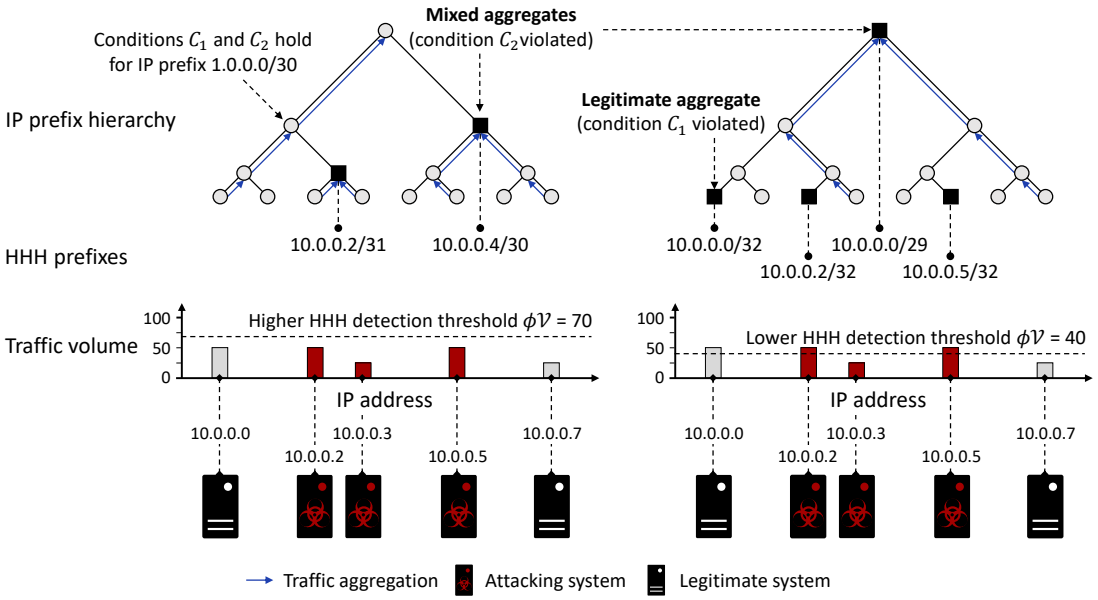


Figure 4.1: Traffic aggregation results in HHH prefixes representing mixed or legitimate aggregates. This cannot be avoided by adjusting the HHH detection threshold $\phi\mathcal{V}$.

$\phi\mathcal{V} = 40$) causes a legitimate aggregate for the HHH prefix `10.0.0.0/32` to appear by violating condition (C_1) (e.g., in the subnet `10.0.0.0/30`). A threshold above 25 (the attack traffic volume sent from IP address `10.0.0.3`) also causes a mixed aggregate for HHH prefix `10.0.0.0/29`. This aggregate occurs at the highest level of the depicted hierarchy due to the distance of the contributing traffic sources. A corresponding filter rule would impact the entire traffic. Avoiding this mixed aggregate would require that condition (C_2) holds for the subnet `10.0.0.0/29` (by choosing $\phi\mathcal{V} < 25$), which would result in another legitimate aggregate for prefix `10.0.0.7/32`.

Consequently, no HHH detection threshold can satisfy both conditions for all subnets. Mixed or legitimate aggregates cannot be avoided in this scenario.

In volumetric DDoS scenarios, the number of traffic sources can be much larger than in the previous example. Finding HHH detection thresholds that minimize the aggregation of legitimate traffic into HHH prefixes is particularly challenging when considering the following two cases:

(1) The presence of individual high-volume legitimate traffic sources.

(2) Wide-spread attacks in which attacking systems send low-volume traffic.

The first case requires high HHH detection thresholds to avoid legitimate aggregates and high false positive rates. Conversely, the second case requires low HHH detection thresholds to reduce the risk of repeatedly mixing attack and legitimate traffic originating from common subnets. To distinguish both cases, threshold selections have to be made in the absence of knowledge about which part of the traffic is attack traffic. While Example 4.1.1 used this knowledge for illustration, it is not explicitly provided by HHH algorithms. Threshold adjustments would need to be made through trial and error. When both cases occur simultaneously, maintaining low false positive rates by changing HHH detection thresholds becomes infeasible.

4.1.2 Estimating Traffic Purity

To generate filter rules with low impact on legitimate traffic, it is not necessary to avoid mixed or purely legitimate aggregates when computing HHH prefixes. The approach presented in this thesis is to identify aggregates containing a significant legitimate traffic volume *after* HHH prefix computation. Specifically, to provide an estimate of the following quantity for all HHH prefixes detected by Aggregate Monitoring:

Definition 4.1: Traffic Purity

Let p be an HHH prefix. Further, let A_p and V_p be the attack and total traffic volume sent from the subnet of prefix p during a monitoring interval. Then the **traffic purity** $\pi_\mathsf{p} \in [0,1] \subset \mathbb{R}$ of the prefix p is given by

$$\pi_\mathsf{p} \stackrel{\text{def}}{=} \frac{A_\mathsf{p}}{V_\mathsf{p}}\,. \tag{4.1}$$

The traffic purity of an aggregate can be used to choose HHH prefixes for filter rule generation. A traffic purity of $\pi_\mathsf{p} = 1$ indicates that an aggregate is composed of pure attack traffic, whereas a traffic purity of $\pi_\mathsf{p} < 1$ indicates that an aggregate contains legitimate traffic. With knowledge of the traffic purity, an HHH prefix with a low traffic purity can be rejected to reduce the risk of removing legitimate traffic.

Definition 4.1 uses ground truth knowledge of the attack and total traffic volume to determine the traffic purity of an aggregate. While HHH algorithms provide an estimation of the total traffic volume V_p, the attack traffic volume A_p remains unknown. Hence, the challenge is to estimate the unknown quantity A_p.

To enable an estimation of the traffic purity, knowledge about legitimate traffic and attack vector characteristics can be used in addition to the volumetric information provided by HHH algorithms. For example, many amplification attack vectors rely on UDP (see Section 2.1.2). Greatly increased UDP traffic from a particular subnet can be an indicator for the presence of such an amplification attack. If the number of bytes sent via UDP during non-attack periods is known, the ratio by which the UDP byte count increases provides (at least some) information about the attack traffic volume within the corresponding aggregate. In addition, several amplification attacks send a high number of frames with large frame size (e.g., DNS amplification attacks), whereas large frames typically occur less often in a network during non-attack periods. The UDP byte count and the distribution of frame sizes constitute features that can be used in combination to estimate the attack traffic volume within a subnet.

Combining multiple features provides more information on the traffic composition of an aggregate and can therefore improve estimations of the traffic purity. The following definition denotes the combined set of features that are used to gain insight into the traffic composition of an aggregate.

Definition 4.2: Aggregate Features

Let p be an HHH prefix, and let $f_1^{(\mathsf{p})}, \ldots, f_n^{(\mathsf{p})} \in \mathbb{N}_0$ be the number of occurrences of $n \in \mathbb{N}$ distinct features in the traffic originating from the subnet of the prefix p during a monitoring interval. Then the vector $\mathsf{f}^{(\mathsf{p})} \stackrel{\text{def}}{=} \left(f_1^{(\mathsf{p})}, \ldots, f_n^{(\mathsf{p})}\right) \in \mathbb{N}_0^n$ denotes the **aggregate features** of the prefix p.

Aggregate features provide a separate count of each different feature within an aggregate. The specific selection of features is not restricted by Definition 4.2. In fact, selecting and monitoring aggregate features is challenging. An HHH algorithm returns only the combined traffic volume for each HHH prefix when queried. To conduct aggregate-based monitoring on multiple different features, HHH algorithms need to be extended to also provide aggregate features. This requires additional counters to keep track of each additional feature value during monitoring and thus increases the memory requirements of HHH algorithms. However, HHH algorithms are designed for memory efficiency to perform efficient traffic processing. Hence, one of the main challenges in using aggregate features is the following:

Selecting memory-efficient features. Selecting features to determine the amount of attack traffic in an aggregate is a trade-off between the information gained by combining multiple features and the resulting increase in the memory utilization of HHH algorithms. Therefore, feature selection should focus on a low number of

features that provide sufficient insight into traffic composition to enable reliable estimations of the traffic purity.

When aggregate features can be obtained for each HHH prefix, the task of Traffic Purity Estimation can be defined as follows:

Definition 4.3: Traffic Purity Estimator

Let $\mathcal{H}$ be a set of HHH prefixes. Further, let the vector $\mathsf{f}^{(\mathsf{p})} \in \mathbb{N}_0^n$ ($n \in \mathbb{N}$) denote the aggregate features of any HHH prefix $\mathsf{p} \in \mathcal{H}$. Then, a **traffic purity estimator** is an algorithm $\mathscr{E}$ that upon input $\mathsf{f}^{(\mathsf{p})}$ outputs an estimated traffic purity $\widehat{\pi}_{\mathsf{p}} \in [0,1] \subset \mathbb{R}$. That is, $\mathscr{E}$ defines a mapping

$$\mathscr{E} : \mathsf{f}^{(\mathsf{p})} \mapsto \widehat{\pi}_{\mathsf{p}} \tag{4.2}$$

for every HHH prefix $\mathsf{p} \in \mathcal{H}$.

Definition 4.3 does not require estimations to be close to the true traffic purity, but estimations closer to the true value π_{p} can be considered to be better. Therefore, the performance of a traffic purity estimator can be expressed through its error:

Definition 4.4: Traffic Purity Estimation Error

Let p be an HHH prefix and π_{p} the traffic purity of p according to Definition 4.1. Further, let $\widehat{\pi}_{\mathsf{p}}$ denote the estimated traffic purity of p using a traffic purity estimator $\mathscr{E}$. Then the (absolute) **traffic purity estimation error** of $\mathscr{E}$ for the prefix p is given by

$$\pi_{\mathsf{p}}^{\mathrm{err}} \stackrel{\mathrm{def}}{=} |\pi_{\mathsf{p}} - \widehat{\pi}_{\mathsf{p}}| \,. \tag{4.3}$$

Achieving lower traffic purity estimation errors enables more accurate distinctions between prefixes representing mainly attack traffic and those with significant legitimate traffic. Still, finding a traffic purity estimator that performs well for a wide range of traffic scenarios is challenging, since the aggregate features are influenced by several traffic characteristics:

- The attack traffic volume, the different volumetric DDoS attack vectors, and the distribution of attacking systems across the IP address space.
- The legitimate traffic volume, traffic characteristics of legitimate network services, and the distribution of legitimate systems over the address space.

Due to the potentially large number of distinct aggregate features that can result from combining attack and legitimate traffic, the following challenge arises:

Low traffic purity estimation errors. A traffic purity estimator should yield low traffic purity estimation errors despite performing estimations on a potentially large set of distinct aggregate features. This is particularly challenging when only partial knowledge about the attack and legitimate traffic is available. For example, legitimate traffic can change in an unpredictable manner, and volumetric DDoS attacks can use previously unseen combinations of attack vectors and attack intensities.

Still, it is not necessary that a traffic purity estimator yields strictly minimal errors to achieve low false positive rates during traffic filtering. Instead, errors can be compensated for by realizing a trade-off between false positive and false negative rates. By requiring a slightly higher estimated traffic purity when selecting HHH prefixes to generate filter rules, prefixes can still be rejected if the traffic purity is slightly underestimated. This may result in more prefixes being rejected (and potentially leads to increased false negative rates), but it introduces a tolerance for Traffic Purity Estimation while maintaining low false positive rates.

4.2 Design Goals

HHH algorithms enable efficient monitoring of high-volume traffic aggregates, but they are not designed to distinguish between attack and legitimate traffic. The primary goal of Traffic Purity Estimation is to identify HHH prefixes whose aggregates are mainly composed of attack traffic. This serves to maintain low false positive rates when selecting prefixes for traffic filtering. Based on the challenges discussed previously, this goal is divided into two main aspects:

(1) Designing a traffic purity estimator that can achieve low traffic purity estimation errors in diverse traffic scenarios.

(2) Focusing on important aggregate features to maintain low HHH algorithm memory requirements (while still retaining low traffic purity estimation errors).

4.2.1 Data-Driven Traffic Purity Estimation

Volumetric DDoS attacks can have diverse traffic characteristics. This is because attackers have multiple options in terms of attack vectors, traffic dynamics, and available attack traffic sources. Additionally, legitimate traffic characteristics depend on attack targets and can change over time. To accommodate this diversity, the goal is to automatically learn the relationship between aggregate features and traffic purity from traffic traces. More specifically, to train a machine learning model through (supervised) regression analysis to achieve low traffic purity estimation errors in diverse traffic scenarios. Relying on a data-driven approach enables training of models for various networking environments without changing the training procedure.

This design goal is addressed in Section 4.3, which structures the task of realizing a traffic purity estimator into three subtasks:

(1) Identifying potentially useful aggregate features that can be efficiently monitored and that can indicate the presence of volumetric DDoS attack traffic.

(2) Extending HHH algorithms to monitor multiple aggregate features instead of just traffic volume.

(3) Training deep neural networks to achieve low traffic purity estimation errors upon input of monitored aggregate features.

4.2.2 Memory-Efficient Aggregate Features

HHH algorithms are designed to use a limited number of counters to achieve high monitoring efficiency. As outlined in Section 4.1.2, monitoring multiple aggregate features requires additional counters to track the frequency of different feature values separately. This leads to a trade-off between the memory requirements of HHH algorithms and the information on traffic composition provided by aggregate features. To avoid unnecessary memory utilization, the second design goal is to find memory-efficient aggregate features, i.e., a small subset of the potentially useful features that still enable machine learning models to achieve low traffic purity estimation errors.

Section 4.4 introduces an approach to select memory-efficient aggregate features from an initial set of generic features. The approach addresses two key objectives:

(1) Quantifying the impact that individual features have on the performance of a trained machine learning model (regarding traffic purity estimation errors). This serves to make features comparable in order to distinguish between features of low and high importance.

(2) Selecting a smaller subset of features that maintains low traffic purity estimation errors. The importance of features guides this selection to retain features of high importance while discarding features of lower importance. Alternatively, multiple features may be combined to reduce HHH memory requirements.

A selection of features occurs in conjunction with model training. That is, memory requirements can be calculated and reduced prior to active traffic monitoring.

4.3 Traffic Purity Estimation Through Supervised Learning

The following introduces an approach to realize a traffic purity estimator based on deep neural networks. Section 4.3.3 presents the design for a feed-forward neural network that estimates the traffic purity of an HHH prefix from its aggregate features. Through supervised learning, the neural network can be trained to reduce traffic purity estimation errors in different traffic scenarios. The training process is outlined in Section 4.3.4. It requires training data that contains the aggregate features of multiple HHH prefixes, as well as the true traffic purity. Once the deep neural network is trained, the true traffic purity is no longer required and estimations can be obtained from aggregate features alone.

4.3.1 Aggregate Features Indicating Volumetric DDoS Attacks

In a volumetric DDoS attack, the presence of attack traffic in addition to legitimate traffic changes traffic characteristics. At a minimum, the traffic volume increases. A change in traffic characteristics may not only indicate the presence of attack traffic, but also provide information on attack traffic volume. Quantifying the change requires a selection of features that can be monitored during attack and non-attack periods.

Feature requirements. Selective Aggregate Filtering is designed to use efficient HHH algorithms to acquire monitoring information and to estimate traffic purity from aggregated traffic. For this, features should fulfill several requirements:

(1) Features should provide per-prefix statistics. More precisely, features should provide a quantitative measure of traffic characteristics in arbitrary subnets for both legitimate and attack traffic. Monitoring only specific attack vector characteristics (e.g., certain source ports) may fail to provide insight into the composition of legitimate traffic in an aggregate.

(2) Monitoring feature should not involve complex computations. The effort in monitoring features should be comparable to the counting performed with an HHH algorithm to maintain computational efficiency.

(3) Monitoring features should not require extensive state keeping to maintain memory efficiency. This prohibits the use of features that track individual flows over time (e.g., to monitor flow durations) and features that require a high number of counters (such as distinguishing unique IP source addresses).

The following list of features is chosen to take these requirements into account.

Features. All subsequent features can be monitored through counting, summation, and aggregation over the IP prefix hierarchy. Relying on counting and summation (instead of more complex operations) serves to maintain a monitoring efficiency that is comparable to the counting performed by (unmodified) HHH algorithms. Aggregation simply requires summation of feature values of child prefixes to obtain values for arbitrary parent prefixes. Additionally, deep packet inspection is avoided by limiting features to protocols up to the transport layer.

- **Prefix length**. The prefix length of an aggregate indicates the extent of the IP address range over which aggregated monitoring information is collected. Including the prefix length in aggregate features can indicate whether or not other features are subject to extensive aggregation. The prefix length can be determined when calculating HHHs.[1]

- **Byte count**. The number of bytes being sent is a common indicator for volumetric DDoS attacks, since attackers seek to overload a network infrastructure by sending unsolicited traffic. Consequently, the total byte count increases when attack traffic is present in addition to legitimate traffic.

[1] Note that only the prefix length and not the address information of a prefix should be included to avoid overfitting on specific IP address space regions.

- **Packet count**. Volumetric DDoS attacks increase the number of packets in comparison to non-attack periods. This is particularly noticeable when an attacker relies on attack vectors that generate small-sized frames, since a higher number of frames is required to reach a certain attack traffic volume.
- **Transport protocol**. The presence of certain attack vectors can shift the distribution of used protocols. While TCP is used by a wide variety of networked applications, many amplification attack vectors rely on UDP (e.g., NTP, DNS, SSDP, and OpenVPN amplification attacks). Hence, counting the frequency at which packets use a specific protocol can provide insight into the presence and intensity of attack traffic. Monitoring the occurrence of protocols may indicate the presence of other attack vectors (such as ICMP floods).
- **Source port**. Attack vectors can generate packets with distinctive UDP or TCP source ports. In particular, the majority of amplification attack vectors generate packets with specific source ports or source ports within a limited range (see Section 2.1.2). Since an attacker has no direct control over a reflector, this attack traffic characteristic cannot be avoided when relying on such attack vectors. In contrast, legitimate clients often access remote systems via randomly chosen source ports in the dynamic port range from port 49,152 to 65,535, or via specific well-known ports. The frequency at which source ports occur can be used as a quantitative indicator for attack and legitimate traffic.
- **Frame size**. Several attack vectors can result in frame size distributions that differ from those of legitimate traffic. This is often the case when using reflectors that send IP datagrams with a characteristic size. For example, DNS and NTP amplification attacks can generate large UDP datagrams that get fragmented into smaller IP datagrams. This leads to an increased number of frames with the size of the maximum transmission unit of a network path. In contrast, legitimate traffic often contains smaller frames such as TCP acknowledgements without payload. Therefore, monitoring the distribution of frame sizes can be an indicator for the amount of attack and legitimate traffic in an aggregate.

While a single feature may be insufficient to accurately distinguish between attack and legitimate traffic, several features can be combined to provide more information on traffic composition and result in more accurate estimations of the traffic purity. The amount of information provided by features also depends on the granularity at which feature values are monitored. Monitoring the frequency of all possible source ports separately requires 65,536 different counters for each of the TCP and UDP protocols. While this provides full information on source port distribution,

131,072 additional counters would be required when monitoring source ports for both protocols. This can constitute a prohibitive increase in memory requirements, especially when considering that most ports may not be used at all. Instead, the distribution of feature values can be monitored in more memory-efficient ways.

Binning. Instead of monitoring every possible feature value, a range of feature values can be combined into a **bin**. Each bin requires only a single counter to keep track of the number of times a value falls into its value range. By partitioning the range of possible source ports into 64 evenly-sized bins, the memory consumption to monitor source ports can be reduced by a factor of 1024. Bins can also be chosen to represent arbitrary value ranges. For example, only two bins can be used to count the frequency of source ports falling into the ranges 0 to 1023 and 1024 to 65,535. Through binning, different trade-offs between memory consumption and provided monitoring information can be realized.

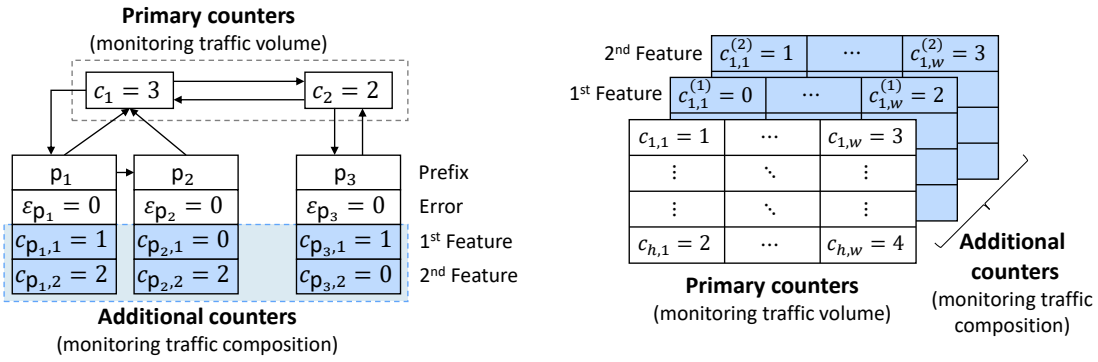
TCP/UDP source port distinction. It is not strictly necessary to distinguish between TCP and UDP source ports. Instead, a single set of counters can be used to monitor the combined source port distribution. This reduces the memory requirements for source port monitoring by a factor of two and can be particularly beneficial when TCP and UDP source port distributions can be expected to be very different.

Section 4.5 evaluates several alternatives for the selection, combination, and binning of different features.

4.3.2 Monitoring Aggregate Features with HHH Algorithms

HHH algorithms monitor traffic volume to detect HHH prefixes. For this, they use one counter for each currently monitored prefix to estimate its traffic volume. These counters are referred to as **primary counters** because the decision to classify a prefix as an HHH prefix is based on traffic volume (see Definition 2.2). The SpaceSaving algorithm uses an additional counter to track the maximum error of traffic volume estimations.

To monitor aggregate features, **additional counters** are required to keep track of the frequency at which individual feature values occur. For example, counting the number of UDP packets requires a separate counter from the primary counter of a prefix. These counters are *not* taken into consideration when determining if the traffic



(a) Space Saving data structure. **(b)** Sketch data structure.

Figure 4.2: Heavy Hitter data structures with two additional counters that track the frequency of two feature values separately to indicate traffic composition.

volume of an aggregate reaches the HHH detection threshold $\phi\mathcal{V}$. Their purpose is instead to provide insight into the traffic composition of an aggregate in order to enable subsequent Traffic Purity Estimation. Since HHH prefix detection depends only on primary counters, HHH algorithms can be modified in a straightforward manner include aggregate features. For this, the data structure of underlying Heavy Hitter algorithms as well as updates and queries need to be adjusted.

Extending Heavy Hitter data structures. To monitor aggregate features with HHH algorithms, the data structure of the underlying Heavy Hitter algorithm can be extended to include a fixed-sized array of additional counters for every monitored prefix. The array contains one counter for each monitored feature as depicted in Figure 4.2 for the Space Saving and sketch data structures. For the Space Saving algorithm, the additional counters reside in the linked list of prefixes, while additional counter arrays are allocated for the sketch data structure. The additional counters increase the data structure's memory requirements, but tracking feature values separately opens up the possibility of distinguishing between attack and legitimate traffic.

Counter updates. The update procedure of an HHH algorithm does not need to be modified to update *primary* counters. Primary counters of a prefix are updated unconditionally. In contrast, only a subset of the additional counters of a prefix may be incremented. For example, updates of counters tracking the frequency of transport layer protocols (such as TCP and UDP) are mutually exclusive. Similarly, only one frame size bin needs to be updated for each packet. Consequently, the number of required counter updates can be significantly lower than the number of monitored features. To update the correct counters, the `update` method of a Heavy Hitter

Algorithm 1: Modified `update` Method of a Count-Min Sketch

▷ *This modified version of the `update` function monitors transport layer protocol and source port distributions. Modified parts are highlighted in blue.*

Input : three-dimensional counter array C (of width w, height h, and depth d), pairwise-independent hash functions hash_i $(i = 1, \ldots, h)$, logarithm (to base 2) of the port bin size ld_bin_size

```
1  Function update(prefix p, count cnt, protocol proto, source port prt):
2      b ← prt ≫ ld_bin_size;                          ▷ Calculate port bin
3      foreach i = 1, ..., h do
4          j ← hash_i(p);                              ▷ Determine index via hash function
5          C_{i,j} ← C_{i,j} + cnt;                    ▷ Update primary counter
6          if proto = UDP then
7              C^(1)_{i,j} ← C^(1)_{i,j} + 1;          ▷ Update UDP count
8          else if proto = TCP then
9              C^(2)_{i,j} ← C^(2)_{i,j} + 1;          ▷ Update TCP count
10         else
11             C^(3)_{i,j} ← C^(3)_{i,j} + 1;          ▷ Update count of other protocols
12         C^(4+b)_{i,j} ← C^(4+b)_{i,j} + 1;          ▷ Update i-th port bin
```

algorithm needs to determine feature bins without incurring high computational complexity. When feature bins are contiguous, aligning the bin size to a power of two enables bin calculation through bit-shifting. For categorical features such as transport layer protocols, case distinctions can be used. Since aggregate features must be selected prior to monitoring, the `update` method can be adjusted accordingly before using a Heavy Hitter algorithm. Algorithm 1 provides an example for a modified `update` method of a Count-Min Sketch that monitors the transport layer protocol and source port distributions. Bin calculation uses bit-shifting for source port features and `if-else` clauses to determine the correct bin for the transport layer protocol.

Querying aggregate features. To obtain all aggregate features, queries to a Heavy Hitter algorithm can simply be modified to return the stored additional counters besides the primary counters when a prefix is classified as a Heavy Hitter. All other operations of a query can remain unchanged. Likewise, HHH algorithms can simply return the aggregated additional counters of a prefix that has been classified as an HHH prefix.

4.3.3 Neural Network Architecture

After querying an HHH algorithm at the start of a monitoring interval, the algorithm returns all aggregate features $\mathbf{f}^{(p)}$ for detected HHH prefixes p. The task is then to assign each prefix p an estimated traffic purity $\widehat{\pi}_p$ with low traffic purity estimation error. Traffic Purity Estimation performs this task using a neural network. With neural networks, the traffic purity of several thousand HHH prefixes can be estimated in parallel through batch processing on a GPU.

Central aspects in designing a neural network architecture to estimate traffic purity are the following:

- **Continuous scalar output**. The neural network architecture must be designed to map higher-dimensional aggregate features to a single scalar output representing the estimated traffic purity. Unlike classification, the output must be able to cover the continuous range of all possible traffic purity values (i.e., the interval $[0, 1] \subset \mathbb{R}$) with floating-point precision.
- **Normalization**. Aggregate features need to be normalized to reduce the risk of vanishing and exploding gradients when relying on deep neural networks. Normalization often determines whether or not a machine learning model can be successfully trained and can also affect model performance (cf. Section 2.4.1).
- **Generalization**. The neural network architecture should be designed to reduce the risk of overfitting the training data. In the context of Traffic Purity Estimation, generalization refers to the ability of a neural network to provide accurate traffic purity estimations when it encounters previously unseen aggregate features during inference. This leads to more robust estimations when legitimate and attack traffic deviate from the training data.

Figure 4.3 outlines the architecture of a deep neural network that takes these considerations into account. The network consists of multiple layers (whose functionality is described in Section 2.4.1) that are processed in a sequential order.

- **Fully-connected layers**. The first fully-connected layer receives the aggregate features $\mathbf{f}^{(p)}$ as input that are provided by an HHH algorithm. Any subsequent fully-connected layer receives the outputs of the preceding layer as input (see below). The layer then processes its input as described in Section 2.4.1, except that an activation is performed by a subsequent layer (see below). For this, each fully-connected layer uses a fixed number of neurons.

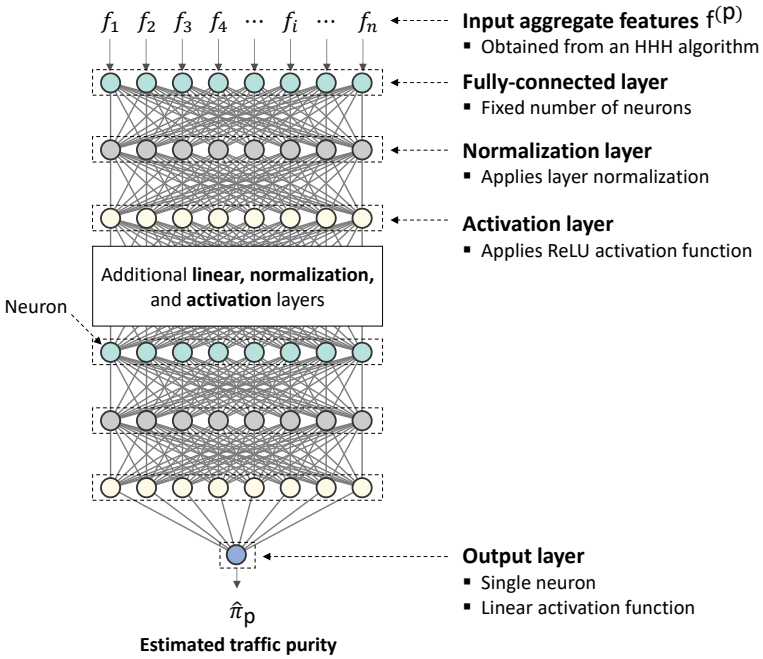


Figure 4.3: Architecture of a feed-forward neural network that outputs an estimated traffic purity $\widehat{\pi}_{\mathsf{p}}$ upon input of aggregate features $\mathsf{f}^{(\mathsf{p})}$ for an HHH prefix p.

- **Normalization layer**. A normalization layer receives the output from the preceding fully-connected layer and applies layer normalization (see Section 2.4.2). The normalized features constitute the input for a subsequent activation layer.

- **Activation layers**. The activation layer applies a ReLU activation function. This activation function introduces non-linearity. Placing the activation layer behind the normalization layer follows current best practices for layer normalization.

- **Output layer**. The final layer of the neural network constitutes a fully-connected layer with a single neuron that outputs the estimated traffic purity $\widehat{\pi}_{\mathsf{p}}$ for the prefix p. It uses a linear activation function to avoid limiting the output value range. This enables it to obtain a proportional feedback for outputs after the forward pass of neural network training.

The neural network architecture can be extended to use multiple fully-connected layers to increase its capacity to learn higher-level features. In this case, every fully-connected layer is (optionally) followed by a normalization layer and an activation layer. This results in a sequence of building blocks composed of these layers.

Further details on implementing neural networks with the described architecture can be found in Appendix A.2.1.

4.3.4 Model Training

The goal of model training in the context of Traffic Purity Estimation is to train a neural network to learn a model M of the relationship between aggregate features and traffic purity for a set of HHH prefixes $\mathcal{H}$. To this end, model training performs stochastic gradient descent (see Section 2.4.2) in order to conduct a regression analysis over multiple training epochs. Each epoch iterates over all samples in a dataset $\mathsf{D} = \left\langle \left(\mathsf{f}^{(\mathsf{p})}, \pi_{\mathsf{p}}\right) \mid \mathsf{p} \in \mathcal{H} \right\rangle$. A sample $\left(\mathsf{f}^{(\mathsf{p})}, \pi_{\mathsf{p}}\right)$ consists of the following:

- Aggregate features $\mathsf{f}^{(\mathsf{p})} = \left(f_1^{(\mathsf{p})}, \ldots, f_n^{(\mathsf{p})}\right)$ of a prefix $\mathsf{p} \in \mathcal{H}$. The aggregate features constitute the input to a neural network in a forward pass. The selection of the n different features is determined when monitoring aggregate features with an HHH algorithm (as described in Section 4.3.2).

- The traffic purity π_{p} of a prefix $\mathsf{p} \in \mathcal{H}$. The traffic purity constitutes the targets of the regression analysis. It is used to calculate losses and to evaluate model performance. The values of π_{p} are calculated from ground truth information (according to Definition 4.1), which can be assumed to be known when creating training data for supervised learning.

The dataset D is split into three disjoint parts $\mathsf{D}_{\mathsf{train}}$, $\mathsf{D}_{\mathsf{val}}$, and $\mathsf{D}_{\mathsf{test}}$ that are used for different purposes:

(1) $\mathsf{D}_{\mathsf{train}}$: Training data that is used for model training. It provides the necessary data for forward passes and back-propagation.

(2) $\mathsf{D}_{\mathsf{val}}$: Validation data used to evaluate model performance during the training process, e.g., to detect when a model starts to overfit.

(3) $\mathsf{D}_{\mathsf{test}}$: Test data that is not used during the training process but can later be used to evaluate model performance after training has been completed.

The objective of model training is to improve the model's performance in terms of achieving low traffic purity estimation errors.

Model performance. The performance of a model that is used as traffic purity estimator is determined based on the realized traffic purity estimation errors for prefixes p in a set of HHH prefixes $\mathcal{H}$ (see Definition 4.4). Specifically, it is quantified in terms of the mean absolute error and the mean squared error, which are defined as follows in the context of Traffic Purity Estimation:

Definition 4.5: Mean Absolute Error (MAE) and Mean Square Error (MSE)

Let $\mathcal{H}$ be a set of HHH prefixes and M a (trained) machine learning model that serves as a traffic purity estimator. Further, let the dataset D contain aggregate features $\mathsf{f}^{(\mathsf{p})}$ for all prefixes $\mathsf{p} \in \mathcal{H}$ and let $\pi_{\mathsf{p}}^{\mathsf{err}}$ denote the traffic purity estimation error realized by M upon input of $\mathsf{f}^{(\mathsf{p})}$. Then, the **mean absolute error** $\pi_{\mathsf{D}}^{\mathsf{MAE}}$ and the **mean squared error** $\pi_{\mathsf{D}}^{\mathsf{MSE}}$ on the dataset D are given by

$$\pi_{\mathsf{D}}^{\mathsf{MAE}} \stackrel{\text{def}}{=} \frac{1}{|\mathcal{H}|} \sum_{\mathsf{p} \in \mathcal{H}} \pi_{\mathsf{p}}^{\mathsf{err}}, \tag{4.4}$$

$$\pi_{\mathsf{D}}^{\mathsf{MSE}} \stackrel{\text{def}}{=} \frac{1}{|\mathcal{H}|} \sum_{\mathsf{p} \in \mathcal{H}} \left(\pi_{\mathsf{p}}^{\mathsf{err}}\right)^2. \tag{4.5}$$

When the dataset D is known from context, the corresponding metrics on $\mathsf{D}_{\mathsf{train}}$, $\mathsf{D}_{\mathsf{val}}$, and $\mathsf{D}_{\mathsf{test}}$ are denoted by $\pi_{\mathsf{train}}^{\mathsf{MAE}}$, $\pi_{\mathsf{val}}^{\mathsf{MAE}}$, $\pi_{\mathsf{test}}^{\mathsf{MAE}}$, $\pi_{\mathsf{train}}^{\mathsf{MSE}}$, $\pi_{\mathsf{val}}^{\mathsf{MSE}}$, and $\pi_{\mathsf{test}}^{\mathsf{MSE}}$.

In general, the performance of a model M is considered to be better when the MAE and MSE are lower.

Several design choices influence the final performance regarding the traffic purity estimation error of a trained neural network:

- **Loss function**. During each forward pass of the training process, the neural network outputs an estimated traffic purity $\widehat{\pi}_{\mathsf{p}}$. During back-propagation, gradient updates are calculated from a loss function based on the estimated traffic purity $\widehat{\pi}_{\mathsf{p}}$ and the true traffic purity π_{p}, which is included in the training data $\mathsf{D}_{\mathsf{train}}$. A lower traffic purity estimation error directly indicates improved model performance but using the mean absolute error or mean squared error as loss function can cause gradients to overshoot or become sensitive to large residuals as outlined in Section 2.4.2. Instead, the Huber loss is used as a surrogate loss metric that also reduces MAE and MSE.
- **Patience**. Training a neural network over too many epochs can result in overfitting on the training data and elevated traffic purity estimation errors during

inference. Evaluating model performance on the validation data D_{val} after each epoch can indicate when a model starts to overfit. In this case, losses on the validation data start to rise whereas they usually decline as long as overfitting does not occur. Instead of training a model for a fixed number of epochs, the training process can be stopped once the losses on the validation data increase. A more tolerant approach is to stop the training process when no improvement has been achieved over a predefined number of epochs. This is referred to as **patience** and allows for short-term increases in validation losses that do not result in immediate termination of model training. Since losses can be subject to small fluctuations between epochs, introducing patience into a training process can result in an overall improved model performance.

- **Checkpoints**. Similar to introducing patience into the training process, checkpoints are designed to achieve a higher model performance when losses fluctuate during training. Every time the model achieves new minimal validation losses, the neural network architecture and its weights are saved. Should the model performance decrease in subsequent epochs, the best model can be restored from a checkpoint once the training ends. This allows a patience training process to execute several more epochs after reaching a minimum validation loss without impacting overall model performance.

- **Batch size**. The batch size determines the number of samples that are processed during a forward pass before losses are calculated, which reduces training times through parallel processing. Back propagation is based on the mean loss taken over an entire batch at the end of a training epoch. This can impact model performance when introducing patience into a training process. When the number of samples in the training data is not a multiple of the batch size, a smaller batch remains for the last training epoch. Losses calculated over smaller batches can exhibit higher variance and cause larger gradients. As a result, the patience can expire prematurely resulting in lower model performance. To avoid the impact of small batches, the last batch in a dataset can be discarded when it has a small size.

- **Sample weights**. The number of detected HHH prefixes can vary greatly depending on the prefix length. For example, only one root IP prefix of length 1 exists, but many fully qualified HHH prefixes could be detected when a large number systems send traffic. Fully qualified prefixes often correspond to pure attack or legitimate traffic. This introduces a bias in the distribution of the (true) traffic purity in training data. To compensate this, the targets of the

training data can be weighted. For this, traffic purity values are partitioned into 100 bins and weighted reciprocally by the square root of the numbers of samples in each bin. Applying a square root reduces the effect of weighting when too many values fall into the same bin. This could otherwise lead to losses becoming too small.

Further details on realizing the training process using a current machine learning framework can be found in Appendix A.2.1.

4.3.5 Postprocessing During Inference

While the output layer of the feed-forward neural network from Section 4.3.3 is required to cover all possible traffic purity values, it can also yield values *outside* of the interval $[0,1] \subset \mathbb{R}$. This can lead to a significant overestimation of the traffic purity ($\widehat{\pi}_p \gg 1$) and negative estimations ($\widehat{\pi}_p < 0$). During training, such estimations are counteracted through back-propagation, but they should be entirely avoided during inference. In particular, considering the hierarchical order of prefixes (as outlined in Section 3.3.5 and further discussed throughout Chapter 5) requires that traffic purity estimations remain within the interval $[0,1] \subset \mathbb{R}$. This can be ensured by performing additional postprocessing that applies the function $x \mapsto \min(\max(0,x),1)$ to an output x of the neural network.

4.4 Reducing HHH Memory Requirements

While Section 4.3.1 discussed several potentially useful features, this section addresses the design goal of selecting memory-efficient aggregate features (see Section 4.2.2) to keep HHH memory requirements low. To this end, Traffic Purity Estimation should focus on features that have the highest impact on traffic purity estimation errors. For example, the 65,536 different UDP source ports can be partitioned into 64 bins, requiring an equal number of counters during HHH monitoring. Still, only a limited subset of the port bins (e.g., only three bins) may be relevant for a trained machine learning model. The goal then resides in finding a small number of features of high importance without significantly impacting traffic purity estimation errors.

This kind of feature selection is a common challenge in machine learning, for which several approaches have been proposed in the past [CS14; SRK15; WLS15; Li+17;

DA22]. Selecting a method to assess the importance of features for the task to reduce HHH algorithm memory footprints has to satisfy several requirements:

- The feature selection method should provide a *quantitative* measure on the importance of individual features to enable a comparison and selection of different features.
- The feature selection method should enable a choice of important features *before* monitoring is conducted. Through this, the unnecessary use of counters in Heavy Hitter data structures can be avoided in the first place. In contrast, computing a reduced number of features from already acquired monitoring information in a later processing stage (e.g., via Principal Component Analysis [JC16]) cannot serve to reduce HHH memory requirements.
- Determining feature importance should not introduce feature values that do not appear in collected datasets.

4.4.1 Feature Importance

The impact of an individual feature on model performance can be assessed by measuring the performance of a trained model on a dataset in which the relationship between the feature and the target value has been broken. In this case, the performance is expected to decrease, as the feature can no longer directly contribute to estimations of the target value. As a result, the reduction in model performance provides a quantitative measure on the importance of the feature to the trained model. To break the correlation between features and target values [Bre01] introduced a method that permutes the values of a single feature across all samples while leaving all other features unchanged.

Figure 4.4 outlines the process of assessing feature importance in the context of Traffic Purity Estimation. The process uses a single dataset D (as described in Section 4.3.4) to calculate the individual importance of each of the n features that comprise the aggregate features of a prefix (see Definition 4.2). The dataset is split into a training dataset $\mathsf{D}_{\mathsf{train}}$, validation dataset $\mathsf{D}_{\mathsf{val}}$, and test dataset $\mathsf{D}_{\mathsf{test}}$ by selecting a subset of the HHH prefixes for each. The three datasets are used throughout the following five-step process to calculate the importance of individual features:

① **Model training**. A model M is trained using the training and validation datasets ($\mathsf{D}_{\mathsf{train}}$ and $\mathsf{D}_{\mathsf{val}}$) as outlined in Section 4.3.4. The values of all features in the

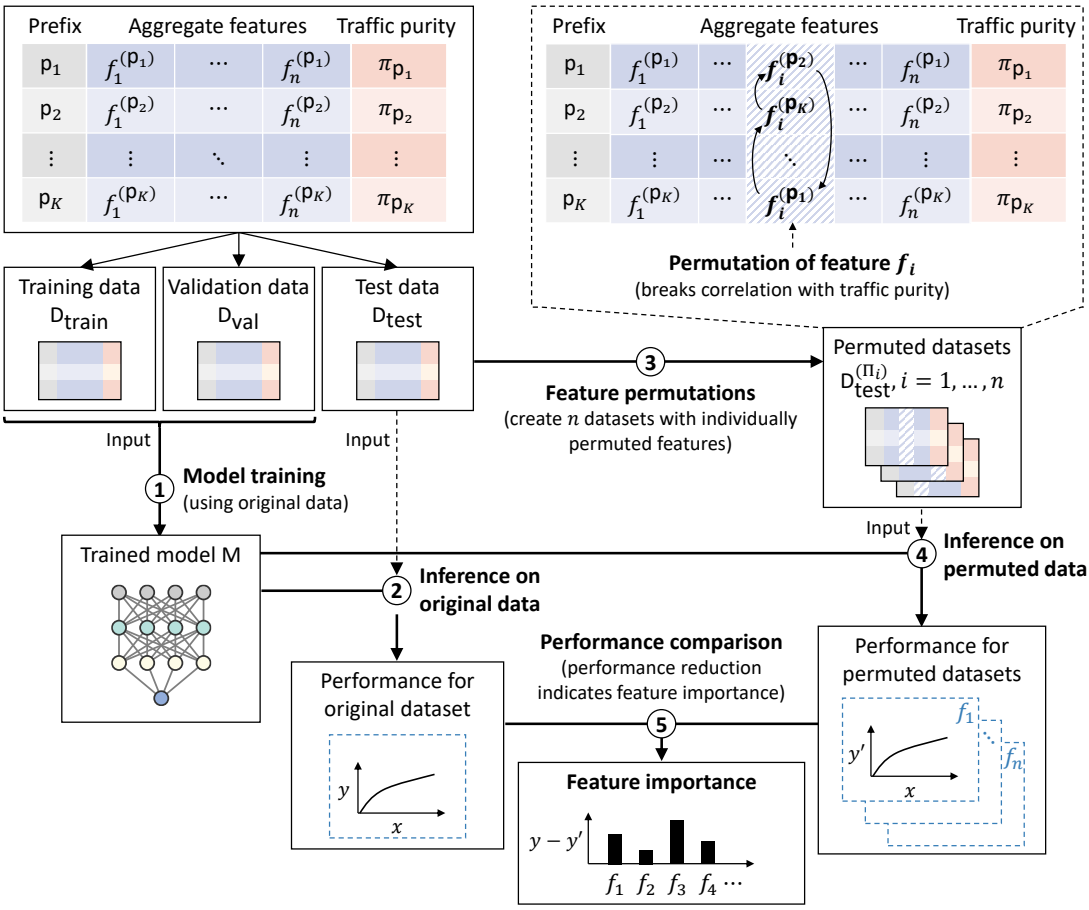


Figure 4.4: Process to assess the impact of individual features on Traffic Purity Estimation by permuting feature values.

training and validation data remain unchanged (i.e., no permutation is applied). Through this, the model can make full use of all features that are related to the traffic purity of a prefix during training.

② **Inference on original data**. Inference is performed on the test dataset D_{test} using the trained model M. This provides estimations on traffic purity $\widehat{\pi}_{\mathsf{p}}$ for all prefixes p in the set of HHH prefixes $\mathcal{H}$ of the test dataset. Based on the estimations, the MAE $\pi^{\text{MAE}}_{\text{test}}$ is calculated (according to Equation 4.4). This constitutes the baseline for later comparisons of model performance on datasets with permuted features.

③ **Feature permutations**. For each of the n different features that comprise the aggregate features $\mathsf{f}^{(\mathsf{p})}$ of HHH prefixes p, a new dataset $\mathsf{D}^{(\Pi_i)}_{\text{test}}$ $(i = 1, \ldots, n)$

is created from the test dataset D_{test}. Each new dataset $\mathsf{D}_{\text{test}}^{(\Pi_i)}$ is created by permuting the values of the i-th feature across all prefixes. Specifically, for each feature, a permutation $\sigma_i \in S_K$ is chosen randomly from the symmetric group S_K (where K denotes the number of HHH prefixes in D_{test}). Using σ_i, the following transformation is applied to all feature values $f_j^{(\mathsf{p}_k)}$ ($k = 1, \ldots, K$; $j = 1 \ldots, n$) in the dataset D_{test}:

$$\Pi_i : f_j^{(\mathsf{p}_k)} \mapsto \begin{cases} f_j^{(\mathsf{p}_{\sigma_i(k)})} & \text{if j} = \text{i}, \\ f_j^{(\mathsf{p}_k)} & \text{otherwise.} \end{cases} \tag{4.6}$$

By permuting the values of the i-th feature, a feature value $f_i^{(\mathsf{p}_{\sigma_i(k)})} \neq f_i^{(\mathsf{p}_k)}$ is no longer directly correlated to the true traffic purity π_{p}. However, the relationship between π_{p} and all other features remains unchanged.

④ **Inference on permuted data**. Similar to step ②, inference is performed on each of the n datasets $\mathsf{D}_{\text{test}}^{(\Pi_1)}, \ldots, \mathsf{D}_{\text{test}}^{(\Pi_n)}$ to obtain estimations on traffic purity with permuted feature values. For this, the *same* trained model M is used. Again, the model's performance is evaluated independently for each dataset $\mathsf{D}_{\text{test}}^{(\Pi_i)}$ to obtain a new MAE value $\pi_{\text{test},\Pi_i}^{\mathsf{MAE}}$, which indicates the model's performance on the permuted dataset (using transformation Π_i from Equation 4.6).

⑤ **Performance comparison**. The result obtained in step ② is compared to all performance results obtained in step ④. Since step ③ broke the correlation of feature values with the traffic purity, inferences on datasets with permuted features typically yield lower performance than on datasets with unchanged feature values. The performance reduction (i.e., the increase in the traffic purity estimation error) resulting from using model M on a dataset $\mathsf{D}_{\text{test}}^{(\Pi_i)}$ provides a quantitative measure of the impact of the i-th feature on model performance.

The performance reduction observed in step ⑤ constitutes the permutation feature importance of a single feature, which is given as follows:

Definition 4.6: Permutation Feature Importance

Let M be a trained model, D_{test} the test data of a dataset D, and $\mathsf{D}_{\text{test}}^{(\Pi_i)}$ the corresponding dataset with permuted values of feature f_i (according to Equation 4.6). Further, let $\pi_{\text{test}}^{\text{MAE}}$ and $\pi_{\text{test},\Pi_i}^{\text{MAE}}$ denote the MAE in estimations of the traffic purity using model M on D_{test} and $\mathsf{D}_{\text{test}}^{(\Pi_i)}$. Then, the **permutation feature importance** $\mathfrak{I}_{f_i}$ of feature f_i on the dataset D is given by

$$\mathfrak{I}_{f_i} \stackrel{\text{def}}{=} \pi_{\text{test},\Pi_i}^{\text{MAE}} - \pi_{\text{test}}^{\text{MAE}} . \tag{4.7}$$

By comparing the importance of individual features, those with the highest impact on model performance can be identified.

Permuting individual feature values does not introduce values other than those that are already present in a dataset. In particular, inferences are performed for the same prefixes (from the dataset D_{test}) and step ③ only replaces features values with existing values of other prefixes. Furthermore, the model architecture and weights do not change while calculating feature importance. The model is trained only once during step ①. Through this, random decisions made during model training (e.g., weight initialization of neural network layers) affect all computations of feature importance in the same way.

Further details on implementing the calculation of feature importance using trained neural networks can be found in Appendix A.2.1.

4.4.2 Selecting Features Based on Feature Importance

The previous section outlined how the importance of individual features can be assessed. With this information, the second design goal of finding memory-efficient aggregate features (as outlined in Section 4.2.2) can be addressed. The task is to derive a lower number of features from an initially large feature set while maintaining low traffic purity estimation errors. To reduce the number of features in a feature set, two options are available:

(a) Discard a feature entirely to remove the counter from the HHH data structure.

(b) Merge multiple bins into a single bin to use only a single counter for the combined range of values covered by the original bins.

Both options can be applied multiple times to reduce overall HHH memory requirements. However, the impact of selecting features on model performance (in terms of MSE and MAE) can only be measured after training a new model that uses the reduced feature set. To avoid excessive feature reduction and to balance memory requirements against traffic purity estimation errors multiple different feature selections should be compared. To enable a comparison between different features, finding memory-efficient aggregate features progresses through the following steps:

(1) Construct an initially large set of aggregate features that includes all feature candidates from Section 4.3.1. When the value range of a feature candidate is divided into bins, the bins cover the entire range of possible feature values (e.g., all 65,536 UDP ports). The initial partitioning into bins determines which features can later be discarded or merged (see Step 4).

(2) Train and evaluate a model that uses the large feature set. Training uses the training data $\mathsf{D}_{\mathsf{train}}$. The evaluation performs inference on the test data $\mathsf{D}_{\mathsf{test}}$ to determine the models performance regarding MSE and MAE. The MSE and MAE serve as a comparison for models with a lower number of features.

(3) Determine the permutation feature importance of all features in the initial large feature set. This provides a guideline to decide which features should be discarded, retained, or merged.

(4) Discard or merge features based on the permutation feature importance from Step 3. In addition to feature importance, domain knowledge about DDoS attack vectors can be used to refine feature selections.

(5) Train and evaluate a new model that uses the reduced feature set from Step 4. Training uses the same data $\mathsf{D}_{\mathsf{train}}$ and $\mathsf{D}_{\mathsf{test}}$ for training and evaluation as in Step 2 to ensure comparability model performance.

Steps 4 and 5 can be repeated multiple times for different choices of discarded, retained, and merged features. This enables the selection of models that realize different trade-offs between traffic purity estimation errors and HHH memory requirements. If a model's errors are too large, the model and its feature set can be entirely disregarded.

The selection of features and bins in Step 1 and Step 4 can impact the overall performance of a model. Therefore, domain knowledge should be taken into consideration when choosing initial bins, to avoid the loss of important information, and to exploit redundancy among features.

Choosing initial bins. The partitioning into bins in Step 1 should be fine-granular enough to enable a flexible selection of features in Step 4. It should also avoid generating an excessive number of features, since this introduces unnecessarily high computational complexity in determining feature permutation importance along with high combinatorial complexity in choosing reduced feature sets. For example, using a distinct feature for each UDP port (corresponding to a bin size of one) would require to evaluate the permutation feature importance of 65,536 features. This also offers 2^{65536} choices to discard ports and $\sim 3.4 \cdot 10^{279}$ different ways to merge bins. In contrast, a bin size of 1024 partitions the UDP ports into 64 bins, reducing computational complexity during feature importance calculation while still offering 2^{64} choices to discard ports and $\sim 1.7 \cdot 10^{6}$ ways to merge bins. Clearly, the latter approach avoids the unnecessarily high computational and combinatorial complexity while offering high flexibility.

Loss of important information. A key consideration in step 4 is that the permutation feature importance assesses the importance of each feature *independently*. Only the values of a single feature are permuted when performing inference on a permuted dataset. In contrast, achieving a low memory footprint requires the removal of *multiple* features from an initially large feature set. The adverse impact on model performance incurred by removing multiple features can be significantly higher than indicated by their permutation feature importance. For this, the $f_1, \ldots, f_{64}$ that partition the TCP and UDP port range into 64 bins of equal size. Individually, many of these features can have low feature importance. However, removing all features of low importance also removes the information on the total number of packets in an aggregate which is implicitly given by the sum $s = f_1 + \ldots + f_{64}$. When additional features (besides port bins) are removed, a model may lose the ability to estimate the total number of packets. Without this point of reference, it can no longer determine the portion of attack traffic contained in an aggregate.

Instead of simply discarding features, features of lower importance can be successively merged into one or more bins. For example, port numbers below 1024 can be represented with a dedicated bin, while higher port numbers are summarized in a common bin. This distinguishes between system ports below 1024, which are often associated with services leveraged as reflectors in amplification attacks, and other ports that do not have a direct association with DDoS attack vectors. Similarly, the explicit distinction of some transport protocols can be avoided, and their occurrence can be summarized in a single bin. For example, the occurrence of the TCP and UDP protocols may be counted in individual bins, while the frequency at which other

protocols are used is counted by a common bin. This removes the necessity to monitor the occurrence of every transport protocol.

Exploiting redundancy. In contrast to avoiding the loss of important information, redundancy among features can be exploited to reduce the number of bins. As outlined above, the total number of packets is given as the sum over all port bins. This information is also provided through the total protocol count. Similarly, the sum over all frame size bins yields the total traffic volume in bytes. Consequently, the total number of packets and bytes can be redundant and removed from the feature set, even if they have high permutation feature importance. The corresponding sums can be recalculated by a neural net as long as complementary features are available.

4.5 Evaluation

The evaluation of the outlined approach to conduct Traffic Purity Estimation focuses on three key questions:

(1) To determine if low traffic purity estimation errors can be obtained from aggregated monitoring information through the training of neural networks when using a wide range of features.

(2) To determine if low traffic purity estimation errors can be maintained when reducing the number of features.

(3) To determine if models trained with specific feature selections generalize to traffic scenarios with different characteristics.

The evaluation is conducted through multiple experiments that begin with model training on a (larger) number of features followed by an evaluation of feature importance. Features of lesser importance are then repeatedly eliminated or their bins are merged according to the considerations in Section 4.4.2 over multiple experiments.

The evaluation uses two traffic scenarios described in Section 4.5.1 for model training and testing under different conditions. Detailed information on model training (including training parameters) is provided in the following section. Based on this, Section 4.5.3 evaluates model performance when an extensive feature set is used. This serves to assess the possibility to conduct Traffic Purity Estimation with neural

Attribute	Description
$\mathcal{P}_{\text{timestamp}}$	A floating-point number indicating the time (in seconds) at which a packet is received
$\mathcal{P}_{\text{size}}$	The size of a packet's ethernet frame (in bytes)
$\mathcal{P}_{\text{src\_addr}}$	The IP source address specified in an IP header
$\mathcal{P}_{\text{protocol}}$	The transport protocol number specified in an IP header
$\mathcal{P}_{\text{src\_port}}$	The source port specified in a TCP or UDP header

Table 4.1: Attributes of simulated packets in traffic scenarios.

networks and provides insight into feature importance. Finally, Section 4.5.4 compares the performance of models that use a reduced feature set and their ability to generalize to previously unseen traffic scenarios.

4.5.1 Traffic Scenarios

Traffic scenarios serve to simulate DDoS attack traffic and legitimate traffic. For this, traffic is either taken from authentic traffic traces or synthesized through random sampling. A scenario combines multiple traffic classes to model characteristics of volumetric DDoS attacks on legitimate systems, including the origin of traffic, traffic volume, and the time-dynamic behavior of traffic among other characteristics. Employing random sampling serves to compensate for limited availability of attack traffic datasets. It also allows for the creation of diverse traffic scenarios based on known attack vector characteristics.

A traffic scenario represents a packet as a five-tuple $\mathcal{P}$ consisting of the attributes listed in Table 4.1:

$$\mathcal{P} = \left(\mathcal{P}_{\text{timestamp}}, \mathcal{P}_{\text{size}}, \mathcal{P}_{\text{src\_addr}}, \mathcal{P}_{\text{protocol}}, \mathcal{P}_{\text{src\_port}}\right)$$

The five packet attributes correspond directly to the information monitored by the HHH algorithms described in Section 4.3.2. The timestamps $\mathcal{P}_{\text{timestamp}}$ and IP source addresses $\mathcal{P}_{\text{src\_addr}}$ of packets define their spatio-temporal distribution (over time and over the IP address space). These two attributes determine the number of packets received from a subnet in any given time frame. The inclusion of the frame size $\mathcal{P}_{\text{size}}$

also indicates the total traffic volume in bytes. In combination, these attributes fully describe the volumetric characteristics of traffic in a scenario. The transport protocol $\mathcal{P}_{\text{protocol}}$ and the source port $\mathcal{P}_{\text{src\_port}}$ represent information that can be monitored with HHH algorithms and may indicate the presence of DDoS attack vectors (e.g., DNS amplification attacks using UDP and source port 53).

To construct challenging scenarios, multiple traffic classes (such as legitimate traffic and different DDoS attack vectors) can be combined in a single scenario. This is done by representing traffic classes as streams, which generate a sequence of packets that can be iterated in ascending order of packet timestamps. By choosing different seeds when performing random sampling, multiple different packet sequences can be generated from a single scenario definition. Additional randomization can be performed through random transformations, which apply to all packets in a stream.

For this evaluation, two distinct traffic scenarios A and B are defined below. The scenarios differ in the legitimate traffic and the modeled attack vectors. Scenario A serves to evaluate the traffic purity estimation errors that are achieved be neural networks and the impact of feature reduction. Scenario B is used to evaluate if trained models generalize to traffic with different characteristics.

Streams and flows. A traffic scenario is logically structured into streams and flows. Streams either replay traffic from a trace directly or they define the characteristics of synthesized traffic. These characteristics are given as fixed values or as probability distributions. When synthesizing traffic, streams are further subdivided into flows. While a stream specifies the common characteristics of all flows, individual flows logically group packets originating from a single IP source address. This structure enables the consistent modeling of the behavior of individual traffic sources. For example, the exact number of packets sent by sources can be controlled as well as the duration of their activity.

- **Streams**. A stream generates the packets of a single traffic class. This is either a replay of packets from a traffic trace or a sequence of synthesized packets with common characteristics. Namely, the following two stream types are used:

  - Replay stream. A replay stream directly represents packets from a traffic trace. It iterates over the packets in a trace and converts them to five-tuples by extracting the attributes listed in Table 4.1 from each packet.

Packets in a trace are assumed to have increasing timestamps $\mathcal{P}_{\text{timestamp}}$ to reflect the correct order in which they are received.[2]

– Synthethic stream. A synthetic stream uses probabilistic sampling to generate a sequence of packets. This is done in two steps:

(1) A synthetic stream defines an IP source address distribution. This is used to sample a specified number of IP source addresses $\mathcal{P}_{\text{src\_addr}}$.

(2) For each sampled address, a stream generates a flow through random sampling (see below). This determines the remaining packet attributes (besides $\mathcal{P}_{\text{src\_addr}}$). A stream either specifies fixed values for the remaining attributes or common probability distributions from which they are sampled.

(3) The packets of all flows are sorted by their timestamps to ensure iteration over the sequence of packets progresses in the correct order.

Using common distributions for flows allows it to describe characteristics of DDoS attack vectors through probabilistic modeling of a stream.

- **Flows**. A flow consists of packets with identical (sampled) IP source addresses. The packets in a flow are generated as follows:

(1) A stream specifies distributions for the start time and the duration of a flow. Sampling from these distributions determines the flow's time frame.

(2) A stream specifies a fixed number of packets contained in each flow. Sampling from a uniform distribution over the flow's time frame determines the timestamp $\mathcal{P}_{\text{timestamp}}$ of each packet.

(3) The remaining attributes $\mathcal{P}_{\text{size}}$, $\mathcal{P}_{\text{protocol}}$, and $\mathcal{P}_{\text{src\_port}}$ are set to fixed values or sampled from distributions specified by the flow's stream.

The total traffic volume of a synthetic stream can be controlled by adjusting the number of flows in a stream, the number of packets in a flow, and the Ethernet frame size of packets. The traffic volume of a flow can then be distributed over time by defining the distributions for the flow start time and flow duration. Similarly, the traffic can be distributed across smaller or larger address space regions by adjusting the source address distribution. In combination, this enables the modeling of high-

[2]This property is not necessarily ensured by common trace formats (such as pcap-files. However, a correct ordering can be achieved through preprocessing with commonly available tools.)

Algorithm 2: Iteration over multiple streams in order of packet timestamps

Input : Empty priority queue Q

```
1  Function initialize_priority_queue(set of streams 𝒮):
       ▷ Initialization of Q enqueues first packets of all streams
2      foreach stream S ∈ 𝒮 do
3          if not S.empty() then                    ▷ Avoid enqueuing empty streams
4              𝒫 ← S.pop();                         ▷ Retrieve first packet 𝒫 from stream S
5              Q.enqueue(𝒫_timestamp, (𝒫, S));      ▷ Enqueue 𝒫 and S into Q and
                                                      preserve order of timestamps
6  Function get_next_packet():
       ▷ Retrieves next packet with lowest timestamp
7      if Q.empty() then
8          return null ;                            ▷ Return no packet if Q is empty
9      (𝒫, S) ← Q.pop();                            ▷ Retrieve next packet and its stream
10     if not S.empty() then
11         𝒫′ ← S.pop();
12         Q.enqueue(𝒫′_timestamp, (𝒫′, S));        ▷ Enqueue next packet 𝒫′ of S and S
13     return 𝒫 ;                                   ▷ Return next packet
```

and low-intensity attack traffic representing short bursts or long-lasting attacks as well as widely spread attacks or attacks originating from smaller address space regions. Controlling the number of flows in a stream also determines if traffic is sparsely or densely distributed over the respective address space region.

Combining multiple streams. Generating a traffic scenario from multiple streams is done by iterating over all packets of all streams in ascending order of timestamps. This yields a single sequence of packets in the order in which they are received. As previously outlined, iteration over packets in a single stream already respects the correct order of packets. All that remains is to ensure that the iteration occurs in the correct order when processing packets from different streams. This is achieved by employing a priority queue, which determines the packet with lowest timestamp across all streams, and by advancing the iteration of individual streams as needed.

Algorithm 2 outlines two functions that provide the required functionality. The function initialize_priority_queue enqueues the first packets (with the lowest timestamp) of all streams into a priority queue. When enqueuing packets in a priority queue (in Line 5), the queue maintains an ascending order of timestamps between the

packets. The function get_next_packet then fetches the packet with the overall lowest timestamp (across all streams) by retrieving the first packet stored in the priority queue (in Line 9 and Line 13). For each packet, the queue also stores a reference to the packet's corresponding stream (in Line 5). Through this, a packet retrieved from the priority queue can be replaced with the next packet from its respective stream (see Line 12). Multiple packets can be retrieved from the same stream in direct succession if this stream continuously yields the lowest timestamps. This occurs, for example, if the stream is the only one that is active within a time frame. The remaining code of Algorithm 2 ensures that empty streams are no longer processed and that no packet is returned once all streams are empty. As a result, repeated invocation of the get_next_packet function yields all packets in ascending timestamp order until no more packets are left in any stream of a traffic scenario.

Stream characteristics. Packet attributes are either specified as fixed values or sampled from probability distributions. The collection of fixed values and distributions denotes the stream characteristics, which define the common characteristics of the traffic that a stream represents. For example, a stream can define that its packets use the UDP protocol, originate from port 53, and have a specific Ethernet frame size distribution (along with other characteristics) to model DNS amplification attacks. To enable the modeling of a broader range of attacks, a stream defines several more characteristics, either through fixed values or by using one of the following probability distributions:

- The uniform distribution $\mathcal{U}[x, y)$ over the interval $[x, y)$.
- The normal distribution $\mathcal{N}(\mu, \sigma)$ with mean μ and standard deviation σ.
- The Weibull distribution $\mathcal{W}(a, b)$ with shape a and scale b.

The following provides an overview of all characteristics that are defined by a stream:

- **IP source address distribution**. IP source addresses of flows are randomly sampled from the IP source address distribution. Streams use one of two distributions throughout this evaluation:
  - The uniform distribution $\mathcal{U}\left[0, 2^{32} - 1\right)$, which distributes IP source addresses uniformly across the entire IPv4 address space. This serves to model a distribution of source addresses that does not exhibit a concentration in any specific address space region.

  - A normal distribution $\mathcal{N}(\mu, \sigma)$ with fixed $\mu = 2^{31}$ and randomized standard deviation σ. The mean $\mu = 2^{31}$ concentrates the distribution of source addresses on a subregion of the address space (the middle of the IPv4 address space). This serves to model address distributions that focus on specific address space regions. The specific region can later be changed (see the random transformations discussed below). The standard deviation σ is randomly sampled from a uniform distribution when generating a new traffic scenario. This causes addresses to have wider or narrower spread across the IP address space to increase the diversity of traffic scenarios.

- **Number of IP source addresses**. The number of IP source addresses determines how many addresses are sampled from the IP source address distribution. This also determines how many flows are generated for each stream. Using a higher or lower number of IP source addresses models traffic sources that are more densely or sparsely distributed across the address space.

- **Flow duration**. The flow duration determines how long a flow is active during a traffic scenario. The flow duration is measured in seconds and sampled from a Weibull distribution with shape parameter $a = 2$. This produces a long-tailed distribution that increases the variance of the number of flows that are simultaneously active compared to using flows of fixed duration. The increased variance serves to make the traffic of a stream more dynamic over time. Using a Weibull distribution allows it to control maximum flow durations independently of the distributions shape up to a certain probability. For this, the shape parameter b of the Weibull distribution is calculated as $b = x/\mathsf{Q}_{99}$, where x denotes the desired maximum flow duration and Q_{99} the 99-percent quantile of the Weibull distribution $\mathcal{W}(2, 1)$. This causes flows to have a maximum duration of x with 99 percent probability. In contrast, uniform and normal distributions would not allow to control maximum flow durations independently of the distribution's shape, since their probability density functions are symmetric.

- **Maximum flow start time**. The maximum flow start time constitutes a window for lower bounds on packet timestamps. Specifically, for each flow a random lower bound (in seconds) is sampled from a uniform distribution $\mathcal{U}[0, x)$ and added to the timestamps of every packet in a flow. In effect, higher values of x spread flows over longer time frames while lower x-values cause the flows to start close together in time. Adjusting the maximum flow start time serves to

model short-term or long-lasting activity of traffic sources in a stream (e.g., to model attack bursts and persistent attacks).

- **Packets per source**. This quantity defines how many packets are generated for each flow. These packets are uniformly distributed across a flow's time frame and use identical IP source addresses. The number of packets per source is specified as a fixed value, which allows the traffic volume of flows and streams to be adjusted in terms of packet and byte counts.

- **Frame size**. The frame size defines fixed values or probability distributions that determine a packet's $\mathcal{P}_{\text{size}}$ attribute. Sampling of specifying $\mathcal{P}_{\text{size}}$ attributes in this manner enables the modeling of common DDoS attack vector characteristics. For example, the frame size in DNS amplification attacks follows a distribution that is highly skewed towards the MTU of a path. Attacks that do not exhibit any special characteristics can be modeled by sampling from a uniform distribution that covers the permissible range of frame sizes.

- **Transport protocol**. The transport protocol is provided as a fixed value, which indicates the use of the UDP, TCP, or other protocols such as ICMP. The value used for the $\mathcal{P}_{\text{protocol}}$ attributed matches that of an IP header. The transport protocol is chosen according to the corresponding DDoS attack vector when modeling attack traffic.

- **Source port**. Similarly to frame sizes, the source port is set to a specific value or randomly sampled from a probability distribution. When the traffic of a DDoS attack vector typically originates from a specific port (such as in a DNS amplification attack) the corresponding value is chosen. If the source ports of the attack traffic are unknown, packets are sampled from a uniform distribution to avoid introducing arbitrary biases.

Random transformations. A random transformation changes one attribute of all packets of a single stream. This occurs in a probabilistic manner to increase scenario diversity. More precisely, a random value is chosen whenever packet generation of a scenario starts. That same value is afterwards used to modify all packets of a stream (without further randomization throughout the scenario). Through this, packets generated by a stream can be altered as a whole while preserving most traffic characteristics. This evaluation uses two random transformations to diversify traffic distributions over time and over the address space:

- **Random address space roll**. A random address space roll applies the following transformation to the IPv4 addresses $\mathcal{P}_{\text{src\_addr}}$ of all packets in a stream:

$$\mathcal{P}_{\text{src\_addr}} \mapsto \mathcal{P}_{\text{src\_addr}} + r \,(\text{mod}\, 2^{32}) \quad \text{with} \quad r \sim \mathcal{U}\left[0, 2^{32} - 1\right).$$

  Addition of the random offset r modulo 2^{32} effectively rotates all addresses through the address space. This serves primarily as a data augmentation technique to compensate for the limited availability of traces and to increase the diversity of authentic traffic. A random address space roll applied to a replay stream produces different traffic patterns over the course of multiple scenarios. This reduces the risk of repeatedly overlapping traffic of different streams in the same address space regions. In addition, a random address space roll can be used instead of explicitly randomizing the mean μ of a normal distribution that is used to define an IP source address distribution of a synthetic stream.[3]

- **Random time shift**. A random time shift adds a random value r from an interval $[x, y) \subset \mathbb{R}$ to all timestamps $\mathcal{P}_{\text{timestamp}}$ of all packets in a stream:

$$\mathcal{P}_{\text{timestamp}} \mapsto \mathcal{P}_{\text{timestamp}} + r \quad \text{with} \quad r \sim \mathcal{U}\left[0, 2^{32} - 1\right).$$

  Applying a random time shift to a stream enforces a lower bound on flow start times that falls into the chosen interval $[x, y)$. This is different from the maximum flow start time, which governs the distribution of flow start times within a stream, but does not enforce a lower bound on timestamps of a stream. The purpose of a random time shift is to (probabilistically) distribute the activity of streams over specific time frames when generating multiple traffic scenarios.

Evaluation scenarios. This evaluation uses two different scenarios A and B to evaluate model performance, the impact of feature selections, and the ability of a model to generalize. Scenario A is used to generate the training, validation, and test datasets $\mathsf{D}_{\text{train}}$, D_{val}, D_{test} used for model training, feature importance calculations, and the evaluation of traffic purity estimation errors (with different aggregate feature selections). Scenario B is used exclusively to generate test data for evaluating feature selections against different traffic characteristics. This assesses if a trained model can generalize to previously unseen scenarios. Table 4.2 summarizes the traffic characteristics and distribution parameters of the two scenarios A and B.

[3] Explicit randomization of the mean μ can result in a normal distribution having a mean close to the borders of the address space. Sampling from such a distribution would result in many outliers outside the address space region, which would have the drawback of requiring special handling. The modulo operation of the random address space roll eliminates the need to handle outliers.

Scenario	Stream	Stream characteristics							
		IP source addr. distribution[1]	#IP source addresses	Flow duration[2] $[s]$	Max. flow start time $[s]$	Packets per source	Frame size [bytes]	Transport protocol	Source port
A	DNS	$\mathcal{U}\left[0, 2^{32}-1\right)$	250	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	1000	see text	UDP	53
	NTP	$\mathcal{U}\left[0, 2^{32}-1\right)$	250	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	1000	$\mathcal{N}(481.1, 10)$	UDP	123
	OVPN	$\mathcal{N}\left(2^{31}, \sigma\right)$	5000	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	180	$\mathcal{N}(64.5, 0.3)$	UDP	1194
	TCP_1	$\mathcal{N}\left(2^{31}, \sigma\right)$	5000	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	180	$\mathcal{U}[60, 748)$	TCP	$\mathcal{U}[49152, 65535]$
	TCP_2	$\mathcal{N}\left(2^{31}, \sigma\right)$	5000	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	180	$\mathcal{U}[748, 1496)$	TCP	$\mathcal{U}[1024, 49151]$
	UDP_1	$\mathcal{N}\left(2^{31}, \sigma\right)$	10000	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	180	$\mathcal{U}[60, 1496)$	UDP	$\mathcal{U}[1, 65535]$
	$LEGIT_1$	MAWI traffic to destination IP address `202.139.222.241`							
B	SSDP	$\mathcal{U}\left[0, 2^{32}-1\right)$	250	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	1000	$\mathcal{N}(347, 9.1)$	UDP	1900
	NTP	$\mathcal{U}\left[0, 2^{32}-1\right)$	250	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	1000	$\mathcal{N}(481.1, 10)$	UDP	123
	OVPN	$\mathcal{N}\left(2^{31}, \sigma\right)$	5000	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	180	$\mathcal{N}(64.5, 0.3)$	UDP	1194
	TCP_3	$\mathcal{N}\left(2^{31}, \sigma\right)$	10000	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	180	$\mathcal{U}[512, 1496)$	TCP	$\mathcal{U}[1024, 49151]$
	TCP_4	$\mathcal{N}\left(2^{31}, \sigma\right)$	5000	$\mathcal{W}\left(2, 45/Q_{99}\right)$	$\mathcal{U}[0, 135)$	180	$\mathcal{U}[60, 1496)$	TCP	$\mathcal{U}[49152, 65535]$
	$LEGIT_2$	MAWI traffic to destination IP address `202.58.173.226`							

[1]The variable σ denotes a randomized standard deviation $\sigma \sim \mathcal{U}[0.02, 0.22) \cdot 2^{32}$.

[2]Dividing the Weibull distribution's scale parameter by Q_{99} results in a maximum duration of 45 seconds with 99% probability.

Table 4.2: Traffic characteristics of the two traffic scenarios A and B used in the evaluation. Different traffic characteristics between the two scenarios are highlighted in blue in scenario B.

All synthetic streams apply a random time shift transformation that uses the uniform distribution $\mathcal{U}[0,720)$. The maximum flow start time distribution is the uniform distribution $\mathcal{U}[0,135)$. Flow durations are sampled from a Weibull distribution $\mathcal{W}\left(2,45/\mathsf{Q}_{99}\right)$, where Q_{99} denotes the distribution's 99 %-quantile. In combination, these choices result in all attack vectors being active across the entire scenario duration (of 900 seconds) for a short amount of time. This causes a high diversity in the attack traffic composition of traffic aggregates, which makes it more challenging to achieve low traffic purity estimation errors. In contrast, restricting an attack vector to a certain time frame would make Traffic Purity Estimation in the remainder of the scenario easier due to lower traffic diversity. Furthermore, all synthetic streams that use a normal distribution to sample IP source addresses apply a random address space roll. This causes different overlaps of streams in the entire IPv4 address space when generating multiple traffic scenarios, thus further increasing scenario diversity.

Scenario A uses seven different streams to overlay legitimate traffic with attack traffic. A replay stream generates the legitimate traffic from a MAWI trace (described below), while six synthetic streams generate attack traffic that models different DDoS attack vectors. The attack traffic uses DNS, NTP, and OpenVPN amplification attacks, as well as TCP- and UDP-based direct-path attacks. The streams set the transport protocol and the source ports of the DNS, NTP, and OpenVPN attacks to fixed values that match the typical characteristics of these attack vectors. The following describes the characteristics that differ between the synthetic streams. When available, frame size distributions are chosen to match empirical observations reported in [KDH21].

- **DNS**. This stream models a DNS amplification attack. The stream uses a lower number of IP source addresses than the OpenVPN and TCP-based attack traffic streams. In contrast, the number of packets per source and the frame size are higher. Consequently, the DNS stream models high-intensity flows originating from few traffic sources. Furthermore, the low number of randomly sampled IP source addresses spreads across the entire IPv4 address space due to the uniform IP source address distribution. This results in a sparse distribution of attack traffic sources. The DNS stream uses a specific frame size distribution to match empiric observations reported in [KDH21], which indicates an average packet size of 1474 and a standard deviation of 59. These statistics can be approximated by sampling frame sizes from the (heavy-tailed) Weibull distribution $\mathcal{W}(185,1481.97)$ and restricting frame sizes to the interval $[0,1496]$ via modulo arithmetic.

- **NTP**. This stream models an NTP amplification attack. The key difference to the DNS stream is the frame size distribution $\mathcal{N}(481.1, 10)$, which generates packets with significantly fewer bytes. Through this, the stream yields a lower attack traffic volume while using the same number of IP source addresses and packets per flow. Due to the lower traffic volume, an aggregation to shorter prefixes and the inclusion of traffic from other streams in aggregates can be expected before reaching an HHH threshold. This results in a more challenging Traffic Purity Estimation compared to the DNS amplification attack.

- **OVPN**. This stream models an OpenVPN amplification attack. In comparison to the previous two streams, the OVPN stream distributes attack traffic over a high number of flows and concentrates attack traffic on smaller address space regions by employing a normal distribution for the IP source addresses. Each flow sends a lower attack traffic volume due to a reduced number of packets per source and lower frame sizes (which use the normal distribution $\mathcal{N}(64.5, 0.3)$ to match observations in [KDH21]). Again, aggregation to shorter HHH prefixes can be expected due to lower per-flow traffic volume, resulting in aggregates containing more diverse traffic.

- **TCP$_1$ and TCP$_2$**. These streams model direct-path attacks using TCP traffic. The stream TCP$_1$ restricts source ports to the range of dynamic ports from 49,152 to 65,535, which is typically used by client applications. In contrast, TCP$_2$ uses the range 1024 to 49,151 of registered ports and allows greater randomization. In comparison to the amplification attacks, source port randomization makes it more difficult to use ports as indicators of attack traffic. Both flows use a low number of packets per source and a high number of source addresses. However, stream TCP$_1$ and TCP$_2$ sample frame sizes uniformly from distinct partitions of the range 0 to 1496. Depending on the temporal activity of streams TCP$_1$ and TCP$_2$, aggregates contain different mixtures of frame sizes and TCP source ports. This is further emphasized by the random address space roll applied to the source addresses sampled from a normal distribution, as it causes different overlaps of the streams in the address space.

- **UDP$_1$**. This stream models a direct-path attack based on UDP traffic. The additional inclusion of UDP$_1$ serves to further diversify the mixtures of traffic in aggregates. Similar to the TCP streams, UDP$_1$ randomizes source ports and frame sizes, but uses the entire port range and frame size range. Furthermore, it models a higher-intensity attack than TCP$_1$ and TCP$_2$ by increasing the number of IP source addresses. This serves to increase the variance of aggregate features.

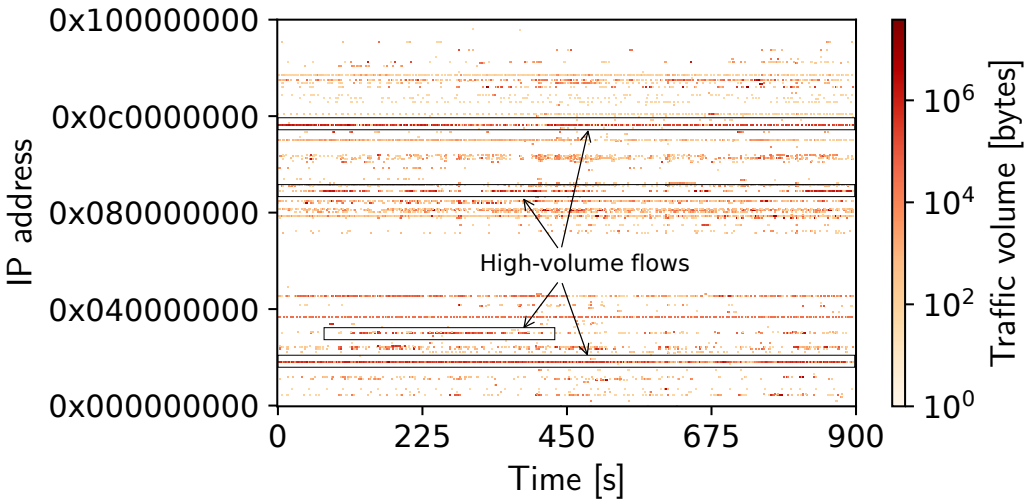


Figure 4.5: Traffic volume distribution of legitimate MAWI traffic used in scenario A. The traffic is sent to IP address `202.139.222.241`.

- **LEGIT**$_1$. This stream replays authentic traffic from the MAWI traffic archive, which provides traffic traces from a Japanese backbone provider. The traffic was captured on September 1st, 2019 between 2:00pm and 2:15pm (at sample point F of the monitoring infrastructure). To avoid confusion between ingress and egress traffic, the trace includes only packets having MAC destination address `88:e0:f3:7a:66:f0`, representing one out of two communicating routers. To create a scenario that differs significantly in the legitimate traffic characteristics, the stream is further reduced to packets sent to IP address `202.139.222.241`. Figure 4.5 visualizes the traffic volume distribution over time and over the address space. The legitimate traffic itself contains several high-volume flows that can directly result in HHH prefixes.

Figure 4.6 provides an example of attack traffic in scenario A. The attack streams overlap in time and in the IP address space. A different randomization changes the distribution and timing of the attack traffic, resulting in diverse aggregate features.

Scenario B retains the NTP and OpenVPN reflection attacks from scenario A, but replaces the DNS amplification with an SSDP amplification attack. In addition, two TCP-based streams with different characteristics are used, and the UDP-based stream is omitted. Since models only use data from scenario A for training, any deviations in scenario B constitute previously unseen traffic characteristics.

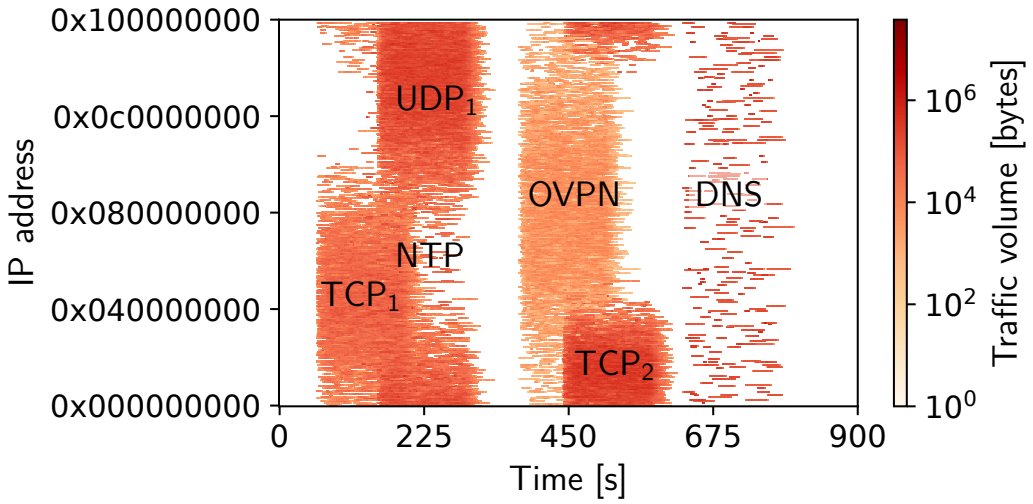


Figure 4.6: Attack traffic volume distribution over time and over the IP address space. Six streams model different attack vectors based on TCP, UDP, DNS, NTP, and OpenVPN (OVPN).

- **SSDP**. This stream replaces the DNS amplification attack and models an SSDP amplification attack instead. The key difference is the frame size distribution that uses the normal distribution $\mathcal{N}(347, 9.1)$. This results in overall lower traffic volume and introduces a new distribution of frame sizes. Additionally, the UDP traffic now originates from a different source port, which serves to evaluate the importance of source ports for Traffic Purity Estimation.

- **TCP_3 and TCP_4**. The two TCP streams model direct-path attacks with different characteristics than the TCP streams used in scenario A. Specifically, TCP_3 and TCP_4 sample from different combinations of frame size ranges and source port ranges. Furthermore, the number of flows is increased for stream TCP_3. This can result in aggregates with different volumes than before.

- **LEGIT_2**. This stream uses different legitimate traffic from the previously described MAWI trace than the stream LEGIT_1. Specifically, the legitimate stream replays traffic sent to IP address `203.78.253.44`, which has a different spatio-temporal volume distribution (cf. Figure 4.7). Consequently, the mixtures of aggregates can be expected to change, providing previously unseen samples to a model performing Traffic Purity Estimation.

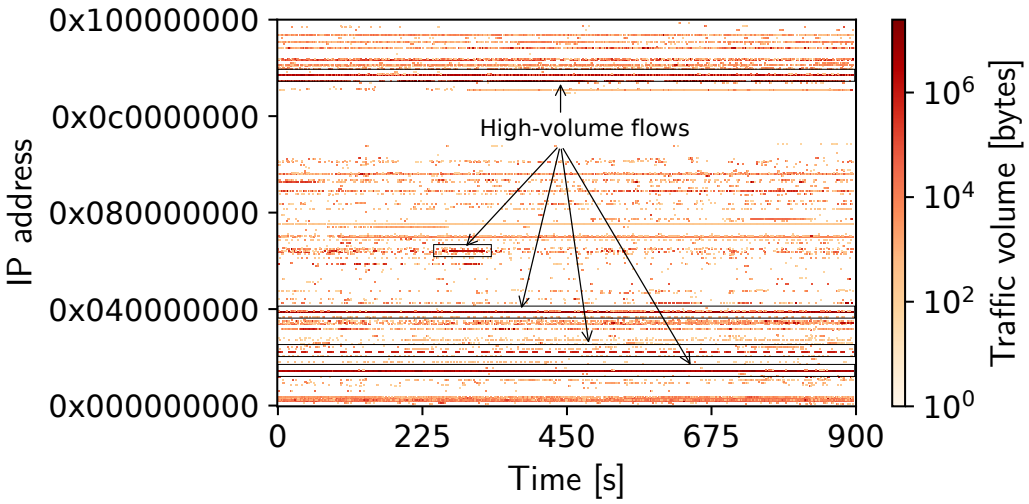


Figure 4.7: Traffic volume distribution of legitimate MAWI traffic used in scenario B. The traffic is sent to IP address `202.58.173.226`. The traffic volume distribution differs significantly from scenario A.

4.5.2 Evaluation Setup

To ensure comparability between models that use different feature selections, model training uses the same training process and parameters for all models. The training uses stochastic gradient descent and follows the process described in Section 4.3.4. The following describes the generation of datasets, the evaluation system's specifications, and specific training parameters used for model training.

Dataset generation. Training and inference are conducted on the traffic scenarios described in Section 4.5. The (randomized) traffic is monitored by an HHH algorithm that uses a selection of aggregate features. To generate the training, validation, and test datasets D_{train}, D_{val}, and D_{test}, the HHH algorithm is queried repeatedly at a monitoring interval duration of one second. This generates the aggregate features and ground truth information on the traffic purity for prefixes of different length. During dataset generation, no sampling is applied, as the training of Traffic Purity Estimation models takes place in advance of an active mitigation. The data is collected over ten randomized instances of scenarios A and B.

Evaluation system specifications. The training and evaluation take place on a bare-metal-server with hardware and software specifications provided in Table 4.3. The HHH monitoring is implemented in the C++ programming language (using a

Category		Specification
Hardware	CPU	AMD EPYC 7402P (3.35 GHz, 24 cores, 48 threads)
	RAM	128 GB DDR4-3200 ECC
	GPU	2x Nvidia A40 (96 GB GDDR6 vRAM)
Software	Ubuntu OS	Version 20.04.6 LTS
	Python	Version 3.8.10
	TensorFlow	Versions 2.10.0
	g++	Version 9.4.0 (configured to C++17 standard)

Table 4.3: Specifications of the system used to evaluate Traffic Purity Estimation.

g++ compiler that is configured to the C++17 standard). It extends the CocoSketch algorithm [Zha+21] to collect aggregate features as described in Section 4.3.2.

Training and evaluation of models uses the TensorFlow framework [Mar+15] for deep learning. The framework provides configurable reference implementations for neural network architectures, training algorithms, and optimizers. With TensorFlow, neural networks can leverage GPU acceleration for training and inference. This is particularly beneficial to calculate permutation feature importance, as it requires multiple inferences on trained neural networks. The process of permutation feature importance that is described in Section 4.4.1 is realized in Python.

Training parameters. Table 4.4 lists parameters choices used in the training of Traffic Purity Estimation models. The training uses an initial learning rate $\lambda = 10^{-4}$, which is controlled by an adam optimizer. The optimizer serves to improve model convergence. The patience parameter controls when the training process is stopped to reduce the risk of overfitting (see Section 4.3.4). The chosen value of 300 allows short term increases in the validation loss to avoid premature termination of model training.

Layer normalization serves to improve model convergence and to achieve lower estimation errors (cf. Section 2.4.2). The number of fully-connected layers and the number of neurons per layer determine how many weights a neural network can use to estimate the traffic purity. The values are chosen high enough to give a neural network sufficiently many weights to be able to achieve low estimation errors. On the other hand, they are chosen to be low enough to restrict the neural network in order to reduce the risk of overfitting on training data. Gradients are calculated based on

Parameter	Value
Learning rate	10^{-4}
Optimizer	adam
Patience	300
Normalization	Layer normalization
Number of fully-connected layers	8
Neurons per fully-connected layer	128 or 256
Loss function	Huber loss
Batch size	32,768
Train/validation/test-split	0.7/0.2/0.1

Table 4.4: Parameters used to train all models.

Huber loss, to reduce the impact of high losses on weight updates. The gradients are averaged across a batch size of 32,768 before a backward pass is performed. This aggregation further reduces the risk of overfitting.

The train/validation/test-split determines the size of datasets $\mathsf{D}_{\mathsf{train}}$, $\mathsf{D}_{\mathsf{val}}$, and $\mathsf{D}_{\mathsf{test}}$ for scenario A. The chosen values follow best practices. For scenario B, the entire dataset is used as test dataset, since this scenario is used exclusively to evaluate model performance on previously unseen traffic.

4.5.3 Model Performance with Extensive Features

The first experiment uses a large number of features to assess whether neural networks can be trained to achieve low traffic purity estimation errors. Table 4.5 lists the full set of features $f_1, \ldots, f_{82}$ of the aggregate features $\mathsf{f}_{\mathsf{ALL}}$. The source port and frame size bins cover their corresponding value ranges with bins of equal size (which is a power of two to facilitate efficient bin computation). The last frame size bin also counts frame sizes beyond 1535 bytes to account for large frames that occur in the legitimate MAWI traffic. The source ports do not differentiate between UDP and TCP transport layer protocols to determine if a model can achieve high performance when this distinction is omitted. The bin sizes and the resulting high number of features are chosen to provide a reference for feature reductions in subsequent experiments.

Features	#Bins	Bin description
f_1	1	One bin representing the **prefix length** of its corresponding aggregate.
f_2	1	One bin representing the **packet count**.
f_3	1	One bin summarizing **traffic volume** in bytes.
f_4	1	One bin counting packets with the **TCP** protocol.
f_5	1	One bin counting packets with the **UDP** protocol.
f_6	1	One bin counting packets that use **other** protocols than TCP or UDP (as indicated by the IP protocol field).
$f_7, \ldots, f_{70}$	64	64 adjacent bins of equal size (1024 ports) counting used **source ports** in the range from 0 to 65,536. Bins do not distinguish between UDP and TCP ports. Packets without source port information are counted as having source port 0 (e.g., ICMP packets).
$f_{71}, \ldots, f_{82}$	12	12 adjacent bins of equal size (128 bytes) counting **frame sizes** in the range from 0 to 1535 bytes. The last bin also counts frame sizes beyond 1535 bytes.

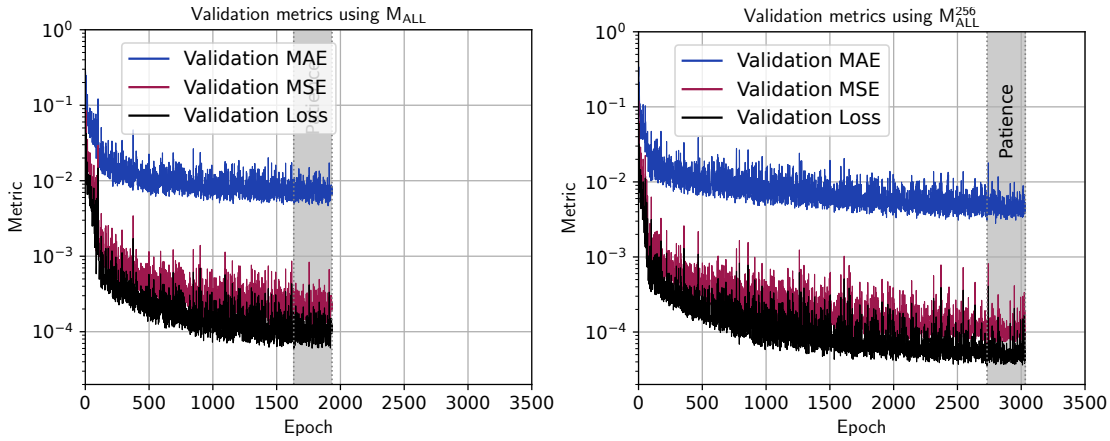
Table 4.5: The aggregate features $\mathsf{f}_{\mathsf{ALL}} = (f_1, \ldots, f_{82})$.

To decide on the number of neurons per layer, the following two models are trained using the neural network architecture described in Section 4.3.3:

- $\mathsf{M}_{\mathsf{ALL}}$: This model uses aggregate features $\mathsf{f}_{\mathsf{ALL}}$ and 128 neurons per layer.
- $\mathsf{M}^{256}_{\mathsf{ALL}}$: This model uses aggregate features $\mathsf{f}_{\mathsf{ALL}}$ and 256 neurons per layer.

Evaluating the two different models serves to assess the impact of the number of weights on generalization to unseen samples.

Model performance. Figure 4.8 shows the progression of the loss, MAE $\pi^{\mathsf{MAE}}_{\mathsf{val}}$, and MSE $\pi^{\mathsf{MSE}}_{\mathsf{val}}$ on the validation data $\mathsf{D}_{\mathsf{val}}$ of scenario A over the course of training. The training duration is determined by the patience of the training process and failure of a model to further reduce losses over multiple epochs. To assess the effect of the number of neurons on model performance, the figure shows the progression for both models $\mathsf{M}_{\mathsf{ALL}}$ (Figure 4.8a) and $\mathsf{M}^{256}_{\mathsf{ALL}}$ (Figure 4.8b). The validation losses of both models are subject to frequent increases. However, the best model is saved during training by using checkpoints. The MAE, the MSE, and the validation losses converge



(a) Model M_{ALL} **(b)** Model M^{256}_{ALL}

Figure 4.8: Progression of loss, MSE, and MAE on validation data during training over multiple epochs using aggregate features f_{ALL}. The grayed patience area indicates training epochs without further loss reductions.

over the course of model training. This indicates that both models are able to learn to estimate traffic purity from aggregate features.

Table 4.6 summarizes the final loss, MAE, and MSE at the end of training. In scenario A, the model M^{256}_{ALL} performs better than the model M_{ALL} except for the MSE on the test data. However, the MAE and MSE of model M_{ALL} are lower in scenario B, indicating better generalization of M_{ALL}. This suggests that a higher number of neurons per layer can increase the risk of overfitting to training data.

Distribution of absolute errors. In addition to the (purely scalar) MAE and MSE statistics, the distribution of absolute traffic purity estimation errors indicates whether a model tends to produce more larger or smaller errors. In general, it is preferable to avoid larger errors, as smaller errors can be partially compensated by requiring a slightly higher estimated traffic purity when selecting HHH prefixes (as discussed in Section 4.1.2). To enable this distinction for models M_{ALL} and M^{256}_{ALL}, Figure 4.9 shows the cumulative distribution function (CDF) of the traffic purity estimation error π_p^{err} over all HHH prefixes monitored in scenarios A and B. The CDF indicates the proportion of errors that are at most as high as the π_p^{err}-value along the x-axis. Table 4.7 also summarizes the numeric values for selected maximum values of π_p^{err}. Both models achieve low errors with high probability on the test data of scenario A, indicating their ability to identify aggregates with high attack traffic volume. For example, models M_{ALL} and M^{256}_{ALL} realize errors of no more than 2 % with

Scenario	Model	Validation loss	MAE		MSE	
			π^{MAE}_{val}	π^{MAE}_{test}	π^{MSE}_{val}	π^{MSE}_{test}
A	M_{ALL}	0.000060	0.0049	0.0037	0.000120	0.000483
	M^{256}_{ALL}	0.000036	0.0031	0.0036	0.000071	0.000544
B	M_{ALL}	N/A	N/A	0.0836	N/A	0.026211
	M^{256}_{ALL}	N/A	N/A	0.1098	N/A	0.044947

Table 4.6: Final loss, MAE, and MSE on validation and test data using aggregate features f_{ALL} in scenarios A and B. Since scenario B uses the entire dataset as test data, non-applicable values are marked with N/A. Best values are highlighted.

Scenario	Model	Proportion of absolute errors π^{err}_{p}			
		$\leq 1\,\%$	$\leq 2\,\%$	$\leq 10\,\%$	$\leq 20\,\%$
A	M_{ALL}	0.9358	0.9691	0.9954	0.9986
	M^{256}_{ALL}	0.9582	0.9764	0.9933	0.9979
B	M_{ALL}	0.6497	0.6714	0.7243	0.7850
	M^{256}_{ALL}	0.6542	0.6855	0.7225	0.7445

Table 4.7: Proportion of absolute traffic purity estimation errors below selected values in scenario A and B using models M_{ALL} and M^{256}_{ALL} with features f_{ALL}. Best values are highlighted.

a high probability of $\sim 97\,\%$. Errors above $10\,\%$ have a probability of $\sim 0.05\,\%$ to $\sim 0.07\,\%$ (for M_{ALL} and M^{256}_{ALL}) and errors of $20\,\%$ or higher occur only with a probability of $\sim 0.2\,\%$.

In comparison, Traffic Purity Estimation becomes less accurate in scenario B. This is shown by the lower CDFs of both models in Figure 4.9. Estimating the traffic purity in scenario B incurs errors of at least $10\,\%$ with higher probabilities of about $28\,\%$ for both models. This constitutes an increase of several orders of magnitude. In addition, errors now exceed $20\,\%$ with a probability of $21.5\,\%$ (M_{ALL}) and $\sim 25.6\,\%$ (M^{256}_{ALL}). If such high errors occur frequently, they can impair the ability to reliably identify aggregates that consist primarily of attack traffic. As a consequence, the ability of models to generalize to diverse traffic scenarios should be improved.

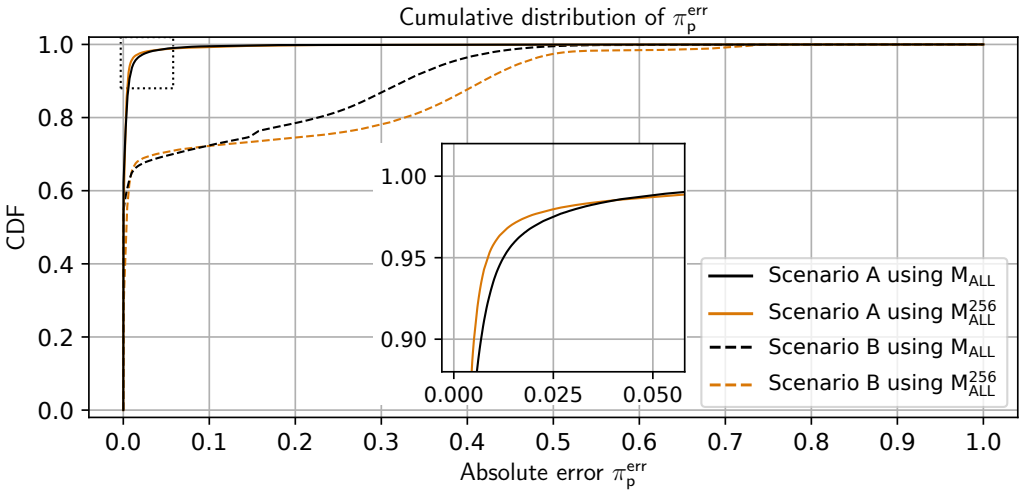


Figure 4.9: Cumulative distribution functions (CDF) of absolute traffic purity estimation errors in scenario A and B using models $\mathsf{M}_{\mathsf{ALL}}$ and $\mathsf{M}_{\mathsf{ALL}}^{256}$ with features $\mathsf{f}_{\mathsf{ALL}}$.

The fact that the performance of model $\mathsf{M}_{\mathsf{ALL}}$ is consistently better than that of model $\mathsf{M}_{\mathsf{ALL}}^{256}$ in scenario B indicates again that a lower number of neurons can facilitate generalization to previously unseen traffic characteristics. Hence, all models used in further evaluations use 128 neurons per layer in their architecture.

Feature importance. The trained model $\mathsf{M}_{\mathsf{ALL}}$ is used to evaluate which of the 82 features of $\mathsf{f}_{\mathsf{ALL}}$ have the highest importance based on the method described in Section 4.4.1. The MAE $\pi_{\mathsf{test}}^{\mathsf{MAE}}$ that is achieved on the HHH prefixes $\mathcal{H}$ of the original (non-permuted) dataset $\mathsf{D}_{\mathsf{test}}$ of scenario A constitutes the baseline for comparisons of model performance. For each feature f_i $(i = 1, \ldots, 82)$ the feature importance $\mathfrak{I}_{f_i}$ is calculated according to Equation 4.7 using test datasets $\mathsf{D}_{\mathsf{test}}^{(\Pi_i)}$ with individually permuted feature values.

Figure 4.10 shows the 25 features with highest importance $\mathfrak{I}$ to the model $\mathsf{M}_{\mathsf{ALL}}$ in descending order of importance. The remaining 57 features consistently have a lower importance $\mathfrak{I} \leq 0.0018$. The importance of all features is shown in Figure A.1 and in Figure A.2 in Appendix A.2.4. The distribution of the feature importance highlights several characteristics of the trained model:

- Each of the seven features $f_3, f_5, f_4, f_8, f_7, f_2$, and f_{82} with highest importance has at least twice the importance of the remaining 75 features. This shows that the trained model focuses mainly on this smaller number of features. The remaining features include the prefix length, the bin for other protocols, as

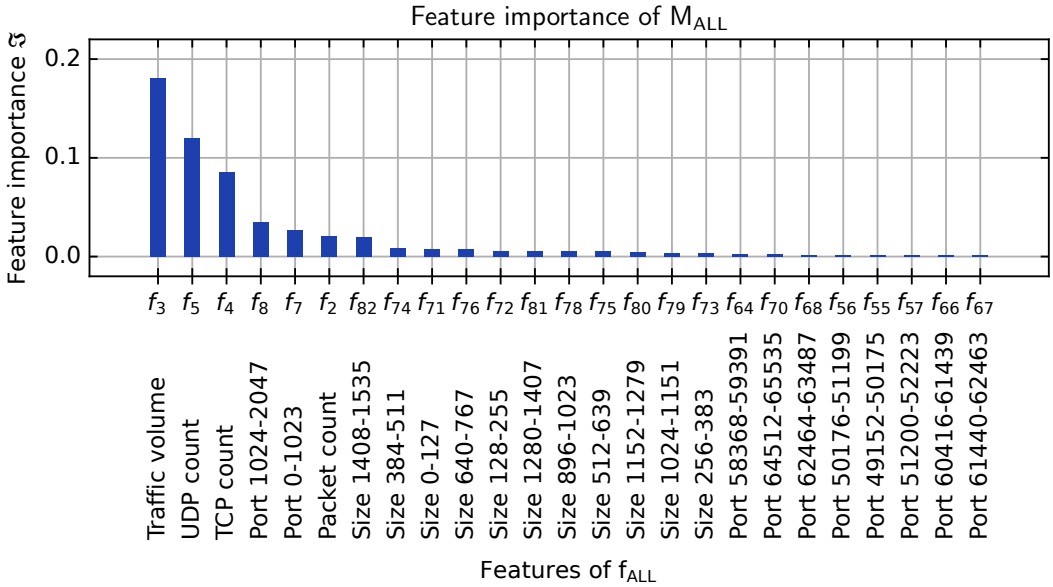


Figure 4.10: The 25 features of f_{ALL} with highest feature importance $\mathfrak{I}$ to M_{ALL} (sorted by feature importance).

well as specific source port and frame size bins. These features are primary candidates to be omitted due to their low importance.

- The traffic volume (feature f_3) has the highest importance. This can be expected, as volumetric DDoS attack traffic is characterized by its high-volume traffic.
- The UDP packet count (feature f_5) has the second highest importance and a higher importance than the TCP packet count (feature f_4). Since three of the attack vectors in scenario A send UDP-based attack traffic (the DNS, NTP, and OpenVPN reflection attacks), the high importance indicates that the model recognizes attack vector characteristics.
- The DNS and NTP attacks send UDP traffic from the port range of feature f_7 (source ports 0 to 1023). Also, the majority of the packets in the legitimate traffic of scenario A use the TCP protocol with source ports in this range (99 % of all packets). The high importance of this feature indicates that the model uses information about the source port distribution to estimate traffic purity.
- Feature f_8 (source ports 1024 to 2047) is of high importance. This can be expected, since no packets of the legitimate traffic of scenario A originate from this source port range, while the OpenVPN attack sends UDP datagrams from port 1194. This further emphasizes that a model can learn specific attack vector

Features	#Bins	Bin description
f_{8-70}	1	One bin counting **source ports** in the range from 1024 to 65,536 (replacing features f_8 through f_{70}).
f_{71-81}	1	One bin counting **frame sizes** in the range from 0 to 1407 bytes (replacing features f_{71} through f_{81}).

Table 4.8: The features f_{8-70} and f_{71-81}, which span larger frame size and source port ranges to reduce the total number of features (compare Table 4.5).

characteristics. However, a strong focus on specific attack vector characteristics can impair a models ability to generalize.

- The number of frames with a size of at least 1408 bytes is of high importance (feature f_{82}). The DNS amplification traffic, a portion of the streams TCP_2 and UDP_1, as well as 85 % of legitimate packets, fall into this range. Even though the legitimate traffic mainly uses this fame size, it appears that the model still uses this feature to identify attack traffic. All other features providing information on frame size distribution (features f_{71} to f_{81}) have only low importance despite the fact that multiple attack streams are restricted in their frame size distribution. This can indicate that other features simply provide more important information, causing the model to focus on these instead.
- The prefix length does not have much significance, which means that the size of the subnet over which traffic is aggregated is not highly relevant for the model.

The relatively low number of features of high importance opens up the possibility to reduce the number of features required to achieve low estimation errors.

4.5.4 Model Performance with Reduced Features

Based on the results from Section 4.5.3 two alternative features are defined in Table 4.8. Their definition is based on the distribution of feature importance (see Figure 4.10):

- Feature f_{8-70} combines source port bins of low importance above port 2047 into a single bin. It also includes the source port range 1024 to 2047 to avoid the explicit use of feature f_8, which risks overfitting the model to specific

attack vector characteristics (of the OpenVPN amplification attack). Using f_{8-70} reduces the feature count by 62.

- Feature f_{71-81} combines the frame size bins with low importance below a frame size of 1408 bytes into a single bin. This alternative choice for bins reduces the feature count by 10.

Based on these new and the previously used features $f_1, \ldots, f_{82}$, new aggregate features are constructed to reduce the total number of features and, consequently, the number of counters that are required by an HHH algorithm. Table 4.9 summarizes the selection of features for all newly constructed aggregate features. The choice of aggregate features is subject to the following considerations:

- $\mathsf{f}_{\mathsf{TOP8}}$ uses feature f_{74} (the eighth most important feature) in addition to the seven features that have at least twice the importance than all other features. This serves to assess the impact on estimation errors when only one more feature is used in addition to the features of high importance.

- $\mathsf{f}_{\mathsf{TOP7}}$ uses the top seven features $f_2, f_3, f_4, f_5, f_7, f_8$, and f_{82} with the highest feature importance. This serves to compare other aggregate features that include or exclude other features than the seven most important features.

- $\mathsf{f}_{\mathsf{TOP6}}$ omits the feature f_{82} for the bin with the largest frame size from $\mathsf{f}_{\mathsf{TOP7}}$. This is to evaluate estimation errors when removing one additional feature.

- $\mathsf{f}^{\mathsf{ports}}_{\mathsf{SEL6}}$ Uses a custom selection of features: $f_3, f_4, f_5, f_7, f_{8-70}$ and f_{82}. This excludes the packet count and replaces it with f_{8-70}, which counts the use of higher port numbers. The idea is to exploit redundancies among features to provide more information on source port distributions. The total packet count is provided by features f_7 and f_{8-70}, since $f_2 = f_7 + f_{8-70}$ and f_7 counts all packets that do not use TCP or UDP (treating them as if they had source port 0). The count of packets that do not use TCP or UDP can also be explicitly calculated as $f_7 + f_{8-70} - f_4 - f_5$.

- $\mathsf{f}^{\mathsf{size}}_{\mathsf{SEL6}}$ Replaces the feature f_{8-70} of $\mathsf{f}^{\mathsf{ports}}_{\mathsf{SEL6}}$ with feature f_{71-81}. This still provides the total packet count as $f_2 = f_{71-81} + f_{82}$, but calculation of the number of packets that use other protocols than TCP or UDP is no longer possible.

- $\mathsf{f}_{\mathsf{TOP4}}$ further reduces the number of features to four by selecting the four most important features f_3, f_4, f_5, f_8. This serves to assess whether low errors can be maintained with reduced memory requirements.

Aggregate features	#Bins	Included features	Bin desciption
$\mathsf{f}_{\mathsf{TOP8}}$	8	$f_2, \ldots, f_5$	Packet count,
		f_7, f_8, f_{74}, f_{82}	Traffic volume, Protocols (TCP, UDP), Source ports ($0-1023$, $1024-2047$), Frame sizes ($384-511$, $1408-1535$)
$\mathsf{f}_{\mathsf{TOP7}}$	7	$f_2, \ldots, f_5$	Packet count,
		f_7, f_8, f_{82}	Traffic volume, Protocols (TCP, UDP), Source ports ($0-1023$, $1024-2047$), Frame sizes ($1408-1535$)
$\mathsf{f}_{\mathsf{TOP6}}$	6	$f_2, \ldots, f_5$	Packet count,
		f_7, f_8	Traffic volume, Protocols (TCP, UDP), Source ports ($0-1023$, $1024-2047$)
$\mathsf{f}^{\mathsf{ports}}_{\mathsf{SEL6}}$	6	$f_3, f_4, f_5,$	Traffic volume,
		f_7, f_{8-70}, f_{82}	Protocols (TCP, UDP), Source ports ($0-1023$, $1024-65535$), Frame sizes ($1408-1535$)
$\mathsf{f}^{\mathsf{size}}_{\mathsf{SEL6}}$	6	$f_3, f_4, f_5,$	Traffic volume,
		f_7, f_{71-81}, f_{82}	Protocols (TCP, UDP), Source ports ($0-1023$), Frame sizes ($0-1407$, $1408-1535$)
$\mathsf{f}_{\mathsf{TOP4}}$	4	f_3, f_4, f_5, f_8	Traffic volume, Protocols (TCP, UDP), Source ports ($1024-2047$)
$\mathsf{f}_{\mathsf{VOLUME}}$	1	f_3	Traffic volume

Table 4.9: Aggregate features with reduced feature sets.

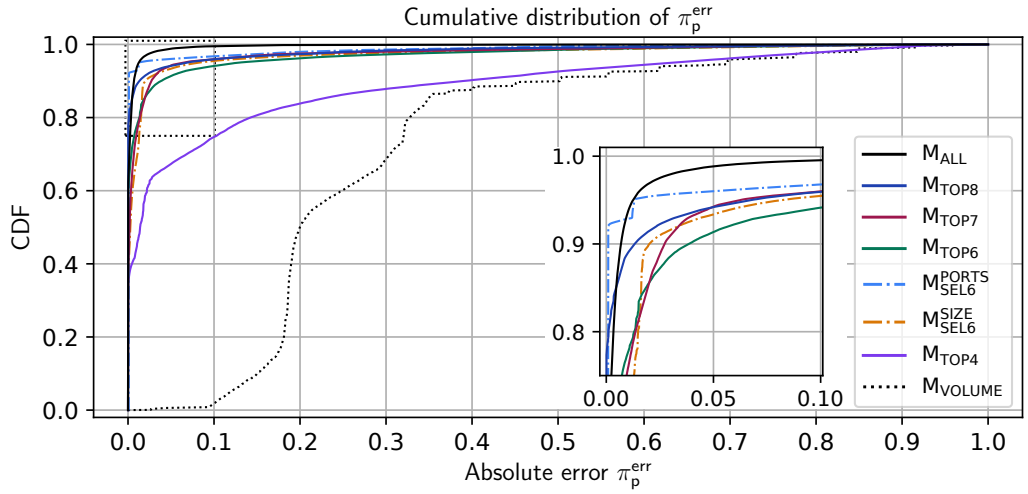


Figure 4.11: CDFs of absolute errors when estimating traffic purity with seven different selections for aggregate features. The traffic purity estimation errors are based on test dataset D_{test} of scenario A.

- $\mathsf{f}_{\text{VOLUME}}$ uses only feature f_3. This choice of a single feature serves as a baseline to evaluate whether the traffic volume alone provides sufficient information to estimate the traffic purity of an aggregate.

For each of the seven aggregate features f_{TOP8}, f_{TOP7}, ... a corresponding model M_{TOP8}, M_{TOP7}, ... is trained using reduced features. The performance of the trained models is evaluated using the test dataset D_{test} of scenario A and the entire dataset D of scenario B to compare estimation errors and the model's ability to generalize.

Model performance in scenario A. Figure 4.11 shows the CDF of traffic purity estimation errors in scenario A for all models that use aggregate features from Table 4.9. For comparison, the figure also includes the CDF for model M_{ALL}. Furthermore, Table 4.10 summarizes the MAE, the MSE, and the proportion of errors for selected maximum error values of 1 %, 2 %, 10 %, and 20 %.

Several results are obtained by comparing model performance and considering the reduction of the number of features in relation to the model M_{ALL}:

- The models $\mathsf{M}_{\text{VOLUME}}$, M_{TOP4}, M_{TOP6}, and M_{ALL} exhibit a clear trend that lower traffic purity estimation errors are achieved with higher probability as the number of features increases. This trend also appears in the MAE and MSE listed in Table 4.10. It would suggest that an increased availability of (relevant) monitoring information directly improves model performance.

Model	MAE	MSE	Proportion of absolute errors π_p^{err}			
			$\leq 1\,\%$	$\leq 2\,\%$	$\leq 10\,\%$	$\leq 20\,\%$
M_{ALL}	0.0037	0.000483	0.9358	0.9691	0.9954	0.9986
M_{TOP8}	0.0167	0.006900	0.8895	0.9148	0.9594	0.9733
M_{TOP7}	0.0214	0.008127	0.7569	0.8567	0.9599	0.9749
M_{TOP6}	0.0262	0.010560	0.7756	0.8559	0.9414	0.9627
M_{SEL6}^{ports}	0.0122	0.004608	0.9286	0.9541	0.9678	0.9798
M_{SEL6}^{size}	0.0245	0.010001	0.6874	0.8997	0.9547	0.9694
M_{TOP4}	0.1050	0.051313	0.4442	0.5871	0.7471	0.8384
M_{VOLUME}	0.2731	0.101588	0.0002	0.0004	0.0185	0.5012

Table 4.10: MAE, MSE, and proportion of absolute traffic purity estimation errors below selected values in scenario A using models with different aggregate features. Best values are highlighted.

- The trend also holds for the models M_{TOP7} and M_{TOP8} with respect to MAE and MSE, but the distribution of errors shown in Table 4.10 is not clearly distinguished from the four previous models. For example, M_{TOP6} achieves errors below 1 % more frequently than M_{TOP7}, but has a higher chance to yield errors above 2 %. The inclusion of an additional feature in f_{TOP8} compared to f_{TOP8} results in a higher probability to incur traffic purity estimation errors below 2 %. In comparison, absolute errors above 10 % can be considered equally likely to occur for both models. This suggests that the additional feature proves effective for Traffic Purity Estimation.

- The performance of model M_{VOLUME} is considerably below that of all other models. The MAE and MSE in comparison to M_{ALL} increase by a factor of 73.8 and 565.4. Errors of at most 10 % are only rarely achieved (with probability $\leq 1.85\,\%$) while errors above 20 % occur at least one order of magnitude more often than for any model that uses six or more features. This indicates that traffic volume alone is insufficient to reliably estimate the attack traffic volume in an aggregate. Consequently, relying on purely volumetric information would result in a strong impact of traffic filtering on legitimate traffic.

- The model M_{TOP4} performs better than M_{VOLUME}, but still has a high chance of 16.1 % to have estimation errors above 20 %.

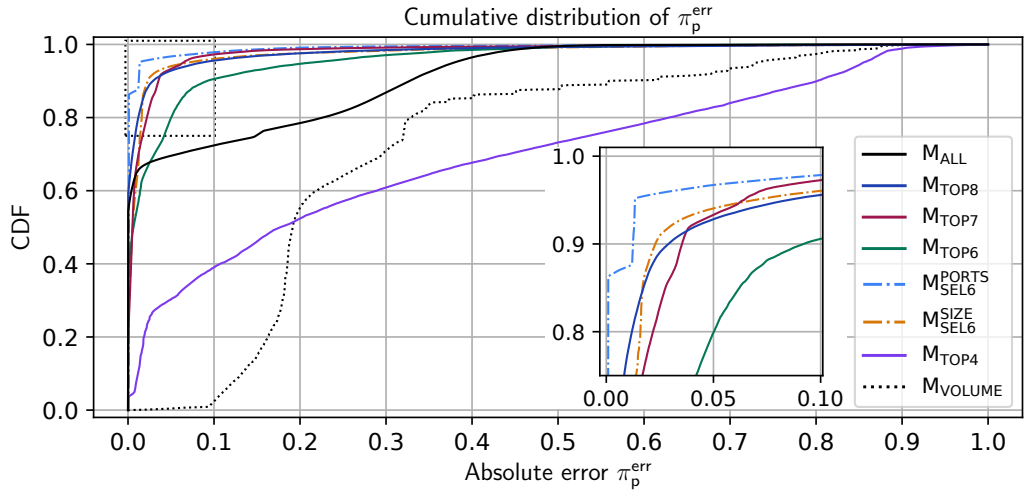


Figure 4.12: CDFs of absolute errors when estimating traffic purity with seven different selections for aggregate features. The traffic purity estimation errors are based on the entire dataset D from scenario B. The CDF of $M_{TCP+UDP+PORTS}$, $M_{TCP+UDP+SIZE}$, and M_{PROTOS} overlap in the upper left corner of the inset figure.

- Models M_{SEL6}^{ports} and M_{SEL6}^{size} can be considered to outperform model M_{TOP6} (that uses an equal number of features). The MAE and MSE are lower and the error is more likely to reach at most 2 %. Hence, the choice of the features f_{8-70} and f_{71-81} proved effective. In particular, using the feature f_{8-70} results in low traffic purity estimation errors. This is either because of the additional information on the source port distribution or due to the ability to calculate the number of packets that use other protocols than TCP and UDP. The model M_{SEL6}^{ports} also consistently outperforms all other models except M_{ALL}. In comparison to M_{ALL} it achieves very low errors more often (see Figure 4.11) but incurs larger errors more frequently. The aggregate features f_{SEL6}^{ports} can still be considered primary candidates for reducing memory requirements. In comparison to M_{ALL}, 76 less counters would be required for an HHH algorithm for each monitored prefix.

In conclusion, the model M_{ALL} would be the preferred choice to consistently maintain low errors, but it requires monitoring of 82 features. When considering trade-offs, the model M_{SEL6}^{ports} is the best candidate, as it requires only 6 features to achieve low traffic purity estimation errors (due to the choice of f_{SEL6}^{ports}). M_{TOP4} and M_{VOLUME} generally suffer from reduced model performance, but M_{TOP4} could still be considered when HHH memory is scarce.

Model	MAE	MSE	Proportion of absolute errors π_p^{err}			
			$\leq 1\,\%$	$\leq 2\,\%$	$\leq 10\,\%$	$\leq 20\,\%$
M_{ALL}	0.0836	0.026211	0.6497	0.6714	0.7243	0.7850
M_{TOP8}	0.0191	0.006115	0.7760	0.8687	0.9558	0.9758
M_{TOP7}	0.0187	0.003922	0.6694	0.7836	0.9727	0.9870
M_{TOP6}	0.0394	0.009220	0.5463	0.6522	0.9058	0.9470
M_{SEL6}^{ports}	0.0090	0.002950	0.8741	0.9557	0.9785	0.9911
M_{SEL6}^{size}	0.0213	0.006396	0.6255	0.8812	0.9605	0.9761
M_{TOP4}	0.2966	0.175726	0.0816	0.2154	0.3904	0.5239
M_{VOLUME}	0.2726	0.107163	0.0006	0.0012	0.0241	0.5540

Table 4.11: MAE, MSE, and proportion of absolute traffic purity estimation errors below selected values in scenario B using models with different aggregate features. Best values are highlighted.

Model performance in scenario B. Evaluating the performance of models that are trained with data from scenario A on the entire dataset of scenario B serves to assess the ability of models to generalize to different scenarios. Figure 4.12 shows the CDF of traffic purity estimation errors in scenario B while Table 4.11 summarizes MAE, MSE, and errors up to the thresholds of $1\,\%, 2\,\%, 10\,\%$, and $20\,\%$.

The key finding of performing Traffic Purity Estimation in scenario B is that the availability of more monitoring information no longer consistently improves model performance. The relationship between the number of features and model performance is also not reciprocal. This is made clear through the following observations:

- Model M_{VOLUME} still has low performance and uses only a single feature.
- Models M_{ALL} and M_{TOP4} are the next two worst-performing models, despite using the largest and second-smallest aggregate features. The MAE and MSE of M_{TOP4} are particularly high due to a high frequency at which high errors occur.
- The model M_{SEL6}^{ports} can be considered the best-performing model since it achieves the lowest errors while using the lowest number of features that still prove effective (cf. Table 4.11). In particular, it clearly outperforms model M_{TOP6}, which used the six features with highest importance. Instead, the custom choice of aggregate features for f_{SEL6}^{ports} produced more relevant monitoring information.

- The model M_{TOP7} has the second best performance regarding traffic purity estimation errors. In comparison, M^{ports}_{SEL6} should be preferred due to its lower errors and memory requirements, but M_{TOP7} did not require a custom feature selection. Instead, its features were determined in an automatic fashion.

- The inclusion of an additional feature in f_{TOP8} no longer proves effective. While M_{TOP8} was among the best performing models in scenario A, it no longer shows an advantage over M_{TOP7} despite requiring more memory.

The reduced performance of models M_{ALL} and M_{TOP4} can be attributed to the fact that scenario B has different traffic characteristics from scenario A and that these models tend to overfit on specific features. The elimination of features can not only reduce HHH memory requirements, but it can also reduce the risk of overfitting. This makes M_{TOP7} and M^{ports}_{SEL6} the two best performing models overall. Nevertheless, introducing a greater lack of monitoring information risks diminishing model performance in previously unseen traffic scenarios, as can be seen in the case of M_{TOP4}.

4.6 Related Work

The following discusses related work that aligns with the design goals of Traffic Purity Estimation. The focus is not only on detecting attack traffic but specifically on quantifying the attack traffic volume or ratio. Furthermore, an approach to monitoring informative features to characterize volumetric DDoS traffic with Heavy Hitters is discussed.

Estimating attack traffic volume. The work of [Agr+11] uses deep neural networks to estimate the total attack traffic volume of volumetric DDoS attacks through regression analysis. Specifically, a neural network is assumed to be deployed or co-located with a router to conduct estimations in real-time. The neural networks use a simpler architecture of two layers with a maximum of 20 neurons per layer. The evaluation of simulated direct-path attacks reports that a minimum MSE of 1.8 for estimations of the absolute attack traffic volume is achieved (which is not directly comparable to the ratio of attack traffic). Similarly to the previous approach, [Gup+11a], [Gup+11b], and [GJM12] estimate the traffic volume of an attack. However, these approaches rely on multiple different regression models, such as polynomial, logarithmic, and exponential regression. Values of at least 29.81 are reported for MSE.

The attack traffic in all evaluations consists of UDP streams that are generated by 100 attacking systems, while the simulated legitimate traffic uses TCP. This is a single attack vector, whose traffic features are not elaborated in detail. In comparison, the evaluation of Traffic Purity Estimation considered attack traffic and traffic aggregates comprising multiple different attack vectors. Furthermore, aggregates represent only a portion of the combined attack traffic, which makes it more challenging to quantify the attack traffic due to less converged features.

Memory-efficient monitoring of volumetric DDoS features. The approach presented in [ABF19] aligns with the goal of identifying important features of high-volume attacks. It uses Heavy Hitter algorithms to extract previously unknown features from monitored attack traffic. Specifically, the Heavy Hitter algorithms are used in online monitoring to identify characteristic string sequences that indicate the presence of attack traffic. This constitutes a promising research direction to efficiently monitor more complex features of volumetric DDoS traffic. Such features may provide highly relevant information for distinguishing between attack and legitimate traffic, which could facilitate more accurate Traffic Purity Estimation.

4.7 Conclusion

This chapter outlined that a distinction between HHH aggregates composed of attack and legitimate traffic cannot be reliably achieved based on traffic volume alone. To enable such a distinction, the concept of Traffic Purity Estimation was introduced, which estimates the ratio of attack traffic in an aggregate. This is an important building block in generating effective filter rules from HHH prefixes. To conduct Traffic Purity Estimation, a novel method based on supervised learning was presented. A key benefit of this method is that the relationship between traffic features and traffic purity can be automatically learned through a data-driven approach. The chapter described the details of the machine learning-based system, including relevant traffic features, modifications to HHH algorithms necessary to collect required monitoring information, as well as the design and training process of a neural network.

To facilitate efficient HHH monitoring, a primary design goal was to achieve low traffic purity estimation errors using only a few features. For this, a feature selection method based on permutation feature importance was presented. The method allows it to retain important features while eliminating features of low relevance. Through this, the memory requirements of HHH algorithms can be reduced.

A key insight is that the data-driven approach can achieve low traffic purity estimation errors with a low number of features. Specifically, two traffic scenarios with distinct traffic characteristics were created to train and evaluate models under different conditions. Both scenarios mixed authentic, legitimate traffic with synthesized attack traffic that modeled multiple DDoS attack vectors. The evaluation showed that models can learn to estimate the traffic purity. The model $\mathsf{M}^{\text{ports}}_{\text{SEL6}}$ that used using a custom feature selection maintained traffic purity estimation errors below 2 % with a probability beyond 95 % in a scenario with new attack vectors that were not included during training. Furthermore, mixtures of multiple attack vectors occurred in traffic aggregates. Through the selection of specific features, the memory requirements were also reduced, from an initial 82 features down to 6. This translates to a reduced number of counters required for the HHH algorithm of Aggregate Monitoring.

Chapter 5

Filter Rule Refinement

Chapter 4 introduced Traffic Purity Estimation to provide insight into the composition of traffic aggregates. This serves to estimate the purity of attack traffic sent from an IP subnet. Still, Traffic Purity Estimation considers aggregates independently and does not take into account that subnets are organized hierarchically. If traffic from a large subnet has a high attack traffic purity, the subnet may still contain address space regions from which only legitimate traffic originates. In this case, attack traffic should be traced back to smaller subnets to reduce the impact of traffic filtering on legitimate traffic. Filter Rule Refinement introduces a method to decide whether attack traffic can be attributed primarily to smaller subnets. This allows short IP prefixes with unnecessarily high impact on legitimate traffic to be rejected when generating filter rules.

This chapter identifies and addresses several challenges that arise from considering the hierarchical relationship between prefixes when relying on HHH algorithms for efficient monitoring. HHH algorithms provide only partial insight into the distribution of traffic volume over the IP prefix hierarchy, and the provided monitoring information may change frequently. This makes it challenging to maintain the effectiveness of blacklists over time.

Section 5.1 elaborates the challenges involved in selecting prefixes for filter rules based on information provided by HHH algorithms in greater detail. The following two sections introduce novel approaches for filter rule selection that take the hierarchical organization of prefixes into account and compensate for frequently changing monitoring information. Section 5.4 evaluates the feasibility of Filter Rule Refinement based on dynamic traffic scenarios. Related work is discussed in Section 5.5.

5.1 Design Goals

The overarching design goal of Filter Rule Refinement is to consider the hierarchical relationship between HHH prefixes when generating filter rules over consecutive monitoring intervals. This serves to assess the impact each individual prefix has on attack and legitimate traffic in order to provide (stable) estimates on the precision that a corresponding filter rule achieves. Given this estimation, blacklist generation can select filter rules by applying the following threshold:

Definition 5.1: Minimum Required Precision

The **minimum required precision** π^* constitutes the threshold on the estimated filtering precision that all IP prefixes must reach in order to be included in a blacklist.

To select prefixes based on this threshold, their hierarchical relationship should be considered. If parts of a subnet are filtered by a set of filter rules, the question arises if it would be better to use only a single filter rule to remove traffic from the entire subnet. This depends on how much attack and legitimate traffic would be additionally affected. Considering this hierarchical relationship between subnets serves to reduce the risk of accepting filter rule prefixes that would have an unnecessarily high impact on legitimate traffic (and effectively a low filtering precision). In particular, Filter Rule Refinement compensates for two adverse effects that arise from selecting filter rule prefixes from an IP prefix hierarchy: overaggregation and HHH instability.

5.1.1 Compensating Overaggregation

The more the traffic is aggregated across the IP prefix hierarchy during the computation of HHHs, the more difficult it is to determine the original source of the traffic. As a prefix's length increases, the corresponding subnet sizes increase exponentially. At a certain point, an HHH can be considered to become **overaggregated** in the context of filter rule generation, as larger subnet sizes increase the risk of removing legitimate traffic. This makes it dangerous to accept short prefixes into a blacklist, even if they have a high attack traffic purity. The attack traffic can originate from small sub-regions of the IP address space, while a short prefix would cover the entire spanning subnet, including regions that send legitimate traffic only. The challenge is to decide whether short prefixes should be accepted into a blacklist. Including

them can be beneficial because it allows the elimination of redundant child prefixes to reduce the size of the blacklist.

The first design goal is to provide an estimate of filter rule precision when disregarding traffic that could be filtered by rules that use longer HHH prefixes. This serves to assess the impact of removing traffic from a larger subnet compared to filtering traffic in smaller sub-regions of the same subnet.

5.1.2 Compensating HHH Instability

When network traffic is monitored over time with HHH algorithms, HHH prefixes can frequently vanish and reappear due to several reasons:

- Efficient HHH algorithms use a limited number of counters to monitor the traffic volume of IP prefixes. This can lead to estimation errors in traffic volumes and frequent pruning of monitored prefixes. As a result, HHH prefixes can change over time, although the network traffic remains unchanged.
- The network traffic in a subnet can change over time. If the (conditional) traffic volume (see Definition 2.2) is close to the absolute HHH detection threshold $\phi\mathcal{V}$, minor changes in the traffic volume can influence the detection of HHH prefixes. This is especially the case when the traffic changes frequently.
- The (relative) HHH detection threshold ϕ and the total traffic volume $\mathcal{V}$ can change over time. This can result in different HHH prefixes being detected across the entire IP subnet hierarchy.

Traffic Purity Estimation and thus the generation of filter rules depend on the detection of HHH prefixes. Consequently, unstable HHH prefixes can result in filter rules being repeatedly established and withdrawn. This can reduce mitigation effectiveness. For example, it can lead to attack traffic not being removed when a filter rule is withdrawn. In addition, it becomes more challenging to consider the hierarchical relationship between two prefixes over time when compensating for overaggregation.

The second design goal of Filter Rule Refinement is to compensate for instabilities in HHH monitoring information during the generation of blacklists. More specifically, to compute a smoothed estimate of filter rule precision that compensates for missing information on traffic purity over short periods of time.

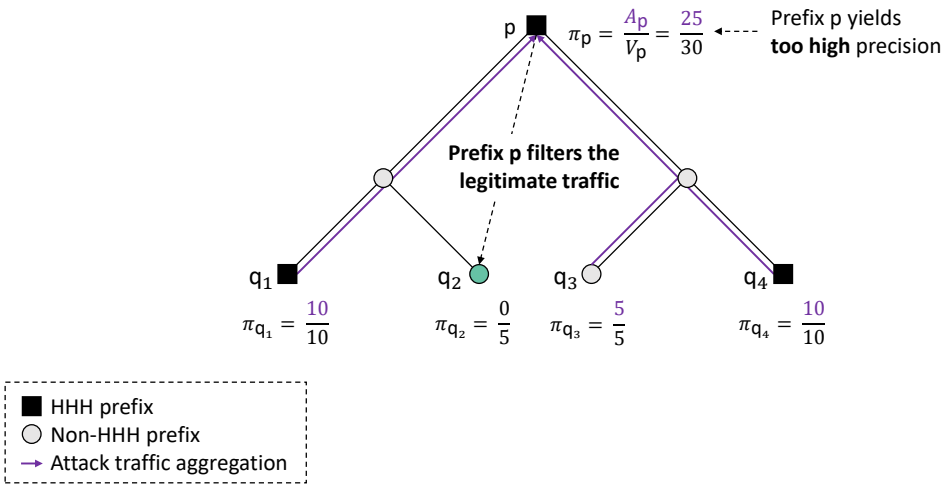


Figure 5.1: Example attack and legitimate traffic distribution with prefixes $q_1, \ldots, q_4$ and common parent prefix p. All prefixes with precision $\pi \geq 4/5$ are accepted into a blacklist resulting in filtering of all legitimate traffic (of q_2).

5.2 Hierarchical Aggregation of Attack Traffic

The attack traffic volume A_p for a prefix p can be estimated from the traffic volume V_p and traffic purity estimations. This allows it to estimate the precision $\pi_p = A_p/V_p$ of a filter rule. However, the decision to accept a filter rule should not be based on precision alone. The following example illustrates how overaggregation of attack traffic can impact filtering effectiveness when relying only on precision estimates.

Example 5.2.1. Consider the four distinct prefixes $q_1, \ldots, q_4$ of equal length with a common longest parent prefix p and the following traffic volumes:

- The attack traffic volume of q_1 and q_4 is $A_{q_1} = A_{q_4} = 10$ and the total traffic volume is $V_{q_1} = V_{q_4} = 10$.
- The attack traffic volume of q_2 is $A_{q_2} = 0$ and the total traffic volume is $V_{q_2} = 5$.
- The attack traffic volume of q_3 is $A_{q_3} = 5$ and the total traffic volume is $V_{q_3} = 5$.

Figure 5.1 depicts the prefixes with their associated traffic volume. The total traffic volume V_p is 30 and the (combined) attack traffic volume A_p is 25. When including all prefixes r with a precision $\pi_r = A_r/V_r \geq 4/5$ in a blacklist, the inclusion of the

common parent prefix p (with precision $\pi_p = 5/6 \geq 4/5$) results in the removal of the purely legitimate traffic in the subnet of q_2. The resulting false positive rate is 1.

This example illustrates that a simple prefix precision metric is insufficient to enable effective filtering decisions. Relying on per-prefix precision while disregarding shorter prefixes results in a worst-case false positive rate due to the wide address space coverage of the short parent prefix p. More specifically, the notion of prefix precision does not adequately capture the impact of p on legitimate traffic. When the precision of prefixes q_1 and q_4 is known and both prefixes are accepted into a blacklist, the corresponding attack traffic volumes A_{q_1} and A_{q_4} can be disregarded in the calculation of π_p. The attack traffic of q_1 and q_4 can be filtered by blacklisting these longer prefixes. In that case, only the remaining attack traffic is of relevance to determine the filtering impact of p. Consequently, the high false positive rate incurred by adoption of p can be avoided by subtracting the combined attack traffic volume $A_{q_1} + A_{q_4}$ from A_p prior to determining π_p. In that case, the prefix p is rejected because $\pi_p = A_{q_2} + A_{q_3}/V_p = 1/6 < 4/5$ and the resulting false positive rate remains zero.

However, monitoring HHH prefixes does not provide complete information on traffic distribution. When q_2 and q_3 are not HHH prefixes, their total and attack traffic volumes remain unknown. In that case, they cannot be blacklisted individually, even when they correspond to pure attack traffic (contrary to the previous example). When q_1, q_4, and p are HHH prefixes but q_2 and q_3 are not, the traffic volumes of q_2 and q_3 are reflected only in the aggregated traffic volume of the parent prefix p. However, in that case, high attack traffic volumes A_{q_1} and A_{q_4} can dominate the attack traffic contribution of A_{q_2} and A_{q_3} to the attack traffic volume A_p of the parent prefix p. This can result in artificially lowered precision when subtracting A_{q_1} and A_{q_4} from A_{q_1} and, consequently, the rejection of an effective filter rule. The following example (shown in Figure 5.2) illustrates this effect.

Example 5.2.2. Assume that traffic volume distribution is monitored by an HHH algorithm that applies a query threshold of $\phi = 1/3$. The arrangement and traffic volume distribution of prefixes $q_1, \ldots, q_4$ and p is identical to Example 5.2.1 with the following exception:

- The attack traffic volume of q_2 is $A_{q_2} = 5$ (instead of 0) while the total traffic volume $V_{q_2} = 5$ remains unchanged.

The combined attack traffic volume $A_p = 30$ now matches the total traffic volume $V_p = 30$, i.e., all traffic should be filtered. However, with the application of a query threshold of $\phi = 1/3$, only the traffic of the HHH prefixes q_1, q_4, and p can be

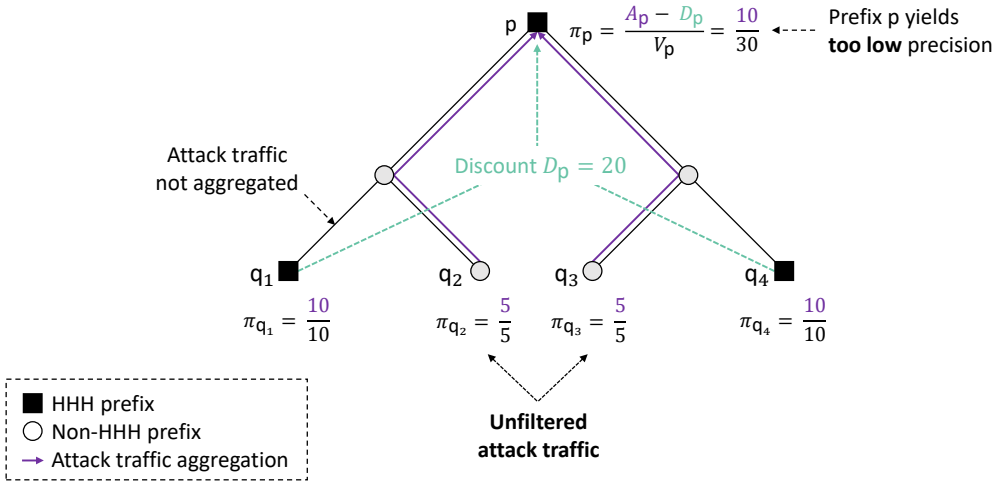


Figure 5.2: Example attack and legitimate traffic distribution with prefixes $q_1, \ldots, q_4$ and common parent prefix p. All prefixes with precision $\pi \geq 4/5$ are accepted into a blacklist. Discounting of attack traffic results in the incorrect rejection of a parent prefix p and the unmitigated attack traffic of q_2 and q_3.

monitored. Specifically, the exact quantities V_{q_2}, A_{q_2}, V_{q_3}, and A_{q_3} remain unknown. Subtracting the combined attack traffic volume $A_{q_2} + A_{q_3}$ only from A_p, but not from V_p, causes the prefix p to have $\pi_p = (A_{q_1} - A_{q_2} - A_{q_3} + A_{q_4})/V_p = 10/30 = 1/3$. With a minimum required precision of 4/5 the prefix p is excluded from the blacklist despite filtering attack traffic only. In contrast, when no HHH monitoring would take place and the traffic volume distribution of all prefixes would be known, the prefixes q_2 and q_3 would be included in the blacklist (due to $\pi_{q_2} = \pi_{q_3} = 1$). This results in the correct removal of all attack traffic.

While the previous examples described specific cases, over and under-estimation of prefix precision should be avoided regardless of traffic distribution and choices of ϕ or π^*. Otherwise, traffic filtering can result in unnecessarily high false positive and false negative rates.

To make effective filtering decisions, both the attack *and* the total traffic volume A_p and V_p of a prefix p must be discounted by the cumulative attack traffic volume $\sum_{q \in \mathcal{B}, q \prec p} A_q$ of all child prefixes q of p that are accepted into a blacklist. Through this, the decision whether to accept a prefix p is based exclusively on the traffic that is directly affected by the inclusion of p *in addition* to longer prefixes. The subtraction of aggregated (attack and legitimate) traffic volumes is referred to as **discounting**

throughout this thesis. The discounting of traffic volumes addresses overaggregation of attack traffic volume and the disregard of shorter blacklisted prefixes. Discounting of both attack and legitimate traffic volume to estimate filter rule precision is captured in the following notion:

Definition 5.2: Discounted Precision

The **discounted precision** $\overline{\pi}_{\mathsf{p}}$ of a prefix p with respect to a minimum required precision $\pi^* \in [0,1]$ and fully qualified child prefixes q of p is defined as

$$\overline{\pi}_{\mathsf{p}} \stackrel{\text{def}}{=} \begin{cases} \dfrac{A_{\mathsf{p}} - D_{\mathsf{p}}}{V_{\mathsf{p}} - D_{\mathsf{p}}} & \text{if } V_{\mathsf{p}} \neq D_{\mathsf{p}}, \\ 0 & \text{otherwise,} \end{cases} \quad \text{with} \quad D_{\mathsf{p}} \stackrel{\text{def}}{=} \sum_{\substack{\overline{\pi}_{\mathsf{q}} \geq \pi^* \\ \mathsf{q} \prec \mathsf{p}}} A_{\mathsf{q}} \,. \tag{5.1}$$

D_{p} denotes the **traffic discount** for prefix p while A_{p}, V_{p} denote its attack and total traffic volumes.

Using $\overline{\pi}_{\mathsf{p}} = 0$ in Equation 5.1 when $V_{\mathsf{p}} - D_{\mathsf{p}} = 0$ causes the rejection of any filter rule that does not have any associated (discounted) traffic volume. Determining the discounted precision $\overline{\pi}_{\mathsf{p}}$ requires iteration from longer to shorter prefixes to enable the evaluation of the condition $\overline{\pi}_{\mathsf{q}} \geq \pi^*$ in Equation 5.1 for child prefixes q and the correct calculation of the traffic discount D_{p} for parent prefixes p. A prefix p without child prefixes immediately yields $D_{\mathsf{p}} = 0$ and $\overline{\pi}_{\mathsf{q}} = A_{\mathsf{q}}/V_{\mathsf{q}}$.

The following example outlines how the application of discounted precision addresses the incorrect inclusion and omission of filter rules in the previous two examples:

Example 5.2.3. Consider the prefix arrangement and traffic distribution from Example 5.2.1 (with purely legitimate traffic originating from q_2) and an applied HHH detection threshold of $\phi = 1/3$ (shown in Figure 5.3). The minimum required precision is set to $\pi^* = 4/5$ (as before). The traffic discount of prefix p is calculated as $D_{\mathsf{p}} = A_{\mathsf{q}_1} + A_{\mathsf{q}_4}$ resulting in a discounted precision $\overline{\pi}_{\mathsf{p}} = (A_{\mathsf{p}} - D_{\mathsf{p}})/(V_{\mathsf{p}} - D_{\mathsf{p}}) = 1/2$. This has two consequences

- The purely legitimate traffic of q_2 (correctly) remains unfiltered. This is clearly a correct decision, since q_2 corresponds to entirely legitimate traffic.
- The pure attack traffic of q_3 remains unfiltered. This can be regarded as an incorrect decision when full information on traffic distributions is available.

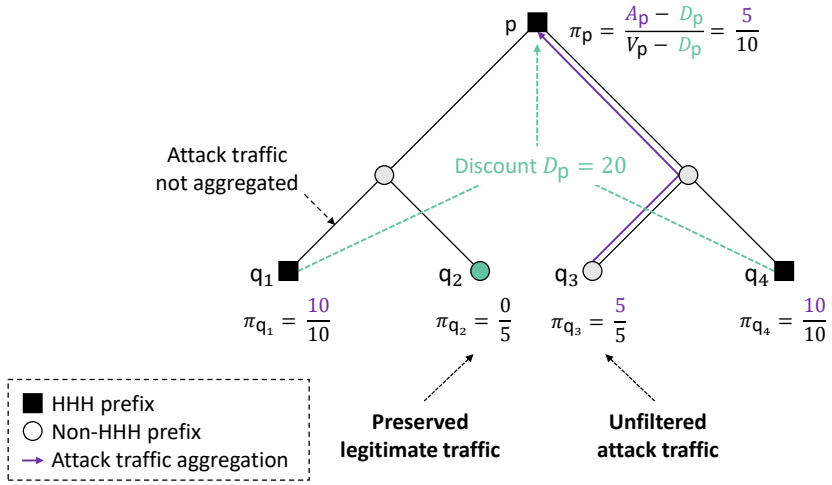


Figure 5.3: Example attack and legitimate traffic distribution with prefixes $q_1, \ldots, q_4$ and common parent prefix p. All prefixes with precision $\pi \geq 4/5$ are accepted into a blacklist. Discounting of attack and legitimate traffic results in the correct rejection of a parent prefix p that has an equal mixture of attack and legitimate traffic.

However, monitoring HHH prefixes does not provide full information on traffic distribution. Only the aggregated traffic volumes $A_{q_2} + A_{q_3} = A_p - A_{q_1} - A_{q_4}$ and $V_{q_2} + V_{q_3} = V_p - V_{q_1} - V_{q_4}$ of q_2 and q_3 are known (through discounting). Requiring a minimum precision $\pi^* = 4/5$ on these traffic volumes makes the rejection of the parent prefix p a correct decision, since blacklisting p removes equal amounts of attack and legitimate traffic. Not filtering the attack traffic should be the preferred choice to avoid the overly strong impact of filtering on the legitimate traffic.

Conversely, when the traffic of q_2 comprises pure attack traffic ($A_{q_2} = 5$) instead, the prefix p has a discounted precision $\overline{\pi}_p = 1$. In contrast to Example 5.2.2, the parent prefix p is now correctly blacklisted, and filtering removes the entire attack traffic.

5.3 Compensating Traffic Dynamics

Blacklists are generated periodically at regular monitoring intervals $\mathcal{I}_t, \mathcal{I}_{t+\Delta t}, \ldots$ of duration Δt to determine subsequent packet filtering. That is, a blacklist generated at the end of the current monitoring interval applies to all packets during the next interval. However, blacklist generation depends on information obtained during the

current interval (the traffic volume reported by HHH algorithms and Traffic Purity Estimation). If traffic changes between monitoring intervals, blacklists generated from previously acquired monitoring information can deviate from the true attack traffic distribution. This can have two adverse effects:

- Attack traffic is not removed after attacking systems temporarily cease to send traffic, resulting in temporary high false negative rates.
- Legitimate traffic is removed after legitimate systems temporarily cease to send traffic, resulting in temporary high false positive rates.

Therefore, ineffective filter rules due to changing traffic patterns can be the result of both dynamic attack *and* legitimate traffic. The following example outlines how a legitimate high-volume flow can result in high false positive rates when it temporarily ceases to send traffic.

Example 5.3.1. Consider three distinct prefixes q_1, q_2, and q_3 with a common longest parent prefix p. Prefix q_2 covers an address space region located between q_1 and q_3. The three prefixes correspond to the following traffic:

- Prefixes q_1 and q_3 represent pure attack traffic of equal volume, which does not change over time i.e., $A_{q_1} = A_{q_3} = V_{q_1} = V_{q_3}$.
- Prefix q_2 corresponds to a single legitimate flow f. Flow f has high traffic volume $V_{q_2} > V_{q_1}$ in every interval $\mathcal{I}_{t+2i\cdot\Delta t}$ and ceases to send traffic ($V_{q_2} = 0$) during every subsequent interval $\mathcal{I}_{t+(2i+1)\cdot\Delta t}$.

No other traffic is present, so that the total traffic volume of all considered prefixes is $N = V_{q_1} + V_{q_2} + V_{q_3}$ If an HHH detection threshold ϕ leads to $V_{q_1} < \phi\mathcal{V} < V_{q_1} + V_{q_3}$ neither q_1 nor q_3 constitute HHH prefixes and they are not blacklisted individually. However, the prefix p is an HHH prefix with (discounted) precision $\overline{\pi}_p = 1$ whenever $V_{q_2} = 0$, i.e., at the end of each monitoring interval $\mathcal{I}_{t+(2i+1)\cdot\Delta t}$. Hence, prefix p is blacklisted during every subsequent monitoring interval $\mathcal{I}_{t+(2i+2)\cdot\Delta t}$ and discards all the traffic of flow f, resulting in a worst-case false positive rate of 1. This is regardless of the choice of the minimum required precision π^*. If V_{q_2} in interval $\mathcal{I}_{t+2i\cdot\Delta t}$ was known ahead of time while generating the blacklist for this interval, p would have a discounted precision of at most $\overline{\pi}_p < \frac{2}{3}$ (since f has high traffic volume $V_{q_2} > V_{q_1}$) and its blacklisting would be prevented with any $\pi^* \geq \frac{2}{3}$.

The example illustrates a worst-case scenario with an alternating legitimate flow and prefixes representing either pure attack or pure legitimate traffic. Still, traffic dynamics can incur increased false positive and false negative rates in more general cases. Whenever the contribution of the legitimate traffic to total traffic volume V_p of a prefix p decreases, the (discounted) precision $\overline{\pi}_\mathsf{p}$ increases due to a reduction of the divisor in Equation 5.1 (the dividend remains unchanged). When this leads to the blacklisting of a prefix p by raising $\overline{\pi}_\mathsf{p}$ over the threshold π^*, an increase in legitimate traffic volume in the next monitoring interval results in an increased false positive rate. This effect applies, for example, to new legitimate flows. Conversely, a reduction in (discounted) attack traffic volume reduces $\overline{\pi}_\mathsf{p}$ by lowering the denominator in Equation 5.1 while leaving the divisor unchanged. As a result, short-term attack traffic bursts can bypass an ingress filter.

To compensate for the effect of traffic dynamics, the traffic volumes V_p, D_p, and A_p of a prefix p can be tracked over time. Remembered traffic volumes from previous monitoring intervals can be considered in the calculation of discounted precision to improve overall filtering effectiveness in two ways:

- A prefix p that had a high $\overline{\pi}_\mathsf{p}$ in previous monitoring intervals can be more readily blacklisted when $\overline{\pi}_\mathsf{p}$ temporarily drops. Through this, remembering previous traffic volumes estimates can lead to reduced false negative rates.
- A prefix p that had a low $\overline{\pi}_\mathsf{p}$ in previous monitoring intervals can be more readily rejected when $\overline{\pi}_\mathsf{p}$ temporarily increases. Through this, historical precision estimates can lead to reduced false positive rates.

The next section introduces a concept to monitor the traffic volumes over time to improve filter rule effectiveness by stabilizing discounted precision estimations in the presence of dynamic traffic.

5.3.1 Traffic Volume Tracking

To retain and utilize historical monitoring information from previous monitoring intervals, the concept of trackers is introduced. Trackers provide smoothed estimates of traffic volumes for individual prefixes using a **weighting function** $\mathsf{E} : \mathbb{R}_{>0} \times \mathbb{R}_{>0} \rightarrow \mathbb{R}_{>0}$ that combines current and historical traffic volumes. Formally, a tracker for a prefix p is defined as follows:

Definition 5.3: Tracker

A **tracker** T_p is a tuple $\mathsf{T}_\mathsf{p} = (A_\mathsf{p}^{\mathsf{cur}}, D_\mathsf{p}^{\mathsf{cur}}, V_\mathsf{p}^{\mathsf{cur}}, A_\mathsf{p}^{\mathsf{prv}}, D_\mathsf{p}^{\mathsf{prv}}, V_\mathsf{p}^{\mathsf{prv}}, L_\mathsf{p})$ that stores current and historical information on attack, discounted, and total traffic volumes for a specific prefix p:

- $A_\mathsf{p}^{\mathsf{cur}}, D_\mathsf{p}^{\mathsf{cur}}, V_\mathsf{p}^{\mathsf{cur}} \in \mathbb{R}_{>0}$ denote current traffic volumes,
- $A_\mathsf{p}^{\mathsf{prv}}, D_\mathsf{p}^{\mathsf{prv}}, V_\mathsf{p}^{\mathsf{prv}} \in \mathbb{R}_{>0}$ denote historical traffic volumes,
- $L_\mathsf{p} \in \mathbb{N}_0$ denotes the **lifetime** of a tracker T_p and is a configurable system parameter.

A tracker T_p is considered to be **active** if $L_\mathsf{p} \neq 0$, otherwise it is **expired**.

Trackers use the weighting function $\mathsf{E} : (x, y) \mapsto \alpha x + (1-\alpha)y$ to combine current and historical traffic volumes x and y. The value of $\alpha \in [0, 1]$ constitutes a configurable system parameter.

Using smoothed traffic volume estimates enables a contribution of historical information to the discounted precision (cf. Equation 5.1) by substituting the variables A_p, D_p, and A_p with the result of the weighting function E:

$$\overline{\pi}_\mathsf{p} = \frac{A_\mathsf{p} - D_\mathsf{p}}{V_\mathsf{p} - D_\mathsf{p}} = \frac{\mathsf{E}(A_\mathsf{p}^{\mathsf{cur}}, A_\mathsf{p}^{\mathsf{prv}}) - \mathsf{E}(D_\mathsf{p}^{\mathsf{cur}}, D_\mathsf{p}^{\mathsf{prv}})}{\mathsf{E}(V_\mathsf{p}^{\mathsf{cur}}, V_\mathsf{p}^{\mathsf{prv}}) - \mathsf{E}(D_\mathsf{p}^{\mathsf{cur}}, D_\mathsf{p}^{\mathsf{prv}})} \tag{5.2}$$

By substituting A_p, D_p, and V_p, short-term variations in traffic volume can be compensated. The following example outlines how the legitimate flow f from Example 5.3.1 can be preserved using a simple weighting function.

Example 5.3.2. Consider the three prefixes q_1, q_2, and q_3 and the legitimate flow f from Example 5.3.1. For illustration, this example uses the weighting function $\mathsf{E} : (x, y) \mapsto \frac{1}{2}(x + y)$. The prefix p yields the discounted precision

$$\overline{\pi}_\mathsf{p} = \frac{\overbrace{\mathsf{E}(A_\mathsf{p}^{\mathsf{cur}}, A_\mathsf{p}^{\mathsf{prv}})}^{=\,2\cdot V_{\mathsf{q}_1}} - \overbrace{\mathsf{E}(D_\mathsf{p}^{\mathsf{cur}}, D_\mathsf{p}^{\mathsf{prv}})}^{=\,0}}{\underbrace{\mathsf{E}(V_\mathsf{p}^{\mathsf{cur}}, V_\mathsf{p}^{\mathsf{prv}})}_{<\,\frac{5}{2}\cdot V_{\mathsf{q}_1}} - \underbrace{\mathsf{E}(D_\mathsf{p}^{\mathsf{cur}}, D_\mathsf{p}^{\mathsf{prv}})}_{=\,0}} < \frac{4}{5}. \tag{5.3}$$

The pure attack traffic remains constant ($A_{\mathsf{p}} = A_{\mathsf{q}_1} + A_{\mathsf{q}_2} = 2 \cdot V_{\mathsf{q}_1}$) while the total traffic volume $V_{\mathsf{p}} = \mathsf{E}(V_{\mathsf{p}}^{\mathsf{cur}}, V_{\mathsf{p}}^{\mathsf{prv}})$ averages out over the alternating traffic volume of the flow f. Including the traffic of q_1 and q_2 and observing the fact that f is a high-volume flow ($V_{\mathsf{q}_2} > V_{\mathsf{q}_1}$), V_{p} remains below the upper bound $\frac{5}{2} \cdot V_{\mathsf{q}_1}$. Hence, enforcing a minimum precision $\pi^* > \frac{4}{5}$ preserves the legitimate traffic, since p is never blacklisted.

5.3.2 Tracker Lifecycle Management

New trackers are created whenever an HHH prefix p is detected that has no corresponding tracker T_{p}. While the mitigation is active, Filter Rule Refinement stores trackers of detected HHH prefixes in a **tracker hash table** $\mathcal{T}$. The tracker hash table uses prefixes as keys to enable fast lookup of trackers. This enables a fast mapping of detected HHHs to corresponding trackers. Furthermore, Filter Rule Refinement manages tracker lifecycles over time with Algorithm 3, specifically with the functions udpate_trackers and transition. Trackers are updated when new information from the HHH algorithm is available at the end of a monitoring interval. Before this, the function transition is invoked to erase expired trackers and to retain historical information. It updates current and historical traffic volumes whenever a monitoring interval elapses and the HHH algorithm provides new monitoring information. Furthermore, Filter Rule Refinement controls the creation and erasure of trackers.

Specifically, at the end of a monitoring interval, Filter Rule Refinement invokes the update_trackers function of Algorithm 3 after querying the HHH algorithm. When the HHH algorithm detects a previously untracked HHH prefix p, a new tracker T_{p} is created and initialized using the create_tracker function (starting in Line 1). The variables $A_{\mathsf{p}}^{\mathsf{prv}}$, $A_{\mathsf{p}}^{\mathsf{cur}}$, $V_{\mathsf{p}}^{\mathsf{prv}}$, $V_{\mathsf{p}}^{\mathsf{cur}}$ are initialized to an undefined state $\perp$ to indicate that no historical or current information is available. The lifetime is initialized to a constant value L_{init} and the newly created tracker is then inserted into the tracker hash table $\mathcal{T}$ using a hash of the prefix p as key. If the prefix p is already tracked in $\mathcal{T}$, the lifetime L_{p} of the existing tracker T_{p} is incremented instead (Line 12 of Algorithm 3). Repeatedly incrementing the lifetime serves to prevent loss of historical information for repeatedly detected HHH prefixes, since trackers are erased when they expire.

Next, line 13 and 14 of the update_trackers function update the current traffic volumes $V_{\mathsf{p}}^{\mathsf{cur}}$ and $A_{\mathsf{p}}^{\mathsf{cur}}$. The value of $V_{\mathsf{p}}^{\mathsf{cur}}$ is obtained from the HHH algorithm (indicated by the get_volume function in Line 13). The current value of $A_{\mathsf{p}}^{\mathsf{cur}}$ is calculated from the

Algorithm 3: Tracker Lifecycle Management

Input : tracker hash table $\mathcal{T}$, ▷ *Maintained by Filter Rule Refinement*
attack traffic volume estimator $\mathscr{E}$,
weighting function E

1 **Function** create_tracker(prefix p): ▷ *Create initialized tracker*
2 $\mathsf{T_p} \leftarrow$ **new** tracker();
3 $\mathsf{T_p}.A_\mathsf{p}^\mathsf{prv} \leftarrow \mathsf{T_p}.A_\mathsf{p}^\mathsf{cur} \leftarrow \mathsf{T_p}.V_\mathsf{p}^\mathsf{prv} \leftarrow \mathsf{T_p}.V_\mathsf{p}^\mathsf{cur} \leftarrow \bot$;
4 $\mathsf{T_p}.L_\mathsf{p} \leftarrow L_\mathsf{init}$;
5 **return** $\mathsf{T_p}$;

6 **Function** update_trackers(HHH result $\mathcal{H}$):
7 **foreach** prefix p **in** HHH result $\mathcal{H}$ **do**
8 **if not** $\mathcal{T}$.contains(p) **then** ▷ *Check if tracker exists*
9 $\mathsf{T_p} \leftarrow \mathcal{T}$.insert(p, create_tracker(p)); ▷ *Initialize new tracker*
10 **else**
11 $\mathsf{T_p} \leftarrow \mathcal{T}$.find(p);
12 $\mathsf{T_p}.L_\mathsf{p} \leftarrow \mathsf{T_p}.L_\mathsf{p} + 1$; ▷ *Increment tracker lifetime*
13 $\mathsf{T_p}.V_\mathsf{p}^\mathsf{cur} \leftarrow \mathcal{H}$.get_volume(p); ▷ *Update $V_\mathsf{p}^\mathsf{cur}$ from HHH query*
14 $\mathsf{T_p}.A_\mathsf{p}^\mathsf{cur} \leftarrow \mathsf{T_p}.V_\mathsf{p}^\mathsf{cur} \cdot \mathscr{E}$(p, $\mathcal{H}$.get_features(p)); ▷ *Estimate $A_\mathsf{p}^\mathsf{cur}$ using $\mathscr{E}$*

15 **Function** transition():
16 **foreach** p **in** $\mathcal{T}$ **do**
17 $\mathsf{T_p} \leftarrow \mathcal{T}$.find(p);
18 **if** $\mathsf{T_p}.L_\mathsf{p} = 0$ **then**
19 $\mathcal{T}$.erase($\mathsf{T_p}$); ▷ *Erase expired tracker*
20 **else**
21 $\mathsf{T_p}.A_\mathsf{p}^\mathsf{prv} \leftarrow \mathsf{E}(\mathsf{T_p}.A_\mathsf{p}^\mathsf{cur}, \mathsf{T_p}.A_\mathsf{p}^\mathsf{prv})$; ▷ *Retain historical information*
22 $\mathsf{T_p}.V_\mathsf{p}^\mathsf{prv} \leftarrow \mathsf{E}(\mathsf{T_p}.V_\mathsf{p}^\mathsf{cur}, \mathsf{T_p}.V_\mathsf{p}^\mathsf{prv})$;
23 $\mathsf{T_p}.L_\mathsf{p} \leftarrow \mathsf{T_p}.L_\mathsf{p} - 1$; ▷ *Decrement tracker lifetime*

volume $V_\mathsf{p}^\mathsf{cur}$ and the traffic purity estimated by an attack traffic volume estimator $\mathscr{E}$. The estimator $\mathscr{E}$ receives aggregate features along with the prefix p as input.

At the end of the monitoring interval, Filter Rule Refinement invokes the transition function of Algorithm 3 (starting in Line 15). The transition function updates $V_\mathsf{p}^\mathsf{prv}$ and $A_\mathsf{p}^\mathsf{prv}$ to retain information on the traffic volumes $V_\mathsf{p}^\mathsf{cur}$ and $A_\mathsf{p}^\mathsf{cur}$. In addition, the transition function decrements the lifetime of a tracker. When a tracker's lifetime L is not incremented by the udpate_trackers function in every monitoring interval, its initial

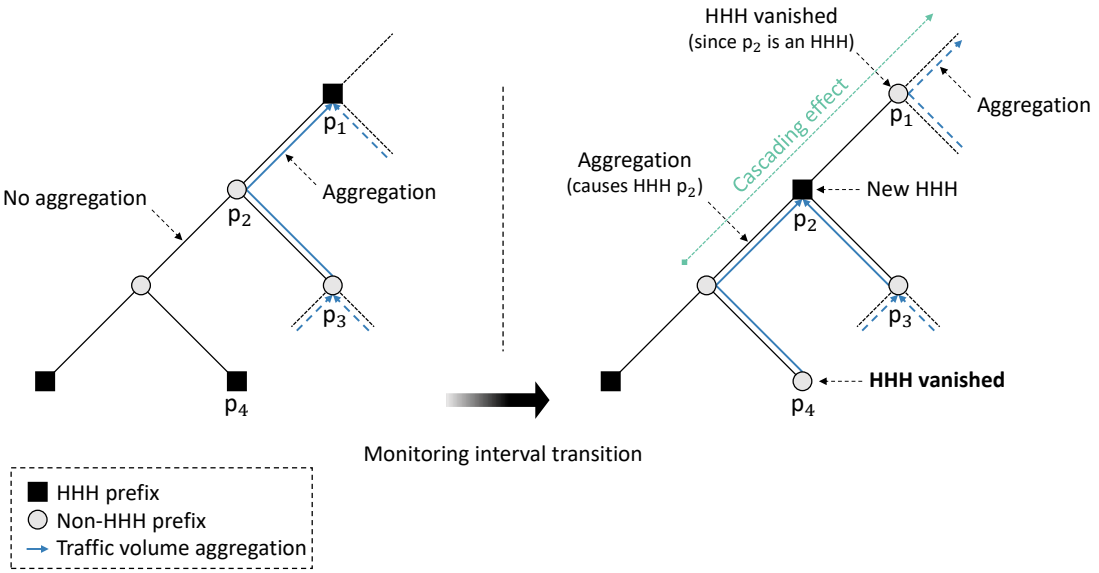


Figure 5.4: Example of a vanishing HHH prefix p_3 causing the creation and vanishing of parent HHH prefixes (p_2 and p_3).

lifetime L_{init} eventually expires. Such expired trackers are removed in line 21. The removal of trackers serves to avoid monotonic growth of the tracker hash table $\mathcal{T}$.

5.3.3 Traffic Volume Substitutions

If HHH prefixes change frequently due to HHH instability, not all the information required to calculate a discounted precision is available. If an HHH prefix vanishes no information on *current* attack and total traffic volumes A_p^{cur} and V_p^{cur} is available. Conversely, no information on *historical* attack and total traffic volumes A_p^{prv} and V_p^{prv} is available when a previously untracked HHH prefix occurs. This limits the calculation of discounted precision to momentary observations, either the (potentially outdated) information on A_p^{prv} and V_p^{prv} or the current information A_p^{cur} and V_p^{cur}.

The effect of a single vanishing or newly detected HHH prefix is not isolated to that specific prefix. Instead, HHH instability can *cascade* over the hierarchy as outlined in the following example.

Example 5.3.3. Consider the prefixes $p_1, \ldots, p_4$ in the hierarchy illustrated in Figure 5.4. When transitioning to a new monitoring interval, the prefix p_4 is no longer detected as an HHH prefix. This causes its traffic volume to be aggregated into

prefix p_2 and (together with the traffic volume of p_3) results in p_2 becoming an HHH prefix (by reaching the HHH detection threshold $\phi\mathcal{V}$). In turn, p_2's parent prefix p_1 is no longer an HHH prefix when its traffic volume drops below $\phi\mathcal{V}$ (since it no longer aggregates p_2's traffic volume). This effect can cascade upward through the hierarchy.

The cascading effect of vanishing or appearing HHH prefixes on filter rules is amplified in dynamic traffic scenarios when the following situations occur:

- Multiple HHH prefixes vanish or appear simultaneously, causing numerous changes throughout the hierarchy. In particular, changes in longer HHH prefixes can have a strong cascading effect throughout the hierarchy, since they can affect the entire path to the root prefix.
- A vanishing or appearing HHH prefix at lower level has a direct impact on a distant parent prefix. Consider the path connecting prefixes p_2 and p_4 in Example 5.3.3. This path can be much longer when the prefixes are on distant hierarchy levels. When the path is free of other HHH prefixes, the traffic volume of p_4 can aggregate over the entire path up to p_2. Thus, a vanishing or appearing HHH prefix with small address space coverage can cause an HHH prefix with wide address space coverage to appear or disappear.

Consequently, HHH instability can frequently disrupt the availability of historical and current monitoring information. This effect destabilizes precision estimates and can lead to accepting ineffective filter rules or rejecting effective filter rules. It can, in turn, lead to high short-term increases in false positive and false negative rates. However, this issue can be addressed by utilizing information stored in existing trackers of child prefixes. Their traffic volume information can (partially) substitute for current attack and total traffic volumes. Specifically, if a prefix p has longer child prefixes q in a tracker hash table $\mathcal{T}$, their aggregated attack and legitimate traffic volumes can be used as **traffic volume substitutions**:

$$\overline{A}_p^{cur} = \sum_{q \prec p, q \in \mathcal{T}} A_q^{cur}, \tag{5.4}$$

$$\overline{V}_p^{cur} = \sum_{q \prec p, q \in \mathcal{T}} V_q^{cur}. \tag{5.5}$$

The quantities $\overline{A}_p^{cur}$ and $\overline{V}_p^{cur}$ provide a (partial) reconstruction of momentarily unavailable traffic volumes A_p^{cur} and V_p^{cur} from other currently available monitoring

information. Thus, they can substitute for the corresponding quantities in the calculation of smoothed discounted precision (in Equation 5.2). The computation of $\overline{A}_{\mathsf{p}}^{\mathsf{cur}}$ and $\overline{V}_{\mathsf{p}}^{\mathsf{cur}}$ follows the same principle and order of iteration over prefixes as the calculation of discounted precision. Therefore, traffic volume substitutions can be calculated in the same pass over the tracker hash map $\mathcal{T}$, eliminating the need to perform multiple iterations during filter prefix generation.

However, exact reconstruction of $A_{\mathsf{p}}^{\mathsf{cur}}$ and $V_{\mathsf{p}}^{\mathsf{cur}}$ is not guaranteed. Child prefixes may cover only a part of the subnet corresponding to the prefix p. In this case, traffic that originates from the uncovered portion of that subnet is not accounted for by $\overline{A}_{\mathsf{p}}^{\mathsf{cur}}$ and $\overline{V}_{\mathsf{p}}^{\mathsf{cur}}$. Instead, it is aggregated higher up in the hierarchy. Consequently, $\overline{A}_{\mathsf{p}}^{\mathsf{cur}}$ and $\overline{V}_{\mathsf{p}}^{\mathsf{cur}}$ can *underestimate* true traffic volumes. In the case where $A_{\mathsf{p}}^{\mathsf{cur}}$ is underestimated, this leads to the rejection of a potentially effective filter rule due to lower $\overline{\pi}_{\mathsf{p}}$ (cf. Equation 5.2). An underestimation of $V_{\mathsf{p}}^{\mathsf{cur}}$ can lead to an unintentional blacklisting of a prefix that affects legitimate traffic due to higher $\overline{\pi}_{\mathsf{p}}$. The effect of underestimating $\overline{V}_{\mathsf{p}}^{\mathsf{cur}}$ is intensified by fact that the discount $D_{\mathsf{p}}^{\mathsf{cur}}$ reduces both the numerator and denominator in Equation 5.2. Hence, an underestimation has a stronger effect on discounted precision than it has on the non-discounted precision $\pi_{\mathsf{p}} = A_{\mathsf{p}}/V_{\mathsf{p}}$.

Underestimating $\overline{V}_{\mathsf{p}}^{\mathsf{cur}}$ is particularly problematic if it leads to overestimating $\overline{\pi}_{\mathsf{p}}$ for a *short* prefix p due to the potential impact on a large subnet. The impact of underestimating $\overline{V}_{\mathsf{p}}^{\mathsf{cur}}$ on $\overline{\pi}_{\mathsf{p}}$ can be reduced by providing a more conservative, potentially higher estimate based on information from previous monitoring intervals. Specifically, by replacing $\overline{V}_{\mathsf{p}}^{\mathsf{cur}}$ in Equation 5.5 with

$$\overline{V}_{\mathsf{p}}^{\mathsf{cur}} = \max\left(\sum_{\mathsf{q}\prec\mathsf{p},\mathsf{q}\in\mathcal{T}} V_{\mathsf{q}}^{\mathsf{cur}}, V_{\mathsf{p}}^{\mathsf{prv}}\right). \tag{5.6}$$

Relying on $V_{\mathsf{p}}^{\mathsf{prv}}$ can provide a short-term overestimation of $V_{\mathsf{p}}^{\mathsf{cur}}$ when the true total traffic volume drops between monitoring intervals. This is useful to reduce the risk of overestimating $\overline{\pi}_{\mathsf{p}}$, which in turn can result in the removal of legitimate traffic by blacklisting ineffective prefixes.

When a new HHH prefix p is detected and no corresponding tracker T_{p} exists, no information on historical traffic volumes is available. In this case, substitutions for historical traffic volumes $A_{\mathsf{p}}^{\mathsf{prv}}$ and $V_{\mathsf{p}}^{\mathsf{prv}}$ can be calculated in a similar manner:

$$\overline{A}_{p}^{prv} = \sum_{q \prec p, q \in \mathcal{T}} A_{q}^{prv}, \tag{5.7}$$

$$\overline{V}_{p}^{prv} = \max\left(\sum_{q \prec p, q \in \mathcal{T}} V_{q}^{prv}, V_{p}^{cur} \right). \tag{5.8}$$

With these traffic volume substitutions, the calculation of $\overline{\pi}_p$ can include (partial) current and historical information in the absence of corresponding HHH prefixes. For this, it leverages the hierarchical structure of the IP prefix hierarchy to utilize information that is stored in trackers of child prefixes. This alleviates the need to exclusively rely on momentary observations and reduces the risk to incur short-term high false positive and false negative rates.

5.3.4 Blacklist Generation

A new blacklist is generated at the end of each monitoring interval after monitoring traffic with the HHH algorithm and updating trackers with the update function of Algorithm 3. During blacklist generation, the discounted precision of currently tracked prefixes is calculated and compared to the minimum required precision π^*. For this, iteration proceeds over all trackers currently stored in the tracker hash map $\mathcal{T}$ from longest to shortest prefixes. This order of iteration resolves the dependency of discounted precision on aggregated discounts of child prefixes. At the same time, traffic volume substitutions can be computed while progressing in this order. This eliminates the need to repeatedly search the tracker hash table $\mathcal{T}$ for currently stored trackers of child prefixes to compute the sums in Equations 5.4, 5.6, 5.7, and 5.8.

The steps involved in blacklist generation are outlined in the generate_blacklist function of Algorithm 4. Initially, the blacklist $\mathcal{B}$ comprises an empty list of prefixes (in Line 2). Calculation of discounted precision in Line 10 for a prefix p requires the aggregated discount, as well as traffic volume substitutions from child prefixes. To calculate the required values while progressing through hierarchy levels, generate_blacklist uses a hash table $\mathcal{A}$ (initialized in Line 3), which stores an **aggregate vector** A_p for each processed prefix p of a specific prefix length L:

$$\mathsf{A}_p = \left(A_p^{prv}, A_p^{cur}, D_p^{prv}, D_p^{cur}, V_p^{prv}, V_p^{cur} \right).$$

Algorithm 4: Blacklist Generation

Input : tracker hash table $\mathcal{T}$, ▷ *Maintained by Filter Rule Refinement*
weighting function E,
maximum prefix length $\mathsf{L}^{\max}$

```
1  Function generate_blacklist(minimum precision π*):
2      B ← new list();
3      A ← new hash_table();
4      for prefix_length L = L^max downto 0 do
5          if L ≠ L^max then
6              A ← generalize_aggregates(A);          ▷ Propagate volumes to parents
7          foreach prefix p, tracker T_p in T with p.length = L do
8              A_p ← find_or_create_aggregate(p, A);
9              T_p ← substitute_volumes(T_p, A_p);    ▷ Substitute missing information
10             π̄_p ← (E(T_p.A_p^prv, T_p.A_p^cur) − E(A_p.D_p^cur, A_p.D_p^prv)) /
                     (E(T_p.V_p^prv, T_p.V_p^cur) − E(A_p.D_p^cur, A_p.D_p^prv));  ▷ Discounted precision
11             if π̄_p ≥ π* then
12                 B.insert(p);                       ▷ Blacklist prefix p
13                 update_discounts(A_p, T_p);        ▷ Memorize D_p^prv, D_p^cur
14             update_substitutions(A_p, T_p);        ▷ Memorize A_p^prv, A_p^cur, V_p^prv, V_p^cur
15     B.delete_redundant_filter_rules();
16     return B;
```

When transitioning to a lower prefix length (in Line 4), traffic volume substitutions and discounts for parent prefixes are determined through point-wise addition of aggregate vectors of their immediate child prefixes. For this, the function generalize_-aggregates of Algorithm 5 determines the parent prefix par of a prefix p. It then performs a lookup into the hash table $\mathcal{A}$ to retrieve the aggregate vector $\mathsf{A}_{\mathsf{par}}$. The find_or_create_aggregate function of Algorithm 5 either retrieves the corresponding stored vector or creates a new aggregate vector for par if none exists. All values of the new entries are initialized to zero to indicate that no prior discounts or traffic volume substitutions have been accumulated. The traffic volumes stored in A_p are afterwards added to the aggregate vector $\mathsf{A}_{\mathsf{par}}$ of the parent prefix (Line 12 of Algorithm 5). Thus, each parent prefix accumulates the traffic volumes of all immediate child prefixes that have aggregate vectors in the hash table $\mathcal{A}$. When transitioning to a shorter prefix length, the generate_aggregates function replaces the current hash table $\mathcal{A}$ with the

Algorithm 5: Aggregate Generalization

```
1  Function find_or_create_aggregate(prefix p, hash_table 𝒜):
2      A_p ← 𝒜.find(p);
3      if A_p = ⊥ then                          ▷ Create zero-initialized entry if none exists
4          A_p ← new aggregate();
5          A_p.A_p^prv ← A_p.A_p^cur ← T_p.D_p^prv ← T_p.D_p^cur ← T_p.V_p^prv ← T_p.V_p^cur ← 0;
6          𝒜.insert(p, A_p);
7      return A_p;
8  Function generalize_aggregates(hash_table 𝒜):    ▷ Propagate volumes to parent
9      t ← new hash_table();                        ▷ Create temporary hash table
10     foreach prefix p, aggregate A_p in 𝒜 do
11         A_par ← find_or_create_aggregate(get_parent(p), 𝒜);
12         A_par ← A_par + A_p;                     ▷ Pointwise addition of A_par and A_p
13     return t;
```

Algorithm 6: Traffic Volume Substitution

```
1  Function substitute_volumes(tracker T_p, aggregate A_p):
2      if T_p.A_p^prv = ⊥ then T_p.A_p^prv ← A_p.A_p^prv;
3      if T_p.A_p^cur = ⊥ then T_p.A_p^cur ← A_p.A_p^cur;
4      if T_p.A_p^prv = ⊥ then T_p.V_p^prv ← max(A_p.V_p^prv, T_p.V_p^cur);
5      if T_p.A_p^cur = ⊥ then T_p.V_p^cur ← max(A_p.V_p^cur, T_p.V_p^prv);
6      return T_p;
```

new hash table of aggregated values that is returned by generalize_aggregates (in Line 6 of Algorithm 4).

Once the aggregated discounts and traffic volume substitutions for prefix length L have been determined, the inner loop (Line 7) of generate_blacklist can determine the discounted precision of all prefixes of length L. If trackers have no current or historical information, the traffic volume substitutions can be retrieved from previously computed aggregate vectors through the substitute_volumes function of Algorithm 6. Recall that traffic volumes of a tracker T are initialized with an undefined state when a tracker is first created (in Line 3 of Algorithm 3). Hence, missing information can be identified and substituted. The function substitute_volumes replaces undefined values of A_p^{prv}, A_p^{cur}, V_p^{prv}, and V_p^{cur} in T_p with the respective aggregated values in A_p.

Algorithm 7: Aggregate Vector Updates

```
1  Function update_discounts(aggregate A_p, tracker T_p):
2      A_p.D_p^prv ← T_p.D_p^prv;
3      A_p.D_p^cur ← T_p.D_p^cur;
4      return A_p;
5  Function update_substitutions(aggregate A_p, tracker T_p):
6      A_p.A_p^prv ← T_p.A_p^prv;
7      A_p.A_p^cur ← T_p.A_p^cur;
8      A_p.V_p^prv ← T_p.V_p^prv;
9      A_p.V_p^cur ← T_p.V_p^cur;
10     return A_p;
```

The values stored in A_p correspond to the traffic volume substitutions $\overline{A}_\mathsf{p}^{\mathsf{prv}}$, $\overline{A}_\mathsf{p}^{\mathsf{cur}}$, $\overline{V}_\mathsf{p}^{\mathsf{prv}}$, and $\overline{V}_\mathsf{p}^{\mathsf{cur}}$ (from Equation 5.4, 5.6, 5.7, and 5.8).

After traffic volume substitution has been performed, the generate_blacklist function calculates the discounted precision $\overline{\pi}_\mathsf{p}$ of a prefix p according to Equation 5.2 using the substituted values of T_p as well as the aggregated discounts $\mathsf{A}_\mathsf{p}.D_\mathsf{p}^{\mathsf{prv}}$ and $\mathsf{A}_\mathsf{p}.D_\mathsf{p}^{\mathsf{cur}}$. If a prefix p reaches the required minimum precision π^*, it is inserted into the blacklist $\mathcal{B}$.

The last steps during iteration over prefixes of a specific length are to update the discounts $D_\mathsf{p}^{\mathsf{prv}}$ and $D_\mathsf{p}^{\mathsf{cur}}$ as well as the traffic volumes $A_\mathsf{p}^{\mathsf{prv}}$, $A_\mathsf{p}^{\mathsf{cur}}$, $V_\mathsf{p}^{\mathsf{prv}}$, and $V_\mathsf{p}^{\mathsf{cur}}$ stored in A_p. For this, the two functions update_discounts and update_substitutions of Algorithm 7 are invoked in Line 13 and Line 14 of Algorithm 4. The update of discounts $D_\mathsf{p}^{\mathsf{prv}}$ and $D_\mathsf{p}^{\mathsf{cur}}$ depends on a prefix p reaching the minimum precision π^* as in Equation 5.1. Hence, only blacklisted prefixes are considered in the discount. In contrast, the update_substitutions function is invoked unconditionally, to collect all available information from child prefixes when reconstructing missing monitoring information.

In a final step, redundant filter rules are removed from the blacklist. For this, all prefixes p in the blacklist are repeatedly generalized, and a hash lookup in $\mathcal{T}$ is performed. If a match is found, the (shorter) matching prefix already covers the subnet of the prefix p, so that p can be deleted.

5.4 Evaluation

The following evaluation assesses how Filter Rule Refinement used in conjunction with Traffic Purity Estimation influences the FPR, FNR, BSIZE metrics. The model $\mathsf{M}_{\mathsf{TOP}}$ from Section 4.5.4 serves as traffic purity estimator $\mathscr{E}$ (see Algorithm 3). This evaluation uses constant parameter choices for ϕ and π^*. Automatic parameter adaptations over the course of a traffic scenario requires a trained RL agent, which is introduced in Chapter 6.

The evaluation is structured into three experiments:

(1) Traffic filtering with discounted precision, but without the use of trackers.

(2) Traffic filtering with trackers, but without traffic volume substitutions.

(3) Traffic filtering with complete Filter Rule Refinement.

Incrementally adopting the mechanisms of Filter Rule Refinement serves to assess the impact of the design decisions on filtering effectiveness.

5.4.1 Mitigation Scenario

This evaluation introduces a mitigation scenario C that is different from the scenarios A and B used for Traffic Purity Estimation (in Section 4.5). The purpose of scenario C is to enable a comparison of the FPR, FNR, and BSIZE achieved during each monitoring interval across different experiments. To compare traffic filtering on different attack vectors, it is necessary to restrict their occurrence to specific time frames. This prevents the use of randomized time shifts that served to increase the diversity of scenario A and B.

The mitigation scenario is a variation of scenario B from Section 4.5.1, which comprised the test dataset for Traffic Purity Estimation. Scenario C separates highly distributed attacks from attacks with fewer high-intensity traffic sources by dividing the scenario duration into four phases ① to ④. The phases extend over 225 seconds each, which corresponds to one fourth of the 15-minute duration of a legitimate MAWI traffic trace. Each phase begins at a specific time to enable a direct comparison of the effectiveness of filter rules across different experiments. In addition, scenario C changes further characteristics of scenario B to evaluate if filter rules remain effective

Scenario	Stream	Phase	Stream characteristics						
			IP source addr. distribution[1]	#IP source addresses	Flow start time [s]	Packets per source	Frame size [bytes]	Transport protocol	Source port
C	SSDP	①	$\mathcal{U}\left[0, 2^{32}-1\right)$	250	$\mathcal{U}[0,135)$	1000	$\mathcal{N}(347, 9.1)$	UDP	1900
	NTP	②	$\mathcal{U}\left[0, 2^{32}-1\right)$	250	$\mathcal{U}[225,360)$	1000	$\mathcal{N}(481.1, 10)$	UDP	123
	UDP_1	③	$\mathcal{N}\left(2^{31}, \sigma\right)$	10000	$\mathcal{U}[450,585)$	180	$\mathcal{U}[512, 1496)$	UDP	$\mathcal{U}[1024, 49151]$
	UDP_2	④	$\mathcal{N}\left(2^{31}, \sigma\right)$	5000	$\mathcal{U}[675,810)$	180	$\mathcal{U}[60, 1496)$	UDP	$\mathcal{U}[49152, 65535]$
	OVPN	④	$\mathcal{N}\left(2^{31}, \sigma\right)$	5000	$\mathcal{U}[675,810)$	180	$\mathcal{N}(64.5, 0.3)$	UDP	1194
	LEGIT_2	MAWI traffic to destination IP address `202.58.173.226`							

[1]The variable σ denotes a randomized standard deviation $\sigma \sim \mathcal{U}[0.02, 0.22) \cdot 2^{32}$.

Table 5.1: Traffic characteristics of the mitigation scenario C. Different traffic characteristics between scenario C and scenario B from Section 4.5.1 are highlighted in blue. Flow durations are the same as in scenario A and B (see Table 4.2).

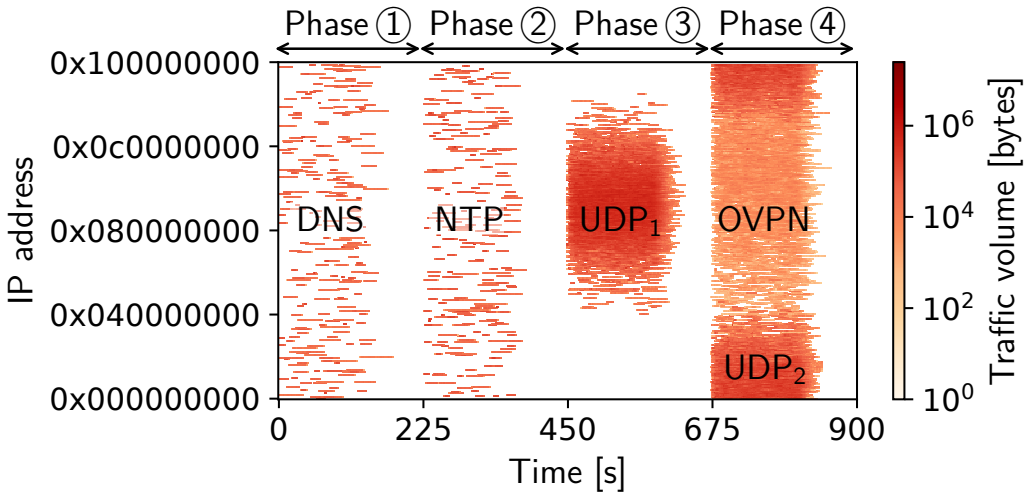


Figure 5.5: Distribution of attack traffic volume across four phases in the mitigation scenario C. The four phases have 225 second durations each. In phase ④, two attack vectors occur simultaneously.

when traffic characteristics change in comparison to the traffic used to train and test Traffic Purity Estimation models.

Table 5.1 lists the stream characteristics of scenario C. The differences between scenario B and C are highlighted, while flow durations are omitted from the table as they remain unchanged.

- Scenario C restricts flow start times to specific intervals to partition the scenario into four phases. The first two phases (starting at 0 and 255 seconds) model attacks that originate from few traffic sources, which send high traffic volumes. The last two phases (starting at 450 and 675 seconds) model highly distributed attacks, where each source sends fewer packets.

- Scenario C uses two UDP-based streams UDP_1 and UDP_2 with randomized source ports instead of the TCP-based streams TCP_4 and TCP_5. This serves to change traffic characteristics in comparison to scenario B.

- In phase ④, scenario C uses the stream UDP_2 together with the stream OVPN that models an OpenVPN amplification attack. This serves to model a distributed attack with two attack vectors in contrast to the single attack vector in phase ③.

Parameter	ϕ	π^*	α	L_{init}
Value	10^{-3}	0.7	0.45	3

Table 5.2: Parameters used for experiments in Sections 5.4.2, 5.4.3, and 5.4.4.

Figure 5.5 visualizes the distribution of the attack traffic in scenario C over time and over the IPv4 address space. As shown, the four phases align with the flow start times specified in Table 5.1 and remain separated across time. Due to the lower number of traffic sources, the SSDP and NTP streams appear more sparsely distributed over the IP address space. In phase ④, the traffic of stream UDP_2 overlaps with the (widely-distributed) traffic of stream OVPN. This is due to randomization of the variance σ of the IP source address distribution. Another random choice of σ can cause a separation of the two streams, leading to attacks from two distinct regions of the address space. Similar to the stream OVPN, traffic of the streams UDP_1 and UDP_2 can span across larger or smaller regions for different choices of σ. This occurs independently for each source address distribution.

Variations in the distribution of attack traffic over the address space can result in different mixtures with legitimate traffic in monitored aggregates. In phase ④, this can also lead to aggregate features that are composed of a mixture of two different attack vectors. The combination can influence the estimation of traffic purity, which may result in different filter rules and variations in FPR, FNR, and BSIZE.

5.4.2 Traffic Filtering with Discounted Precision

The first experiment calculates the discounted precision from Definition 5.2 while generating blacklists for traffic filtering in scenario C. However, no historical information ($A_{\text{p}}^{\text{prv}}$ or $V_{\text{p}}^{\text{prv}}$) is used and no calculation of exponential weighted moving averages occurs. This effectively avoids the use of trackers, while discounts are still applied to adjust the attack and total traffic volumes $A_{\text{p}}^{\text{cur}}$ and $V_{\text{p}}^{\text{cur}}$.

Table 5.2 provides the values of ϕ and π^* that are used for generating filter rules. The choice of ϕ encourages the use of filter rules with short prefixes, while the value of π^* balances FPR against FNR. This serves to evaluate design decisions when trying to generate compact blacklists. The monitoring interval has a duration of one second and the HHH algorithm of Aggregate Monitoring tracks traffic volumes and aggregate features of at most 1000 prefixes. The experiment (and all further

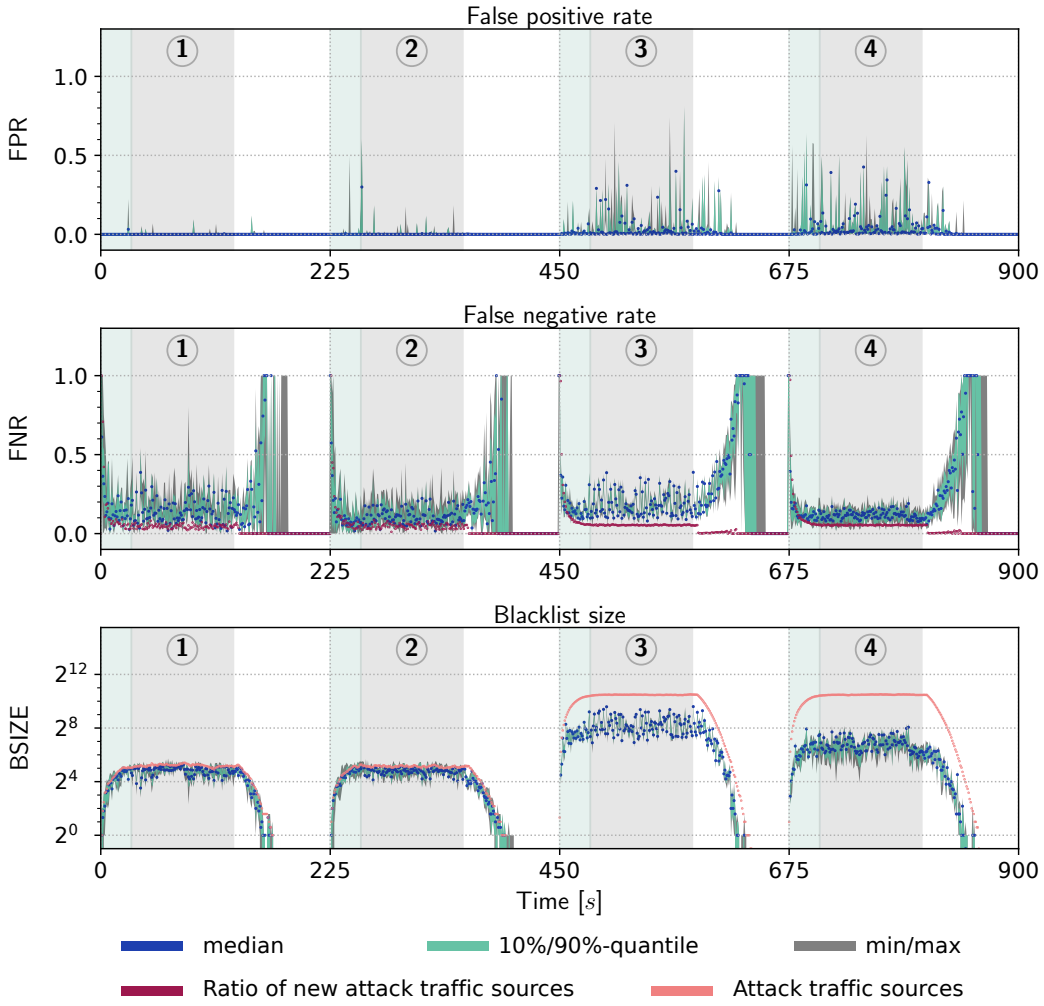


Figure 5.6: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes when filtering traffic in scenario C without using trackers. Green regions within the four phases ① to ④ indicate a 20-second ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

experiments) summarize results over ten randomized episodes of the mitigation scenario. Randomization is initialized with a fixed seed to ensure comparability with subsequent experiments.

The metrics for FPR, FNR, BSIZE are calculated at the end of each interval. Their progression during the scenario is shown in Figure 5.6. The number of attack traffic sources that are active during each interval serves as a baseline for the maximum

Metric		Phase				Entire scenario duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0076	0.0139	0.0000
	mean	0.0009	0.0057	0.0438	0.0535	0.0164
	90 %-quantile	0.0001	0.0020	0.1187	0.1446	0.0297
	maximum	0.0942	0.5991	0.8089	0.6266	0.8089
FNR	median	0.1304	0.1008	0.1643	0.1154	0.1163
	mean	0.1483	0.1133	0.1842	0.1163	0.1718
	90 %-quantile	0.2761	0.2089	0.3028	0.1723	0.3719
	maximum	0.8020	0.4456	0.4121	0.2470	1.0000
BSIZE	median	30.0	28.0	280.5	105.0	28.0
	mean	29.1	28.1	323.0	114.8	75.5
	90 %-quantile	39.0	36.1	562.1	190.1	224.0
	maximum	53.0	51.0	798.0	324.0	821.0

Table 5.3: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE when traffic filtering without trackers. The table shows metrics for the stable states of the four phases of scenario C as well as the entire scenario duration (calculated over ten episodes). High values of FPR and FNR are highlighted in blue.

blacklist size. The ratio of new attack traffic sources quantifies the portion of all attack traffic sources that are new in the corresponding monitoring interval. It provides as a baseline for the false negative rate, since filter rules can only be generated for these sources after they have been detected. This means that filter rules for new attack sources are typically established when the monitoring interval elapses after one second. Still, the FNR can be below the ratio of new attack traffic sources because already established filter rules can discard traffic from new sources.

Table 5.3 summarizes the median, mean, 90 %-quantile, and maximum values of FPR, FNR, and BSIZE for the stable states during each of the four phases (highlighted in gray in Figure 5.6). In addition, corresponding metrics for the entire duration of scenario C are provided.

The mean FPR during phases ① and ② remains low with the exception of temporarily high values at the start of phase ②. During the last two phases, it becomes more challenging to maintain a low FPR as the number of attack traffic sources increases.

The value of ϕ leads to short filter rule prefixes and a reduction of BSIZE below the number of attack traffic sources in phase ③ and ④. This incurs a greater risk of a removing legitimate traffic, as the filter rules cover larger address space regions. As a result, the FPR is frequently above 11 %, as shown by the 90 %-quantile, and the maximum FPR reaches values above 62 %.

Figure 5.6 shows that the FNR is occasionally high during the ramp-up phase of an attack (the first 20 seconds of each phase) and the cool-down phase of an attack (colored in green and white in Figure 5.6). At the start of an attack, attack traffic sources must be detected before filter rules are generated, which causes a short delay before the FNR decreases rapidly. Furthermore, during the ramp-up and cool-down phases, only few attack traffic sources are active, which results in a small divisor being used when calculating FNR (see Definition 3.5). Hence, high false negative rates can occur (especially over the entire scenario duration) when only a few attack traffic sources are missed by traffic filtering. Since the total attack traffic volume is low at the start and end of an attack, the elevated FNR is not critical. However, the FNR in the stable state of phases ① through ③ is consistently higher than the ratio of new attack traffic sources. The mean and median FNR remain above 10 % in all three phases with considerably higher values for the 90 %-quantile and the maximum (above 20 % and 41 %).

The elevated FPR and FNR during the stable state states result from short-term changes in the attack and legitimate traffic. In particular, no historical information can be taken into account to remember the location of legitimate traffic sources in the IP address space. When a legitimate traffic source temporarily reduces its traffic volume, a filter rule for the corresponding address space region can reach a momentarily high discounted precision. This causes newly generated filter rules to remove legitimate traffic when a traffic source increases its traffic volume again. The high FNR is a result of not remembering attack traffic sources. A temporary decrease in the attack traffic volume leads to momentarily reduced discounted precision and the withdrawal of filter rules. Due to this, a portion of the attack traffic can bypass the filter rules.

5.4.3 Traffic Filtering without Traffic Volume Substitutions

The second experiment uses trackers that calculate exponential weighted moving averages from A_p^{prv}, V_p^{prv}, A_p^{cur} and V_p^{cur}, as well as the discounted precision. However, no traffic volume substitutions are performed. In addition to the previously used parameters, trackers use the values α and L_{init} from Table 5.2. The value $\alpha = 0.45$ is

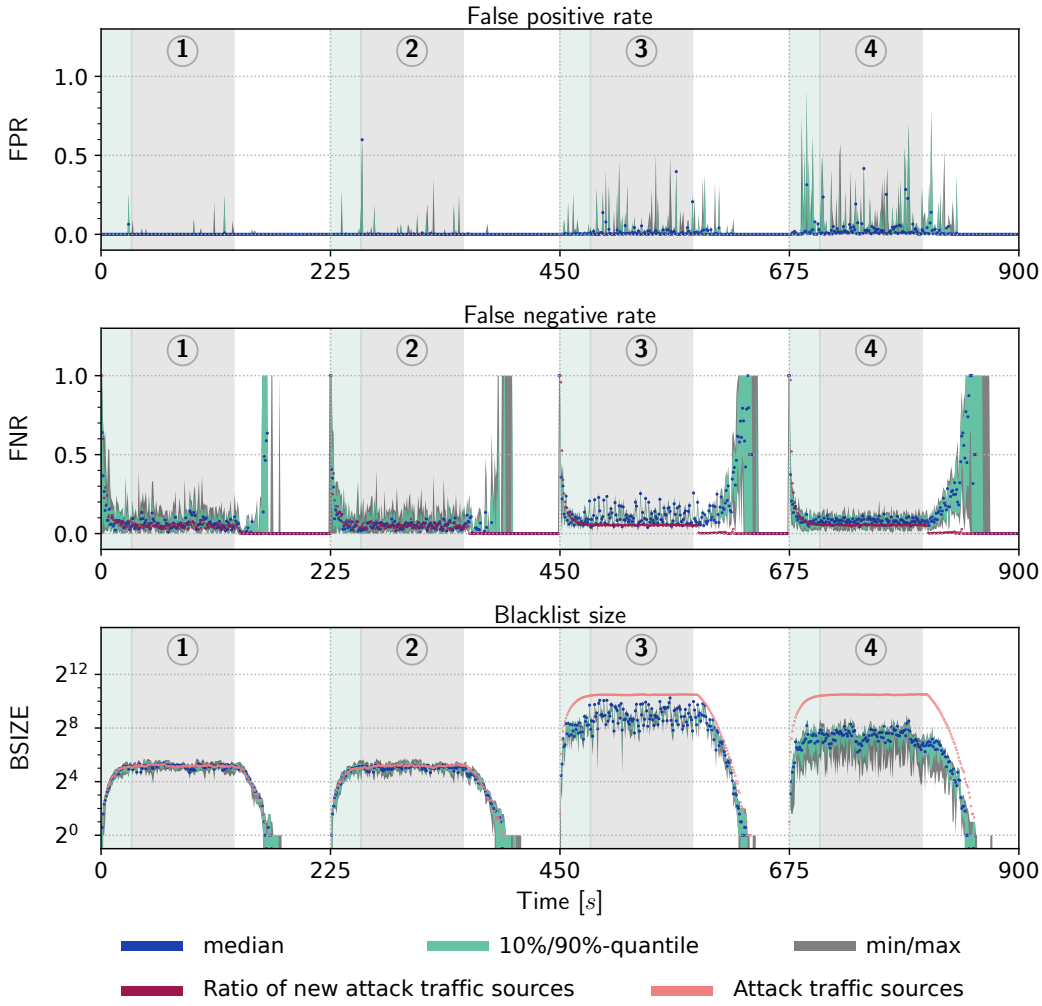


Figure 5.7: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes when filtering traffic in scenario C without using traffic volume substitutions. Green regions within the four phases ① to ④ indicate a 20-second ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

chosen to balance new monitoring information against historical information. The initial tracker lifetime L_{init} causes tackers to expire when no attack traffic is present over a short amount of time.

Figure 5.7 shows the progression of FPR, FNR, and BSIZE over the course of the mitigation scenario. Table 5.4 provides the median, mean, 90 %-quantile, and maximum for FPR, FNR, and BSIZE during the stable states of the four attack phases.

Metric		Phase				Entire scenario duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0024	0.0114	0.0000
	mean	0.0018	0.0078	0.0220	0.0441	0.0130
	90 %-quantile	0.0001	0.0022	0.0421	0.1094	0.0203
	maximum	0.2458	0.5991	0.5073	0.7055	0.9159
FNR	median	0.0571	0.0487	0.0941	0.0783	0.0615
	mean	0.0700	0.0602	0.1085	0.0777	0.1024
	90 %-quantile	0.1420	0.1330	0.1935	0.1233	0.2018
	maximum	0.3937	0.4034	0.2872	0.2158	1.0000
BSIZE	median	36.0	34.0	474.0	166.0	34.5
	mean	35.4	34.1	533.3	173.8	117.3
	90 %-quantile	43.0	42.0	907.1	292.0	364.0
	maximum	53.0	55.0	1259.0	514.0	1259.0

Table 5.4: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE when not performing traffic volume substitutions. The table shows metrics for the stable states of the four phases of scenario C as well as the entire scenario duration (calculated over ten episodes). High values of FPR and FNR are highlighted in blue.

In comparison to the previous experiment, the FNR is significantly reduced and closer to the ratio of new attack sources in all four phases. In particular, the mean and median FNR during the stable state of phase ① and ② are at most 0.7 % (a reduction by an order of magnitude compared to the previous experiment). During these phases, fewer attack traffic sources that send high traffic volumes are active. Despite variations in their traffic volume, filter rules are now able to track these sources over time. This improves filtering effectiveness as more attack traffic is removed with additional filter rules (as indicated by the higher blacklist size). The values for the 90 %-quantile and the maximum in phase ① and ② are also reduced in comparison to the previous experiment. These values remain above 13 % and 39 %, since it is challenging to predict the appearance of entirely new attack traffic sources (which immediately send high-volume traffic during the first two phases).

While the reduction of the FNR is advantageous, the FPR still reaches high values, particularly during phase ③ and ④. This is evident in Figure 5.7 and reflected

in the 90 %-quantile and the maximum values listed in Table 5.4. Specifically, the maximum FPR in the last two phases reaches values of 50.73 % and 70.55 %, while the 90 %-quantile in phase ④ remains at 10.94 %. Despite considering historical information, estimations of the (smoothed) discounted precision are not accurate enough to preserve the legitimate traffic. This also leads to the generation of filter rules that affect large address space regions, causing arbitrary traffic removal. This is indicated by FNR-values below the ratio of new attack traffic sources in Figure 5.7, as such values can only occur due to preemptive traffic removal.

The results of this experiment show that filter rule generation needs more accurate estimations of the smoothed discounted precision to preserve the legitimate traffic.

5.4.4 Traffic Filtering with Filter Rule Refinement

This experiment considers Filter Rule Refinement that utilizes the discounted precision, trackers, and traffic volume substitutions. The parameters listed in Table 5.2 remain unchanged compared to the previous experiments.

Results are visualized in Figure 5.8 and summarized in Table 5.5. In comparison to the experiments in Sections 5.4.2 and 5.4.3, a complete Filter Rule Refinement achieves significantly lower FPR throughout the mitigation scenario. The median FPR remains at or below 0.34 % during the stable state of all four attack phases, while the mean FPR reaches at most 1.35 %. Furthermore, the 90 %-quantile is reduced to no more than 3.15 %, which is a considerable improvement compared to the value of 10.94 % in phase ④ of the previous experiment. The maximum FPR over the entire scenario duration is also reduced by more than 64 %. This shows that traffic volume substitutions prove effective in reducing estimation errors of the smoothed discounted precision, which in turn leads to preserving more legitimate traffic.

A trade-off occurs between the FPR and FNR, since the FNR tends to be higher than in the previous experiment. The highest increase in FNR metrics is in the 90 %-quantile of phase ④ (an increase of 8.63 %). During this phase, the FPR frequently reached high values in the previous experiment. This shows that the use of traffic volume substitutions leads to the rejection of filter rules that would otherwise have a strong impact on legitimate traffic. Consequently, the trade-off can be considered to be effective. A similar trade-off occurs in the other three phases. The exceptions are a reduced maximum FNR during phase ② and a reduced 90 %-quantile in phase ③.

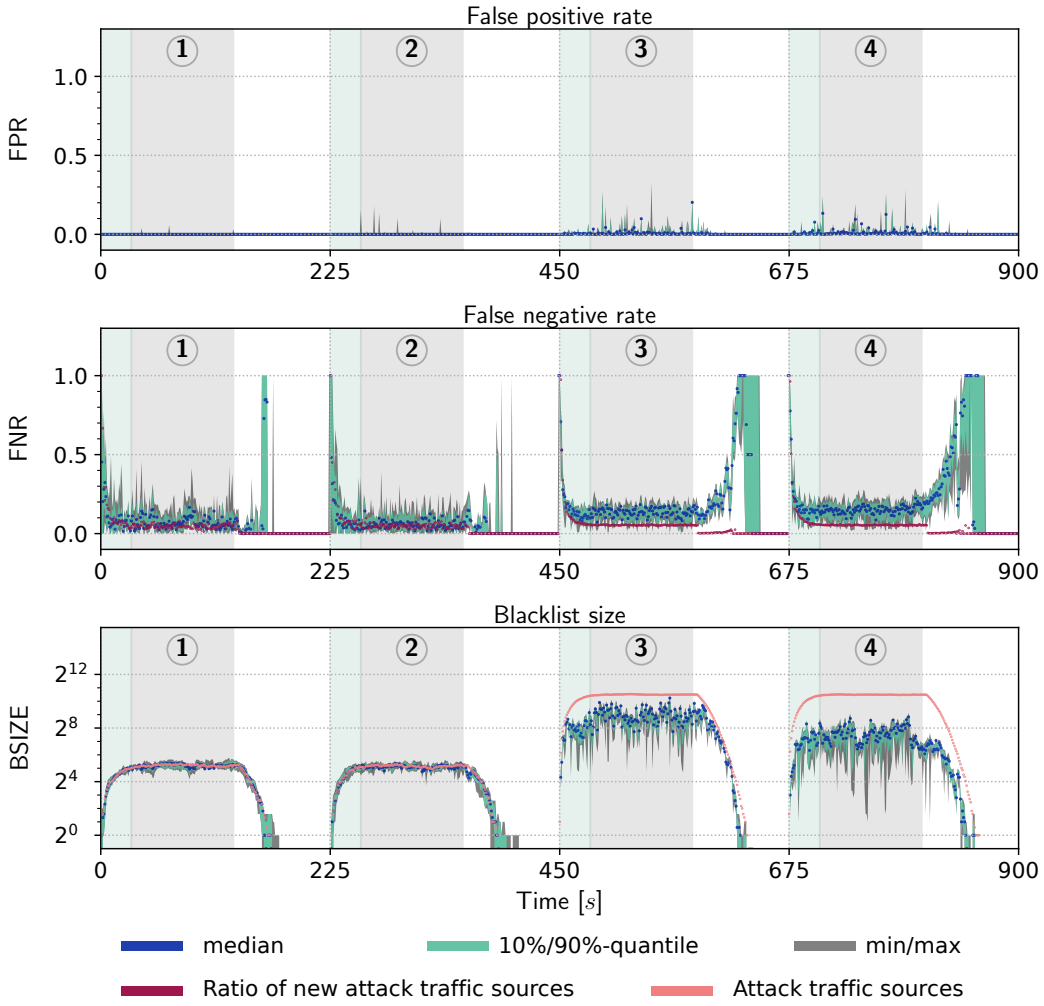


Figure 5.8: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes when filtering traffic with complete Filter Rule Refinement. Green regions within the four phases ① to ④ indicate a 20-second ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

The differences of less than 1 % are consistent with the variations of the FNR over the course of the (dynamic) mitigation scenario.

Blacklist sizes remain close to those of the previous experiment during phase ① and ②. This means that no significant increase in the number of filter rules occurred, but filter rules proved more effective due to a lower maximum FPR. During phase ③, the mean and the 90 %-quantile of BSIZE are lower, while the mean FNR increases

Metric		Phase				Entire scenario duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0021	0.0034	0.0000
	mean	0.0001	0.0010	0.0106	0.0135	0.0038
	90 %-quantile	0.0000	0.0005	0.0303	0.0315	0.0062
	maximum	0.0580	0.1793	0.3268	0.2844	0.3268
FNR	median	0.0653	0.0521	0.1357	0.1505	0.0846
	mean	0.0795	0.0644	0.1305	0.1456	0.1348
	90 %-quantile	0.1640	0.1336	0.1885	0.2096	0.2551
	maximum	0.4579	0.3954	0.2703	0.2853	1.0000
BSIZE	median	36.0	35.0	473.0	178.5	35.0
	mean	36.3	34.7	488.9	191.2	114.1
	90 %-quantile	44.0	42.0	804.2	330.1	369.0
	maximum	54.0	49.0	1264.0	561.0	1264.0

Table 5.5: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE when filtering traffic with complete Filter Rule Refinement. The table shows metrics for the stable states of the four phases of scenario C as well as the entire scenario duration (calculated over ten episodes).

by $\sim 3\,\%$ and the mean FPR is approximately halved. This underscores that more filter rules with impact on legitimate traffic are rejected at the cost of increased FNR. In phase ④, all metrics for BSIZE are higher than in the previous experiment. Simultaneously, the FNR increases by an order of magnitude, which indicates that shorter prefixes are more frequently rejected. This is because the estimation of the (smoothed) discounted precision is stabilized by traffic volume substitutions, which makes it more difficult to achieve (momentarily) high precision estimates in the presence of legitimate traffic. The accompanying decrease in FPR shows that under the same parameter choices performing traffic volume substitutions results in more accurate attack traffic removal.

Other trade-offs could be realized by adjusting the value of ϕ and π^*. A lower value of ϕ generally results in longer prefixes being considered while, a lower value of π^* would reject fewer rules to emphasize lower FNR at the expense of higher FPR.

5.5 Related Work

ZAPDOS [Mis+24] presented an approach for iterative prefix refinement that decides on the length of prefixes used to remove volumetric DDoS attack traffic. The goal is to address memory limitations of switches. The proposed system incrementally splits up prefixes and increases prefix lengths when a prefix has a high likelihood of corresponding to attack traffic. This serves to reduce false positive rates. The likelihood is determined by a random forest [Bre01] machine learning model that uses multiple features. Similarly to Filter Rule Refinement, the likelihood is tracked over time. ZAPDOS has to actively select the prefixes that are monitored to address memory limitation. These prefixes are then refined over multiple iterations. As a result, the evaluation of ZAPDOS shows that it takes over one minute to reduce the false negative rate to $\sim 25\,\%$. In comparison, Filter Rule Refinement monitors high-volume prefixes by default by relying on HHH algorithms. Furthermore, it considers different prefix lengths within the same monitoring interval.

RADAR [Zhe+18] extends the length of prefixes similar to ZAPDOS based on the likelihood that a prefix corresponds to attack traffic. However, RADAR merges longer prefixes with a low likelihood into shorter prefixes once a maximum number of prefixes has been reached. This can reduce the monitoring information on subnets sending high-volume traffic (which are automatically detected by HHH algorithms). Reported experimental results indicate that the time to respond to a volumetric DDoS attack is on the same order of magnitude as ZAPDOS.

ACC-Turbo [Alc+22] is designed to achieve fast reaction to pulsating DDoS attack traffic volumes. The approach monitors multiple features of high-volume aggregates in the data plane to detect high-volume traffic. ACC-Turbo refines the aggregate characteristics in terms of traffic features while performing traffic monitoring in the data plane. Aggregates that incur high drop rates during traffic processing are considered to correspond to attack traffic and subsequently rate-limited. This is essentially a classification based on traffic volume, which can be problematic in the presence of high-volume legitimate traffic. In contrast, Filter Rule Refinement uses the machine learning-based approach of Traffic Purity Estimation that relies on multiple features to identify aggregates with a high attack traffic ratio.

5.6 Conclusion

This chapter identified and addressed two important challenges in using HHH monitoring information for filter rule generation: overaggregation and HHH instability. The first challenge was addressed through the notion of discounted precision, which considers the hierarchical relationships between prefixes. Specifically, it disregards attack traffic that can be removed with longer prefixes when estimating the attack traffic volume of longer parent prefixes. To compensate for HHH instability arising from traffic changes, the concept of trackers was presented. Trackers stabilize filter rule generation by calculating a smoothed discounted precision over time. Several algorithms were presented to calculate smoothed discounted precisions over a prefix hierarchy and to manage the lifecycle of trackers.

The evaluation introduced a mitigation scenario with multiple attack vectors and changing traffic patterns. The results of the evaluation showed that using trackers reduces the risk of incurring high false positive rates due to overaggregation and HHH instability.

Chapter 6

Controlling Mitigation Trade-Offs

A core challenge in volumetric DDoS mitigation is to maintain filtering effectiveness in dynamic traffic situations where the attack and legitimate traffic change over time. When considering a rule-based filtering system, it is challenging to generate a set of filter rules that achieves the following three mitigation goals simultaneously:

- **Low** FPR. The false positive rate FPR should remain low to avoid the unintentional removal of legitimate traffic.
- **Low** FNR. The false negative rate FNR should remain low to prevent attack traffic from bypassing a traffic filter.
- **Low** BSIZE. The blacklist size BSIZE should remain low to facilitate efficient blacklist evaluation.

The three goals can conflict with each other because, depending on the traffic situation, a reduction in one metric may result in an unavoidable increase of another metric. For example, achieving $\mathsf{FPR} = \mathsf{FNR} = \mathsf{BSIZE} = 0$ is only possible in the absence of attack traffic. In the presence of attack traffic, its removal requires at least one filter rule so that either FNR or BSIZE must be non-zero. When attack traffic originates from a high number of sources, a large blacklist size may be required to accurately remove the attack traffic. This makes blacklist evaluation increasingly complex, to the point where the question arises whether a smaller blacklist should be used at the cost of increased FPR or FNR. The question of how to maintain the right balance between the three metrics FPR, FNR, and BSIZE needs to be answered repeatedly in dynamic traffic scenarios. Moreover, trade-offs may be necessary to reconcile conflicting goals so that no single goal is neglected to the point where the overall filtering effectiveness is compromised. For example, over-emphasizing low FPR and low BSIZE can result in a high enough FNR for a volumetric DDoS attack to remain effective.

The central idea to maintain a balance between different mitigation goals is to adjust two key mitigation parameters with a repeatedly executed control loop: the HHH

detection threshold ϕ used in Aggregate Monitoring and the minimum required discounted precision π^* used in Filter Rule Refinement (see Section 5.3.4). These two parameters impact FPR, FNR, and BSIZE in the following way:

- The minimum required discounted precision π^* influences all three metrics. A lower value of π^* typically reduces FNR at the expense of FPR, since filter rules are allowed to have lower discounted precision, resulting in the removal of more legitimate traffic. Conversely, higher π^* values shift the balance between FNR and FPR in favor of a lower FPR. The π^*-value also affects the blacklist size, as a lower π^*-value results in more filter rules being accepted into a blacklist.

- The HHH detection threshold ϕ influences how many HHH prefixes are detected and, consequently, how many filter rules can be generated from these prefixes. Through this, the parameter ϕ ultimately affects the blacklist size BSIZE.

These relationships translate the task of balancing mitigation goals to choosing parameter combinations for ϕ and π^* for a given traffic situation. This is because controlling both parameters simultaneously allows for trade-offs between all three metrics. The π^*-value governs the trade-off between FPR and FNR, while the effects of changing π^* on BSIZE can be counteracted by adjusting ϕ.

To this end, this chapter introduces a novel concept to establish an RL-based control loop. The control loop is executed by the RL-based Control component (introduced in Section 3.3.6), which controls the parameter ϕ used by Aggregate Monitoring and the parameter π^* used by Filter Rule Refinement. The specific design goals of realizing the control loop through RL are stated in Section 6.1. Section 6.3 describes a concept to convey the importance of mitigation goals (along with desired trade-offs) to a RL agent through reward function modeling. Following this, Section 6.4 outlines details on the design of a RL agent (including state and action spaces, as well as neural network architectures). Details on agent training and execution during inference are provided in Section 6.5. The evaluation in Section 6.6 shows that an agent can be trained to adapt the parameters ϕ and π^* in response to changing traffic characteristics. As a result, intended trade-offs between FPR, FNR, and BSIZE (conveyed via reward function parameters) can be realized. Section 6.8 summarizes key findings of this chapter.

6.1 Design Goals

The overarching design goal of controlling mitigation trade-offs is to realize a control loop that enables a reaction to changing traffic characteristics that balances FPR, FNR, and BSIZE against each other. Its primary function is to adjust the mitigation parameters ϕ and π^* when necessary. The control loop is executed by the RL-based Control component (see Section 3.3.6), which provides the chosen parameters to the Aggregate Monitoring and Filter Rule Refinement components.

Finding suitable mitigation parameters is challenging, since it is difficult to determine the impact of parameter combinations on FPR, FNR, and BSIZE for several reasons:

(1) Adjustments to ϕ can cause cascading changes in detected HHH prefixes throughout the IP prefix hierarchy. This effect is described in Section 5.3.3 in the context of HHH instability. The fact that changing ϕ can cause multiple cascading effects to occur simultaneously leads to a complex relationship between ϕ, the number of HHH prefixes, and the size of the generated blacklist.

(2) The parameters ϕ and π^* are not independent. The ϕ-value determines the HHH prefixes available for later blacklist generation, during which the π^*-value takes effect. Changing ϕ can generate different HHH prefixes that result in blacklists with different FPR and FNR if π^* is not adjusted accordingly. Conversely, π^* affects BSIZE as more or less filter rules are accepted into a blacklist. In order to achieve effective trade-offs between the mitigation goals, the dependency between ϕ and π^* must be taken into account.

(3) The impact of a parameter combination on the three mitigation metrics depends on the attack and legitimate traffic. Different distributions of traffic volume over the IP address space can result in different distributions of HHH prefixes across the prefix hierarchy for a fixed choice of ϕ. In addition, the mixture of attack vectors with legitimate traffic within address space regions influences Traffic Purity Estimation. This, in turn, affects the selection of filter rules for a given π^*-value during blacklist generation. As a result, ϕ and π^* must be chosen according to the specific traffic composition and distribution, as well as the desired trade-off.

In short, the parameters, the mitigation metrics, and the traffic form a complex three-way relationship that makes it challenging to explicitly define policies for parameter choices. In particular, the third reason implies that such policies would need to account for highly diverse traffic scenarios. To avoid the explicit definition of policies, this

chapter outlines a concept to learn policies automatically. To this end, the design goal of realizing a control loop is approached with deep RL. More specifically, the chapter focuses on two key objectives:

- The design of an agent that is capable of learning the relationship between the parameter choices, the traffic, and the mitigation metrics.
- A modeling of trade-offs through reward functions so that agent training can be direct to achieve the desired mitigation goals.

The following outlines the goals of these two objectives in greater detail. When an agent can be trained successfully, it can be used to control the Aggregate Monitoring and Filter Rule Refinement parameters during an active mitigation.

6.1.1 Automatic Mitigation Parameter Adaptation

Choosing effective mitigation parameters needs to take the relationship between different parameters, the traffic, and achievable mitigation metrics into account. As outlined above, this constitutes a challenging task. Defining explicit policies needs to take diverse traffic scenarios into account. When traffic patterns change over time, the policies must be updated accordingly. Instead, policies can be learned from interacting with other mitigation components and observing the result of traffic filtering. To enable automatic mitigation parameter adaptations, deep RL is applied.

The learning-based approach requires the design of an agent and, in particular, its state and action spaces as well as its neural network architecture. This leads to three key questions, which are addressed in Section 6.4:

(1) Which parameter combinations should an agent be able to choose?

(2) What data can be used to inform an agent of the current traffic situation?

(3) How to design the neural network architecture of the deep RL agent?

6.1.2 Conveying Mitigation Goal Importance

Directing an RL agent's training requires reward in a form that establishes a linear order among actions.[1] This is necessary to make actions comparable in order to identify preferable actions. In the context of traffic filtering, the effectiveness of taking an action is expressed through the resulting FPR, FNR, and BSIZE. These constitute three independent metrics that cannot be used directly to model reward because they do not impose a linear order. To make actions comparable, the independent metrics must be projected onto a one-dimensional space through a reward function. That is, the reward function must provide a scalar reward (in $\mathbb{R}$) that is derived from the achieved FPR, FNR, and BSIZE after taking an action.

The key challenge in finding a suitable reward function is that the reward must reflect multiple functional requirements of traffic filtering:

- A reward should reflect the importance of FPR, FNR, and BSIZE in relation to each other. The relative importance should be quantifiable to enable the modeling of effective trade-offs. For example, the reward function should specify which increase in FPR is acceptable for a simultaneous decrease in FNR.
- A reward function must be able to indicate the complete failure of traffic filtering. This is necessary to reduce the risk of an agent taking corresponding actions. For example, the reward should be minimal when a blacklist discards all traffic.
- A reward function should avoid over-emphasizing any single mitigation goal. The risk is that an agent may disregard any two of the metrics FPR, FNR, and BSIZE whenever the third metric provides the most significant contribution to the overall reward. In this case, the two disregarded metrics may no longer be balanced against each other.

The design goal of conveying mitigation goal importance is addressed in Section 6.3, where the task of reward function modeling is structured into two subtasks:

(1) An analysis of mitigation goal importance, conflicts, and interdependencies.

(2) The definition of a reward function that reflects the functional requirements of traffic filtering.

[1] A linear order (also called total order) ensures any two rewards $\mathbf{r}_1, \mathbf{r}_2 \in \mathbb{R}$ are strongly connected through a binary relation $\leq$, i.e., either $\mathbf{r}_1 \leq \mathbf{r}_2$ or $\mathbf{r}_2 \leq \mathbf{r}_1$ holds.

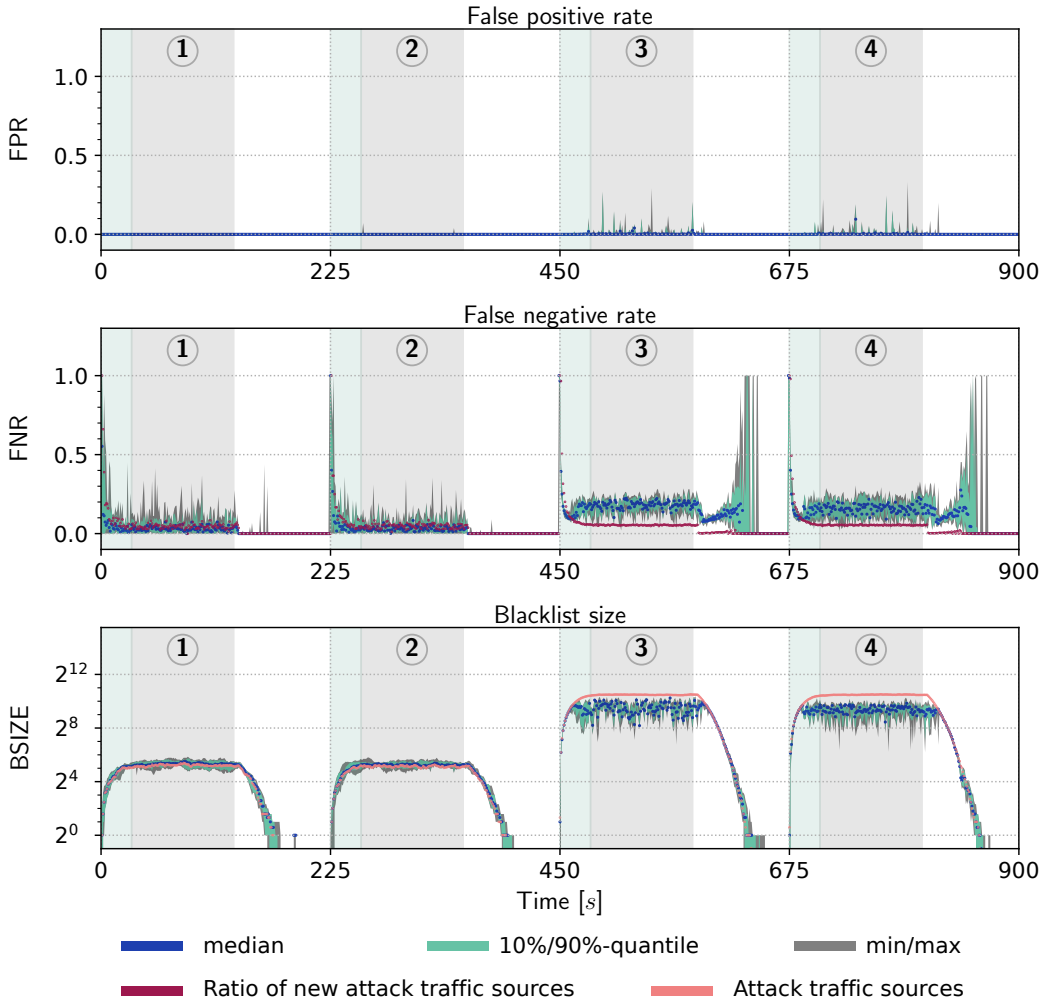


Figure 6.1: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes of the mitigation scenario C. Green regions within the four phases ① to ④ indicate the ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

6.2 An Example Mitigation Result

The following provides an example of an RL agent controlling filter rule generation in the mitigation scenario C from Section 5.4.1. The agent is trained to preserve legitimate traffic while also prioritizing a low blacklist size. Details on agent training are omitted, but will be elaborated throughout the remainder of this chapter.

Figure 6.1 shows the progression of FPR, FNR, and BSIZE over the four phases of the scenario. After each monitoring interval of a one-second duration, the agent receives a reward based on these three metrics. The agent then determines new parameters for ϕ and π^*, which are applied to Aggregate Monitoring and Filter Rule Refinement when generating a new blacklist. For each monitoring interval, the figure visualizes the median, mean, 90 %-quantile, and maximum of the three metrics. These quantities are calculated over ten randomized episodes of the mitigation scenario.

The number of attack traffic sources that are active in each monitoring interval serves as a baseline for the maximum blacklist size. This number changes over time when transitioning between different attack phases. In particular, significant changes occur during the ramp-up and cool-down phase of an attack, but also when transitioning to the phases ③ and ④, where low-volume attack traffic originates from a high number of attacking systems compared to phases ① and ②. The ratio of attack sources per second serves as a reference for the false negative rate as the mitigation can only react to newly appearing attack sources once these are detected. The FNR can still be below the number of new attack sources because already established filter rules can discard traffic from new attack sources.

Figure 6.1 shows that the FPR remains low most of the time with the exception of momentary increases. These are the result of traffic dynamics occurring in the mitigation scenario. In particular, it is challenging to rely on shorter filter rule prefixes (in order to reduce the blacklist size) without affecting legitimate traffic that originates from newly emerging traffic sources. Still, the impact of such filter rules is limited to short-term durations, as Filter Rule Refinement and RL-based Control work in conjunction to withdraw ineffective filter rules.

Since the objective is to use a low number of filter rules while also preserving the legitimate traffic, a trade-off between FNR and BSIZE is necessary. Therefore, reducing BSIZE below the number of active attack traffic sources results in the FNR being significantly higher than the ratio of new attack traffic sources. Consequently, the agent realizes an intended trade-off that is conveyed through the agent's reward. Details on modeling reward functions for this purpose are provided in Section 6.3.

The trade-off is a result of the agent's choices of the HHH detection threshold ϕ and the (minimum) required discounted precision π^*. Figure 6.2 shows the median, 10 %-90 %-quantiles, and maximum values of the parameters chosen by an agent. Clearly, the trained agent reacts to the start and end of an attack by adapting parameter values (during the ramp-up and cool-down phases). In addition, the distribution of parameter combinations is different in phases with fewer and more attack traffic

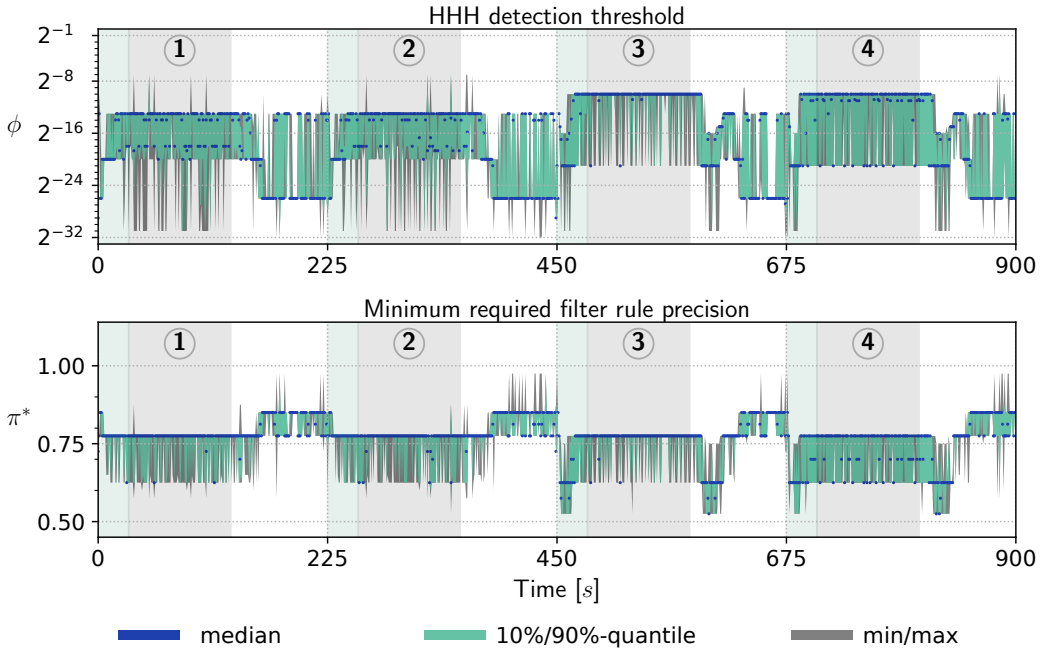


Figure 6.2: Progression of parameters ϕ and π^* in the mitigation example. The median, 10 %-90 %-quantiles, and maximum are calculated from ten episodes.

sources (phases ① and ② in comparison to phases ③ and ④). By choosing appropriate parameters when the traffic changes occur over the course of a mitigation scenario, the agent receives a higher reward. In turn, the agent maintains the intended balance between the three mitigation metrics.

6.3 Modeling Trade-Offs Through Reward Functions

Section 6.2 outlined an example in which two agents realize different trade-offs during mitigation. To guide an agent to realize effective trade-offs, the three metrics FPR, FNR, and BSIZE need to be balanced against each other. More specifically, a reward function must map the three metrics to a scalar reward. This mapping needs to express the priority of the mitigation goals. For example, a decrease in FPR may be of higher priority than an equal decrease in FNR. This implies that the reward for a lower FPR should be higher than the reward for a lower FNR. However, the priority of a mitigation goal can increase or decrease in comparison to other goals as its corresponding metric progresses. This can lead to a point where one mitigation

goal becomes more important than another and a priority reversal occurs. A reward function must be able to express such relationships by modeling the progression of the (scalar) reward in accordance to mitigation goal priority. Furthermore, none of the mitigation goals may be neglected entirely, and the mitigation effectiveness must not be compromised.

To model reward functions that consider the priority of mitigation goals, Section 6.3.1 provides a general assessment of mitigation goal importance. Furthermore, conflicts and interdependencies between different goals are discussed in Section 6.3.2 and Section 6.3.3. Considerations on mitigation goal priorities and interdependencies lead to the reward function modeling presented in Section 6.3.4. In addition, Section 6.3.5 discusses further reward function tuning that allows it to explicitly model priority reversal between mitigation goals.

6.3.1 Mitigation Goal Importance

Balancing the three mitigation goals of low FPR, low FNR, and low BSIZE against each other requires a general assessment of how important these goals are in relation to each other. To this end, the following considerations can be taken into account in reward function modeling.

As a general tendency, low false positive rates have higher priority than low false negative rates for the following reasons:

- Low false positive rates around 1 % or 2 % can reduce throughput in the presence of TCP-based traffic due to incurred packet loss and interference with congestion control algorithms. In contrast, a false negative rate of 1 % or 2 % indicates that 98 % to 99 % percent of the attack traffic has been discarded. In this case, attack traffic needs to be orders of magnitude greater than the bottleneck link capacities to have a significant impact on throughput.
- When network links are over-provisioned, a residual attack traffic volume can be allowed to enter a network infrastructure without impeding service availability. This leads to a tolerance to low false negative rates. In contrast, the unintentional removal of legitimate traffic can have a direct negative impact on service availability.

To assess the priority of BSIZE in relation to FPR and FNR, two aspects regarding the blacklist size can be considered:

- Filter rules are intended to conduct prefix-based matching in the data plane. However, the time to evaluate filter rules and the memory in the data plane (such as TCAM) are typically limited. This places an upper bound on the blacklist size, so that BSIZE becomes increasingly important as it approaches the upper bound (up to the point where priority reversal with both FPR and FNR can occur).
- A small blacklist size must be acceptable in any case as long as traffic filtering is to be used to counteract volumetric DDoS attacks. This places a low priority on the blacklist size (in comparison to FPR and FNR) as long as BSIZE remains close to 0.

The ability to efficiently evaluate blacklists in the data plane depends on the capabilities of the network infrastructure. For example, recent work considered distributing flow table entries across switches that have spare capacity [BZ16], offering the potential to apply larger blacklists. To take such options into account, a reward function should be adjustable to account for different infrastructure capabilities instead of targeting specific trade-offs.

6.3.2 Mitigation Goal Conflicts

A conflict between mitigation goals exists, when achieving one goal makes it infeasible to achieve another. The specific conflicts between achieving a low FPR, low FNR, and low BSIZE can be summarized as follows:

Conflict between FPR and BSIZE. When the entire attack traffic originates from a single subnet and no legitimate traffic originates from the same subnet, a single filter rule can be sufficient to achieve $\mathsf{FPR} = 0$. In contrast, when distributed volumetric attack sources are interspersed with legitimate traffic sources in the IP address space, multiple filter rules are required to achieve $\mathsf{FPR} = 0$. Specifically, one filter rule is required for each subnet that contains attack traffic sources but no legitimate traffic sources. In this case, any reduction of BSIZE necessarily results in increased FPR. This means that FPR and BSIZE are in conflict whenever attack and legitimate traffic originates from interspersed traffic sources.

Conflict between FPR and FNR. A conflict between FPR and FNR occurs when the blacklist is bounded below the number of distinct (non-adjacent) subnets emitting attack but no legitimate traffic. In this case, the mitigation system can resort to filter rules with longer or shorter prefixes. For traffic patterns with interspersed attack and

legitimate sources, using longer prefixes reduces the risk of inadvertently discarding legitimate traffic and consequently incurs lower FPR. Still, with a limited blacklist size, some attack traffic sources may be missed, resulting in elevated FNR. Conversely, resorting to shorter prefixes can reduce FNR by removing more attack traffic at the cost of increased FPR.

Conflict between FNR and BSIZE. A conflict between FNR and BSIZE occurs when the size of a blacklist should be reduced without incurring higher FPR (e.g., when FPR is considered to have high priority). When a minimal set of filter rules discards all attack traffic, the only option to maintain a given FPR and to reduce the blacklist size is to remove some of the filter rules. This results in higher FNR because the initial set of filter rules had minimal size. Conversely, if a given FPR is to be maintained, lower FNR can only be achieved by adding additional filter rules. Thus, when a reduction of FPR is not feasible, FNR and BSIZE can be in conflict.

These conflicts result from networking requirements, but also from changing attack and legitimate traffic patterns. In order to maintain an effective mitigation, it can become necessary to frequently resolve conflicts over time.

6.3.3 Mitigation Goal Interdependencies

Modeling the progression of the contribution of individual metrics to overall reward independently does not necessarily achieve the best mitigation results. This becomes clear when considering a linear reward function $\mathcal{R}_{\text{LINEAR}} : [0,1]^n \times \mathbb{R}_{>0}^n \to \mathbb{R}_{>0}, n \in \mathbb{R}$ that applies a vector of weights $\mathsf{w} = (w_1, \ldots, w_n)$ to a vector $\mathsf{x} = (x_1, \ldots, x_n)$:

$$\mathcal{R}_{\text{LINEAR}} : (\mathsf{x}, \mathsf{w}) \mapsto \overline{\mathsf{x}} \cdot \mathsf{w}^\top = \sum_{i=1}^{n} \overline{x_i} \cdot w_i \tag{6.1}$$

Here, the operator $\overline{\cdot} : [0,1] \to [0,1]$ is used for convenience to express a reciprocal relationship where a lower value results in a higher reward[2]:

$$\overline{\cdot} : x \mapsto 1 - x\,. \tag{6.2}$$

[2]The operator is applied to each coefficient when applied to a vector.

The weights w_i weigh each element x_i of the vector x independently when applying $\mathcal{R}_{\mathsf{LINEAR}}$. That is, the contribution of each term $w_i\overline{x_i}$ does not influence the contribution of other terms to overall reward. While $\mathcal{R}_{\mathsf{LINEAR}}$ projects the n-dimensional vector x into a single scalar reward, several problems would arise from guiding agent training with $\mathcal{R}_{\mathsf{LINEAR}}$. This becomes clear when choosing $\mathsf{x} = (\mathsf{FPR}, \mathsf{FNR}, \mathsf{BSIZE})$ and $\mathsf{w} = (w_{\mathsf{FPR}}, w_{\mathsf{FNR}}, w_{\mathsf{BSIZE}})$. In this case, the linear relationship between the metrics and the reward implied by $\mathcal{R}_{\mathsf{LINEAR}}$ suffers from multiple drawbacks.

Failure to indicate critical failure. When FNR and BSIZE are strictly positive, the mitigation system fulfills the goals of low FNR and low BSIZE at least partially. In this case, $\mathcal{R}_{\mathsf{LINEAR}}$ yields reward $\mathbf{r} > 0$ for $\mathsf{FPR} = 1$. Despite non-zero reward, the overall mitigation effort must be considered a complete failure. In fact, $\mathsf{FPR} = 1$ implies that the DDoS mitigation system itself discards the entire legitimate traffic and perfectly realizes the attacker's goal. Still, when the blacklist consists of only a few filter rules that discard the entire traffic (resulting in $\mathsf{FNR} = 0$ and $\mathsf{BSIZE} \approx 0$) $\mathcal{R}_{\mathsf{LINEAR}}$ yields reward close to $w_{\mathsf{FNR}} + w_{\mathsf{BSIZE}}$. Thus, for any non-negligible weights w_{FNR} and w_{BSIZE} the reward function $\mathcal{R}_{\mathsf{LINEAR}}$ inadequately reflects the importance of FPR by yielding non-negligible reward for a complete failure.

Failure to optimize for secondary goals. A straightforward approach to indicate critical failure (as described above) is to place a high emphasis on FPR by choosing $w_{\mathsf{FPR}} \gg w_{\mathsf{FNR}}, w_{\mathsf{BSIZE}}$. In this case, overall high reward can only be achieved for low false positive rates. However, the maximum reward for simultaneously achieving a low false negative rate and a low blacklist size is bounded by the comparatively small sum $w_{\mathsf{FNR}} + w_{\mathsf{BSIZE}}$. Since FPR can be subject to frequent changes due to traffic dynamics, the contribution of $w_{\mathsf{FPR}} \cdot \mathsf{FPR}$ to the reward can introduce significant fluctuations into the reward that an agent receives during training. This can obscure the optimization potential for FNR and BSIZE, since an agent will primarily focus on achieving high reward from FPR and disregard the relationship between FNR and BSIZE. In this case, optimization for secondary goals becomes infeasible.

Failure to realize trade-offs. When transitioning from shorter to longer prefixes (e.g., by adjusting the HHH detection threshold ϕ), the blacklist size can grow exponentially. Yet, when attack traffic sources are interspersed with legitimate traffic sources over the IP address space, short prefixes are required to achieve low FPR. In such cases, the contribution of $w_{\mathsf{BSIZE}} \cdot \overline{\mathsf{BSIZE}}$ to the reward diminishes for low FPR (due to the extensive blacklist size). The exponential increase in blacklist size and corresponding decrease of $w_{\mathsf{BSIZE}} \cdot \overline{\mathsf{BSIZE}}$ cannot be compensated for by increasing w_{BSIZE}. The necessary increase would yield exceedingly high reward for low BSIZE,

so that the reward contribution from $w_{\mathsf{FPR}} \cdot \overline{\mathsf{FPR}}$ eventually becomes negligible in comparison. This guides an agent to always prefer blacklists of small size. In effect, agent training can either emphasize low FPR and disregard low BSIZE or vice versa. As a result, $\mathcal{R}_{\mathsf{LINEAR}}$ fails to express the reciprocal relationship between low FPR and low BSIZE and does not account for the two metrics to progress over different orders of magnitude.

6.3.4 Harmonic Reward Functions

A shortcoming of a linear reward function $\mathcal{R}_{\mathsf{LINEAR}}$ is that reward is generated independently for each mitigation goal. This is a result of combining multiple metrics through addition, so that reward generated by a single metric has no impact on the reward contribution of other metrics. To model an interdependency between multiple mitigation goals, a weighted harmonic mean can be used instead.

Definition 6.1: Weighted Harmonic Mean

The **weighted harmonic mean** [Fer31] $H_n : \mathbb{R}^n_{>0} \times \mathbb{R}^n_{>0} \to \mathbb{R}_{>0}, n \in \mathbb{N}$ of a vector $\mathsf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n_{>0}$ and weight vector $\mathsf{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n_{>0}$ is given by

$$H_n : (\mathsf{x}, \mathsf{w}) \mapsto \frac{\sum_{i=1}^n w_i}{\sum_{i=1}^n w_i x_i^{-2}}. \tag{6.3}$$

The index n is omitted when it is clear from the context.

A weighted harmonic mean H cannot be directly used as a reward function, as it is undefined for any coefficient $x_i = 0$ and it does not reflect the reciprocal relationship between the priority of low false positive rates and low false negative rates and their corresponding metrics FPR and FNR (e.g., low FNR has high priority). Instead, H can be extended to a **harmonic reward function** $\mathcal{R}_{\mathsf{HARMONIC}} : (\mathbb{R}_{>0} \cup 0)^n \times \mathbb{R}^n_{>0} \to \mathbb{R}_{>0}$:

$$\mathcal{R}_{\mathsf{HARMONIC}} : (\mathsf{x}, \mathsf{w}) \mapsto \frac{\sum_{i=1}^n w_i}{\sum_{i=1}^n w_i \left[\!\left[\overline{x_i}^{-2}\right]\!\right]}. \tag{6.4}$$

The operator $[\![\cdot]\!] : (\mathbb{R}_{>0} \cup 0) \to \mathbb{R}_{>0}$ maps its argument to a strictly positive number that is not smaller than a chosen small value $\epsilon \in \mathbb{R}_{>0}$:

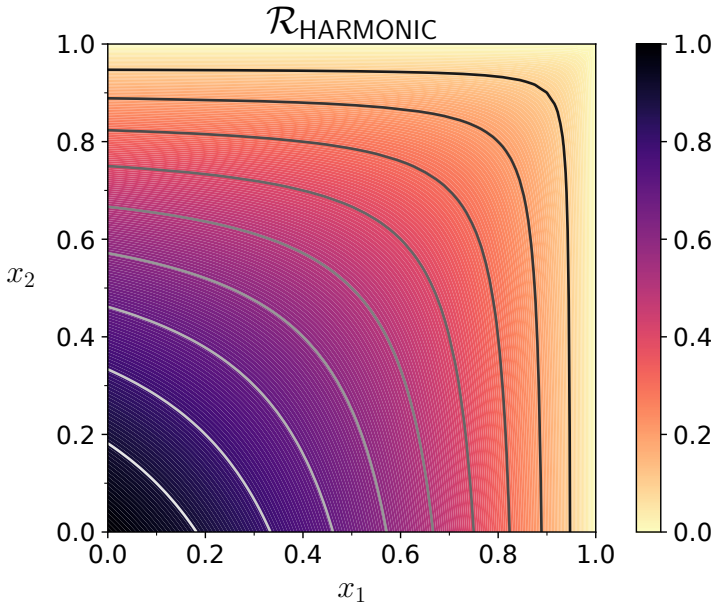


Figure 6.3: Image of the reward function $\mathcal{R}_{\text{HARMONIC}}(\mathsf{x}, \mathsf{w})$ for $\mathsf{x} = (x_1, x_2) \in [0,1]^2$ and fixed weights $\mathsf{w} = (w_1, w_2) = (1,1)$. Contour lines indicate equal reward.

$$[\![\cdot]\!] : x \mapsto \max(x, \epsilon)\,. \tag{6.5}$$

In the following, the operator $[\![\cdot]\!]$ uses a value of $\epsilon = 10^{-7}$ to avoid division by zero and to reduce numerical instabilities arising from division by smaller numbers. In addition, Equation 6.4 applies the operator $\overline{\cdot}$ from Equation 6.2 to coefficients x_i that represent metrics having a reciprocal relationship with their mitigation goals (i.e., FPR and FNR).

Figure 6.3 illustrates an example of the image of a harmonic reward function $\mathcal{R}_{\text{HARMONIC}}$ over the two-dimensional interval $[0,1]^2$ for equal weights $\mathsf{w} = (w_1, w_2) = (1,1)$. Notice that $\mathcal{R}_{\text{HARMONIC}}$ assumes a value close to zero along the axes $x_1 = 1$ and $x_2 = 1$. Through this, the reward generated by $\mathcal{R}_{\text{HARMONIC}}$ is minimal along these axes, which indicates complete failure of the mitigation if a single mitigation goals fails.

The contour lines in Figure 6.3 indicate equal reward $\mathbf{r}$ generated by $\mathcal{R}_{\text{HARMONIC}}$ for different vectors $\mathsf{x} = (x_1, x_2)$. The area enclosed by contour lines, the origin $(0,0)$, as well as the x_1 and x_2 axes is convex, so that any fixed reward $\mathbf{r}$ imposes a restriction

on the maximum coefficient values of x. Bounding the coefficients in this way limits the potential trade-offs between mitigation goals that can be realized at a fixed reward. If the enclosed area would be concave, at least one coefficient of x may become unbounded. In this case, trade-offs can no longer be controlled through the reward function, as a high coefficient value (e.g., one resulting from a high FNR) could still yield high reward. An agent would then learn to take ineffective actions. In the case where multiple coefficients become unbounded, extreme trade-offs are possible at equal reward levels. This can lead to an agent learning a policy that alternates between actions causing high FPR, FNR, or BSIZE.

6.3.5 Reward Term Shaping

While harmonic reward functions can be used to model dependencies of mitigation goals, additional adjustments are necessary to reflect the priorities of mitigation goals in relation to each other. The function $\mathcal{R}_{\text{HARMONIC}}$ offers control over trade-offs by adjusting the weight vector w, but cannot account for a priority reversal of two mitigation goals. This is because the term $w_1 x_1^{-2}$ in Equation 6.3 is always less than term $w_2 x_2^{-2}$ for $w_1 < w_2$ and $x_1 = x_2$. For priority reversal to occur, x_2 must have stronger effect on reward than x_1 when x_1 and x_2 progress uniformly. That is, $w_2 x_2^{-2}$ must become smaller than $w_1 x_1^{-2}$ as $x_1 = x_2$ increases.

In the context of RL-based Control, priority reversal serves to reduce an agent's reward significantly once the blacklist approaches a certain size. Through this, BSIZE can have low importance as long as the blacklist remains small, but its importance can eventually exceed that of FPR and FNR as the blacklist size increases. This does not offer exact control over the maximum blacklist size, as it may not be possible to achieve effective trade-offs for an arbitrarily low blacklist size. Still, emphasizing low BSIZE through reward function modeling guides an agent to prioritize smaller blacklists in a best-effort manner. An increasing priority of the blacklist size can be modeled by controlling the reward contribution of BSIZE with a piecewise linear shaping function, which is defined as follows:

Definition 6.2: Shaping Function

Let $\vartheta_1 < \vartheta_2 < \ldots < \vartheta_n \in \mathbb{N}_0$ be a sequence of $n \in \mathbb{N}$ thresholds and let $\zeta_1 > \zeta_2 > \ldots > \zeta_n \in [0,1]$ be fixed corresponding function values. Then, the piecewise linear **shaping function** $\sigma_{\mathsf{BSIZE}} : \mathbb{N}_0 \to (\mathbb{R}_{>0} \cup 0)$ is given by

$$\sigma_{\mathsf{BSIZE}} : \mathsf{BSIZE} \mapsto \begin{cases} 1 & \text{if } \mathsf{BSIZE} < \vartheta_1, \\ \zeta_i + \dfrac{\zeta_{i+1} - \zeta_i}{\vartheta_{i+1} - \vartheta_i}(\mathsf{BSIZE} - \vartheta_i) & \text{if } \vartheta_i \leq \mathsf{BSIZE} < \vartheta_{i+1}, \\ 0 & \text{if } \mathsf{BSIZE} \geq \vartheta_n. \end{cases} \tag{6.6}$$

To control the effect of the blacklist size on overall reward, the shaping function σ_{BSIZE} is applied to BSIZE before calculating reward with the function $\mathcal{R}_{\mathsf{HARMONIC}}$. Function values of σ_{BSIZE} are linearly interpolated between the thresholds $\vartheta_i, \ldots, \vartheta_n$. Choosing lower function values ζ_i for the thresholds ϑ_i increases the priority of maintaining a lower blacklist size, as a lower value of σ_{BSIZE} decreases the value of $\mathcal{R}_{\mathsf{HARMONIC}}$. Consequently, controlling the slope $(\zeta_{i+1} - \zeta_i)/(\vartheta_{i+1} - \vartheta_i)$ (by choosing function values and thresholds) models a slow or steeply decreasing reward for increasing BSIZE. This can encourage smaller or larger reductions in blacklist size depending on parameter choices. A plateau of equal reward can be used to completely deprioritize a reduction of BSIZE. In particular, setting $\zeta_1 = 1$ serves to allow an agent focus on low FPR and low FNR when BSIZE is below ϑ_1.

Example 6.3.1. Figure 6.4 visualizes example progressions of four shaping functions $\sigma^{(i)}_{\mathsf{BSIZE}}$ $(i = 1, \ldots, 4)$ and their derivatives. Each shaping function $\sigma^{(1)}_{\mathsf{BSIZE}}, \ldots, \sigma^{(4)}_{\mathsf{BSIZE}}$ uses a different initial threshold $\vartheta^{(1)}_1 = 700, \vartheta^{(2)}_1 = 900, \vartheta^{(3)}_1 = 1100$, and $\vartheta^{(4)}_1 = 1300$. The two next higher thresholds $\vartheta^{(i)}_2$ and $\vartheta^{(i)}_3$ are set at intervals of 200 so that $\vartheta^{(i)}_2 = \vartheta^{(i)}_1 + 200$ and $\vartheta^{(i)}_3 = \vartheta^{(i)}_1 + 400$. Corresponding function values are fixed at $\zeta_1 = 1, \zeta_2 = 0.98$, and $\zeta_2 = 0.8$. The value of all functions $\sigma^{(i)}_{\mathsf{BSIZE}}$ reaches $\zeta_1 = 0$ at $\vartheta_4 = 2000$. A lower function value of $\sigma^{(i)}_{\mathsf{BSIZE}}$ for a given blacklist size indicates a higher priority on maintaining a low blacklist size. Consequently, $\sigma^{(4)}_{\mathsf{BSIZE}}$ places more emphasis on small blacklist sizes than $\sigma^{(1)}_{\mathsf{BSIZE}}$.

The choice of $\zeta_1 = 1$ eliminates the incentive to reduce BSIZE below the initial threshold. Through this, an agent can focus on low FPR and FNR without needlessly optimizing for lower blacklist sizes. The choice of $\zeta_2 = 0.98$ close to 1 encourages a reduction of the blacklist size when possible, but it is more likely that maintaining low FPR and low FNR is of greater importance to achieve high reward. Once ϑ_2 and ϑ_3 are exceeded, reward declines rapidly so that BSIZE takes increasing precedence

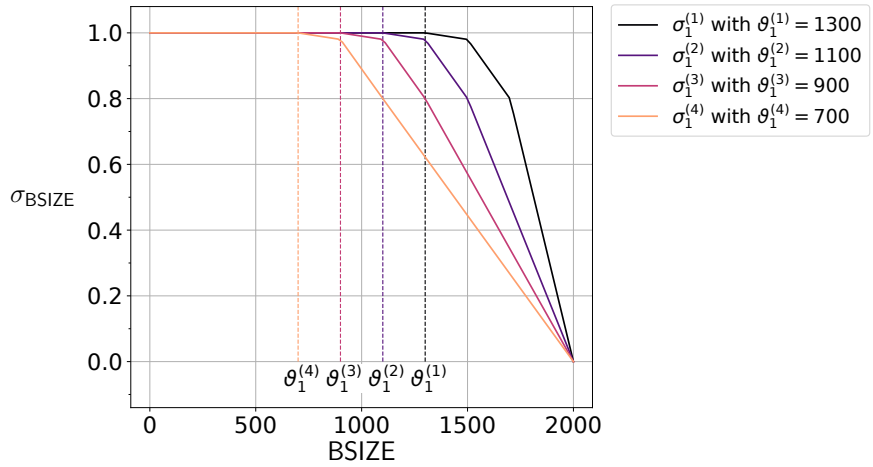


Figure 6.4: Four examples $\sigma_{\text{BSIZE}}^{(1)}, \ldots, \sigma_{\text{BSIZE}}^{(4)}$ of shaping functions with fixed choices $\zeta_1 = 1, \zeta_2 = 0.98$, and $\zeta_3 = 0.8$. Each function $\sigma_{\text{BSIZE}}^{(i)}$ uses a different initial threshold $\vartheta_1^{(i)}$ (indicated by vertical dotted lines) as well as $\vartheta_2^{(i)} = \vartheta_1^{(i)} + 200$ and $\vartheta_3^{(i)} = \vartheta_1^{(i)} + 400$. All shaping functions use $\zeta_4 = 0$ starting at $\vartheta_4 = 2000$.

over FPR and FNR. When BSIZE reaches the maximum value $\vartheta_4 = 2000$, the value of σ_{BSIZE} decreases to 0. In this case, the overall reward is minimal (due to $\mathcal{R}_{\text{HARMONIC}}$ considering mitigation goal dependencies), which indicates a critical failure.

Combining σ_{BSIZE} with $\mathcal{R}_{\text{HARMONIC}}$ results in a shaped harmonic reward function that is designed to model a priority reversal of BSIZE with FPR and FNR.

Definition 6.3: Shaped Harmonic Reward Function

Let $\text{FPR}, \text{FNR} \in [0,1]$ and $\text{BSIZE} \in \mathbb{N}$ denote the false positive rate, false negative rate, and the blacklist size. Further, let the vector $\mathsf{w} = (w_{\text{FPR}}, w_{\text{FNR}}, w_{\text{BSIZE}}) \in \mathbb{R}^3_{>0}$ denote corresponding weights and σ_{BSIZE} denote the shaping function from Definition 6.2 (with fixed thresholds ϑ_i and function values ζ_i). Then, the **shaped harmonic reward function** $\mathcal{R}_{\text{HARMONIC}}^{\sigma_{\text{BSIZE}}} : \mathbb{R}^3 \times \mathbb{R}^3_{>0} \to \mathbb{R}_{>0}$ is given by

$$\begin{aligned} &\mathcal{R}_{\text{HARMONIC}}^{\sigma_{\text{BSIZE}}} : (\text{FPR}, \text{FNR}, \text{BSIZE}; w_{\text{FPR}}, w_{\text{FNR}}, w_{\text{BSIZE}}) \\ &\qquad \mapsto \frac{w_{\text{FPR}} + w_{\text{FNR}} + w_{\text{BSIZE}}}{\frac{w_{\text{FPR}}}{[\![\overline{\text{x}_{\text{FPR}}}^2]\!]} + \frac{w_{\text{FNR}}}{[\![\overline{\text{x}_{\text{FNR}}}^2]\!]} + \frac{w_{\text{BSIZE}}}{[\![\sigma_{\text{BSIZE}}(\text{BSIZE})^2]\!]}} \end{aligned} \tag{6.7}$$

In contrast to Equation 6.4, the operator $\overline{\cdot}$ is not applied to σ_{BSIZE} in Equation 6.7. This is because the reciprocal relationship between increasing BSIZE and decreasing reward is already expressed by the declining slope of σ_{BSIZE}.

The following example uses two different parameter combinations for $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$ to compare the generated reward and achievable trade-offs.

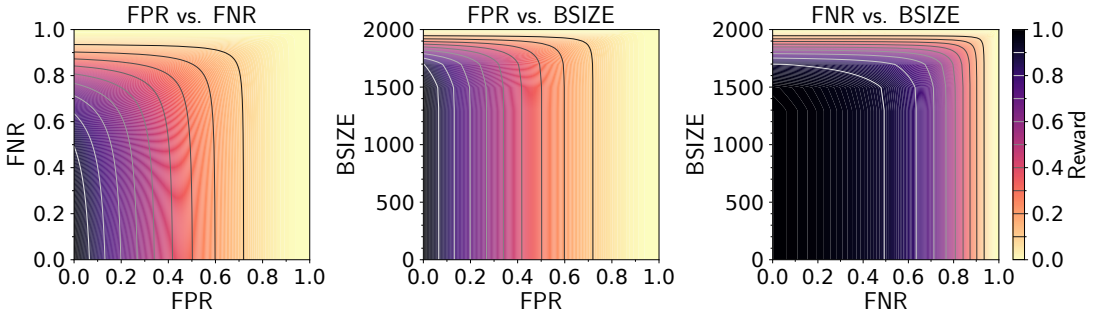
Example 6.3.2. Table 6.1 lists two parameter combinations A and B for $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$. The function values of σ_{BSIZE} corresponding to the listed thresholds are chosen as $\zeta_1 = 1, \zeta_2 = 0.98, \zeta_3 = 0.8$, and $\zeta_4 = 0$ like in the previous example. Parameter combination A uses higher thresholds for σ_{BSIZE} and a larger weight w_{FNR}, while both combinations use the same weight $w_{\text{FPR}} = 4$. Figure 6.5 visualizes the resulting reward generated by $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$. The contour lines indicate where the reward $\mathbf{r}$ reaches values of $0.1, 0.2, \ldots, 0.9$.

On the left, Figure 6.5 shows the reward for FPR and FNR (for BSIZE $= 0$). The contour lines intersect at lower values with the FPR axis than with the FNR axis as the reward decreases monotonically. This is a consequence of w_{FPR} being larger than w_{FNR} and leads to a stronger reward reduction when FPR increases than when FNR increases by the same amount. In order to maintain a certain reward, an agent has to achieve lower FPR than FNR, which corresponds to a higher priority of FPR. The contour lines also occur at lower FNR values in Figure 6.5a compared to Figure 6.5b. This indicates that reward decreases more rapidly for parameter combination A when FNR increases. Consequently, choosing a higher value for w_{FNR} in parameter combination A places a higher priority on low FNR.

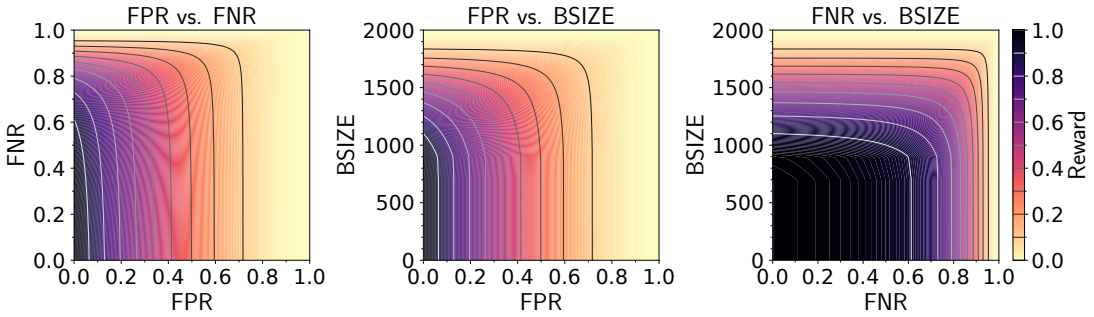
The middle and right side of Figure 6.5 compare the progression of FPR and FNR to that of BSIZE. While the reward for growing FPR decreases quickly due to high w_{FPR}, the thresholds of the shaping function σ_{BSIZE} allow the reward to stay high while BSIZE approaches the threshold ϑ_1 from below. This means that the FPR is restricted to low values for trade-offs with high reward as long as BSIZE is not larger than ϑ_1.

Parameter combination	w_{FPR}	w_{FNR}	w_{BSIZE}	ϑ_1	ϑ_2	ϑ_3	ϑ_4
A	4	0.2	1	1300	1500	1700	2000
B	4	0.1	1	700	900	1100	2000

Table 6.1: Parameter combinations for $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$ used in Example 6.3.2.



(a) Parameter combination A



(b) Parameter combination B

Figure 6.5: Reward generated by $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$ using parameter combinations from Table 6.1. Each plot visualizes the reward for two out of the three metrics FPR, FNR, and BSIZE. The contour lines indicate where the reward $\mathbf{r}$ reaches values of $0.1, 0.2, \ldots, 0.9$.

For example, a blacklist size of no more than $\vartheta_1 = 1300$ requires FPR to be at most 0.061 to achieve a reward of at least 0.9 using parameter combination A. When the overall achievable reward is lower, the reward function allows FPR and BSIZE to become larger. Nevertheless, trade-offs between FPR and BSIZE remain constrained by the convex shape of $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$ (as indicated by the contour lines). This is also the case for trade-offs between FNR and BSIZE, but the reward decreases less rapidly for growing FNR since w_{FNR} is much smaller than w_{FPR} in both parameter combinations.

Compared to Figure 6.5b, the contour lines are closer together (for both FPR and FNR) in the upper part of Figure 6.5a, where BSIZE assumes high values. This is because ϑ_3 is higher for parameter combination A, while both combinations use the same value $\vartheta_4 = 2000$. Consequently, the shaping function σ_{BSIZE} for combination A decreases faster above ϑ_3 (where the greatest reward reduction occurs). In contrast, σ_{BSIZE}

decreases earlier but more slowly for parameter combination B due to the lower thresholds. As a result, combination B places a higher priority on lower blacklist size, but allows for a wider range of trade-offs when an overall high reward cannot be achieved.

For any combination of metrics and for both parameter combinations, the reward shown in Figure 6.5 assumes a minimal value close to zero when FPR, FNR, or BSIZE have maximum values. That is, the reward function $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$ indicates a critical failure when at least one of the mitigation goals fails.

The previous example outlines how different choices of the weights w_{FPR} and w_{FNR} can be used to model different trade-offs between FPR and FNR. In combination with reward term shaping, adjustments to the thresholds $\vartheta_1, \ldots, \vartheta_4$ offer control over trade-offs with BSIZE. The dependencies between mitigation goals are expressed through the convex shape of $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$, which restricts the trade-offs that are possible at different reward levels. It also indicates when a mitigation critically fails.

6.4 Agent Design

To maintain mitigation effectiveness, an agent needs to learn the relationship between an applied action, the resulting mitigation state, and the received reward. While reward function modeling is discussed in Section 6.3, this section focuses on the design of the RL agent. For this, three important parts of the agent's design are discussed:

- **Action space**. The action space defines which mitigation parameter values an agent can choose. A larger set of available parameter values offers an agent greater control over the mitigation, but it can make the learning task more difficult. Design decisions for an action space to control filter rule generation are discussed in Section 6.4.1.
- **Mitigation state**. The mitigation state informs an agent about the impact of choosing an action on the (filtered) traffic. The design of the mitigation state determines whether an agent can distinguish between different traffic distributions and impacts the agent's ability to choose effective parameter combinations. Section 6.4.2 provides details on modeling the mitigation state that serves as input for an agent.

- **Neural network architecture**. A RL agent's neural network architecture determines how the mitigation state is processed and how new actions are chosen. Depending on the RL algorithm, an agent may employ multiple neural networks (e.g., in an actor-critique fashion). Section 6.4.3 outlines the considerations in the design of neural network architectures.

In combination, the three parts determine the full decision-making process of a trained agent that observes a mitigation state and outputs parameter values for a subsequent monitoring interval.

6.4.1 Action Spaces

An action space determines the control a RL agent has over the mitigation. More precisely, an agent controls the mitigation by choosing an action that represents a specific combination of ϕ and π^*-values. The chosen values are then applied to Aggregate Monitoring and Filter Rule Refinement. The full set of actions available to an agent comprises a parameter action space, which is defined as follows:

Definition 6.4: Parameter Action Space

A **parameter action space** is a subset $\mathbf{A}$ of the two-dimensional vector space $\mathbb{R}^2$. The elements $\mathbf{a} \in \mathbf{A}$ represent the actions an agent can take to control mitigation parameters. Each action $\mathbf{a} = (\phi, \pi^*) \in \mathbf{A} \subset \mathbb{R}^2$ identifies a unique combination of parameter values for the HHH detection threshold ϕ and the minimum required discounted precision π^*.

The design of the parameter action space influences the success of agent training. A larger parameter action space increases the complexity of a learning task, whereas smaller action spaces offer fewer options to choose effective mitigation parameters. This raises the question of which parameter combinations should be available to an agent. Two key considerations on the range and the progression of parameter values are taken into account in the design of parameter action spaces.

Parameter value range. Both parameters ϕ and π^* fall into the interval $[0, 1]$. Neither the (discounted) precision of a filter rule nor the fraction ϕ of the total traffic volume $\mathcal{V}$ can be outside the interval boundaries. This interval can be further refined for both parameters:

- The π^*-value should be at least $\frac{1}{2}$. Accepting a lower bound on (discounted) precision can result in filter rules that remove more legitimate than attack traffic, which would undermine the purpose of traffic filtering.

- To generate small blacklists, it is sufficient to choose a maximum ϕ-value of $\frac{1}{3}$ (or below). With $\phi = \frac{1}{3}$ the Aggregate Monitoring generates at most three HHH prefixes, since each HHH prefix needs to account for at least one third of the total traffic volume. Prefixes representing predominantly legitimate traffic can further be rejected through Filter Rule Refinement, so that blacklists of size three and below can be achieved.

Parameter value progression. The parameter value progression determines the distribution of available parameter values over the value range.

- ϕ-values of a parameter action space should approximately follow a geometric progression (e.g., $\phi_i = 2^{-i}, i = 2,3,\ldots$). This is due to the structure of the prefix hierarchy, where each increment in prefix length doubles the subnet size. If the traffic volume is evenly distributed across the IP address space, the volume in each subnet halves as the prefix length increases. To generate increasingly longer prefixes, the ϕ-value must progress accordingly to be able to distinguish between the traffic volumes of differently sized subnets.

- π^*-values should be evenly distributed. An uneven distribution of π^*-values concentrates a subset of parameter values on a smaller value range. This translates to focusing on specific ratios of legitimate traffic volume to attack traffic volume (as π^* governs filter rule precision). Such a bias should be avoided, since attackers may influence the ratio of attack to legitimate traffic in such a way that it falls outside of the focused value range. Furthermore, the ratio depends on the distribution of attack and legitimate traffic over the IP address space and, consequently, also on the aggregation of traffic throughout the IP prefix hierarchy. This can lead to many different ratios appearing across the IP prefix hierarchy. As a result, modeling a bias towards specific π^*-values into an action space can impede an agent's ability to realize effective trade-offs.

To design the parameter action space according to the previous considerations, parameter values for ϕ and π^* can be defined using two different (linear and geometric) action spaces:

Definition 6.5: Linear Action Space

A **linear action space** $\mathbf{L}_{a,b,n}$ ($a, b \in \mathbb{R}, n \in \mathbb{N}$) is a set of n (real-valued) elements $x_1, x_2, \ldots, x_n \in [a, b]$. The elements are given as $x_i = a + i \cdot \frac{b-a}{n-1}$ $\forall i = 1, \ldots, n$.

Definition 6.6: Geometric Action Space

A **geometric action space** $\mathbf{G}_m$ ($m \in \mathbb{N}$) is a set of m (real-valued) elements $x_1, x_2, \ldots, x_m \in \left[\frac{1}{2}, \frac{1}{2^m}\right]$. The elements are given by the following geometric progression: $x_1 = \frac{1}{2}, x_2 = \frac{1}{4}, x_3 = \frac{1}{8}, \ldots, x_i = \frac{1}{2^i}, \ldots, x_m = \frac{1}{2^m}$.

A geometric action space $\mathbf{G}_m$ models the geometric progression of ϕ-values, while a linear action space $\mathbf{L}_{a,b,n}$ distributes π^*-values equidistantly. The full parameter action space containing all (ϕ, π^*)-combinations is given as the cross product of $\mathbf{G}_m$ and $\mathbf{L}_{a,b,n}$ and denoted as $\mathbf{A}^m_{a,b,n}$:

$$\mathbf{A}^m_{a,b,n} \stackrel{\text{def}}{=} \left\{(\phi, \pi^*) \mid \phi \in \mathbf{G}_m, \pi^* \in \mathbf{L}_{a,b,n}\right\} = \mathbf{G}_m \times \mathbf{L}_{a,b,n} \tag{6.8}$$

Example 6.4.1. Figure 6.6 illustrates the actions $\mathbf{a}_1, \ldots, \mathbf{a}_{672}$ of the parameter action space $\mathbf{A}^{32}_{0.5,1,21} = \mathbf{G}_{32} \times \mathbf{L}_{0.5,1,21}$. The geometric action space $\mathbf{G}_{32}$ offers 32 different choices for ϕ-values. When traffic is uniformly distributed across the IP address space, the ϕ-values can generate up to 2^{32} HHH prefixes to accurately match individual traffic sources. In contrast, the 21 values of $\pi^* \in \mathbf{L}_{0.5,1,21}$ are evenly spaced across the value range at a distance of 0.025. This allows fine-granular adjustments of π^* and avoids placing an arbitrary emphasis on any subregion of the parameter value range. The combined parameter value range of both parameters spans the (two-dimensional) interval $\left[2^{-32}, 2^{-1}\right] \times \left[\frac{1}{2}, 1\right]$.

Appendix A.3.1 describes how a parameter action space containing a selection of ϕ and π^*-values can be implemented in the Python programming language.

6.4.2 Mitigation State Representation

To control the mitigation, an RL agent requires information on the current state of the filter rule generation system, as well as the traffic distribution. Based on this information, the agent can react to changing traffic patterns and maintain mitigation

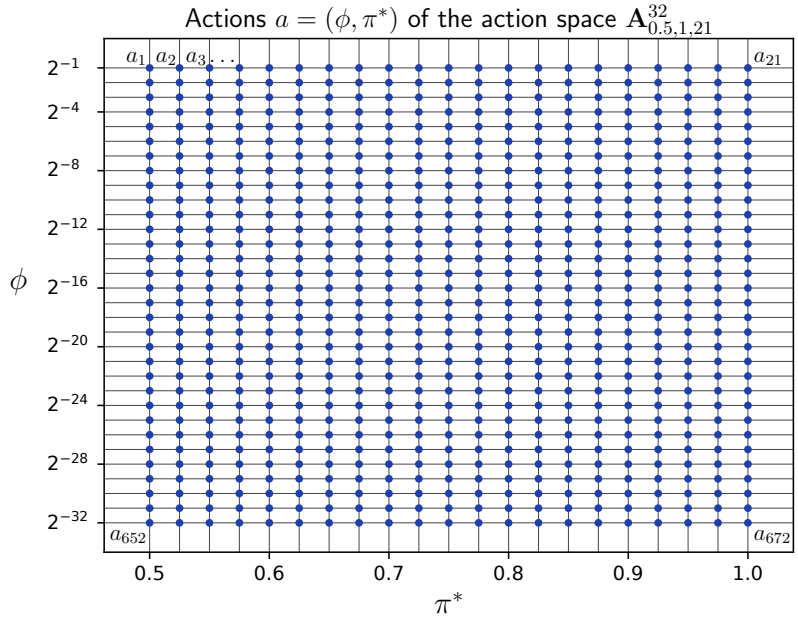


Figure 6.6: Actions $\mathbf{a}_1, \ldots, \mathbf{a}_{672}$ of the parameter action space $\mathbf{A}^{32}_{0.5,1,21} = \mathbf{G}_{32} \times \mathbf{L}_{0.5,1,21}$ arranged in the interval $\left[2^{-32}, 2^{-1}\right] \times \left[\frac{1}{2}, 1\right]$. Values of $\phi \in \mathbf{G}_{32}$ progress geometrically (along the log-scale ϕ-axis), while values of $\pi^* \in \mathbf{L}_{0.5,1,21}$ are evenly spaced.

effectiveness by adapting the parameters ϕ and π^*. In RL, a new state $\mathbf{s}$ is generated whenever the environment finishes executing a single step. An action for a subsequent step is then selected in consideration of the new state $\mathbf{s}_t$. In the context of filter rule generation, taking a step corresponds to updating the blacklist, querying the HHH algorithm, conducting Traffic Purity Estimation, performing Filter Rule Refinement, and processing the traffic over the course of a monitoring interval. Relevant changes in system components and in the traffic occurring over a monitoring interval need to be communicated to the RL agent. The data provided to the agent is herein referred to as the **mitigation state**.

The main goal in designing the mitigation state is to enable a distinction between different situations, so that an agent can learn effective parameter combinations and adapt parameter values when necessary. To describe the current situation, the agent is provided with information on the blacklist, the HHH query result, and the traffic volume distribution encoded in three different features:

- f_{BSIZE}: This feature encodes the blacklist size BSIZE resulting from executing Filter Rule Refinement. The size is encoded as a single scalar value. By informing

Feature	Domain	Description
f_{BSIZE}	$\mathbb{N}$	The number of applied filter rules
f_{HIST}	$\mathbb{N}^{33}$	33 counters that represent the number of HHH prefixes generated at each level of the IPv4 prefix hierarchy
f_{VOLUME}	$\mathbb{R}^{6\times33\times1024}$	A tensor representing the traffic volume distribution over the IPv4 prefix hierarchy as two-dimensional image with six channels

Table 6.2: The features of the mitigation state $\mathbf{s}_t$ and their corresponding domains.

an agent about the blacklist size, the agent can learn to balance BSIZE against FNR and FPR.

- f_{HIST}: A histogram that counts the number of times an HHH prefixes with a specific lengths occurs after querying the HHH algorithm of Aggregate Monitoring. This information serves to enable an agent to learn the relationship between using shorter or longer prefixes and the resulting FPR, FNR, and BSIZE.
- f_{VOLUME}: A tensor that represents the traffic volume distribution across the IPv4 prefix hierarchy during a monitoring interval. The tensor is interpreted as a two-dimensional image and processed with CNNs embedded in the agent's neural network architecture (see Section 6.4.3). It serves to inform an agent of the traffic volume originating from different subnets and to indicate changes in the traffic distribution over the IP address space.

Together, the three features comprise the mitigation state $\mathbf{s} = (f_{\text{BSIZE}}, f_{\text{HIST}}, f_{\text{VOLUME}})$. Table 6.2 provides an overview of the corresponding domains of feature values. Further details on the information encoded by each feature and how the features are computed are provided below.

Blacklist size f_{BSIZE}. The feature f_{BSIZE} encodes the size of the blacklist BSIZE. This serves to indicate whether a lower or higher number of filter rules was applied during a monitoring interval. The blacklist size is calculated after generating a new blacklist through Filter Rule Refinement (with the `generate_blacklist` of Algorithm 4).

Prefix length histogram f_{HIST}. This feature informs an agent whether longer or shorter prefixes have been generated after applying the HHH detection threshold ϕ. For this, the feature encodes the number of occurrences of HHH prefixes of each specific length. Since prefixes over the IPv4 address space have a length between 0

to 32, the feature uses a total of 33 counters. The feature f_{HIST} is calculated after querying the HHH algorithm of Aggregate Monitoring but before performing Filter Rule Refinement. By observing the difference between the sum of the counters in f_{HIST} and the value of f_{BSIZE}, an agent can infer the number of rejected HHH prefixes. This information may be taken into account during neural network training to facilitate more effective choices of π^*.

Traffic volume distribution f_{VOLUME}. The tensor f_{VOLUME} encodes the distribution of traffic volume over the IP prefix hierarchy. This information serves multiple purposes:

- An agent is enabled to learn to distinguish between different traffic volume distributions. This includes distinguishing between attacks with a high or low number of attack traffic sources, concentrated or widely-distributed attack traffic sources, and situations with and without attack traffic. In effect, an agent can learn to react to different situations by adapting mitigation parameters (e.g., by requiring a high minimum required precision π^* in the absence of attack traffic).
- The agent receives information about the traffic volume over the IPv4 prefix hierarchy. Through this, an agent can correlate the total traffic volume with the traffic volume occurring at different levels of the hierarchy. This information can be used to adjust the HHH detection threshold ϕ in order to generate shorter or longer HHH prefixes.

While the traffic volume distribution over the IPv4 prefix hierarchy can provide substantial information, its full representation requires a prohibitive number of distinct counters ($2^{33}-1$ counters to account for each subnet). Such a high-dimensional state would result in high memory consumption and may prevent the use of data structures required by RL algorithms, such as replay buffers. In addition, high-dimensional input layers may be required in the agent's neural network architecture, which leads to high computational complexity during agent training and inference. Instead of representing the traffic volume of each prefix with a separate counter, the feature f_{VOLUME} represents the traffic volume distribution in a more compact fashion. f_{VOLUME} constitutes a tensor that is interpreted as a two-dimensional image with a reduced number of counters. The shape of f_{VOLUME} is determined by two factors:

(1) **Bin-count B.** The bin-count partitions the IPv4 address space into B address bins of equal size. Through this, $2^{32}/B$ IP address are mapped to the same bin, which reduces the number of required counters by a factor of 2^B compared

Algorithm 8: Generating f_{VOLUME}

▷ *This algorithm computes the mitigation state feature f_{VOLUME} from counters stored by Aggregate Monitoring.*

Input : the bin-count B of f_{VOLUME} (a power of two)

```
1  Function get_subnet_bounds(prefix p):
       ▷ Return first and last address of the subnet of a prefix.
2      len ← p.get_length();
3      first_addr ← p.get_address();
4      last_addr ← first_addr + 2^(32−len) − 1;
5      return first_addr, last_addr;

6  Function generate_traffic_image(HHH counters C):
7      f_VOLUME ← new tensor(33, B);        ▷ A 0-initialized tensor of shape 33 × B
8      foreach counter c in C do            ▷ Iterate over all stored counters
9          p ← c.get_prefix();
10         V ← c.get_volume();
11         first_addr, last_addr ← get_subnet_bounds(p);
12         for bin = first_addr/B to last_addr/B do
               ▷ Update traffic volume of bins in the prefixes address range
13             f_VOLUME[p.length][bin] ← f_VOLUME[p.length][bin] + V;
14     return f_VOLUME;
```

to a full representation of all IP addresses. To enable efficient computation of address bins via bit-shifting, the bin-count is chosen as a power of two.

(2) **Maximum prefix length**. The tensor f_{VOLUME} not only encodes the traffic volume of specific IP addresses, but also summarizes the traffic volume of entire subnets that are represented by their IP prefix. To distinguish prefixes of different lengths, it is necessary to allocate one set of address bins for each distinct prefix length (ranging from 0 to 32). This distinction contributes a factor of 33 to the dimension of f_{VOLUME}.

Given the bin-count B, the shape of f_{VOLUME} is $33 \times B$. Throughout this thesis, a bin-count of $B = 1024$ is used as trade-off between mitigation state size and provided information. This reduces the number of entries in f_{VOLUME} to 33,792 compared to a full representation of the IPv4 subnet hierarchy that requires $2^{33} - 1$ counters.

The tensor f_{VOLUME} is generated from traffic volume information stored by the HHH algorithm of Aggregate Monitoring. To utilize all available information, the full set of stored counters is used (i.e., no HHH detection threshold is applied). The traffic volume stored for each monitored prefix `p` is accumulated in corresponding entries of the tensor f_{VOLUME}. The first dimension of f_{VOLUME} determines the hierarchy level of a prefix, while the second dimension determines the range of the address space covered by `p`. This can be interpreted as rendering a two-dimensional image of the traffic volume distribution.

The rendering process uses Algorithm 8. First, all entries in f_{VOLUME} are initialized to zero (in Line 7). The algorithm then iterates over all counters `c` and associated prefixes `p` stored in the data structure of an HHH algorithm (Line 8). The length `len` of each prefix `p` determines the index along the first dimension of f_{VOLUME} (i.e., the hierarchy level). Subsequently, the algorithm updates one or more entries along the second dimension of f_{VOLUME}. For this, it calculates the first and last address of the prefixes `p` subnet (using the `get_subnet_bounds` function in Line 1). These addresses determine the first and last bins of f_{VOLUME} that need to be updated. The `for` loop in Line 12 then increments the entries of all affected bins by the traffic volume stored in the counter `c` of prefix `p`. After iteration over all prefixes, f_{VOLUME} represents the cumulative sum of the traffic volume in each subnet, which is furthermore aggregated through binning. The rendered information represents the distribution of the traffic over the IP address space and over the IP prefix hierarchy.

Figure 6.7 visualizes an example of the tensor f_{VOLUME} rendered from traffic volume information collected over one monitoring interval. Darkened regions indicate the presence of traffic, while white regions indicate its absence. Each bin corresponds to a range of 2^{22} consecutive IP addresses as a result of choosing a bin-count $B = 1024$.

6.4.3 Neural Network Architectures

The neural network architecture of an agent maps input features to selected actions. In the context of RL-based Control, it maps the three features f_{BSIZE}, f_{HIST}, and f_{VOLUME} to a single action $\mathbf{a}$ in the action space $\mathbf{A}^{32}_{0.5,1,21}$. The three features have different dimensionalities, which requires input processing before an action can be chosen by a common neural network. Therefore, the neural network architecture is structured into input processing followed by a neural network for action selection.

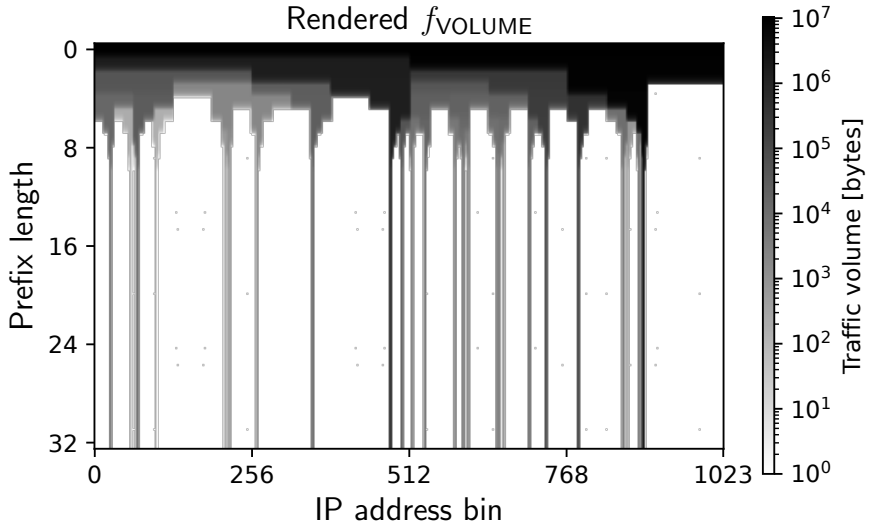


Figure 6.7: Example rendering of the feature f_{VOLUME} from traffic volume information monitored over one monitoring interval. Each bin corresponds to 2^{22} IP addresses. The traffic volume is displayed on a logarithmic scale for improved visibility.

Figure 6.8 provides an overview of the resulting neural network architecture of an agent conducting RL-based Control. The neural network processes input features through the following three steps:

(1) Initially, the neural network receives the three input features f_{BSIZE}, f_{HIST}, and f_{VOLUME}. Since the two-dimensional shape of the tensor f_{VOLUME} cannot be directly processed by a feed-forward neural network, a dimensionality reduction is required. For this, the neural network architecture embeds a CNN that computes an embedding $f_{\text{EMBED}} \in \mathbb{R}^{64}$ with a one-dimensional shape. This embedding can be used by a feed-forward neural network to select an action (in Item 3).

(2) The embedding f_{EMBED} is combined with the remaining input features. The vectors f_{BSIZE} and f_{HIST} remain unchanged until they are concatenated with f_{EMBED} to generate a single vector $f_{\text{CONCAT}} \in \mathbb{R}^{98}$. This vector represents the combined information on the blacklist size, the prefix length distribution, and the most relevant information on the traffic volume distribution that is extracted by the CNN.

(3) In the final step, the vector f_{CONCAT} provides the input for a feed-forward neural network. This neural network consists of a sequence of stacked linear layers

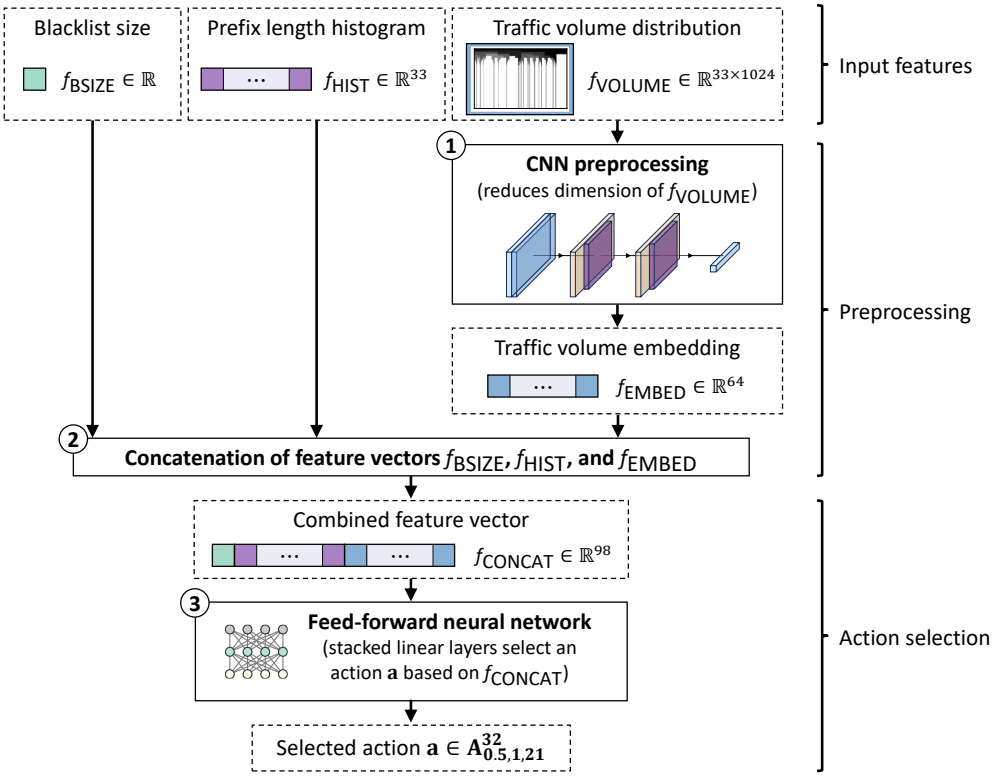


Figure 6.8: Architecture of an RL agent's neural network. Input and intermediate features are included to illustrate the flow of data in the neural network.

with multiple neurons that process f_{CONCAT}. This sequence is followed by a single neuron that outputs an action $\mathbf{a}$ from the action space $\mathbf{A}^{32}_{0.5,1,21}$. While the CNN only processes the traffic volume distribution, the feed-forward neural network can correlate the traffic volume with the blacklist size and the prefix length histogram. Consequently, an agent can learn the relationship between all features comprising the mitigation state, chosen actions, and the achieved reward (through back-propagation).

Further details on the architecture of the CNN and the feed-forward neural network are provided below.

CNN architecture. The CNN architecture consists of several layers that are processed in a sequential fashion. Table 6.3 provides the exact specifications of each layer, while Figure 6.9 visualizes the CNN's architecture. The neural network is structured into two parts:

Building block	Layer	Input shape	Kernel size	Stride	# Filters	Activation function	Output shape	Description
Convolutional part								
$\mathbf{B}_1$	Conv_1	$(33, 1024)$	$(2, 3)$	$(1, 2)$	8		$(8, 32, 511)$	2D convolutional layer
	BN_1	$(8, 32, 511)$				ReLU	$(8, 32, 511)$	Batch normalization
	MP_1	$(8, 32, 511)$	$(1, 2)$	$(1, 2)$			$(8, 32, 255)$	Max-pooling layer
$\mathbf{B}_2$	Conv_2	$(8, 32, 255)$	$(2, 3)$	$(1, 2)$	16		$(16, 31, 127)$	2D convolutional layer
	BN_2	$(16, 31, 127)$				ReLU	$(16, 31, 127)$	Batch normalization
	MP_2	$(16, 31, 127)$	$(2, 2)$	$(2, 2)$			$(16, 15, 63)$	Max-pooling layer
$\mathbf{B}_3$	Conv_3	$(16, 15, 63)$	$(2, 2)$	$(1, 1)$	32		$(32, 14, 62)$	2D convolutional layer
	BN_3	$(32, 14, 62)$				ReLU	$(32, 14, 62)$	Batch normalization
	MP_3	$(32, 14, 62)$	$(2, 2)$	$(2, 2)$			$(32, 7, 31)$	Max-pooling layer
Linear part								
$\mathbf{B}_4$	Flatten	$(32, 7, 31)$					6944	Flatten operation
	Linear_1	6944				ReLU	64	Linear layer
	Linear_2	64				ReLU	64	Linear layer

Table 6.3: The sequence of layers (top to bottom) used by CNN to process the traffic volume distribution f_{VOLUME}. The table specifies the layer parameters where applicable.

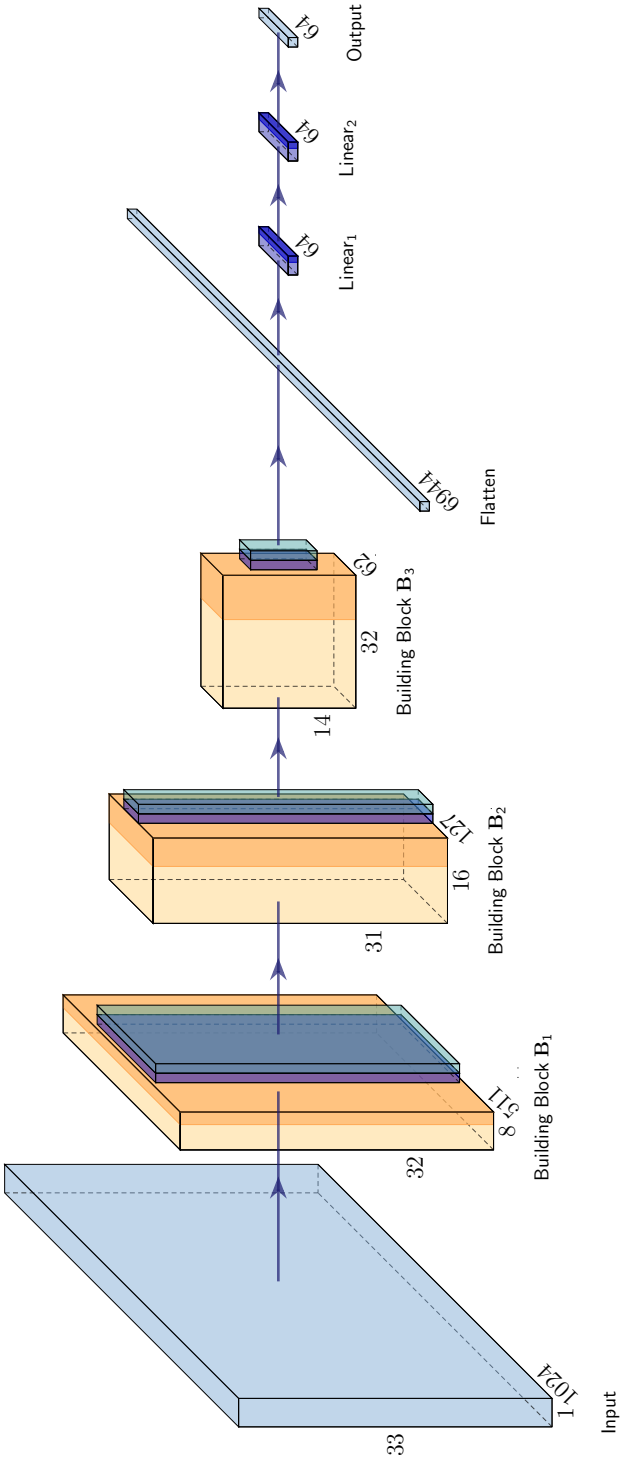


Figure 6.9: CNN used for traffic image feature extraction. Depicted numbers specify the dimension of tensors (after applying convolutional layers, the flatten operation, and linear layers). Tensor dimensions after applying batch normalization and max-pooling layers are omitted for clarity.

- **Convolutional part**. The first part uses three building blocks $\mathbf{B}_1, \ldots, \mathbf{B}_3$. Each building block consists of a two-dimensional convolutional layer, a batch normalization layer, an activation function, and a max-pooling layer. The layers and building blocks are processed sequentially.

- **Linear part**. The linear part consists of a single building block $\mathbf{B}_4$. This building block reshapes the output of the convolutional part to have a one-dimensional shape and processes the result with a sequence of linear layers.

Processing starts on the feature f_{VOLUME} with the first building block of the convolutional part. Within each building block, it progresses through the following steps:

(1) Initially, a two-dimensional convolutional layer Conv_i processes an input tensor. This applies (trainable) convolutions with fixed kernel size and stride to the entries of the input tensor. The resulting tensor consists of multiple channels that encode different extracted features of the input data. Relevant features are learned during neural network training. Furthermore, a convolutional layer reduces the dimension of each channel depending on the specified kernel size and stride. The repeated application of convolutional layers enables a neural network to create additional features from previous convolutional layer passes.

(2) A batch normalization layer BN_i is applied to the output of convolutional layers with multiple sequential layers. This layer normalizes input tensors to have a mean of zero and unit variance per batch. Normalizing in this fashion reduces the internal covariance shift of a neural network [IS15]. In effect, this reduces the risk of vanishing and exploding gradients and facilitates training of deep neural networks.

(3) Each normalization layer is followed by a ReLU activation function. The activation function introduces necessary non-linearity into neural network processing. Using the ReLU activation function for convolutional neural network architectures is a common design choice. Its placement after batch normalization layers follows current best-practices in CNN architecture design.

(4) A max-pooling layer MP_i is applied to the output of the ReLU activation function. Max-pooling maps the input of a kernel of fixed size to a single scalar value that denotes the kernels maximum. Through this, the output tensor of a max-pooling layer has fewer coefficients than the original input. The repeated application of max-pooling layers serves to reduce the dimension of processed tensors in order to avoid highly complex neural networks.

Layer	Input shape	Activation function	Output shape	Description
Linear_1	98	ReLU	200	Linear layer
Linear_2	200	ReLU	200	Linear layer
Linear_3	200	none	672	Linear layer

Table 6.4: The sequence of layers (top to bottom) of the feed-forward neural network used for action selection.

(5) Neural network processing continues with the convolutional layer of the next building block of the convolutional part. If the convolutional part has been fully processed, it continues with the linear part instead.

The output of the convolutional part is a tensor with a three-dimensional shape. This tensor cannot be directly processed by the feed-forward neural network of an RL agent. To reduce the tensor to a one-dimensional shape, the linear part performs the following two steps:

(1) The operation Flatten reduces the output tensor of the convolutional part to a vector with 6944 coefficients and a one-dimensional shape.

(2) Two linear layers Linear_1 and Linear_2 with 64 output neurons are applied after the Flatten operation. Each layer applies the ReLU activation function to introduce further non-linearity. The additional layers serve to further reduce the dimension of the output vector and to train the CNN to extract the most relevant information from the traffic volume distribution.

The result of CNN preprocessing is a tensor f_{EMBED} of one-dimensional shape with 64 coefficients. This tensor provides a part of the input for subsequent action selection.

Action selection. After preprocessing, the tensor f_{EMBED} is concatenated with the (unchanged) features f_{BSIZE} and f_{HIST}. The resulting tensor with 98 coefficient constitutes the input for the feed-forward neural network that chooses an agent's action. Table 6.4 specifies the architecture of this neural network. Specifically, it consists of a sequence of three linear layers with the following properties:

- The first layer Linear_1 has an input shape of 98, which matches the number of coefficients of the concatenated input tensor f_{CONCAT}. The layer has 200 neurons and corresponding output shape.

- The intermediate layer Linear_2 uses 200 neurons to map the input from Linear_1 to Linear_3 without changing its shape.

- The final layer Linear_3 outputs a tensor with 672 coefficients. Each coefficient represents the q-value of a single action among the 672 actions in the action space $\mathbf{A}^{32}_{0.5,1,21}$.

The first two layers use the ReLU activation function to introduce non-linearity into neural network processing. The final layer does not apply an activation function, so that output values can span an arbitrary value range. The number of layers and neurons per layer is chosen sufficiently high for an agent to learn to distinguish between different traffic situations and to choose values for ϕ and π^* accordingly. While higher numbers may offer more capacity to store learning information, overly complex neural networks can impede an agent's ability to learn.

A forward pass through the neural network outputs q-values for all actions. Subsequently, the action with the highest q-value is selected. This exploits learned information encoded in the neural network in an effort to choose effective parameter combinations of ϕ and π^* and to maximize an RL agent's reward.

6.5 Training and Inference

The following sections elaborate how DQN agents can be trained to conduct RL-based Control. In particular, the choice of hyperparameters values applied during training is discussed in Section 6.5.1. In addition, Section 6.5.2 outlines key differences between the training process an the execution of an agent during inference.

6.5.1 Training with the DQN Algorithm

The following outlines the design choices that determine how an agent is trained with the DQN algorithm to perform RL-based Control. In all other regards, the training is performed as described in Section 2.5.

Replay buffer capacity and initialization. Experience replay using a replay buffer allows a DQN agent to learn from past transitions, which is intended to improve agent training (as outlined in Section 2.5). The replay buffer capacity constitutes a hyperparameter in DQN training and must be tuned to the learning task to achieve overall high reward. For the task of controlling filter rule generation, the probability of choosing effective ϕ and π^* values must be taken into account when choosing the buffer capacity. Only a small subset of the action space $\mathbf{A}^{32}_{0.5,1,21}$ (of size 672) may prove effective for a specific traffic pattern. For example, choosing high ϕ-values generates short filter rule prefixes, which will be rejected when choosing a high π^*-value at the same time (assuming the presence of legitimate traffic). The result is a high FNR and low reward. Conversely, combining a low ϕ-value and low π^*-value generally results in high FPR and high BSIZE. Only few actions in the action space $\mathbf{A}^{32}_{0.5,1,21}$ (one percent or less) may be suitable to achieve the highest reward after training. In effect, a random choice of actions results in a bias towards low-reward transitions. This bias is more pronounced the fewer options for parameter selections are effective throughout a traffic scenario.

When agent training starts, the replay buffer is empty and must be filled so that an agent can begin to learn from an initial set of transitions. Since no trained policy is available at this time to choose effective actions, the DQN algorithm samples actions uniformly at random from the action space $\mathbf{A}^{32}_{0.5,1,21}$. The sampled actions control the environment while filling the replay buffer to its full capacity (without training the agent's neural networks). Since actions are chosen at random, they disregard the mitigation state, the reward, and any relationship between ϕ and π^*. Due to the bias towards low-reward transitions, only a small subset of the initial transitions in the replay buffer will have high reward.

The initial bias in the replay buffer can have a (potentially long-lasting) adverse effect on agent training. First, sampling from the replay buffer results in low-reward transitions more frequently, so that an agent has a reduced chance to learn from transitions with high reward. Second, the effect lasts longer with increasing replay buffer capacity as new transitions replace old transitions at a one-to-one ratio with each training step. Consequently, a larger replay buffer stores the initial low-reward transitions for a longer time. To reduce the impact of having a low chance to randomly choose effective ϕ and π^* combinations while filling the replay buffer, the capacity of the buffer can be limited. Choosing a replay buffer capacity of 10,000 transitions replaces all initial transitions after an equal number of training steps. Thereafter, the agent primarily learns from transitions, which result from using a policy network that is continuously trained to achieve higher reward. Through this, the initial impact of

the sampling bias towards low-reward transition diminishes when training an agent over longer periods (in the order of millions of training steps).

During training (and after replay buffer initialization) a large replay buffer can be the cause of so-called catastrophic forgetting [MC89; Rat90]. When an agent learned a sub-optimal policy over an extended period of training steps, transitions in the replay buffer are biased towards low or intermediate reward. Each time the agent's neural networks learn from these transitions, neural network weights that contribute to choosing effective actions are also affected by the weight updates of the latest training step. Through this, the neural network effectively forgets effective parameter choices, which can lead to continuously decreasing reward as training progresses.

In contrast, the drawback of low replay buffer capacity is that agent training may fail to achieve overall high reward. When an agent learns a suboptimal policy during training, the agent can be prone to choosing actions that yield only intermediate reward. With lower replay buffer capacity, the resulting transitions are more likely to replace existing transitions in the buffer that have high reward. In this case, the agent primarily trains on transitions generated from choosing suboptimal actions.

To account for the bias towards ineffective combinations of ϕ and π^*, a replay buffer capacity of 10,000 transitions is chosen when training DQN agents for the task of filter rule generation. This hyperparameter value constitutes a middle ground between large and small replay buffer capacities. It avoids catastrophic forgetting and proves effective to achieve overall high reward for all experiments outlined in Section 6.6.

Exploration rate schedule. During training, a DQN algorithm applies an ϵ-greedy strategy to sample actions uniformly at random from the action space instead of applying the action chosen by an agent's policy. The probability at which actions are sampled at random can be adjusted with an exploration schedule over the course of agent training. This probability constitutes a trade-off between exploring more combinations of mitigation states, actions, and the resulting reward and further optimizing a well-trained policy. Defining an exploration schedule must take into account that only a small subset of the action space $\mathbf{A}^{32}_{0.5,1,21}$ may prove effective for a specific traffic pattern (as outlined above).

At the start of agent training, the exploration rate should be high to account for the previously outlined bias of sampling transitions with low reward more frequently than those with high reward. The bias in combination with a low exploration rate carries the risk of settling for policy networks that choose ineffective actions during the early phase of agent training. In contrast, a high exploration rate causes frequent

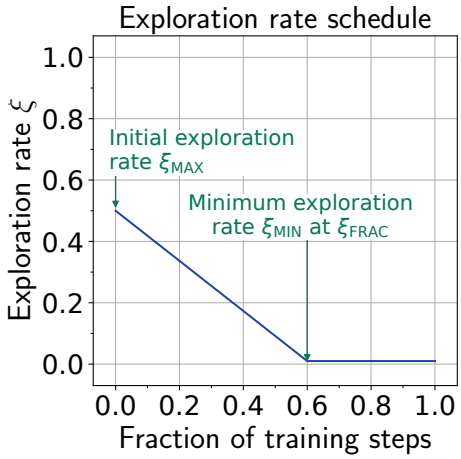


Figure 6.10: Exploration rate schedule with initial exploration rate $\epsilon_{\text{MAX}} = 0.5$ and minimum exploration rate $\epsilon_{\text{MIN}} = 0.01$ (reached at a fraction of $\epsilon_{\text{FRAC}} = 0.6$ of the total training steps).

weight updates in an agent's policy network due to repeatedly sampling high-reward transitions. This bootstraps agent training so that more effective parameter combinations of ϕ and π^* are chosen as the training progresses. This also counteracts the bias towards low-reward transitions as a result of random replay buffer initialization, as the initial transitions are increasingly replaced by transitions with higher reward.

As the policy network of an agent learns to select actions that yield higher reward, the exploration rate should decrease. This reduces the risk that transitions with low reward cause detrimental weight updates to an agent's neural network, which may prevent an agent from achieving an overall high reward. As a further consequence, the exploration rate should be significantly reduced at later training stages. Through this, any lingering impact arising from earlier random sampling of low-reward transitions can be reduced, and the agent can be trained to choose effective actions more consistently. Still, the exploration rate should not completely vanish, as the distribution of transitions generated from interacting with a mitigation environment is non-stationary. In other words, later interactions with the environment can result in new traffic patterns to which an agent should still learn to react.

Reducing the learning rate during training with a linear exploration schedule accommodates these considerations. This technique is often applied to achieve high reward when training RL agents. A linear exploration rate schedule starts with a chosen initial exploration rate ϵ_{MAX}. The schedule decreases the exploration rate until it reaches a minimum exploration rate ϵ_{MIN} at a predefined fraction ϵ_{FRAC} of the total

training steps. Figure 6.10 visualizes an exploration rate schedule with $\epsilon_{\text{MAX}} = 0.5$, $\epsilon_{\text{MIN}} = 0.01$, and $\epsilon_{\text{FRAC}} = 0.6$. This schedule is applied when using the DQN algorithm for agent training. The high value of ϵ_{MAX} results in random actions being applied to the environment half of the time at the start of training. After 60 % of the total training steps, only 1 % of all actions are sampled at random. The significantly lower value of $\epsilon_{\text{MIN}} = 0.01$ is necessary to improve an agent's policy network until it consistently chooses ϕ and π^* values that realize effective trade-offs. This is due to the fact that slight parameter variations can cause strong variations in FPR, FNR, and BSIZE and reduce mitigation effectiveness. When an agent is trained on randomly sampled transitions during the final part of training, its ability to realize effective trade-offs can be diminished due to detrimental updates of its neural network.

Optimizer and learning rate schedule. Using a constant learning rate during agent training can prevent agents from achieving high reward. This is because the gradients calculated for weight updates during a backward pass are proportional to the losses when the learning rate remains constant. If losses do not decrease, the weights of a neural network will not converge. In deep RL, this leads to an agent frequently choosing actions that result in suboptimal reward. Therefore, training uses an adam optimizer to adjust the learning rate in each step. In addition, a learning rate schedule continuously reduces the maximum learning rate of all weights to ensure that neural networks eventually stabilize.

Soft-updates. A DQN algorithm uses a neural network (the value network) to learn the q-values of actions in the action space for a given observation. However, selecting actions directly with the value network during training can lead to reduced reward and diminish an agent's performance. Instead, a second neural network (the target network) is often used as a surrogate for the value network. The target network is updated less frequently than the value network, so that actions determined by the target network are more stable during training. Specifically, the target network's parameters θ' are periodically updated from the value network's parameters θ. The number of steps between updates denotes the **target update intervals** $\upsilon_{\text{INTERVAL}} \in \mathbb{N}$. When using so-called soft updates [Lil+19] the target network receives weighted updates. In this case, the parameters of the target network are updated as follows: $\theta' \leftarrow \upsilon_{\text{WEIGHT}}\theta + (1-\upsilon_{\text{WEIGHT}})\theta'$, where $\upsilon_{\text{WEIGHT}} \in \mathbb{R}$ denotes the **target update weight**. The choice of $\upsilon_{\text{INTERVAL}}$ and υ_{WEIGHT} can have a critical impact on DQN training (e.g., by causing catastrophic forgetting). The values for $\upsilon_{\text{INTERVAL}}$ and υ_{WEIGHT} used in this thesis were experimentally determined. Specific values are provided in the evaluation in Section 6.6.2.

Parallel training environments. The DQN algorithm used in this thesis collects transitions from multiple training environments in parallel to reduce training time. The environments are separated via process isolation to avoid side effects. This includes isolation of neural networks performing Traffic Purity Estimation, which are embedded into the environment. Furthermore, each environment receives a unique random seed to avoid duplicate transitions that would result in a bias of the distribution of entries in the replay buffer.

Reward discount factor. The reward discount factor γ determines the extent to which future transitions are considered in the calculation of the reward for choosing an action. In the context of generating filter rules for volumetric DDoS mitigation, each observation is considered to be independent of future network states. This is because the traffic sent by traffic sources is independent of actions chosen by RL-based Control. Consequently, the reward discount factor γ is set to zero.

6.5.2 Inference with the DQN Algorithm

To evaluate an agent's ability to adapt mitigation parameters, its neural network is processed in an inference mode. That is, the training of the neural network is suspended and only forward-passes are performed. In particular, inference affects the following aspects of an RL agent.

Random choices. During inference, randomization that occurs during training is disabled. This includes the random choice of actions as part of the exploration as well as random sampling from a replay buffer (which is not used during inference). In effect, a DQN agent responds in a deterministic fashion to observations. Still, randomization can occur during traffic filtering due to probabilistic updates of HHH data structures and random sampling of packets. To ensure consistent results across multiple experiments conducted in the evaluation in Section 6.6, random generators used in traffic processing are initialized with fixed seeds. This also facilitates reproducibility of evaluation results.

Neural network parameters. In contrast to approaches like online learning, the parameters of an agent's neural network are no longer updated during inference. This also applies to normalization parameters such as the mean and variance used by batch normalization. Once these parameters have been learned during training, they are applied without change to new observations collected during inference. Therefore,

an agent's ability to realize effective trade-offs depends in part on the normalization technique's ability to generalize to previously unseen traffic patterns.

6.6 Evaluation

This section evaluates the ability of RL-based Control to realize effective trade-offs under different conditions. Specifically, multiple experiments evaluate the ability of RL-based Control to maintain effective trade-offs when the blacklist size is increasingly prioritized. Reducing the blacklist size makes it more challenging to balance FPR and FNR against BSIZE.

The results presented in this section are based on event-discrete simulation of traffic filtering with RL-based Control over the course of a mitigation scenario. Section 6.6.1 outlines the general evaluation procedure while Section 6.6.2 describes the evaluation setup, including simulation system specifications, hyperparameters of RL algorithms, and mitigation environment parameters. Experiments under different conditions are presented throughout Section 6.6.3 to Section 6.6.5. The chapter concludes with a discussion of the most relevant findings in Section 6.6.6.

6.6.1 Evaluation Procedure

The evaluation conducts multiple experiments under different conditions, each of which progresses through the following common steps:

(1) Parameters for Aggregate Monitoring, Filter Rule Refinement, and the reward function $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$ (from Definition 6.3) are selected to model different conditions under which simulated traffic filtering is performed.

(2) An agent is trained over a fixed number of training steps by interacting with a training environment. The environment simulates traffic filtering in a mitigation scenario. In particular, the environment performs Aggregate Monitoring, Traffic Purity Estimation, and Filter Rule Refinement so that agents can observe the mitigation state resulting from the interaction of these components. In addition, the environment calculates the reward for achieved FPR, FNR, and BSIZE in order to inform the agent of the effectiveness of its chosen actions. During training, instances of the mitigation scenario are repeatedly randomized to continuously generate new traffic patterns.

(3) The model of the trained agent is used to perform inference on a separate evaluation environment. The evaluation environment simulates traffic filtering over ten differently randomized traffic scenario instances, to evaluate mitigation effectiveness on diverse traffic patterns. To reduce the risk of overfitting on specific patterns, the evaluation environment uses a different random seed than training environments for traffic generation. Furthermore, using a fixed random seed ensures comparability of different experiments. The evaluation environment records the achieved FPR, FNR, and BSIZE for each monitoring interval.

The metrics obtained from the evaluation environment are used in the remainder of this section to evaluate and discuss experiment results. Recording metrics during evaluation instead of training ensures that random choices made for the purpose of training an agent do not occur during an active mitigation. Otherwise, such choices can reduce mitigation performance. For example, sampling actions at random (due to exploration) can have a detrimental effect on the effectiveness of filter rules.

Traffic scenario. All experiments are conducted on the mitigation scenario C with four distinct phases ① to ④ that is defined in Section 5.4.1. Both training and evaluation environments use the scenario C, but each environment uses a different fixed random seed. This results in different traffic episodes for each environment, so that training and evaluation of RL models uses traffic with distinct characteristics. The results of each experiment are calculated over ten (randomized) traffic episodes, which corresponds to traffic filtering over a duration of 9000 seconds. Using a common scenario and fixing the random seed of the evaluation environment ensures that results of different experiments remain comparable despite randomization of traffic episodes.

Note that the mitigation scenario C differs from the scenarios A and B used to train and test Traffic Purity Estimation models (described in Section 4.5.1). This takes into account that traffic characteristics can change between the training of Traffic Purity Estimation models and conducting traffic filtering at a later time. If this is disregarded, the Traffic Purity Estimation model may be overfitted and produce artificially low Traffic Purity Estimation errors. Such an overfitting makes it easier to accurately choose the parameter π^* and may lead to artificially improved traffic filtering results under RL-based Control. Using a different mitigation scenario C evaluates whether RL-based Control still proves effective when Traffic Purity Estimation needs to generalize to previously unseen traffic characteristics.

		Phase			
		①	②	③	④
Attack sources per monitoring interval	median	36.0	35.0	1420.0	1421.0
	mean	36.3	35.2	1407.4	1407.1
	90 %-quantile	44.0	42.0	1471.0	1470.0
	maximum	55.0	52.0	1532.0	1525.0

Table 6.5: Median, mean, 90 %-quantile, and maximum number of attack sources sending traffic in each monitoring interval. The listed values distinguish between four attack phases ① to ④ of the mitigation scenario.

Trying to reduce the blacklist size while also reducing FPR and FNR becomes increasingly challenging the more BSIZE is below the number of attack traffic sources. To provide a reference, Table 6.5 summarizes the mean and maximum number of attack traffic sources that send traffic during each monitoring interval throughout the four phases ① to ④ of scenario C. Recall that during the first two phases a low number of uniformly distributed attack traffic sources send high-volume traffic, while a higher number of normally distributed sources send low-volume attack traffic during phase ③ and ④ (see Table 5.1). Due to these distributions, the average distance of attack traffic sources is lower during phase ③ and ④ compared to the previous two phases. The lower distance facilitates using filter rules with shorter prefixes that discard traffic from multiple attack traffic sources. In contrast, attempting to use a single filter rule to discard traffic from two attack traffic sources in phase ① and ② carries a high risk of inadvertently removing legitimate traffic originating from the (larger) intermittent address space. Therefore, blacklist size reductions can be expected to occur primarily during the last two phases. Still, the inclusion of the first two phases serves to evaluate whether RL-based Control can learn to adapt to changes in attack traffic distributions that occur over time.

Experiment roadmap. The evaluation conducts a total of seven experiments Exp_1 to Exp_7 to evaluate mitigation performance under different limitations on the blacklist size. Agents are trained to realize different trade-offs by shaping the reward term for BSIZE. This is achieved by using different shaping functions $\sigma^{(1)}_{\mathsf{BSIZE}}$ to $\sigma^{(7)}_{\mathsf{BSIZE}}$ in the nine different experiments (see Equation 6.6). The choice of the threshold $\vartheta 1$ for each shaping function $\sigma^{(1)}_{\mathsf{BSIZE}}$ is given in Table 6.6. This threshold marks the start of a reduction in reward for increasing blacklist size. Two additional thresholds $\vartheta_2 = \vartheta_1 + 200$ and $\vartheta_3 = \vartheta_2 + 200$ denote when reward decreases more rapidly. The

Experiment	Threshold ϑ_1
Exp_1	1500
Exp_2	1300
Exp_3	1000
Exp_4	900
Exp_5	800
Exp_6	700
Exp_7	600

Table 6.6: Parameter ϑ_1 used by the shaping function σ_{BSIZE} in experiments Exp_1 to Exp_7.

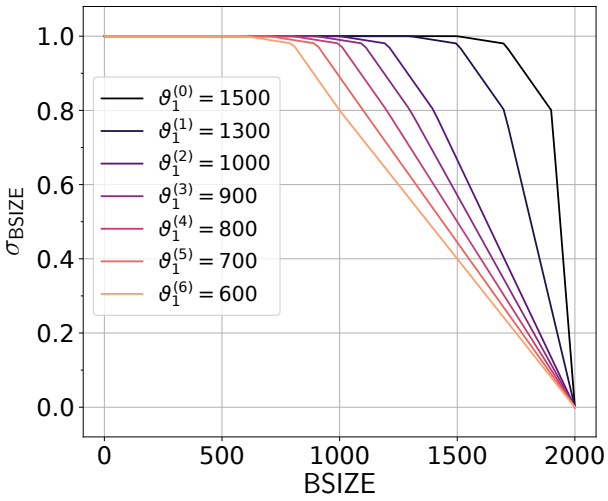


Figure 6.11: Shaping functions $\sigma^{(i)}_{\mathsf{BSIZE}}$ using $\vartheta_2^{(i)}$ and $\vartheta_3^{(i)}$ of experiment Exp_i from Table 6.6 and $\vartheta_3^{(i)} = \vartheta_2^{(i)} + 200 = \vartheta_1^{(i)} + 400$.

corresponding rewards are $\mathbf{r}_2 = 0.98$ at ϑ_2 and $\mathbf{r}_3 = 0.8$. The resulting shaping functions are depicted in Figure 6.11. This progression of rewards models low and high incentives to decrease the blacklist size. The initial slight decrease to $\mathbf{r}_2 = 0.98$ offers a tolerance to emphasize lower FPR and FNR while approaching ϑ_2. Blacklist sizes larger than ϑ_2 should generally be avoided to maintain high reward. Each successive experiment reduces the thresholds to evaluate under which conditions RL-based Control can maintain effective trade-offs.

6.6.2 Evaluation Setup

The following provides an overview of the framework used to evaluate RL-based Control. In addition, system specifications, hyperparameters, and mitigation environment parameters are described.

Evaluation framework overview and system specifications. The evaluation uses simulation to train multiple agents to generate filter rules and control traffic filtering under different conditions. Figure 6.12 illustrates implementation details of the evaluation framework used to conduct experiments throughout this section. The evaluation framework is divided into two main parts: a Python-based RL framework and a C++-based environment. The RL framework performs agent training and

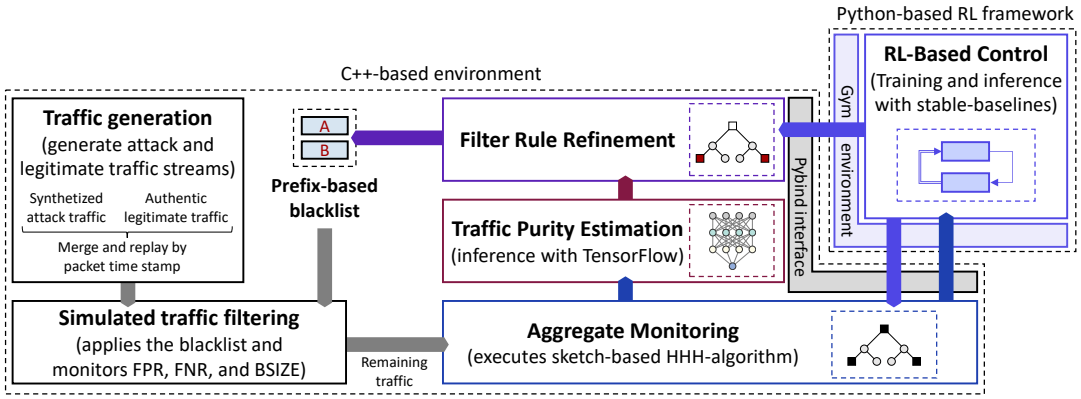


Figure 6.12: Overview of the evaluation framework.

evaluates a trained agent's policy through inference. The environment is responsible for traffic generation, Aggregate Monitoring, Traffic Purity Estimation, and Filter Rule Refinement. In addition, it simulates traffic filtering by applying generated filter rules, keeps track of BSIZE, and calculates the achieved FPR and FNR in each monitoring interval. The interaction between an agent and an environment consists of choosing ϕ and π^* parameters at the beginning of a monitoring interval, as well as observing the resulting monitoring state at the end of the interval. To facilitate training on multiple environments, several instances of the environment can be executed in parallel without interfering with each other.

The evaluation framework is executed on a bare-metal-server with hardware and software specifications provided in Table 6.7. RL agent training utilizes the Stable Baselines 3 library [Raf+21], which provides a configurable reference implementation for the DQN algorithm. For neural network processing, Stable Baselines 3 relies on the PyTorch library [Pas+19], which offers GPU-accelerated tensor computation and automatic calculation of gradients. In combination, these libraries provide the basic functionality for agent training.

The environment is implemented in C++ (using the C++17 standard) to facilitate efficient event-discrete simulation. Traffic generation either replays static capture files (based on the pcap file format [HR24]) or synthesizes packets through random sampling. Packet randomization utilizes the C++ standard template library to implement random distributions required to model the mitigation scenario. During a monitoring interval, traffic generation replays all packets of a traffic scenario that fall into the corresponding time frame. The environment then simulates traffic filtering by discarding packets that fall into the address range of filter rule prefixes. All other packets

Category		Specification
Hardware	CPU	AMD EPYC 7402P (3.35 GHz, 24 cores, 48 threads)
	RAM	128 GB DDR4-3200 ECC
	GPU	2x Nvidia A40 (96 GB GDDR6 vRAM)
Software	Ubuntu OS	Version 20.04.6 LTS
	Python	Version 3.8.10
	TensorFlow	Versions 2.9.3 of the C-API
	PyTorch	Version 2.0.1
	Stable Baselines 3	Version 1.8.0
	Gym	Version 0.21.0
	g++	Version 9.4.0 (configured to C++17 standard)
	pybind11	Version

Table 6.7: Specifications of the system used to evaluate RL-based Control.

are processed by aggregate monitoring, which computes HHHs using a CocoSketch algorithm [Zha+21]. To perform Traffic Purity Estimation with pre-trained neural networks on aggregate information, it is necessary to link the TensorFlow runtime library into the C++-based environment. A context switch to the standard Python API of TensorFlow would incur prohibitive overhead, as it requires running Python in a subprocess and serializing the features of all aggregates in each monitoring interval. Instead, the environment utilizes TensorFlow's C++ API directly. The C++-based Filter Rule Refinement generates new blacklists in each monitoring interval using the algorithms introduced in Chapter 5.

Coupling the Python-based RL framework with the C++-based environment is achieved by using two additional libraries: Gym and pybind11. Python's Gym library offers a streamlined interface to implement environments and to exchange observations, actions, and rewards between an environment and an agent. Through this, implementation details of the environment can be hidden from the RL framework in order to facilitate a modular design. For example, the DQN algorithm can be replaced by other RL algorithms without changes to the environment. The pybind11 library binds C++ functions to Python functions and manages the transfer of function arguments and return values. This enables seamless interaction between the two programming

Hyperparameter	Value
Learning rate schedule	$\lambda_{\text{MAX}} = 10^{-4}, \lambda_{\text{MIN}} = 10^{-5}$
Exploration rate schedule	$\epsilon_{\text{MAX}} = 0.5, \epsilon_{\text{MIN}} = 0.01, \epsilon_{\text{FRAC}} = 0.6$
Soft-updates	$\upsilon_{\text{INTERVAL}} = 1000, \upsilon_{\text{WEIGHT}} = 0.001$
Reward discount	$\gamma = 0$
Optimizer parameters	$\beta_1 = 0.5, \beta_2 = 0.99$
Total training steps	1,500,000
Replay buffer capacity	10,000
Batch size	64
Parallel training environments	4

Table 6.8: Hyperparameters used for DQN training.

languages, so that event-discrete simulation can run efficiently in C++ while agent training can utilize Python-based RL frameworks.

The RAM capacity of the server state in Table 6.7 is sufficient to store all transitions collected during RL training in memory. Furthermore, the GPU's vRAM capacity allows it to run training and inference of all neural networks in parallel. This includes the neural network of an RL agent as well as the neural networks that perform Traffic Purity Estimation on traffic aggregates in multiple environments. The multi-core CPU also supports the execution of several environments in parallel.

Hyperparameters. To ensure comparability between different experiments, agent training uses common hyperparameters for all agents (except for adjusted reward function parameters). The values used for all hyperparameters are summarized in Table 6.8. Each hyperparameter choice has been discussed in Section 6.5.1. While the Stable Baselines 3 library offers customization of additional hyperparameters of the DQN algorithm, these parameters are left at their default values for experiments conducted in the evaluation.

Mitigation environment parameters. Similarly to hyperparameters, all agents use common parameters for the mitigation environment to ensure comparability among experiments. Table 6.9 summarizes all parameters that influence the generation of filter rules in the environment.

Environment parameter	Value
Monitoring interval duration	$\Delta t = 1\,s$
Sketch capacity	1000 entries
Traffic Purity Estimation model	$\mathsf{M}_{\mathsf{TOP}}$
Tracker parameters	$\alpha = 0.45, L_{\text{init}} = 3$

Table 6.9: Parameters of the mitigation environment.

The monitoring interval duration is fixed to one second to balance HHH stability against filter rule generation time. With a reduced monitoring interval duration, HHH algorithms monitor less information, which can result in unstable HHHs and can cause fluctuations in the blacklist. This effect depends on the traffic volume sent by traffic sources during a monitoring interval, since HHHs become more stable when traffic characteristics converge.

The sketch capacity determines the number of entries that are available to monitor the traffic distribution over the IP address space. Each entry stores all features required for Traffic Purity Estimation. The number of counters is restricted to conduct aggregate-based monitoring instead of monitoring each individual traffic source. Consequently, monitoring information provided by the HHH algorithm can incur errors in traffic volume and traffic features.

The mitigation environment performs Traffic Purity Estimation using the model $\mathsf{M}_{\mathsf{TOP}}$ from Section 4.5.4. This leads to seven features being stored in each sketch entry, which constitutes a trade-off between low Traffic Purity Estimation errors and sketch memory requirements.

Filter Rule Refinement parameters are chosen as in Section 5.4.2. The parameter value of $\alpha = 0.45$ balances historical against current monitoring information. The chosen value serves to stabilize filter rules while offering the ability to react to changes in traffic patterns. The tracker lifetime $L_{\text{init}} = 3$ allows it to keep track of historical information when aggregates vanish for short periods of time. Still, it causes the timely removal of trackers to avoid a continuous growth in the number of active trackers.

Experiment	Weights of $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$			Thresholds of σ_{BSIZE}			
	w_{FPR}	w_{FNR}	w_{BSIZE}	ϑ_1	ϑ_2	ϑ_3	ϑ_4
Exp_1	4	0.15	1	1500	1700	1900	2000
Exp_2	4	0.15	1	1300	1500	1700	2000

Table 6.10: Reward function parameters for experiments Exp_1 and Exp_2.

6.6.3 Prioritizing low FPR and low FNR

The first two experiments Exp_1 and Exp_2 train agents to emphasize low FPR and low FNR while allowing high BSIZE. This provides a reference on achievable FPR and FNR for comparison with further experiments that place a higher priority on reduced blacklist size. The intended trade-off is conveyed to an agent by choosing the weights of the reward function $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$ and the thresholds $\vartheta_1, \ldots, \vartheta_4$ of the shaping function σ_{BSIZE}. The specific weights and thresholds are chosen as specified in Table 6.10 for the following reasons:

- w_{FPR}. The value $w_{\text{FPR}} = 4$ is high in comparison to other weights to prioritize the preservation of legitimate traffic over the other mitigation goals.
- w_{FNR}. The value $w_{\text{FNR}} = 0.15$ is on a different order of magnitude compared to w_{FPR}. This takes into account that it may be necessary to accept a fraction of the attack traffic in order to retain low FPR.
- w_{BSIZE}. The weight w_{BSIZE} is normalized to 1 throughout all experiments, since the reward contribution of BSIZE can be directly controlled through σ_{BSIZE}. This is achieved by specifying the thresholds $\vartheta_1, \ldots, \vartheta_4$ along with corresponding rewards $\mathbf{r}_1, \ldots \mathbf{r}_4$ (see Section 6.6.1).

The difference between experiment Exp_1 and Exp_2 resides in the choice of thresholds for σ_{BSIZE}. They are chosen based on the mean and maximum number of active attack traffic sources in phase ③ and ④ of the mitigation scenario (cf. Table 6.5).

- Exp_1. The first experiment uses high thresholds that are consistently above the mean number of active attack traffic sources in each monitoring interval. A slight reduction in reward can still occur when the number of filter rules exceeds the threshold $\vartheta_1 = 1500$. This could be the case when the number of attack traffic sources in phase ③ and ④ is close to the maximum and a filter rule is generated for each source. Modeling reward for BSIZE in this manner

retains some importance of the blacklist size so that features regarding the blacklist size can still influence agent training. Thresholds ϑ_3 and ϑ_4 are chosen at intervals that are consistent with subsequent experiments, although they are never reached in experiment Exp_1.

- Exp_2. The second experiment sets the thresholds $\vartheta_1 = 1300$ and $\vartheta_2 = 1500$ around the mean number of active attack traffic sources in phase ③ and ④. This encourages an agent to reduce the blacklist size below the mean number of attack sources (as σ_{BSIZE} is below the maximum value of 1 when exceeding ϑ_1). In contrast, reward reduction in Exp_1 only occurs when approaching the maximum number of attack sources. When the number of attack sources is close to the maximum in experiment Exp_2, an additional reward reduction occurs. In this case, the value of σ_{BSIZE} declines from $\mathbf{r}_2 = 0.98$ to $\mathbf{r}_3 = 0.8$ between ϑ_2 and ϑ_3. Similarly to Exp_1, threshold ϑ_4 is never reached.

While experiment Exp_1 discourages only excessive blacklist size (beyond 1500), experiment Exp_2 encourages a reduction of BSIZE below $\vartheta_1 = 1300$ and the mean number of active attack sources. Exp_2 furthermore discourages excessive blacklist size in a stronger manner. Still, neither of the two experiments causes a large reward reduction for blacklist sizes around the mean number of active attack sources (as σ_{BSIZE} yields a value above 0.98 in this range).

Mitigation results for experiment Exp_1. The mitigation results presented outline the progression of FPR, FNR, and BSIZE over time and provide summary metrics for different attack phases and the entire scenario duration.

The progression over time for experiment Exp_1 is shown in Figure 6.13. The figure highlights the initial ramp-up (the first 30 seconds) of each attack phase in green and the following stable state phase (of 100 seconds) in gray. During the cool-down phase (i.e., the remaining 95 seconds), the attack traffic declines, but attack traffic can still occur before a new attack is initiated. This is due to the choice of probability distributions (for flow start start times and flow durations) that govern the activity of attack traffic sources. The curves in Figure 6.13 show the mean, minimum, and maximum, as well as 10 % to 90 %-quantiles calculated over ten episodes progress over time.

The false positive rate remains close to zero throughout traffic filtering over all episodes with occasional occurrences of elevated maximum FPR. The FPR reaches a maximum value of 0.1867 in a single episode at time step 357 (at the beginning of the cool-down phase of attack phase ②). The reason for the momentarily increased FPR

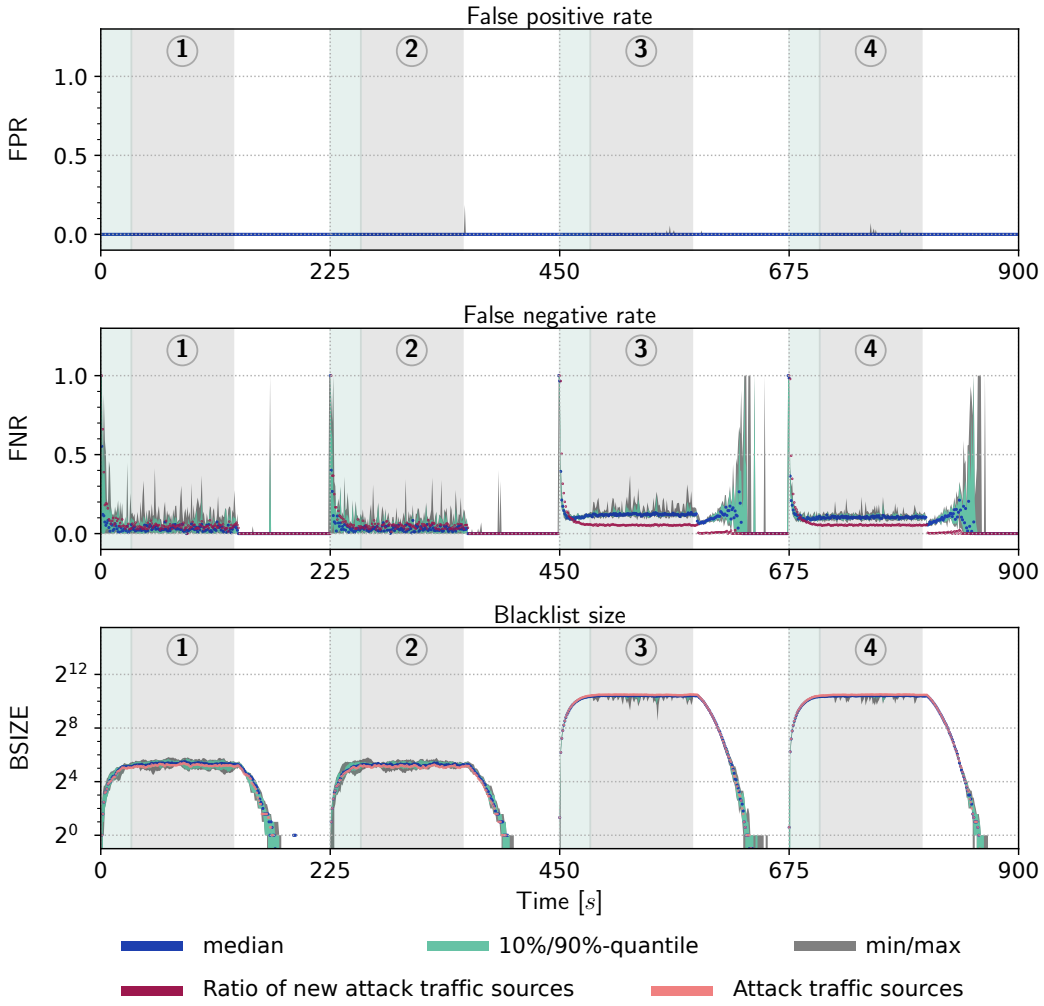


Figure 6.13: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes of experiment Exp_1. Green regions within the four phases ① to ④ indicate the ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

are errors in the estimation of the traffic volume of two IP prefixes of length 32. The errors lead to a reduced (current) traffic discount $D_{\mathsf{p}}^{\mathsf{cur}}$ and an elevated discounted precision $\overline{\pi}_{\mathsf{p}}$ for a shorter prefix p (cf. Equation 5.2). This leads to the short-term inclusion of the shorter prefix in the blacklist and unintentional removal of legitimate traffic. However, the filter rule is immediately withdrawn in the following monitoring interval. Such estimation errors occur due to two design choices that can lead to reduced traffic visibility:

(1) Limiting the number of counters in the sketch data structure can result in errors in traffic volume estimations. Still, using (compact) sketch data structures facilitates maintaining low memory requirements for Aggregate Monitoring.

(2) Applying traffic filtering upstream requires probabilistic traffic sampling to retain insight into traffic composition and distribution. This can also cause errors in traffic volume estimations. However, the upstream placement of a traffic filter enables the immediate removal of attack traffic before it can enter a network infrastructure (and before it needs to be processed by Aggregate Monitoring).

In essence, these design choices constitute a deliberate trade-off between maintaining low memory requirements, protecting the network infrastructure, and accepting a short-term impact on legitimate traffic.

In comparison to FPR, the false negative rate remains considerably higher throughout all episodes. This is due to several reasons:

- Prioritizing low FPR over low FNR by choosing the w_{FPR} considerably higher than w_{FNR} trains an agent to avoid filtering legitimate traffic at the cost of not removing attack traffic.

- Deprioritizing BSIZE and emphasizing low FPR discourages the use of short filter rule prefixes. This results in filtering attack traffic sources individually. Interval-based traffic filtering requires at least one monitoring interval to detect and filter newly occurring attack traffic sources. This causes the initial portion of traffic sent by each traffic source to remain unaffected (unless previously established filter rules apply). Insofar, the ratio of new attack traffic sources appearing in each monitoring interval serves as a reference for achievable FNR.

- Trackers are deliberately configured to weigh historical monitoring information stronger than current information (by choosing $\alpha = 0.45$ when calculating the smoothed discounted precision $\overline{\pi}_{\mathsf{p}}$). This serves to reduce the risk of incurring high FPR but causes an elevated FNR (as prefixes are accepted later into a blacklist). This is a deliberate trade-off to preserve the legitimate traffic.

The FNR is exceptionally high at the beginning of the ramp-up phase and during the cool-down phase. During these periods, only a few attack traffic sources are active, which will result in high FNR if their traffic is not removed. This is because the denominator of Equation 3.4 used to calculate FNR is equally low (see Definition 3.5).

Despite the high FNR values, a volumetric DDoS attack remains ineffective during these periods, as the total attack traffic volume is low.

During phase ① and ②, the median FNR remains close to this ratio of new attack traffic sources, while it is consistently higher during phase ③ and ④. This is because the absolute number of attack traffic sources increases during the later phases, so more filter rules become outdated and a larger number of new filter rules need to be created. The greater need for adaptation makes it more challenging to achieve low FNR. In addition, experiment Exp_1 places a high priority on low FPR, so that RL-based Control must rely primarily on filter rules with long prefixes. Otherwise, short prefixes would increase the risk of inadvertently removing legitimate traffic from unrelated address space regions. Consequently, fine-granular adaptation of the blacklist is necessary to realize the intended trade-off.

Overall, the median FPR and FNR remain low regardless of changes in the attack traffic source distribution and traffic composition over time. This indicates that blacklist generation adapts to changing attack and legitimate traffic patterns.

Experiment Exp_1 is designed explicitly deprioritize BSIZE. As a result, the blacklist size is not reduced in comparison to the total number of attack traffic sources. This is indicated by the two overlapping curves for median BSIZE and the number of attack traffic sources at the bottom of Figure 6.13. Also, the minimum and maximum follow the progression of the number of attack traffic sources closely during all four phases. This shows that traffic filtering under RL-based Control primarily seeks to generate filter rules for individual traffic sources and disregards the blacklist size as intended.

Table 6.11 summarizes the median, mean, and maximum FPR, FNR, and BSIZE for the stable states of all four attack phases in experiment Exp_1. It also provides metrics for the entire duration of traffic filtering, including the ramp-up and cool-down phases. Overall, the high threshold $\vartheta_1 = 1500$ allows RL-based Control to maintain a low mean FPR of 0.0001 over the entire scenario duration (by deprioritizing BSIZE). In 90 % of all monitoring intervals no legitimate traffic is discarded, as indicated by the zero value of the 90 %-quantile in the last column of the table.

During the stable states of phase ① no legitimate traffic is discarded. In phase ②, at most 0.55 % of legitimate traffic is removed during the stable state. In both phases, the legitimate traffic remains mostly unaffected by traffic filtering as indicated by the zero values of the median, mean, and 90 %-quantile of the FPR.

Metric		Phase				Entire scenario duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0000	0.0000	0.0000
	mean	0.0000	0.0000	0.0003	0.0004	0.0001
	90 %-quantile	0.0000	0.0000	0.0002	0.0002	0.0000
	maximum	0.0000	0.0054	0.0557	0.0722	0.1867
FNR	median	0.0295	0.0290	0.1203	0.1027	0.0383
	mean	0.0425	0.0406	0.1262	0.1050	0.0700
	90 %-quantile	0.0965	0.0943	0.1479	0.1254	0.1314
	maximum	0.4130	0.3530	0.3684	0.2334	1.0000
BSIZE	median	40.0	38.0	1359.5	1357.0	38.5
	mean	40.0	38.5	1337.9	1341.8	418.6
	90 %-quantile	48.0	45.1	1402.0	1396.0	1365.0
	maximum	57.0	56.0	1443.0	1464.0	1464.0

Table 6.11: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE in experiment Exp_1. Metrics outline the stable states of the four phases of the mitigation scenario as well as the entire scenario duration (calculated over ten episodes).

During phase ③ and ④, the mean FPR and the 90 %-quantile reach values of 0.0003 and 0.0002. This is because the attack traffic is distributed across a larger number of attack traffic sources. While attack traffic sources send high-volume traffic during phase ① and ②, aggregates corresponding to individual attack traffic sources now have a considerably lower traffic volume. This makes (probabilistic) traffic volume estimation with sketches and Traffic Purity Estimation more challenging. It especially affects the short filter rule prefixes (that aggregate less traffic), which are required to maintain low FPR. As a result, legitimate traffic is more frequently removed, which is evident in the value of the 90 %-quantiles of 0.0002. Still, maximum incurred FPR remains at 0.0557 and 0.0722 so that traffic filtering has low impact on the legitimate traffic during any one monitoring interval.

Similarly to the difference in FPR between the first and last two attack phases, the median, mean, and 90 %-quantile values of the FNR are consistently higher during phase ③ and ④. Again, this is the result of the attack traffic distributing across more traffic sources. During the later attack phases, this results in a delay of one

monitoring interval, which is required to detect and filter traffic from a new source to occur more often. In addition, the absolute number of attack sources with short flow duration increases, as the attack flow duration in all phases is drawn from the same Weibull distribution. With a higher number of short-lived attack flows, blacklist adaptations become more challenging.

The 90 %-quantiles listed in Table 6.11 indicate that over 90 % and 85 % of the attack traffic is reliably discarded during the first and last two attack phases. The median and mean FNR values further indicate that blacklists typically remove additional attack traffic before it can enter a network infrastructure. The maximum FNR over the entire scenario reaches one as it is not possible to immediately counteract the initial attack traffic sources appearing at the beginning of an attack (due to the delay in filter rule generation). The high maximum FNR values that occur during the stable states of the four attack phases are limited to short durations (cf. Figure 6.13).

The metrics for BSIZE in Table 6.11 outline a considerable difference between attack phases with high and low numbers of attack traffic sources. While the maximum number of filter rules is bounded by values of 57 and 56 during phase ① and ②, BSIZE reaches values up to 1443 and 1464 during the later phases. This is a result of the modeling of the mitigation scenario and prioritizing low FPR by choosing reward function parameters accordingly. RL-based Control relies on long prefixes during filter rule generation to filter individual traffic sources and to reduce the impact on legitimate traffic.

The median, mean, 90 %-quantile, and maximum of the blacklist size exceed the number of attack sources that are active in a monitoring interval in phase ① and ② (listed in Table 6.5). This is because some trackers can still yield a high smoothed discounted precision $\overline{\pi}_p$ for a short period when attack traffic already vanished. When a lower number of attack traffic sources are active, this can cause a higher BSIZE. This is also the case for the mean and the 90 %-quantile in phase ③ and ④. In contrast, the metrics for blacklist size remain below number of active attack traffic sources during phase ③ and ④. This shows that a portion of the attack traffic sources remain unfiltered when their number (temporarily) approaches its maximum. In particular, BSIZE also remains well below the threshold $\vartheta_1 = 1500$. Due to this, the blacklist generation would not reduce the reward an agent receives, and agent training can focus on optimizing for low FNR and low FPR.

Mitigation results for experiment Exp_2. The second experiment has a higher potential to reduce the reward for large blacklists. By choosing the first threshold $\vartheta_1 = 1300$ the reward starts to decline before BSIZE reaches the median, mean, and 90 %-quantile

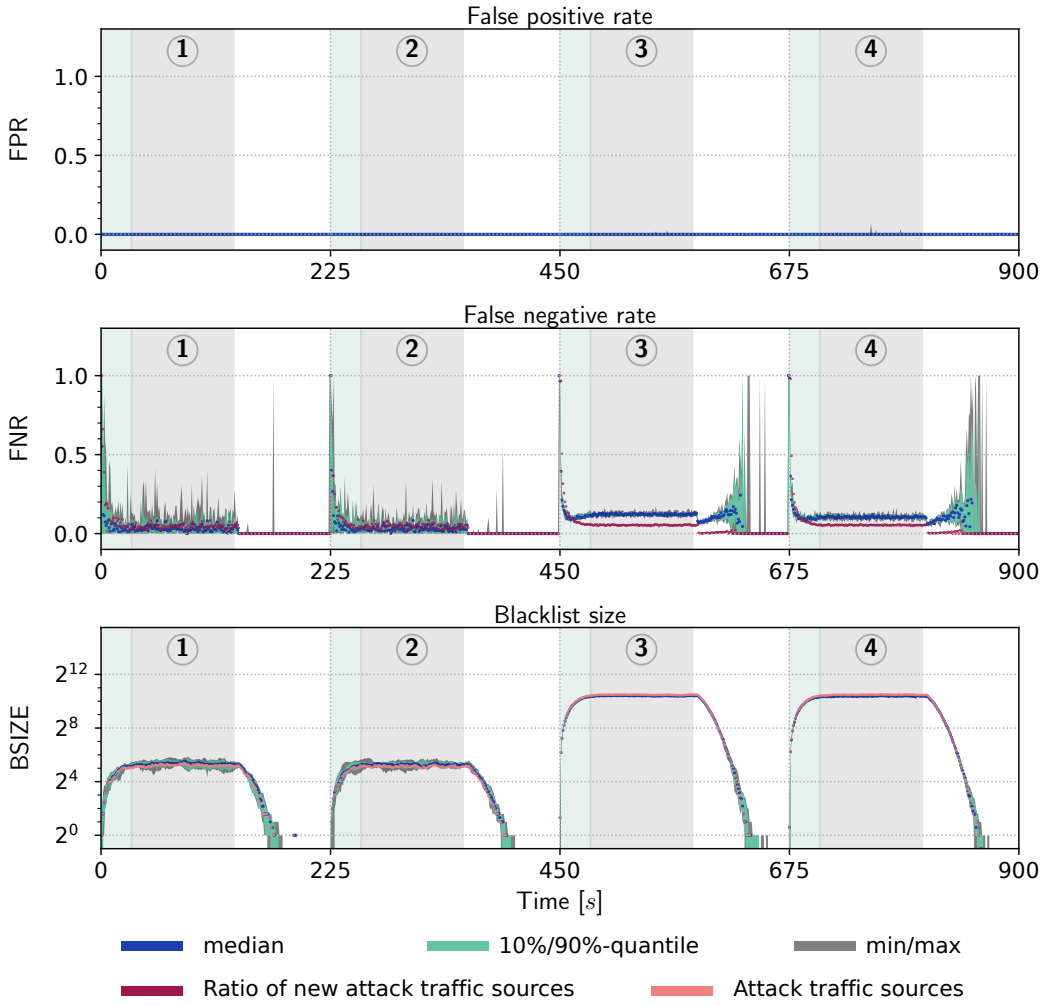


Figure 6.14: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes of experiment Exp_2. Green regions within the four phases ① to ④ indicate the ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

of active attack traffic sources listed in Table 6.5. In addition, the maximum number of attack traffic sources can exceed the threshold $\vartheta_2 = 1500$. At this point, the reward for corresponding BSIZE declines more rapidly. This could not occur in the previous experiment.

Figure 6.14 visualizes the progression of FPR, FNR, and BSIZE (for ten episodes) over time for experiment Exp_2. Notably, the short-term increase in FPR during phase ② of the previous experiment does not occur again (cf. Figure 6.13). In fact, legitimate

Metric		Phase				Entire scenario duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0000	0.0000	0.0000
	mean	0.0000	0.0000	0.0002	0.0002	0.0001
	90 %-quantile	0.0000	0.0000	0.0001	0.0001	0.0000
	maximum	0.0000	0.0018	0.0263	0.0721	0.0721
FNR	median	0.0296	0.0291	0.1217	0.1046	0.0390
	mean	0.0436	0.0409	0.1223	0.1050	0.0693
	90 %-quantile	0.0986	0.0950	0.1378	0.1237	0.1305
	maximum	0.4130	0.3530	0.1716	0.1618	1.0000
BSIZE	median	40.0	38.0	1362.0	1339.0	38.5
	mean	40.0	38.5	1361.5	1338.3	420.3
	90 %-quantile	48.0	45.0	1402.0	1390.0	1360.0
	maximum	58.0	55.0	1460.0	1444.0	1460.0

Table 6.12: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE in experiment Exp_2. Metrics outline the stable states of the four phases of the mitigation scenario as well as the entire scenario duration (calculated over ten episodes).

traffic remains largely unaffected by traffic filtering throughout the duration of the scenario, despite variations in the attack and legitimate traffic.

The progression of FNR and BSIZE are close to that in experiment Exp_1. In fact, increases in FNR occur approximately at the same time during the cool-down phase of each attack. This is a result of using identical seeds when generating randomized traffic episodes in the evaluation environment. During phase ③ and ④, high short-term increases in FNR do no longer occur (compared to Exp_1) as the FNR remains close to the median value. This is quantified in the summary results below.

Table 6.12 lists the metrics for experiment Exp_2. As outlined above, the values for median, mean, and 90 %-quantiles of the FPR are at or close to zero. The maximum FPR over the entire scenario duration remains at 0.0721, as the previous error in traffic volume estimation does not occur or has no impact on filter rule generation. The maximum FPR of the stable state in all phases remains on the same order of magnitude as in experiment Exp_1. The slightly reduced FPR in phases ② and ③ can be attributed to different random choices made during training (such as exploration

and weight initialization of neural networks) and during traffic processing (such as probabilistic sketch updates and traffic sampling).

Similarly to the FPR, results on FNR are consistent with the previous experiment, except for the maximum FNR in phase ③ and ④. The maximum FNR is reduced by 19.68 and 7.16 percentage points compared to Exp_1. The cause of the FNR reduction is elaborated in the discussion of parameter adaptations below.

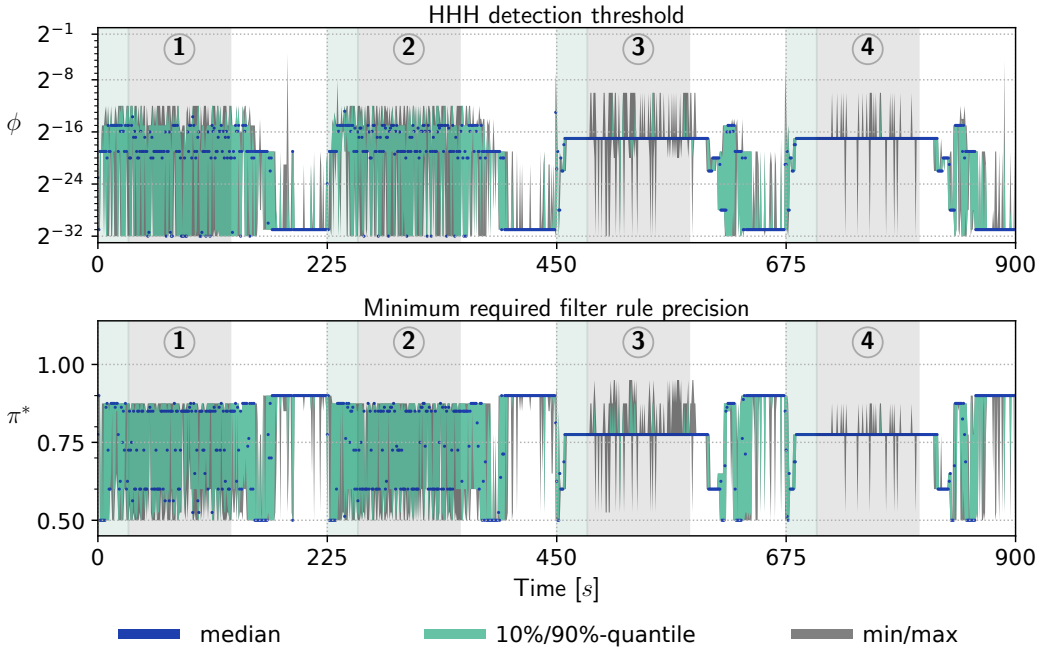
Despite choosing a lower threshold $\vartheta_1 = 1300$ no significant changes in the blacklist size occur. In fact, in phase ③ the mean, 90 % quantile, and maximum values of BSIZE are higher than in experiment Exp_1. This also holds for maximum BSIZE in phase ①. In effect, the agent learns to place a higher emphasis on low FPR and low FNR higher than seeking to avoid minor reward reductions due to higher blacklist sizes.

Parameter adaptations in experiment Exp_1 and Exp_2. Figure 6.15 compares the parameter choices of trained agents in experiment Exp_1 and Exp_2. The figure includes the median, 90 %-quantiles, and maximum values of the HHH detection threshold ϕ and the minimum required filter rule precision π^*.

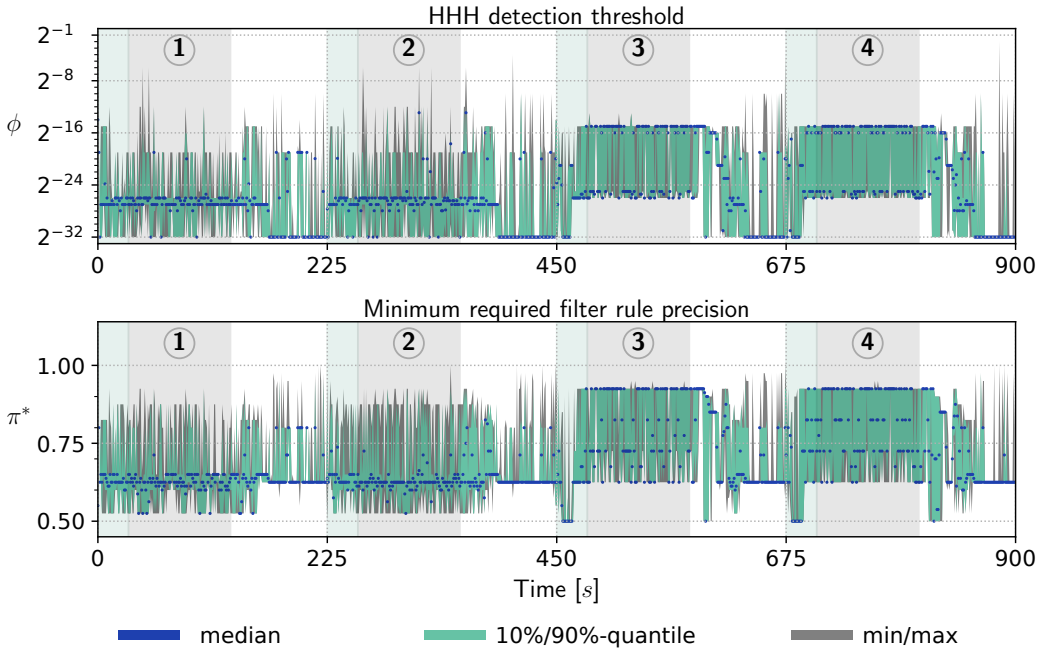
The distribution of chosen parameter combinations deviates significantly between the two experiments. The median values of π^* and ϕ in experiment Exp_1 distribute over larger ranges during phase ① and ② compared to experiment Exp_2. During phase ③ and ④ this relationship is reversed. This is because agents have a high degree of freedom in choosing parameters when the blacklist size can be disregarded. Under this condition, an agent has different options to reduce FPR and FNR:

(1) An agent can choose a low HHH detection threshold ϕ that results in the generation of HHH prefixes for individual traffic sources (of legitimate and attack traffic). In this case, the aggregate of a prefix p consists purely of legitimate or attack traffic with high probability. Due to this, the (smoothed) discounted precision $\overline{\pi}_\mathsf{p}$ is either close to zero or it reaches high values. To achieve a low FPR it is sufficient to apply a minimum required filter rule precision π^* that is higher than the (low) $\overline{\pi}_\mathsf{p}$-value of legitimate traffic aggregates. This allows π^* to be chosen from a large value range. It also yields low FNR as long as π^* is not chosen high enough to cause the unnecessary rejection of prefixes.

(2) An agent can choose a high minimum required filter rule precision π^*. This serves to reduce FPR, but the FNR depends on the choice of ϕ. Choosing a high ϕ value causes strong aggregation and can result in the mixture of attack and legitimate traffic. This creates the risk that prefixes covering large address



(a) Experiment Exp_1.



(b) Experiment Exp_2.

Figure 6.15: Progression of parameters ϕ and π^* in experiment Exp_1 and Exp_2. The median, 10 %-90 %-quantiles, and maximum are calculated from ten episodes.

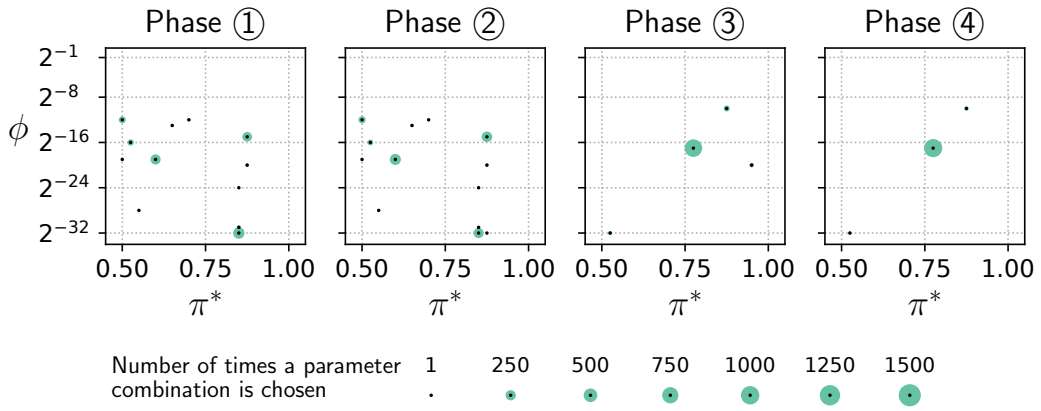
space regions will be rejected due to having low $\overline{\pi}_{\mathrm{p}}$, leading to an increase in FNR. Consequently, this imposes an upper limit on ϕ that depends on the traffic pattern. However, an agent can still choose lower HHH detection thresholds when blacklist size is of no concern.

Figure 6.15 illustrates that the trained agents of experiment Exp_1 and Exp_2 use different options to maintain low FPR and low FNR. This is because agent training is affected by multiple sources of randomness, such as exploration, random traffic sampling, and probabilistic updates of sketch data structures. Still, both agents achieve comparable results that realize the intended trade-offs.

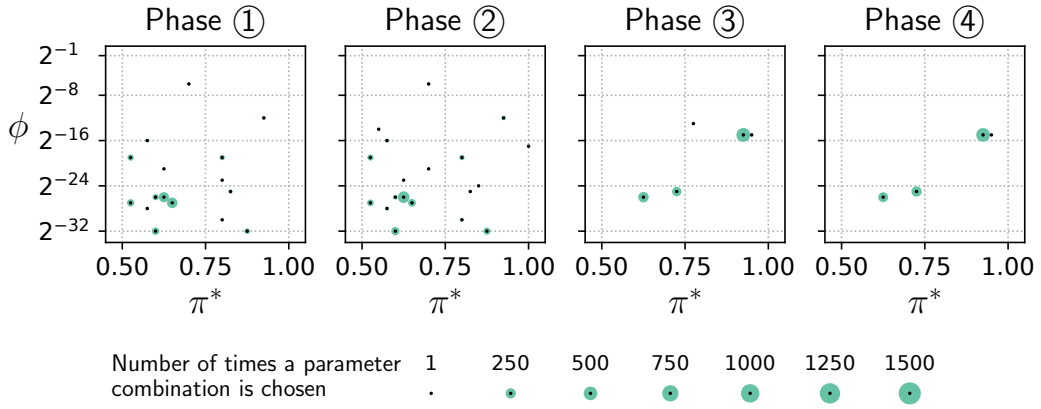
The progression of parameter choices shows that both agents differentiate between different traffic patterns. Parameter choices vary noticeably when transitioning from a stable state to a cool-down phase. The agent of experiment Exp_1 gradually adapts parameters to reduce ϕ while raising π^* consistently while progressing through cool-down phases. As a result, filter rules will only be generated if they have long prefixes and high $\overline{\pi}_{\mathrm{p}}$. This serves to reject prefixes corresponding to aggregates composed of the legitimate traffic, which is still being sent during the cool-down phase. The agent of experiment Exp_2 also adjusts parameters when progressing through the cool-down phases, but chooses different parameter values. Still, the result remains essentially the same, as the legitimate traffic is preserved.

In addition to adapting to cool-down phases of an attack, agents react differently to the stable states of phase ① and ② in comparison to phase ③ and ④. Figure 6.16 visualizes how often each specific parameter choice occurs during these periods. The radius of a green circle around the center point marking each parameter combination is proportional to the number of times each combination is chosen. During phase ① and ② the choices of the agent in experiment Exp_1 alternate between different options to choose parameters. A single parameter combination occurs at most 331 and 306 times out of the 1000 monitoring intervals that comprise the stable states of the two phases in ten episodes. The choices are the result of agent training and the reaction to variations in the monitored traffic. In effect, all parameter combinations chosen during the first two phases achieve the intended goal of maintaining low FPR and FNR. During phase ③ and ④, the agent focuses on a single parameter combination that occurs 901 and 972 times. The chosen parameter combination is $\phi = 2^{-17}$ and $\pi^* = 0.775$. This corresponds to the option of choosing a high π^* value while keeping ϕ sufficiently low to avoid the unnecessary rejection of prefixes.

In experiment Exp_2, the agent focuses more on lower π^* and ϕ values during phase ① and ②. While the parameter choices differ from experiment Exp_1, the FPR and FNR



(a) Experiment Exp_1.



(b) Experiment Exp_2.

Figure 6.16: Distribution of parameters ϕ and π^* in experiment Exp_1 and Exp_2 in the stable states of phases ① to ④. The radius of green-colored circles indicates how often the parameter combination at the center point is chosen.

remain comparable. Despite occasionally choosing a high ϕ value of 2^{-6} the FPR remains at zero. This can be caused by reacting to a momentarily low total traffic volume, so that high-volume attack traffic still generates long filter rule prefixes. During phase ③ and ④, the agent chooses the most frequently occurring parameter combination at most 524 and 512 times while also selecting two additional combinations more than 200 times. The different choices lead to the previously described reduction of the maximum FNR. By using lower ϕ values (of 2^{-25} and 2^{-26}) and lower π^* values (of 0.725 and 0.625) more often, longer HHH prefixes are generated and more readily accepted in the blacklist. Consequently, agent training in

experiment Exp_2 can be considered to be more effective, as the agent adapts better to traffic changes and maintains low FNR more reliably.

Insights. The results of experiment Exp_1 and Exp_2 show that RL-based Control can maintain low FPR and FNR close to the ratio of new attack traffic sources in a monitoring interval. This despite changes in the legitimate and attack traffic that can occur frequently in the dynamic mitigation scenario. Trained agents realize the trade-offs modelled through the reward function parameters as intended. Insofar, the mitigation goals can be conveyed during agent training. To achieve this, it is necessary for the agents to learn effective parameter combinations of ϕ and π^*. Otherwise, either false positive rates and false negative rates can be expected to be higher, when choosing parameters statically or more randomly.

6.6.4 Reducing Blacklist Size

The following three experiments are designed to train an agent to place a higher priority on lower blacklist size. This generally results in using shorter filter rule prefixes more frequently to reduce BSIZE. Specifically, reward function parameters are adjusted in the following way:

- Exp_3 **and** Exp_4. These experiments decrease the threshold ϑ_1 to 1000 and 900. Other thresholds are reduced proportionally (at intervals of 200). Through this, an agent receives a lower reward for high BSIZE compared to experiment Exp_1 and Exp_2.
- Exp_5. This experiment decreases ϑ_1 to 800 and additionally reduces w_{FNR} to 0.1. With shorter prefixes, the risk of incurring higher FPR increases, as shorter prefixes cover larger address space regions from which legitimate traffic can originate. Reducing w_{FNR} enables an agent to realize a trade-off between FPR and FNR in favor of FPR. For example, an agent can choose a higher value for π^* to reject more prefixes with lower discounted precision.

Table 6.13 summarizes the resulting reward function parameters.

Mitigation results. Figure 6.17 shows the progression of FPR, FNR, and BSIZE over time for experiment Exp_3. This experiment uses the threshold $\vartheta_1 = 1000$ (and higher thresholds at intervals of 200). In comparison to the previous experiments, the FPR during phase ③ and ④ is increased for individual, non-consecutive monitoring

Experiment	Weights of $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$			Thresholds of σ_{BSIZE}			
	w_{FPR}	w_{FNR}	w_{BSIZE}	ϑ_1	ϑ_2	ϑ_3	ϑ_4
Exp_3	4	0.15	1	1000	1200	1400	2000
Exp_4	4	0.15	1	900	1100	1300	2000
Exp_5	4	0.10	1	800	1000	1200	2000

Table 6.13: Reward function parameters for experiments Exp_3, Exp_4, and Exp_5.

intervals.[3] This is a consequence of using shorter prefixes to reduce BSIZE while also reducing FNR. The shorter prefixes enable attack traffic removal in a larger address space region but are more likely to impact legitimate traffic than short prefixes that affect individual traffic sources only. The corresponding reduction in blacklist size is reflected in the median value of BSIZE falling below the number of attack traffic sources. Using shorter prefixes also affects FNR, which is subject to frequent fluctuations due to the following effects:

- Shorter prefixes correspond to larger address space regions, which increases the probability that an already established filter rule affects newly emerging attack traffic sources. Consequently, it no longer requires one initial monitoring interval to detect and remove new attack traffic, which lowers the overall FNR.
- Attack traffic sources may remain unaffected by filtering when using a limited number of filter rules. When attack traffic is concentrated on a small region of the address space, few filter rules may be sufficient to discard the attack traffic. When the attack traffic is more widely distributed, it becomes more challenging to generate short filter rules with low impact on legitimate traffic. In this case, filter rules are more likely to be rejected in order to achieve low FPR. As a result, variations in the attack traffic source distribution can result in short-term increases and reductions of the FNR.

The second effect also accounts for the lower FNR in phase ④ in comparison to phase ③. Phase ④ uses two attack vectors, whereas phase ③ uses only a single one (see Table 5.1). The variance of the attack source address distribution is independently randomized for each attack vector. With two different attack vectors it is more likely that at least one vector has a low variance and more frequently concentrates attack

[3] Figure 6.17 shows results for multiple episodes where the aggregated FPR may be above zero for several consecutive monitoring intervals. In a single episode, a (non-aggregated) FPR above zero occurs only for individual, non-consecutive monitoring intervals.

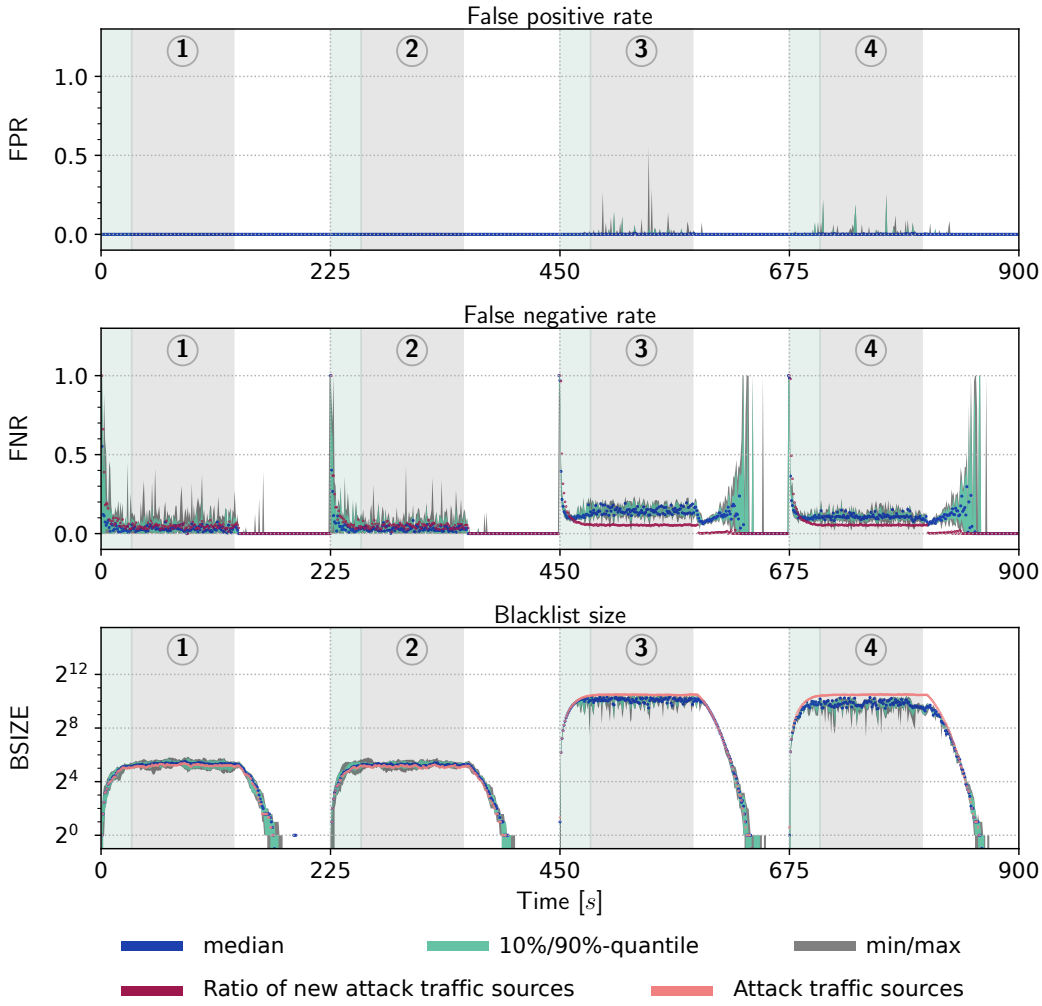


Figure 6.17: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes of experiment Exp_3. Green regions within the four phases ① to ④ indicate the ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

traffic sources on smaller address space regions. Due to this, the attack traffic can be removed more consistently with fewer, shorter prefixes so that FNR is lower during phase ④ than during phase ③.

Other characteristics of the progression of metrics over time are consistent with experiment Exp_1 and Exp_2. In particular, the FPR, FNR, and BSIZE during phase ① and ② remain unaffected. This is because the number of attack traffic sources stays

Metric		Phase				Entire scenario duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0002	0.0002	0.0000
	mean	0.0000	0.0000	0.0044	0.0049	0.0012
	90 %-quantile	0.0000	0.0000	0.0052	0.0064	0.0005
	maximum	0.0000	0.0000	0.5534	0.2573	0.5534
FNR	median	0.0296	0.0284	0.1397	0.1065	0.0380
	mean	0.0423	0.0403	0.1453	0.1092	0.0736
	90 %-quantile	0.0967	0.0945	0.1943	0.1523	0.1596
	maximum	0.4130	0.4241	0.2372	0.2239	1.0000
BSIZE	median	40.0	38.0	1118.0	896.5	38.5
	mean	40.0	38.5	1086.7	896.0	320.1
	90 %-quantile	48.0	45.0	1306.1	1197.0	1082.0
	maximum	58.0	56.0	1430.0	1346.0	1430.0

Table 6.14: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE in experiment Exp_3. Metrics outline the stable states of the four phases of the mitigation scenario as well as the entire scenario duration (calculated over ten episodes).

far below the threshold $\vartheta_1 = 1000$. This allows attack traffic from individual sources to be discarded without having to balance BSIZE against FPR or FNR.

The results for stable states in the experiment are quantified in Table 6.14. The low number of attack traffic sources in the first two phases results in an FPR of zero and FNR on the same order of magnitude as in experiment Exp_1 and Exp_2. The most notable differences occur in the higher maximum FPR and the consistently reduced metrics for BSIZE in phase ③ and ④. While the FPR is at most 0.0052 and 0.0064 at least 90 % of the time, in phase ③ a maximum value of 0.5534 is reached once in ten episodes. It occurs at time index of 537 seconds and is the result of a probabilistic update of a sketch data structure in the previous monitoring interval. The probabilistic update attributes attack traffic to a wrong prefix p (of length 32), resulting in a high (smoothed) discounted precision $\overline{\pi}_{\mathsf{p}}$. During the monitoring interval starting at 537 seconds, a legitimate traffic source sends high-volume traffic from that prefix, which is unintentionally removed. The sudden, sharp increase in legitimate traffic is causing the momentarily high FPR value.

During other monitoring intervals, the acceptance of shorter prefixes in the blacklist results in the removal of traffic from larger address space regions. If low-volume legitimate traffic originates from the corresponding regions, its removal may be necessary to balance FPR against FNR and BSIZE. High-volume legitimate that persists over multiple monitoring intervals leads to the rejection of prefixes. Still, if the legitimate traffic volume increases suddenly and significantly, a momentary high FPR can occur. The prefixes are then withdrawn in the following monitoring interval to maintain a low FPR. This leads to an elevated maximum FPR, but consistently low values for the median, the mean, and the 90 %-quantile. Essentially, the short-term rise in FPR is the result of the intended trade-off that is conveyed by the reward function parameters.

The blacklist size in phase ③ and ④ is significantly reduced in experiment Exp_3. For example, the median value of BSIZE is reduced by an additional 31.5 and 33.0 percentage points during phases ③ and ④ in comparison to experiment Exp_2. In general, a stronger reduction occurs in phase ④, where the maximum blacklist size remains below the mean and median number of active attack traffic sources per monitoring interval. This is due to attack traffic sources being concentrated on smaller address space regions more frequently, which enables more effective filtering with fewer filter rules. In combination, the results for phase③ and ④ show that adjusting the thresholds of σ_{BSIZE} affects agent training and offers control over the size of generated blacklists. Note that the steepest decline in reward caused by σ_{BSIZE} begins at the threshold $\vartheta_3 = 1400$, which is only exceeded when BSIZE approaches its maximum during phase ③. Otherwise, BSIZE remains within regions where the reward declines less rapidly. In particular, the median and mean values are below $\vartheta_2 = 1200$ so that the value of σ_{BSIZE} is typically reduced by less than 2 % (due to choosing $\mathbf{r}_2 = 0.98$ for ϑ_2).

The progression of metrics in experiment Exp_4 is visualized in Figure 6.18. The threshold ϑ_1 is configured to a value of 900 in this experiment to further reduce the blacklist. The FPR rises again in phase ③ and ④ compared to Exp_1 and Exp_2, but the large increase to a value of 0.5534 that occurs in experiment Exp_3 can be avoided. In addition, the FNR increases. This is the effect of an agent focusing increasingly on shorter prefixes to account for reduced reward from high BSIZE. As a result, the effect of FNR showing larger variations in experiment Exp_3 is further emphasized in experiment Exp_4.

The trade-off between FPR, FNR, and BSIZE is also evident in the summarized results in Table 6.15. Compared to experiment Exp_3, the median, mean, and the 90 %-quantile

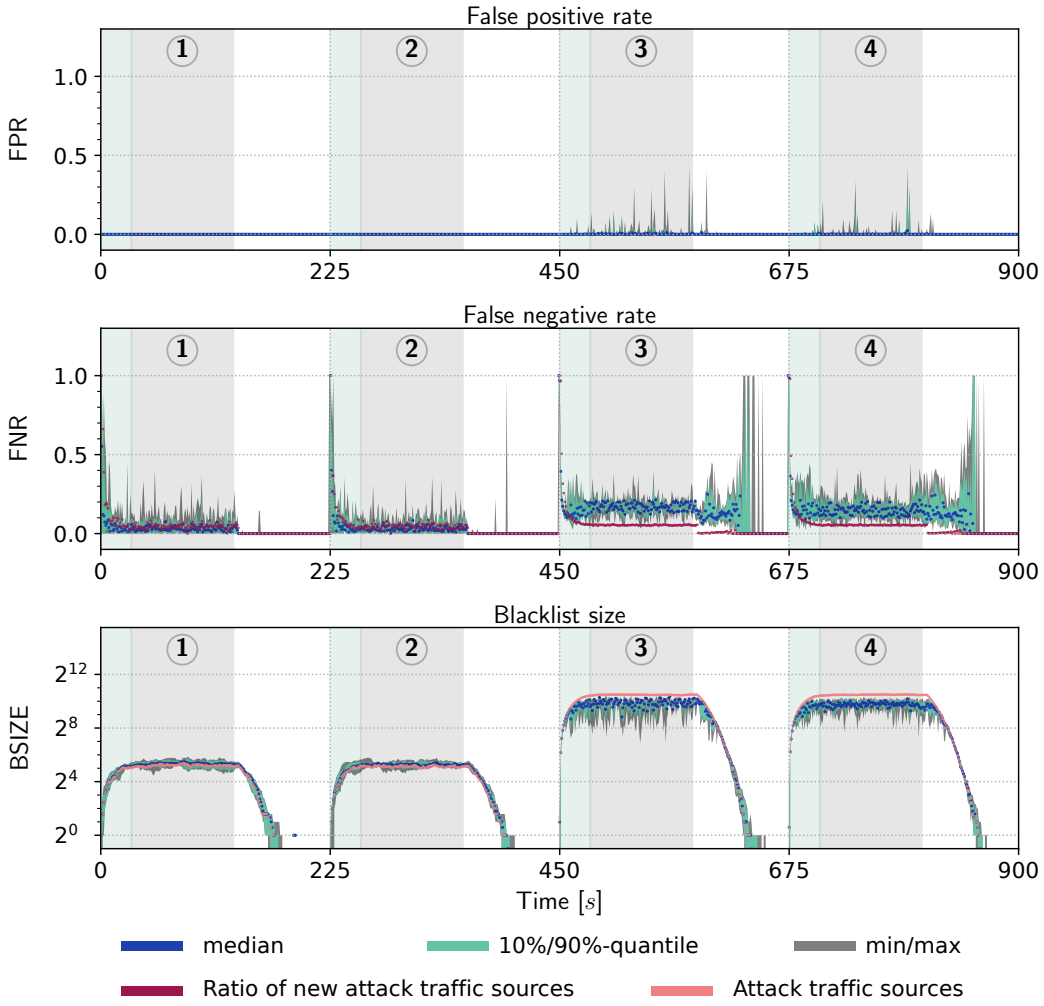


Figure 6.18: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes of experiment Exp_4. Green regions within the four phases ① to ④ indicate the ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

of FNR increase between 2.3 and 3.3 percentage points during phase ③. In phase ④ they increase by 3.4 to 7.1 percentage points. In addition, the FNR reaches higher maximum values of 0.3984 and 0.4548 compared to 0.2372 and 0.2239 in phase ③ and ④ of experiment Exp_3. In contrast, all values of BSIZE decrease. For example, the median values are reduced by an additional 15.4 and 2.8 percentage points.

Furthermore, reducing blacklist size also results in temporarily higher FPR. In phase ③, the 90 %-quantile increases by 0.0062 compared to experiment Exp_3

Metric		Phase				Entire scenario duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0005	0.0002	0.0000
	mean	0.0000	0.0000	0.0063	0.0053	0.0016
	90 %-quantile	0.0000	0.0000	0.0114	0.0058	0.0008
	maximum	0.0000	0.0000	0.4393	0.4310	0.4393
FNR	median	0.0298	0.0285	0.1718	0.1407	0.0380
	mean	0.0420	0.0403	0.1690	0.1500	0.0856
	90 %-quantile	0.0947	0.0943	0.2209	0.2228	0.2021
	maximum	0.4130	0.3591	0.3984	0.4548	1.0000
BSIZE	median	40.0	38.0	945.5	871.5	38.5
	mean	40.0	38.5	896.5	865.4	288.4
	90 %-quantile	48.0	45.0	1190.1	1125.0	984.0
	maximum	58.0	55.0	1375.0	1291.0	1375.0

Table 6.15: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE in experiment Exp_4. Metrics outline the stable states of the four phases of the mitigation scenario as well as the entire scenario duration (calculated over ten episodes).

and reaches a value beyond 1 %. The maximum FPR increases to 0.4393 and 0.4310. The fact that the median and mean values remain on the same order of magnitude as before indicates that the trade-off between FPR and BSIZE primarily results in short-term increases of the FPR. Again, this is the result of generating shorter filter rule prefixes. If a single source suddenly starts sending high-volume legitimate traffic, it can cause momentarily high FPR before a filter rule can be withdrawn. Under the constraints imposed by the reward function parameters, this effect cannot be fully compensated by Filter Rule Refinement or RL-based Control due to a trade-off between BSIZE and the other mitigation goals being necessary.

Experiment Exp_5 further reduces the threshold ϑ_1 to a value of 800 while lowering w_{FNR} to 0.1. Consequently, the priority of low BSIZE increases while the priority of low FNR decreases. The resulting FPR, FNR, and BSIZE are shown in Figure 6.19 and quantified in Table 6.16. The FPR during phase ① and ② does not remain at zero as in the two previous experiments. During phase ② a momentary increase to an FPR of 0.0707 occurs, which is the result of an underestimation of traffic volume by the

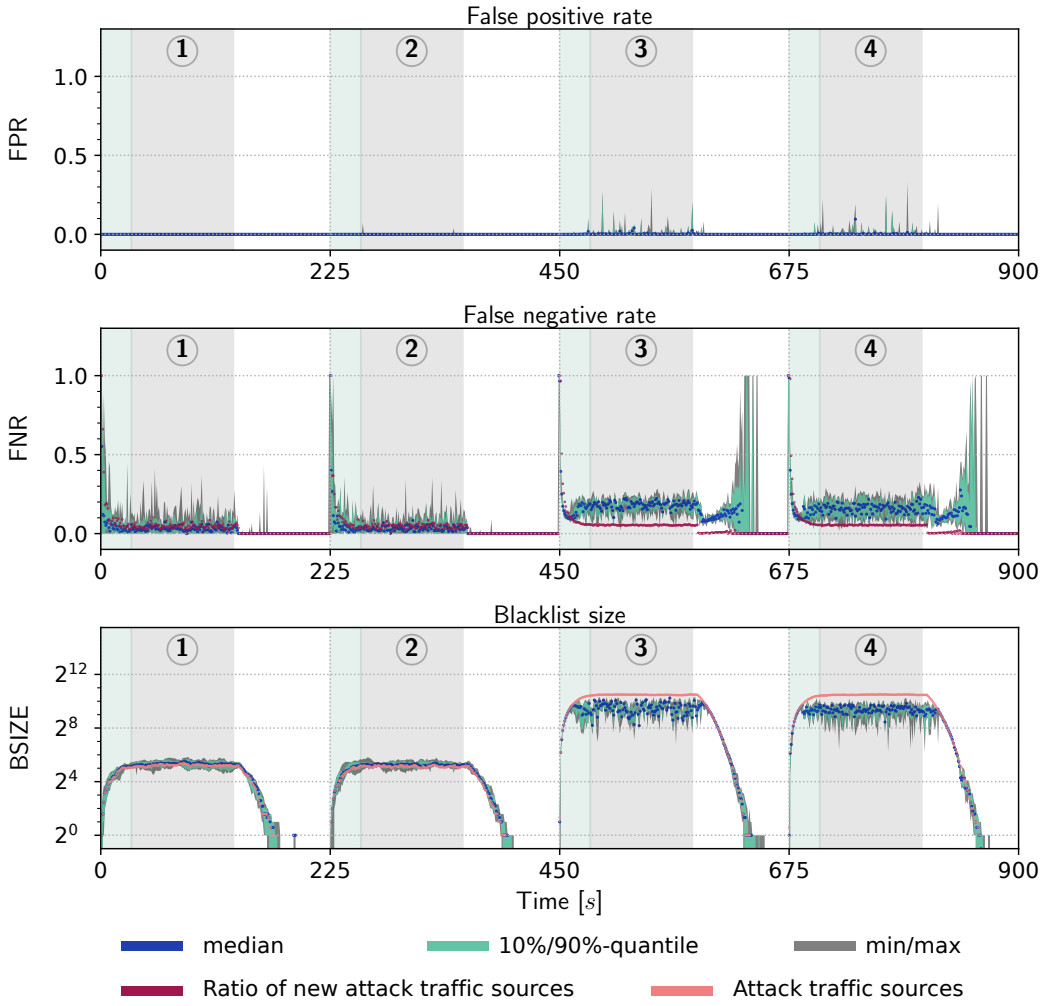


Figure 6.19: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes of experiment Exp_5. Green regions within the four phases ① to ④ indicate the ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

HHH algorithm. This error causes an overestimation of the (smoothed) discounted precision $\overline{\pi}_{\mathsf{p}}$ of a short prefix p. The resulting filter rule affects legitimate traffic during one monitoring interval, but it is subsequently withdrawn.

The 90 %-quantile reaches or exceeds a value of 1 % in phase ③ and ④, whereas this occurred only in phase ③ in the previous experiment. The general trend that the 90 %-quantile of the FPR increases throughout the experiments indicates that it becomes more and more challenging to reliably preserve legitimate traffic as the

Metric		Phase				Entire scenario duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0007	0.0005	0.0000
	mean	0.0000	0.0001	0.0059	0.0064	0.0017
	90 %-quantile	0.0000	0.0000	0.0115	0.0100	0.0012
	maximum	0.0070	0.0707	0.2917	0.3319	0.3319
FNR	median	0.0307	0.0295	0.1816	0.1600	0.0413
	mean	0.0438	0.0417	0.1775	0.1578	0.0865
	90 %-quantile	0.0990	0.0960	0.2297	0.2314	0.2047
	maximum	0.4130	0.3530	0.2859	0.3292	1.0000
BSIZE	median	40.0	38.0	712.5	673.0	38.5
	mean	39.8	38.3	724.0	660.9	240.9
	90 %-quantile	48.0	45.0	1076.3	908.0	797.0
	maximum	58.0	56.0	1318.0	1226.0	1319.0

Table 6.16: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE in experiment Exp_5. Metrics describe the stable states of the four phases of the mitigation scenario as well as the entire scenario duration (calculated over ten episodes).

priority of low BSIZE increases. In contrast, to the 90 %-quantile the maximum FPR during phase ③ and ④ decreases again to 0.2917 and 0.3319. This outlines that short-term variations in traffic patterns are challenging to control when relying on small blacklists. Still, around 99 % of the legitimate traffic remain unaffected by traffic filtering with a probability of at least 90 %.

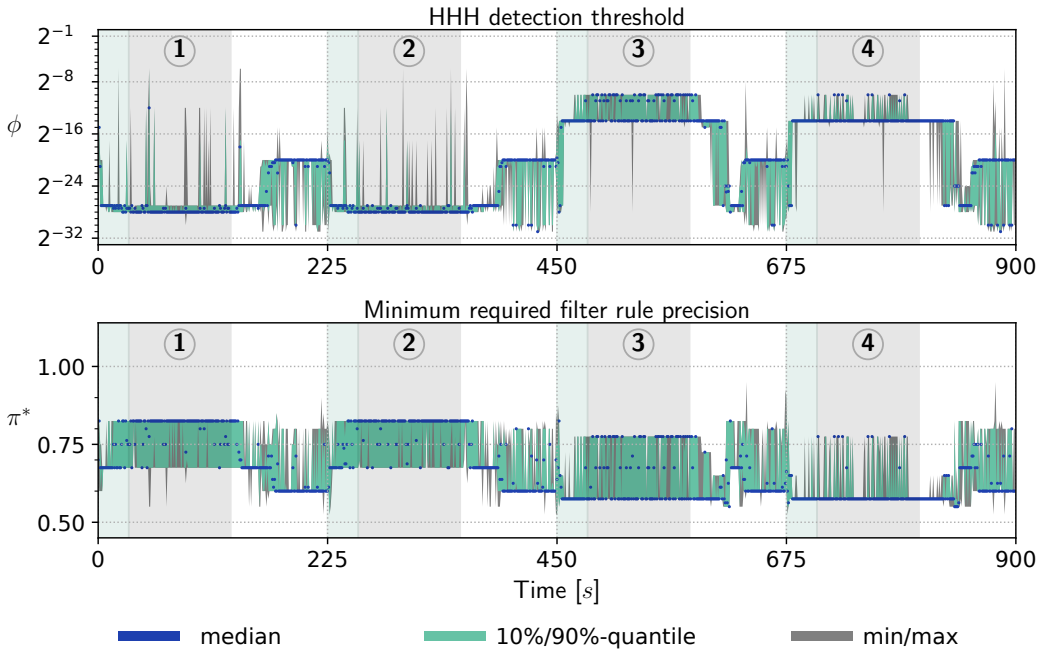
The FNR progression in Figure 6.19 shows variations like in experiment Exp_4. This shows that with a reduced number of filter rules it becomes consistently more challenging to match attack traffic sources in dynamic traffic situations. As a result of lowering w_{FNR}, mean, median, and 90 %-quantile of the FNR in phase ③ and ④ increase. They still remain within a 2 %-margin compared to the previous experiment, but the blacklist size is considerably reduced. In particular, the median value of BSIZE decreases by 24.6 % and 22.8 %. A similar reduction occurs in the mean (19.2 % and 23.6 %) and the 90 %-quantile of phase ④ (19.29 %). There are smaller reductions in the maximum blacklist size (4.2 % and 5.0 %), which shows that RL-based Control must occasionally resort to generating longer prefixes to reduce FPR. This is a result of

still placing a high priority on low FPR by choosing a higher value for the weight w_{FPR}. While reducing ϑ_1 does not affect BSIZE in the first two phases, the mean blacklist has been reduced to 50 % and 42.9 % of the mean number of attack traffic sources in phase ③ and ④. This is despite the fact that in the later stages a larger number of attack traffic sources are used, which can be interleaved with legitimate traffic sources throughout the address space.

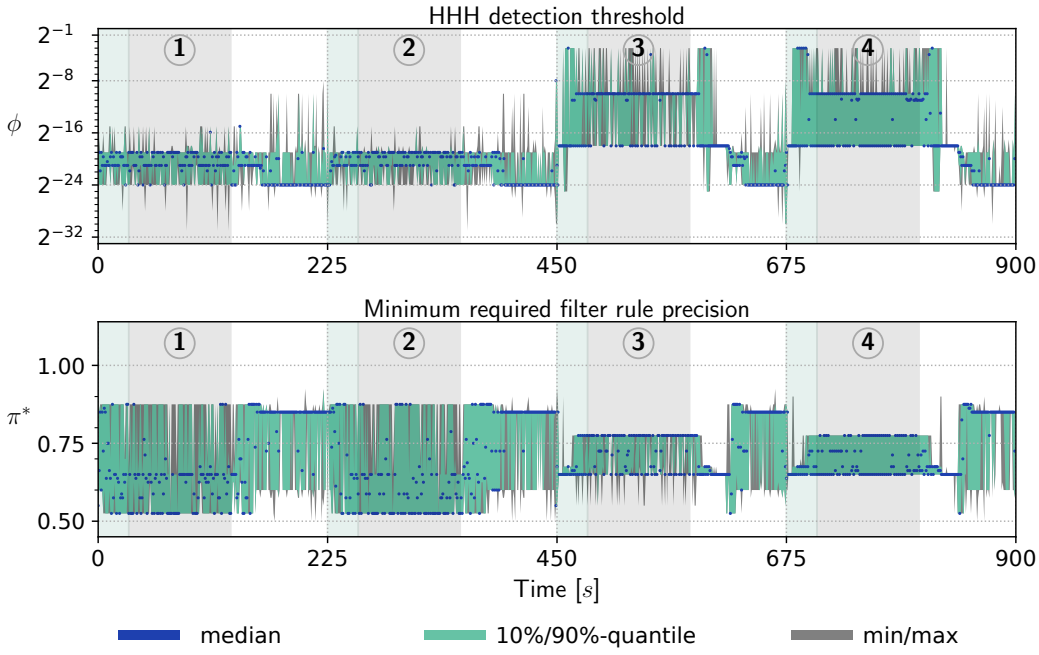
Parameter adaptations in experiment Exp_3, Exp_4, and Exp_5. Figure 6.20 visualizes the parameters ϕ and π^* chosen by trained agents in experiment Exp_3 and Exp_4. The first two and last two phases are clearly distinguishable by parameter choices in both experiments. Furthermore, different parameter choices are selected during the later parts of the cool-down phases. In both experiments, the trained agents exhibit consistent behavior during this time. However, the agents resort to different parameter combinations as a result of random choices occurring during the training. These observations also apply to experiment Exp_5, for which the parameter choices are shown in Figure 6.21. Essentially, all three agents learn to distinguish different traffic situations and adapt the thresholds in response to changing traffic patterns.

The agents of experiments Exp_4 and Exp_4 choose HHH detection thresholds ϕ from a small value range during the first two attack phases. In contrast, the agent in experiment Exp_1 selected ϕ from a larger value range. Both approaches yielded comparable results, as different parameter choices can have the same effect on traffic filtering when few attack sources send high-volume traffic.

During phase ③ and ④ all three agents frequently choose higher values for the HHH detection threshold ϕ than in experiment Exp_1 and Exp_2, where ϕ was mainly chosen at or below values of 2^{-17} and 2^{-15}. This highlights that RL-based Control reduces the blacklist size by causing the generation of shorter prefixes when σ_{BSIZE} uses lower thresholds. The choice of ϕ also affects the choice of π^*. The agents in experiments Exp_1 and Exp_2 primarily selected π^* values of 0.725 or higher. In experiment Exp_3 and Exp_4, values of π^* below 0.7 occur more frequently. Choosing a lower π^*-value typically results in reduced FNR as less prefixes are rejected during Filter Rule Refinement. This is particularly the case when shorter HHH prefixes are generated by the HHH algorithm as a result of choosing a higher ϕ. In this case, more traffic is aggregated before the HHH detection threshold is reached, which results in mixing legitimate and attack traffic and generally lower discounted precision. By choosing lower values for ϕ, the agents reduce the FNR in experiment Exp_3 and Exp_4.



(a) Experiment Exp_3.



(b) Experiment Exp_4.

Figure 6.20: Progression of parameters ϕ and π^* in experiment Exp_3 and Exp_4. The median, 10 %-90 %-quantiles, and maximum are calculated from ten episodes.

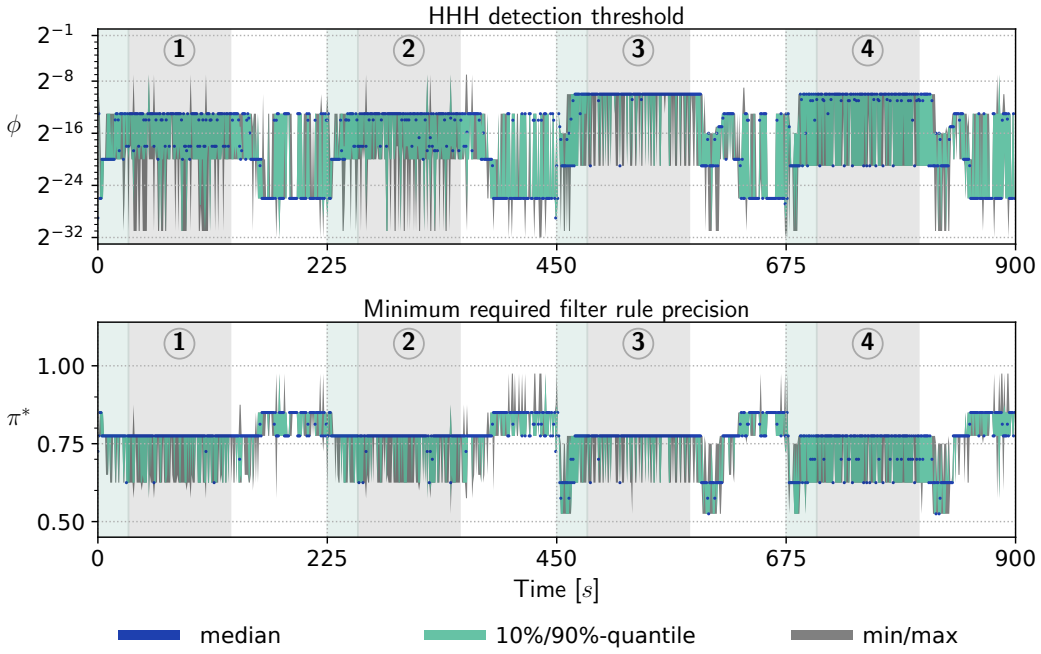
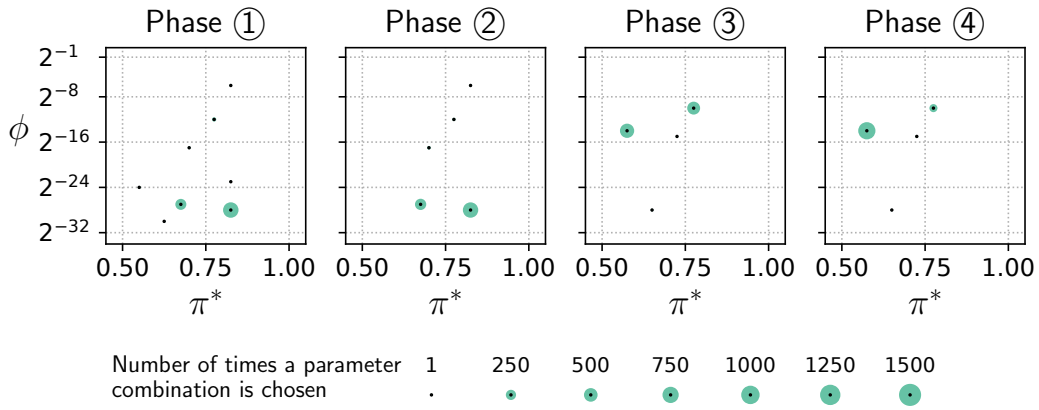
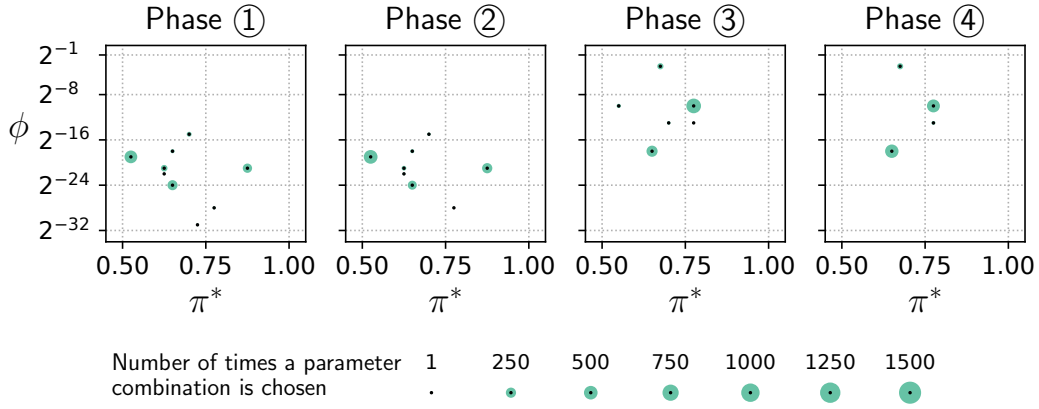


Figure 6.21: Progression of parameters ϕ and π^* in experiment Exp_5. The median, 10 %-90 %-quantiles, and maximum are calculated from ten episodes.

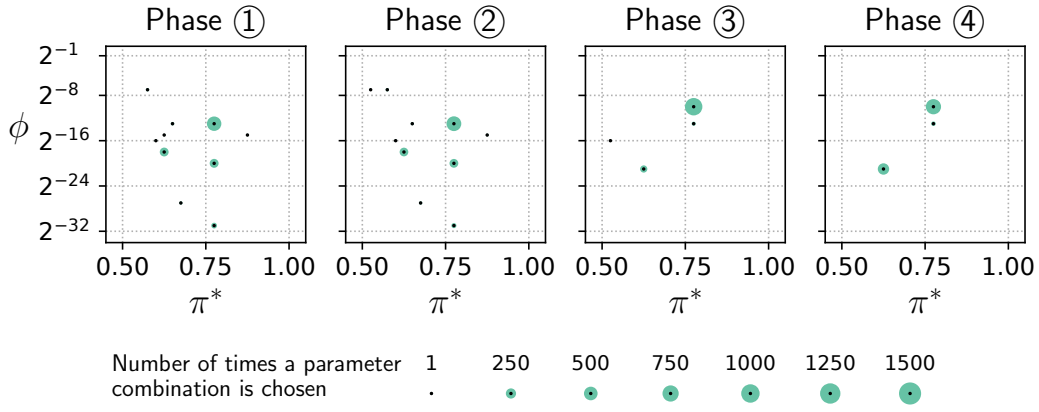
Figure 6.22 shows how often a parameter combination of ϕ and π^* is selected during the stable states of all four attack phases. The distribution of parameter values during the first two phases is a result of having multiple options to achieve comparable filtering results. Throughout experiment Exp_3 to Exp_5, the ϕ parameter reaches higher values in the last two attack phases than than in experiment Exp_1 and Exp_2 (cf. Figure 6.16). Specifically, the agents choose ϕ-values of 2^{-10} in 577, 1522, and 1531 out of the 2000 monitoring intervals of phase ③ and ④ in experiments Exp_3, Exp_4, and Exp_5. Such high values occurred only 80 times in experiment Exp_1 and did not occur in experiment Exp_2. Furthermore, the agents use π^*-values below 0.7 more often in phase ③ and ④ of experiment Exp_3 and Exp_4 to reduce FNR. Such values occur in 1419 out of 2000 monitoring intervals in Exp_3 and 939 times in Exp_4. In contrast, values of π^*-values below 0.7 are selected 429 times in experiment Exp_5. The tendency to use higher π^* value in the last experiment results from reducing the weight w_{FNR}. This allows an agent to reject more prefixes (and incur higher FNR) at equal reward levels. Essentially, the agent's choice of parameter combinations reflect the intended trade-offs conveyed by the reward function parameters of experiments Exp_3 to Exp_5.



(a) Experiment Exp_3.



(b) Experiment Exp_4.



(c) Experiment Exp_5.

Figure 6.22: Distribution of parameters ϕ and π^* in experiment Exp_3, Exp_4, and Exp_5 in the stable states of phases ① to ④. The radius of green-colored circles indicates how often the parameter combination at the center point is chosen.

Experiment	Weights of $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$			Thresholds of σ_{BSIZE}			
	w_{FPR}	w_{FNR}	w_{BSIZE}	ϑ_1	ϑ_2	ϑ_3	ϑ_4
Exp_6	4	0.10	1	700	900	1100	2000
Exp_7	4	0.10	1	600	800	1000	2000

Table 6.17: Reward function parameters for experiments Exp_6 and Exp_7.

Insights. Experiments Exp_3 to Exp_5 show that agent training can be controlled through the tuning of reward function parameters in a consistent manner. The blacklist size is reduced as the thresholds of the shaping function σ_{BSIZE} are lowered. Naturally, this requires trade-offs between the different mitigation goals. However, the results presented in this section indicate that these trade-offs follow the prioritization of mitigation goals. The FPR increases by fractions of percentage points, while the FNR increases in the order of single-digit percentage points. This corresponds reciprocally to the relationship of the weights w_{FPR} and w_{FNR}. A challenge arises from compensating for errors in the estimation of traffic volume by probabilistic HHH algorithms. These errors can impact the calculation of discounted precision and may result in the temporary acceptance of unnecessarily short prefixes in a blacklist, causing temporarily elevated FPR.

6.6.5 Further Reduction of the Blacklist Size

The final two experiments are designed to evaluate RL-based Control's ability to maintain effective trade-offs under tight resource constraints regarding the blacklist size. For this, the reward function parameters are configured as specified in Table 6.17 for experiment Exp_6 and Exp_7. In both experiments, the reward declines before the blacklist size approaches half of the mean number of attack traffic sources in the last two phases.

Figure 6.23 visualizes the progression of FPR, FNR, and BSIZE in experiment Exp_6, while Table 6.18 quantifies the results for stable states. While no significant changes appear during phase ① and ②, the FNR increases considerably in the later phases. Stronger fluctuations than in previous experiments can also be observed. In particular, the median FNR increases by 0.0521 and 0.0497 compared to experiment Exp_5 in phase ③ and ④. In addition, the 90 %-quantile of the FNR increases by 0.137 and

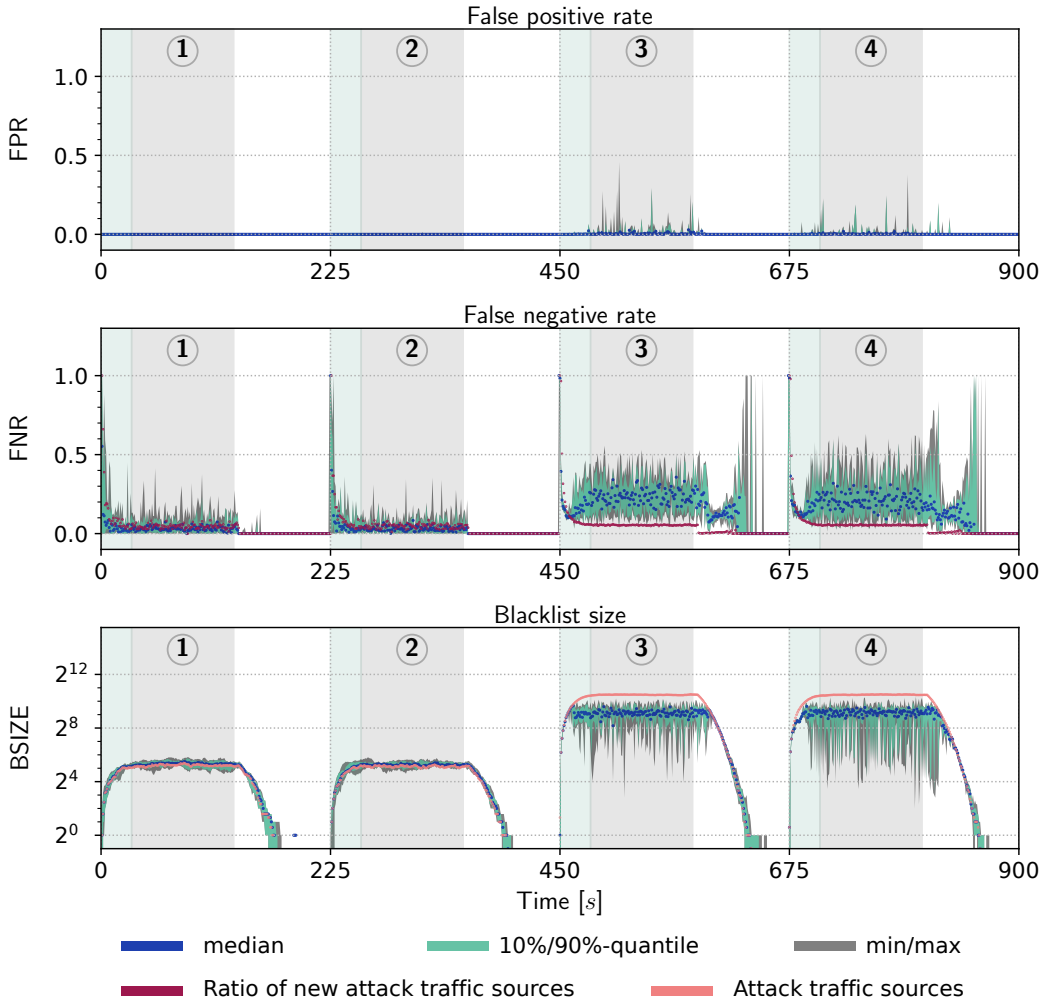


Figure 6.23: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes of experiment Exp_6. Green regions within the four phases ① to ④ indicate the ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

0.159, while the maximum FNR reaches values of 0.5564 and 0.6309. The 90 %-quantile of the FPR also increases by 0.0122 in phase ③ but it remains close to the value of Exp_5 in phase ④. The maximum FPR is a result of a short-term impact on legitimate traffic as in the previous experiments.

In return for increased FPR and FNR, the blacklist size is reduced in the last two phases. For example, this is reflected in the 90 %-quantile, which decreases by 23.3 % and 15.3 % compared to Exp_5. The maximum value of BSIZE is also reduced. During

Metric		Phase				Entire duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0014	0.0009	0.0000
	mean	0.0000	0.0000	0.0096	0.0076	0.0024
	90 %-quantile	0.0000	0.0000	0.0237	0.0122	0.0023
	maximum	0.0018	0.0139	0.4579	0.3776	0.4579
FNR	median	0.0304	0.0291	0.2337	0.2097	0.0388
	mean	0.0430	0.0407	0.2367	0.2234	0.1035
	90 %-quantile	0.0969	0.0949	0.3671	0.3904	0.2824
	maximum	0.4130	0.3530	0.5564	0.6309	1.0000
BSIZE	median	40.0	38.0	553.0	579.0	38.5
	mean	39.8	38.4	545.9	538.2	194.4
	90 %-quantile	48.0	45.0	825.1	769.2	645.0
	maximum	57.0	56.0	1172.0	1281.0	1281.0

Table 6.18: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE in experiment Exp_6. Metrics describe the stable states of the four phases of the mitigation scenario as well as the entire scenario duration (calculated over ten episodes).

the majority of monitoring intervals, the blacklist size remains below the threshold $\vartheta_2 = 900$, indicating that BSIZE can be controlled through the tuning of reward function parameters.

Still, the significant increase in FNR can be considered an ineffective trade-off. Note that the value of BSIZE often decreases significantly during the later phases as shown in Figure 6.23. Such an effect has not been observed in previous experiments. Instead, BSIZE was more consistently centered around its median value. This indicates that the agent is prone to choosing parameter combinations that frequently cause too many prefixes to be rejected.

Since the only changes that occurred in experiment Exp_6 compared to previous experiments are the thresholds applied to the shaping function σ_{BSIZE}, this effect can be considered a result of agent training. Therefore, it may be considered more challenging to train an RL agent to maintain effective trade-offs under increasingly strict blacklist size constraints.

Metric		Phase				Entire duration
		①	②	③	④	
FPR	median	0.0000	0.0000	0.0033	0.0030	0.0000
	mean	0.0000	0.0000	0.0159	0.0145	0.0045
	90 %-quantile	0.0000	0.0000	0.0357	0.0328	0.0067
	maximum	0.0001	0.0424	0.4620	0.3370	0.4620
FNR	median	0.0328	0.0312	0.1206	0.1178	0.0421
	mean	0.0448	0.0427	0.1192	0.1187	0.0763
	90 %-quantile	0.1013	0.0968	0.1610	0.1724	0.1636
	maximum	0.4130	0.3934	0.2316	0.2898	1.0000
BSIZE	median	40.0	38.0	469.0	213.5	38.5
	mean	39.9	38.3	498.9	264.3	137.8
	90 %-quantile	48.0	45.0	789.0	552.2	450.0
	maximum	57.0	55.0	1282.0	880.0	1282.0

Table 6.19: Median, mean, 90 %-quantile, and maximum of FPR, FNR, and BSIZE in experiment Exp_7. Metrics describe the stable states of the four phases of the mitigation scenario as well as the entire scenario duration (calculated over ten episodes).

Experiment Exp_7 further emphasizes the priority of BSIZE by lowering the thresholds of σ_{BSIZE} by an additional value of 100. The progression of FPR, FNR, and BSIZE over the course of the four phases is shown in Figure 6.24. Corresponding metrics are quantified in Table 6.19. Given the results of the previous experiment, a similar increase in at least one of the metrics could be expected. As in the previous experiment, FPR, FNR, and BSIZE in the first two phases remain largely unaffected by the changes to σ_{BSIZE}. However, in contrast to experiment Exp_6 the FNR remains close to the ratio of new attack sources in each monitoring interval during the later phases while the blacklist size shows a considerable reduction.

Specifically, the median value of BSIZE is reduced by 34.2 % and 68.2 % in the last two phases compared to experiment Exp_5. The mean value decreases similarly, while the 90 %-quantile is reduced by 26.7 % and 39.2 %. In effect, the blacklist size reaches only one-third of the mean number of attack traffic sources on average. Meanwhile, the 90 %-quantile of the FNR remains at 0.1610 and 0.1724. This can be considered an effective trade-off between the two metrics. This trade-off comes at

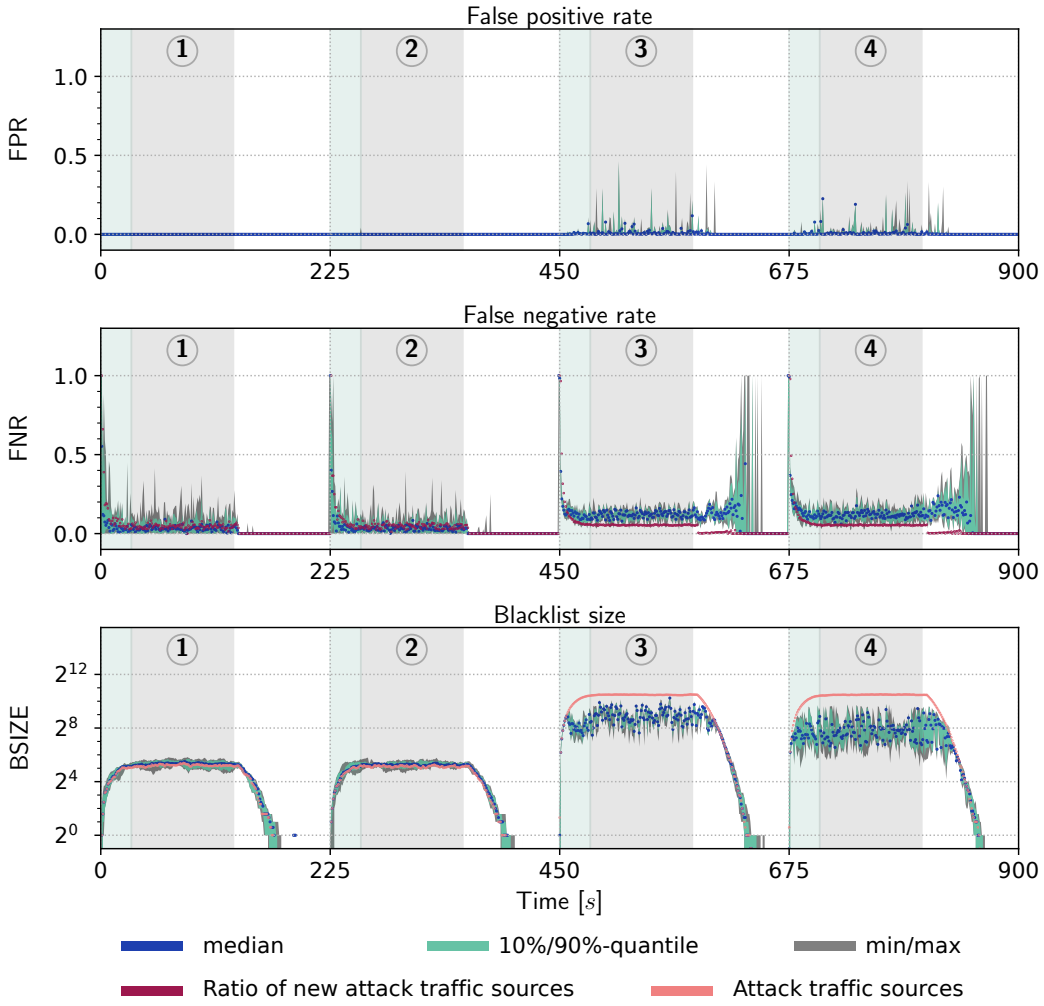
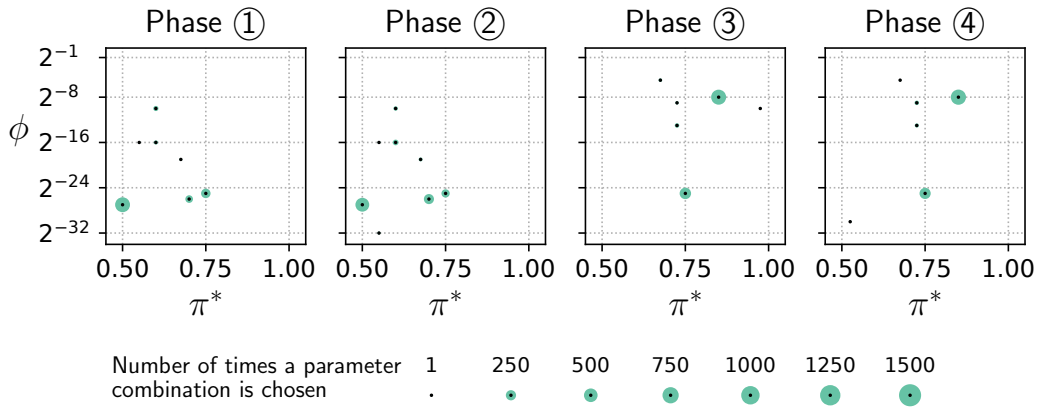
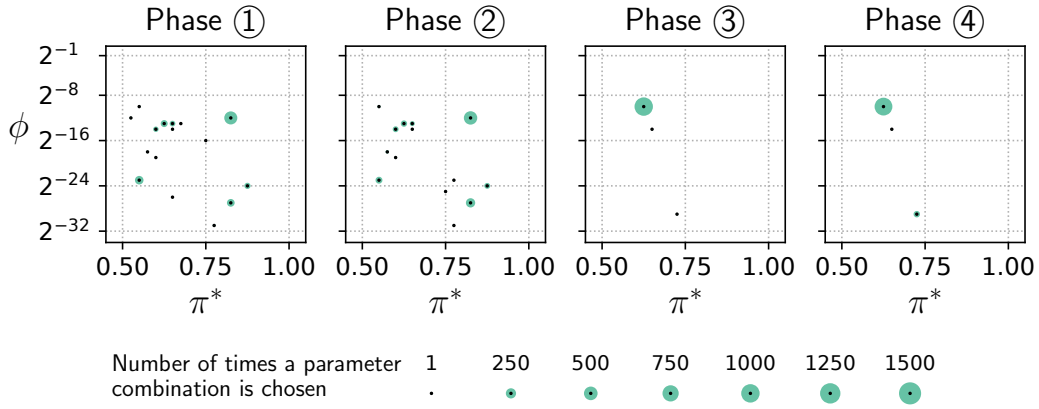


Figure 6.24: Achieved FPR, FNR, and BSIZE for each monitoring interval calculated over ten episodes of experiment Exp_7. Green regions within the four phases ① to ④ indicate the ramp-up phase of an attack while gray regions indicate a stable state during an attack phase. White regions mark the cool-down phase of an attack.

the cost of increased FPR as the value of the 90 %-quantile increases to 0.0357 and 0.0328 (a difference of 0.0242 and 0.0228 in comparison to Exp_5). Still, given the significantly lower blacklist size and the lower FNR, the trade-off realized by the agent in experiment Exp_7 can be considered to be more effective than in experiment Exp_6. This is despite the fact that an agent must learn to maintain trade-offs under stricter resource constraints.



(a) Experiment Exp_6.



(b) Experiment Exp_7.

Figure 6.25: Distribution of parameters ϕ and π^* in experiment Exp_6 and Exp_7 in the stable states of phases ① to ④. The radius of green-colored circles indicates how often the parameter combination at the center point is chosen.

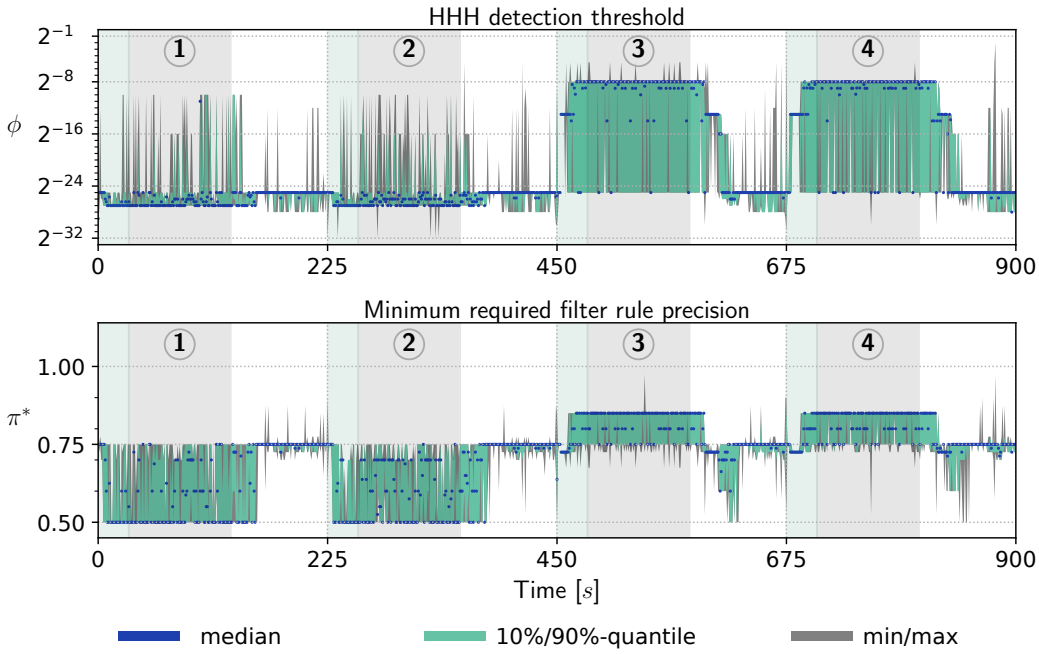
The difference in the results of experiments Exp_6 and Exp_7 can be attributed to the result of agent training, which is subject to random choices. To mitigate the potential of an agent realizing ineffective trade-offs, an agent can be evaluated and retrained (e.g., with slightly different parameters) if it exhibits reduced effectiveness. The results of experiment Exp_6 are presented here to outline that RL is inherently a random process, that does not guarantee overall consistent results. In particular, when it becomes challenging to realize trade-offs under strict constraints. The consequences of that in the context of volumetric DDoS mitigation become apparent in the elevated FNR in phase ③ and ④ of Exp_6.

Parameter adaptations in experiment Exp_6 and Exp_7. Figure 6.25 shows how often agents select specific parameter combinations in experiment Exp_6 and Exp_7. The choices during the first two phases produce similar results as in previous experiments. During the last two phases, the most frequently chosen parameter values of ϕ are 2^{-8} in Exp_6 and 2^{-10} in Exp_7. While these indicate a common preference for short prefixes, the corresponding value of π^* differs considerably.

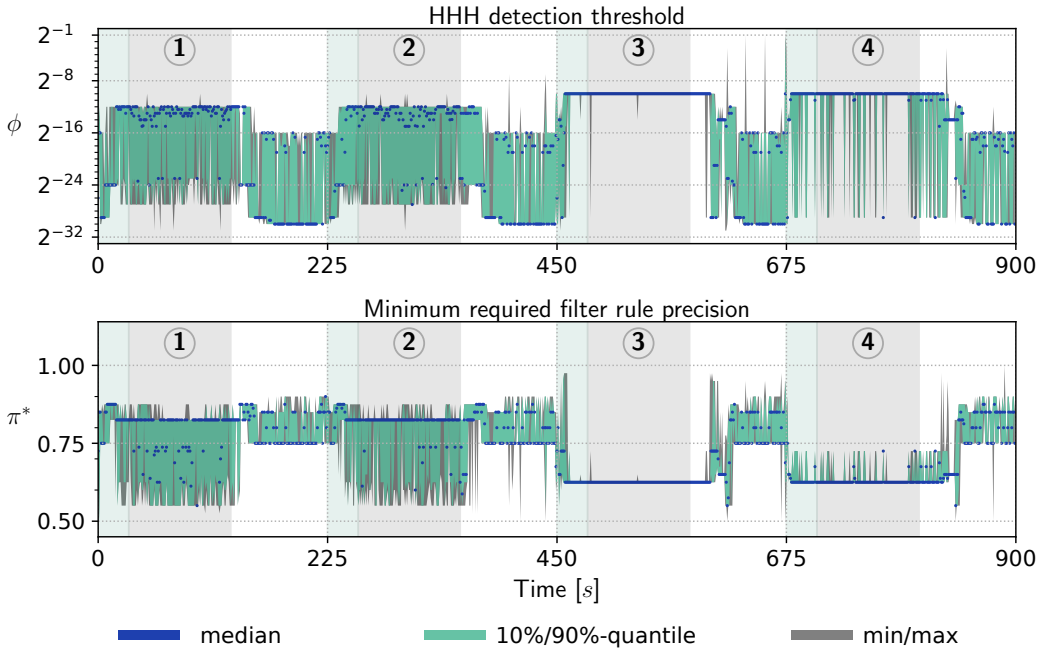
In experiment Exp_6, the agent chooses $\pi^* = 0.85$ while the preferred choice is $\pi^* = 0.625$. These values are chosen 1272 and 1924 times during the 2000 monitoring intervals that comprise the stable state phases over the ten evaluation episodes. The lower value $\pi^* = 0.625$ allows the agent in experiment Exp_7 to maintain a reduced FNR but also leads to traffic filtering discarding legitimate traffic. The value $\pi^* = 0.85$ leads to the rejection of prefixes and causes the higher FNR in experiment Exp_6. The second most frequently chosen parameter combination in Exp_7 is $\phi = 2^{-25}$ and $\pi^* = 0.75$ (selected 644 times). The lower ϕ results in generating longer prefixes, which cannot be maintained under the imposed blacklist size constraints.

The preference for a single parameter combination during the last two phases of experiment Exp_7 is also apparent in the progression of chosen parameter values over time (shown in Figure 6.26). In contrast, the agent of experiment Exp_6 alternates between generating shorter and longer filter rule prefixes due to its choice of the ϕ-value. In conjunction with the dynamic traffic, this results in the fluctuations of FNR and BSIZE in Figure 6.23.

Insights. The final two experiments outlined two primary results. First, experiment Exp_6 showed that RL training is not guaranteed to yield consistent results due to its inherent probabilistic nature. This can cause a diminished effectiveness of filter rule generation and lead to ineffective trade-offs. Second, experiment Exp_7 outlined that a well-trained agent can maintain effective trade-offs even under tight resource constraints (regarding blacklist size). Through this, BSIZE remained below one third of the number of active attack traffic sources during the last two phases. The (necessary) trade-off for this is an increase of the FPR.



(a) Experiment Exp_6.



(b) Experiment Exp_7.

Figure 6.26: Progression of parameters ϕ and π^* in experiment Exp_6 and Exp_7. The median, 10 %-90 %-quantiles, and maximum are calculated from ten episodes.

6.6.6 Findings

The sequence of experiments Exp_1 to Exp_7 was designed to evaluate RL-based Control ability to maintain trade-offs when a low blacklist size is increasingly prioritized. The experiments yielded the following key findings.

First, an RL agent can be trained to differentiate between different traffic situations. This is indicated by the parameter choices over time, which are different when a low or high number of attack traffic sources are active. An agent responds to these changes in traffic patterns by adjusting the thresholds ϕ and π^*. Furthermore, parameter choices occur during the ramp-up and cool-down phases where an agent reacts to newly emerging attack traffic or withdraws previously established filter rules. In conclusion, the design of the agent enables the automatic learning of the relationship between a current mitigation state and the impact of choosing mitigation parameters.

Second, trade-offs can be controlled by tuning the parameters of the reward function $\mathcal{R}^{\sigma_{\mathsf{BSIZE}}}_{\mathsf{HARMONIC}}$. Choosing higher thresholds $\vartheta_1, \ldots, \vartheta_3$ for σ_{BSIZE} in experiment Exp_1 and Exp_2 results in comparatively high BSIZE but reduced FPR and FNR. This corresponds to the intended higher priority of lower FPR and FNR conveyed by the reward function parameters. In comparison, the gradual reduction of thresholds throughout the experiments results in agent training increasingly focusing on lower blacklist sizes. When BSIZE is significantly reduced below the number of active attack traffic sources, an increase in FPR or FNR is challenging to avoid. In particular, momentarily high FPR can occur in dynamic traffic situations when shorter filter rules are used to reduce BSIZE. Still, the results of experiment Exp_5 outline that BSIZE can be reduced to 50 % and 42.9 % of the mean number of active attack traffic sources in phase ③ and ④ of the mitigation scenario. The resulting 90 %-quantile of the FPR remains at ≈ 1 %. A stronger reduction of BSIZE to one third of the number of active attack traffic sources is achieved in experiment Exp_7. In this case, the 90 %-quantile of the FPR remains at ≈ 3 %, which can be considered an effective trade-off when a low blacklist size is important.

Third, experiments Exp_1 to Exp_5 and Exp_7 show a consistently decreasing BSIZE, which is accompanied by steady increases in FPR and FNR. In comparison, the increase in FNR in experiment Exp_6 is not consistent with this general trend. The reason for this lies in the probabilistic nature of RL agent training, as no changes were made to the mitigation scenario or other system components. Consequently, under the challenging conditions in experiment Exp_6 (which placed a high priority

on low BSIZE) the end result of agent training should be supervised to ensure that agents realize effective trade-offs.

6.7 Related Work

A recent approach [Li+23] considers traffic filtering based on IP-prefixes that imposes constraints on the number of required filter rules, the volume of removed attack traffic, and the volume of preserved legitimate traffic. Monitoring information is aggregated over a data structure called F-tree, which models a hierarchy over the IP source addresses of monitored traffic sources. Filter rule generation then enables a selection of rules that ensures that two out of the three constraints are met. In contrast, RL-based Control balances all three metrics. The proposed system relies on external feedback to estimate how much legitimate and attack traffic is impacted by traffic filtering (which is estimated by Traffic Purity Estimation that relies on efficient HHH algorithms in this thesis). Evaluation results based on an emulated DDoS scenario are not directly comparable as the traffic source distributions are unknown, but indicate that a three-second delay occurs before filtering takes place.

AutoDefense [MMW22] employs RL to distinguish between regions in the IP address space that send high attack or legitimate traffic volumes. Specifically, it samples packets to monitor congestion signals (such as TCP retransmissions) that occur when legitimate flows are impacted by DDoS attacks. Based on this information, AutoDefense clusters IP source addresses and congestion signals through unsupervised learning based on decision trees. A RL agent (based on DQN) is trained to control the pruning of nodes in the tree to detect clusters that mainly correspond to legitimate traffic. From this, rules are generated that exclude the IP address regions corresponding to the detected clusters from rate-limiting. Pruning the tree reduces the number of clusters and, therefore rules, while the selection of clusters serves to reduce the impact of rate-limiting on legitimate traffic. AutoDefense samples packets from the data plane to build the decision tree, whereas Selective Aggregate Filtering relies on HHH monitoring that can be performed directly in the data plane. Furthermore, the importance of removing attack traffic can be explicitly conveyed by the reward function in RL-based Control.

6.8 Conclusion

This chapter introduced the concept of RL-based Control, which serves to control trade-offs between FPR, FNR, and BSIZE resulting from filter rule generation in dynamic traffic scenarios. To this end, a design for a RL system was presented that controls filter rule generation via two essential parameters: the HHH detection threshold of Aggregate Monitoring and the (minimum) required discounted precision π^*. A key challenge in the RL system's design is to convey desired trade-offs to an agent. This challenge was addressed in Section 6.3, which introduced an approach to reward function modeling based on the harmonic mean. In particular, a reward function $\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$ was presented that conveys the priority and interdependencies of the three mitigation goals of low FPR, low FNR, and low BSIZE. Key design decisions regarding the modeling of state and action spaces for a DQN-based RL agent, as well as its neural network architecture, were discussed in Section 6.4. Section 6.5 covered details of agent training such as important hyperparameter choices and its execution during inference.

The experiments presented in Section 6.6 outlined that the design of the agent in conjunction with reward function modeling enables RL-based Control to learn and adapt to different traffic patterns. In particular, the results outline that the priority of mitigation goals can be conveyed to an agent by adjusting reward function parameters. This enabled different trade-offs to be realized in a simulated, dynamic mitigation scenario. In fact, an agent can be trained to either prioritize lower false positive rates and lower false negative rates (by generating longer filter rule prefixes) or to reduce the blacklist size. The largest reduction in blacklist size was achieved in experiment Exp_7 (in Section 6.6.5). There, BSIZE was reduced to one third of the number of active attack traffic sources in the later phases of the mitigation scenario, primarily at the cost of an increase in FPR to $\approx 3\,\%$ (in the 90 %-quantile).

Chapter 7

Conclusion

This thesis introduced Selective Aggregate Filtering, a novel approach to protect attack targets and network infrastructures against volumetric DDoS attacks. The concepts presented guided HHH-based filter rule generation with deep learning techniques (supervised learning and RL) to counteract attacks through early upstream attack traffic removal. It was shown that Selective Aggregate Filtering can maintain a balance between the blacklist size and filtering effectiveness when mitigation attacks from individually strong or highly distributed sources. With compact blacklists, memory limitations of switches can be addressed to facilitate fast attack traffic removal.

The HHH algorithms served as a fast and memory-efficient solution for detecting IP subnets of various sizes that send high-volume traffic. However, these algorithms cannot distinguish between attack and legitimate traffic. To overcome this limitation and to leverage the efficiency of HHH algorithms for filter rule generation, Traffic Purity Estimation was introduced in this thesis. It uses deep learning in combination with additional traffic features (besides volume) to estimate the attack traffic purity in traffic aggregates. This information allows it to estimate the distribution of attack traffic volume across the IP subnet hierarchy and to identify address space regions that mainly contain (distributed) attacking systems. While Traffic Purity Estimation requires HHH algorithms to monitor additional features, it balances estimation errors and memory requirements by identifying highly relevant traffic features.

It was shown that knowledge of the distribution of attack traffic over the IP prefix hierarchy is not enough to select prefixes for filter rules. A large subnet can have a high attack traffic purity, but it may still contain regions with purely legitimate traffic. To preserve legitimate traffic, the novel concept of Filter Rule Refinement accounts for the hierarchical relationship between prefixes. It traces the origin of attack traffic back to smaller address space regions to enable more precise traffic filtering. In addition, Filter Rule Refinement tracks filter rule precision over time to

compensate for HHH prefixes that vanish and reappear over time. This effect would otherwise lead to unstable monitoring information and fluctuating filter rules.

Finally, this thesis addressed the challenge of maintaining a balance between false positive rates, false negative rates, and blacklist sizes. This is particularly challenging in scenarios with many attack traffic sources that would require extensive blacklists for accurate attack traffic removal. This challenge was addressed by introducing RL-based Control. It continuously controls the required precision and prefix length of filter rules to generate compact, effective blacklists. Through this, RL-based Control maintains effective trade-offs in diverse volumetric DDoS scenarios.

7.1 Summary of Contributions

The following lists the main contributions of this thesis, which are made by the introduction and the evaluation of the novel concept of Selective Aggregate Filtering.

- A modular architecture for Selective Aggregate Filtering was presented and the workflow of filter rule generation was described. The architecture integrates Traffic Filtering, Aggregate Monitoring (based on HHH algorithms), Traffic Purity Estimation, Filter Rule Refinement, and RL-based Control into a continuously executed control loop that generates and adapts filter rules.
- Traffic Purity Estimation was introduced as a novel deep learning-based method to estimate the ratio of volumetric DDoS attack traffic in traffic aggregates. In particular, this thesis described a neural network architecture, the process of model training, and HHH algorithms that can monitor multiple volumetric DDoS traffic features. To balance HHH memory requirements against estimation errors, a method to identify important attack traffic features was presented.
- Attack scenarios with authentic legitimate traffic and synthetic multi-vector DDoS attack traffic were designed. An evaluation of Traffic Purity Estimation based on these scenarios showed that low estimation errors can be achieved while focusing only on highly relevant traffic features.
- Filter Rule Refinement introduced the notion of discounted precision. It allows the hierarchical relationship between IP prefixes to be taken into account when estimating the impact of filter rules on legitimate traffic. Multiple algorithms were presented to track the discounted precision of IP prefixes over time in order to compensate for frequent changes in HHH monitoring information.

- Filter Rule Refinement was evaluated on the attack scenarios. The findings show that tracking the discounted precision can frequently avoid high false positive rates during traffic filtering.
- RL-based Control introduced a novel concept for maintaining effective trade-offs between false positive rates, false negative rates, and blacklist sizes in dynamic volumetric DDoS scenarios. The design of a DQN agent that controls mitigation parameters was presented, including the agent's neural network architecture, its training process, as well as its state and action spaces. The key challenge of conveying desired trade-offs to an RL agent was addressed through reward function modeling based on harmonic functions.
- A comprehensive evaluation of RL-based Control showed that agents can be trained to maintain desired trade-offs while counteracting volumetric DDoS attacks with different characteristics. In particular, the findings showed that trade-offs with mean FPR and FNR can reduce the mean blacklist to 50 % or less of the number of attack traffic sources.

7.2 Perspectives for Future Research

Selective Aggregate Filtering is designed to mitigate volumetric DDoS attacks, but its concepts can be generalized to a broader range of use cases. The deep learning-based approach of Traffic Purity Estimation quantifies the ratio of traffic with certain characteristics in an aggregate. This could also be used for traffic engineering, for example, to identify and prioritize IP subnets that mainly send traffic of interactive applications (such as video conferencing and gaming applications with specific traffic characteristics). Alternatively, Traffic Purity Estimation could be adapted to detect and locate other kinds of distributed attacks and anomalies. For example, counting packets with set TCP SYN flags can serve to detect high-intensity distributed network scans. The main challenge would be to select and monitor use-case specific features, whereas the general concept of Traffic Purity Estimation would remain largely unchanged.

In addition to choosing different features, the actions of flow rules can be changed to support specific use cases. For example, traffic can be assigned to low-priority queues, rerouted, or forwarded to other systems for further inspection instead of being discarded immediately. Extending Selective Aggregate Filtering to support multiple actions simultaneously would offer differentiated control with a limited number of automatically generated flow rules while processing traffic at line speed.

However, it currently remains an open question how to choose between multiple different actions based on monitoring data that is aggregated over IP subnets.

This thesis considered aggregating traffic features over the IP subnet hierarchy with HHH algorithms to generate prefix-based filter rules. However, HHHs can also be computed over multi-dimensional domains [Cor+08], for example, to detect combinations of IP source prefixes and TCP/UDP source port ranges that correspond to high-volume traffic. A promising research direction is to extend the concepts of Filter Rule Refinement to track (discounted) filter rule precision over a multi-dimensional domain. This can enable the generation of stabilized filter rules with multiple matches (instead of just IP prefixes) that counteract specific attack vectors, such as amplification attacks using fixed UDP ports.

Several advances in deep learning offer further optimization potential for Selective Aggregate Filtering. RL algorithms such as Proximal Policy Optimization [Sch+17] and Soft Actor-Critic [Haa+18] support continuous (real-valued) action spaces, which offer more fine-grained choices for the parameter values that control filter rule generation. Furthermore, Graph Neural Networks [Wu+21] can process traffic volume information over the IP subnet hierarchy directly without the need to generate images for CNNs used by RL-based Control. In addition, both Traffic Purity Estimation and RL-based Control may benefit from rigorous hyperparameter tuning (e.g., to optimize neural network architectures, learning rates, and batch sizes).

Finally, additional trade-offs could be automatically controlled through RL-based Control. For example, monitoring intervals have a fixed duration. Shorter intervals give HHH algorithms less time to collect data, which can increase errors in Traffic Purity Estimation due to less converged traffic features. The resulting impact on filter rule effectiveness may still be acceptable to achieve faster reaction times. Such a trade-off can be controlled by including the monitoring interval duration as an additional parameter in an agent's action space. Likewise, the sampling rate of Aggregate Monitoring could be adjusted dynamically. However, additional trade-offs must be carefully considered, as they may require changing the reward functions and can make training RL agents more challenging, as more parameters must be controlled simultaneously.

Appendices

Appendix A

Terminology

A.1 Mathematical Notations and Important Symbols

The following tables provide an overview of mathematical notations, symbols, and important variables used in this thesis.

Mathematical Notations	
Symbol	**Description**
$\mathbb{N}_0$	The set $\mathbb{N} \cup \{0\}$ of natural numbers and 0
$\mathbb{R}_{>0}$	The subset $\{x \in \mathbb{R} \mid x > 0\}$ of strictly positive real numbers
$\mathbb{E}[X]$	The expected value of a random variable X
(x, y)	The open interval $\{z \in \mathbb{R} \mid x < z < y\}$ of real numbers
$[x, y]$	The closed interval $\{z \in \mathbb{R} \mid x \leq z \leq y\}$ of real numbers
$[x, y)$	The half-open interval $\{z \in \mathbb{R} \mid x \leq z < y\}$ of real numbers
$(x, y]$	The half-open interval $\{z \in \mathbb{R} \mid x < z \leq y\}$ of real numbers
$\mathsf{x}^\top$	The transposed (column) vector of a a (row) vector x
$\mathsf{X}^\top$	The transposed $n \times m$ matrix of an $m \times n$ matrix X
$\frac{\partial \mathcal{L}}{\partial w}$	The partial derivative of a function $\mathcal{L}$ w.r.t. a variable w
X^*	The set of all words over a finite alphabet X
$\mathcal{U}[x, y)$	The uniform distribution over half-open the interval $[x, y)$
$\mathcal{U}[x, y]$	The uniform distribution over the closed interval $[x, y]$
$\mathcal{N}(\mu, \sigma)$	The normal distribution with mean μ and standard deviation σ
$\mathcal{W}(a, b)$	The Weibull distribution with shape and scale parameters a and b

Table A.1: List of mathematical notations

Important Symbols		
Symbol	**Description**	**Reference**
p	An IPv4 prefix	
ϕ	A threshold on traffic volume used to detect (hierarchical) Heavy Hitter prefixes	Definition 2.1 Definition 2.2
$\mathcal{V}$	The total traffic volume in a data stream	Definition 2.2
$\mathcal{H}$	A set of hierarchical heavy hitters	Definition 2.2
c	A counter in the data structure of a heavy hitter algorithm	Section 2.3
ϵ	Exploration rate used by the DQN algorithm	Section 2.5
λ	Learning rate used by to train agents	Section 2.5
γ	Reward discount factor used in the calculation of agent reward	Section 2.5
$\mathbf{A}$	An action space of a reinforcement learning agent	Section 2.5
$\mathbf{a}$	An action from an action space	Section 2.5
$\mathbf{S}$	The state space of a reinforcement learning agent	Section 2.5
$\mathbf{s}$	A state from the state space	Section 2.5
$\mathbf{r}$	Reward gained by a reinforcement learning	Section 2.5
$\mathcal{A}$	A reinforcement learning agent	Section 2.5
$\mathcal{F}$	A filter rule that discard packets based on their IPv4 source address	Definition 3.1
$\mathcal{B}$	A blacklist (set of filter rules)	Definition 3.2
FPR	The false positive rate of traffic filtering	Definition 3.3
FNR	The false negative rate of traffic filtering	Definition 3.4
BSIZE	The size of the blacklist applied for traffic filtering	Definition 3.5
$\mathcal{I}$	A monitoring interval	Definition 3.6
t	A timestamp in seconds	Section 3.3.7
V_{p}	The total traffic volume originating from the subnet of a prefix p	Definition 4.1
A_{p}	The attack traffic volume originating from subnet of a prefix p	Definition 4.1

D	A dataset used to train, validate, or test a machine learning model	Section 4.3.4
$\mathsf{D}_{\text{train}}$	A dataset used to train a machine learning model	Section 4.3.4
D_{val}	A dataset used to validate a machine learning model	Section 4.3.4
D_{test}	A dataset used to test a machine learning model	Section 4.3.4
$\mathsf{D}_{\text{test}}^{(\Pi)}$	A permuted dataset with test data used to assess permutation feature importance	Section 4.4.1
f	A feature used as input for a machine learning model	Definition 4.2
f	A vector of features used as input for a machine learning model	Definition 4.2
π_{p}	The traffic purity of an aggregate	Definition 4.1
$\mathscr{E}$	A traffic purity estimator	Definition 4.3
$\widehat{\pi}_{\mathsf{p}}$	The estimated traffic purity of an aggregate, i.e., the output of a traffic purity estimator $\mathscr{E}$	Definition 4.3
$\pi_{\mathsf{p}}^{\text{err}}$	The absolute error in traffic purity estimation for a prefix p	Definition 4.4
$\mathfrak{I}_f$	The importance of a feature	Definition 4.6
$\pi_{\mathsf{D}}^{\text{MAE}}$	The mean absolute error of traffic purity estimation on a dataset D	Definition 4.5
$\pi_{\text{train}}^{\text{MAE}}$	The mean absolute error of traffic purity estimation on a training dataset $\mathsf{D}_{\text{train}}$	Definition 4.5
$\pi_{\text{val}}^{\text{MAE}}$	The mean absolute error of traffic purity estimation on a validation dataset D_{val}	Definition 4.5
$\pi_{\text{test}}^{\text{MAE}}$	The mean absolute error of traffic purity estimation on a test dataset D_{test}	Definition 4.5
$\pi_{\mathsf{D}}^{\text{MSE}}$	The mean squared error of traffic purity estimation on a dataset D	Definition 4.5
$\pi_{\text{train}}^{\text{MSE}}$	The mean squared error of traffic purity estimation on a training dataset $\mathsf{D}_{\text{train}}$	Definition 4.5
$\pi_{\text{val}}^{\text{MSE}}$	The mean squared error of traffic purity estimation on a validation dataset D_{val}	Definition 4.5
$\pi_{\text{test}}^{\text{MSE}}$	The mean squared error of traffic purity estimation on a test dataset D_{test}	Definition 4.5

$\mathcal{P}$	A five-tuple representing a packet in an evaluation scenario	Section 4.5.1
$\mathcal{P}_{\text{timestamp}}$	The timestamp of a packet in an evaluation scenario	Section 4.5.1
$\mathcal{P}_{\text{size}}$	The size of of a packet in an evaluation scenario	Section 4.5.1
$\mathcal{P}_{\text{src\_addr}}$	The IPv4 source address of a packet in an evaluation scenario	Section 4.5.1
$\mathcal{P}_{\text{protocol}}$	The value of the IPv4 protocol field of a packet in an evaluation scenario	Section 4.5.1
$\mathcal{P}_{\text{src\_port}}$	The source port of a packet in an evaluation scenario	Section 4.5.1
M	A machine learning model	Section 4.5
D_{p}	The discounted volume of a prefix p used in Filter Rule Refinement	Definition 5.2
$\overline{\pi}_{\mathsf{p}}$	The discounted precision of a prefix p used in Filter Rule Refinement	Definition 5.2
π^*	The minimum filter rule precision required by Filter Rule Refinement	Section 5.2
T	A tracker used by Filter Rule Refinement	Definition 5.3
$\mathcal{T}$	A hash table of trackers (indexed by prefixes)	Section 5.3.2
L_{init}	The initial lifetime of a tracker used by Filter Rule Refinement	Section 5.3.2
E	A weighting function used by trackers	Definition 5.3
α	A smoothing parameter used for tackers	Definition 5.3
A_{p}	A vector storing aggregate features for a prefix p	Equation 5.9
$\mathcal{A}$	A hash table used in Filter Rule Refinement to accumulate aggregate vectors	Section 5.3.4
$\mathbf{L}$	A linear action space representing linearly progressing parameter choices	Definition 6.5
$\mathbf{G}$	A geometric action space representing geometrically progressing parameter choices	Definition 6.6
$\dot{-}$	Operator used to express a reciprocal relationship of the metrics FPR and FNR to their importance	Equation 6.2
$[\![\cdot]\!]$	Operator used to ensure that values are not below a fixed small value ϵ (e.g., $\epsilon = 10^{-7}$)	Equation 6.5

H_n	A harmonic mean of n variables	Definition 6.1
$\mathcal{R}_{\text{HARMONIC}}$	A harmonic reward function	Equation 6.4
σ_{BSIZE}	A sigmoid shaping function used to adjust the impact if BSIZE on the reward generated by harmonic reward functions	Definition 6.2
ζ_i	The i-th specified function value of the shaping function σ_{BSIZE}	Definition 6.2
ϑ_i	The i-th threshold on the blacklist size in the shaping function σ_{BSIZE}	Definition 6.2
$\mathcal{R}^{\sigma_{\text{BSIZE}}}_{\text{HARMONIC}}$	A harmonic reward function that shapes the progression of BSIZE before reward calculation	Definition 6.3

Table A.2: List of important symbols

A.2 Traffic Purity Estimation

A.2.1 Python-Based Traffic Purity Estimation Model

Listing A.1 describes how a model with the architecture discussed in section 4.3.3 can be built in Python using the TensorFlow machine learning framework. The arguments of the `build_model` function specify the neural network architecture and the learning rate applied by the optimizer. The function builds a sequentially structured neural network, that consists of a multiple blocks with three layers:

- The `layers.Dense` class represents a fully-connected layer. The number of neurons in the layer is specified in the constructor argument. The activation function is disabled. Instead, activations are performed in a subsequent layer.
- The `LayerNormalization` class represents a layer that performs layer normalization on its input (provided by the previous fully-connected layer).
- The `Activation` class represents a layer that applies an activation function only. The specific layer uses the ReLU function to introduce non-linearity into the model.

```
1  from tensorflow.keras import Sequential
2  from tensorflow.keras import layers, losses, optimizers

4  def build_model(layers = 8, neurons = 128, learning_rate = 1e-4)
       :
5      """ Build a model with the specified architecture
6          and learning rate.
7      """
8      model = Sequential()
9      # Add sequence of dense (fully-connected) and dropout layers
10     for _ in range(layers):
11         # Add fully-connected layer
12         model.add(layers.Dense(neurons, activation = None))
13         # Add normalization layer
14         model.add(layers.LayerNormalization())
15         # Add activation layer
16         model.add(layers.Activation('relu'))
17     # Add final layer with a single neuron
18     model.add(layers.Dense(1, activation = 'linear'))
19     # Set optimizer and loss function
20     optimizer = optimizers.Adam(learning_rate = learning_rate)
21     loss = losses.Huber()
22     # Build the model
23     model.compile(optimizer = optimizer, loss = loss)
24     return model
```

Listing A.1: Simplified code to build a neural network for Traffic Purity Estimation in Python with TensorFlow.

A total of eight blocks of these three layers are added to the neural network. An additional fully-connected layer with a single neuron is then added to the neural network. The neural uses a linear activation function to output estimations on the traffic purity. The model is compiled with an `adam` optimizer that adjusts the learning rate over the course of model training. In addition, the Huber loss function is used to calculate losses during training.

A.2.2 Training of a Traffic Purity Estimation Model

Listing A.2 provides code for the training process of a previously built model (using the `build_model function from Listing A.1`). The arguments of the `train` function specify the batch size and the patience that are applied during model training. A portion of the input features `x_trn` and targets `y_trn` are potentially discarded to

ensure that the number of samples in the training data is a multiple of the batch size. This serves to avoid high losses resulting from small batches. The `train` function then adds two callbacks to the constructed model:

- An `EarlyStopping` callback that continuously monitors the validation loss. If the validation loss does not decrease within a number of training steps specified by the parameter `patience`, the callback terminates model training. This is intended to reduce the risk of overfitting.
- A `ModelCheckpoint` callback that saves the currently trained model whenever the validation loss decreases. Through this, an earlier model can be restored should the validation loss increase before the model training is terminated by the `EarlyStopping` callback. Only the best model is saved at any time.

The `model.fit` function of the TensorFlow framework executes the forward and backward passes of stochastic gradient descent to train the model.

A.2.3 Permutation Feature Importance in Python

Listing A.3 shows Python code to implement a method to determine permutation feature importance. The function `feature_importance` first obtains a reference for the MAE of a trained model on the dataset provided as parameter `data`. It then iterates over the number of features in the dataset. Ten iterations are performed for each feature, in which one feature in the dataset is shuffled by the `permute_feature` function. After shuffling, the model is executed in inference to measure the MAE on the permuted dataset. The reference value of the MAE is then subtracted from the newly measured MAE to calculate the importance of the feature. After all iterations are complete, the function returns the mean importance of each individual feature.

```
from tensorflow.keras import callbacks, layers

def train(data, batch_size = 32768, patience = 300):
    """ Train a model on the provided data. """
    # Split the provided data set into training and
    # validation data.
    # x_trn: the training dataset
    # y_trn: the training data ground truth
    # x_val: the validation dataset
    # y_val: the validation data ground truth
    x_trn, y_trn, x_val, y_val = data

    # Drop incomplete last batch from training data
    last_batch_size = x_trn.shape[0] % batch_size
    if last_batch_size < batch_size / 2:
        x_trn = x_trn[:-last_batch_size]
        y_trn = y_trn[:-last_batch_size]

    # Build the model
    model = build_model(layers = 8, neurons = 128)

    # Define callbacks that are invoked during training
    callbacks = [
        # Cause early stopping when validation losses fail
        # to increase for (patience) number of epochs.
        callbacks.EarlyStopping(monitor = 'val_loss',
            restore_best_weights = True, patience = patience),
        # Save last model that achieves lowest loss
        callbacks.ModelCheckpoint(filepath = ...,
            monitor = 'val_loss', save_best_only = True)]

    # Train the model on the training data using specified
    # parameters. Early stopping evaluates model performance
    # against the validation data.
    model.fit(x_trn, y_trn, validation_data = (x_val, y_val),
        batch_size = batch_size, callbacks = callbacks)
    return model
```

Listing A.2: Training a model in Python with TensorFlow.

```python
import numpy as np

def mae(model, data, targets):
    """ Calculate mean absolute error (mae) of the model
        on the provided data w.r.t. the ground truth (targets)
    """
    # Perform inference on batch
    estimation = model.predict_on_batch(data)
    # Limit estimations to the interval [0, 1]
    estimation[estimation > 1.] = 1.
    estimation[estimation < 0.] = 0.
    # Calculate mean absolute error
    return np.abs(targets - estimation).mean()

def permute_feature(data, index):
    """ Randomly permute the column of feature values at
        the specified index.
    """
    shuffled = np.array(data)
    np.random.shuffle(shuffled[:, index])
    return shuffled

def feature_importance(model, data, targets, features, N = 10):
    """ Calculate feature importance of all features.
        model: a trained model
        data: a test dataset
        targets: ground truth
        features: the number of features
        N: The number of repetitions per feature
    """
    # Calculate mean absolute error (mae) on the original data
    ref = mae(model, data, targets)
    importance = np.zeros(shape = (N, features))
    for f in range(features):
        for n in range(N):
            # Create permuted dataset
            perm = permute_feature(data, f)
            # Calculate increase in mean absolute error
            # compared to performance on original data
            importance[f, n] = mae(model, perm, targets) - ref
    # Calculate and return mean mae increase across repetitions
    return np.array(importance).mean(axis = 1)
```

Listing A.3: Calculating feature importance in Python using a trained model.

A.2.4 Evaluation of Feature Importance

Figure A.1 and Figure A.2 provide supplementary information on the evaluation of feature importance conducted in Section 4.5. Feature importance was evaluated for the model $\mathsf{M}_{\mathsf{ALL}}$ that uses the 82 different features listed in Table 4.5

The feature importance $\mathfrak{I}_{f_i}$ for each feature f_i $(i = 1, \ldots, 82)$ is calculated according to Equation 4.7. Figure A.1 lists the features in order of their index while Figure A.2 lists features in order of importance.

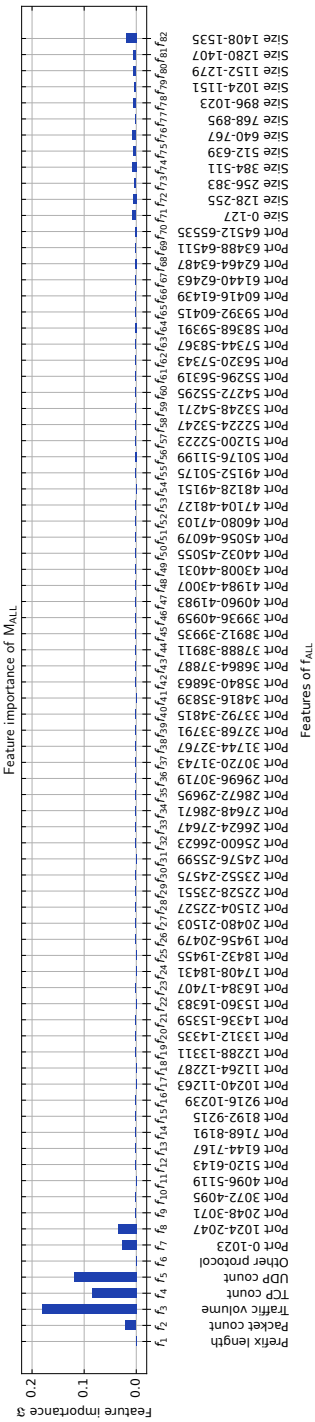


Figure A.1: Feature importance of 82 different features w.r.t. MAE (sorted by feature).

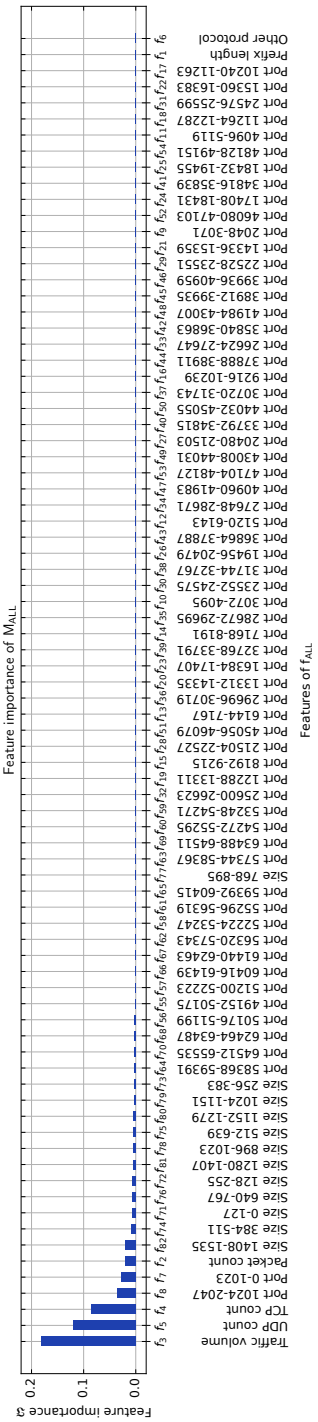


Figure A.2: Feature importance of 82 different features w.r.t. MAE (sorted by MAE increase).

A.3 RL-based Control

A.3.1 Construction of Parameter Action Spaces using Python

Listing A.4 outlines the construction of the parameter action space $\mathbf{A}^{32}_{0.5,1.0,21}$ in Python. The class `ActionsDiscrete32x21` constructs parameter value combinations for the HHH detection threshold ϕ and the minimum required (discounted) filter rule precision π^*. The different combinations are represented by individual actions in the `gymspace` class member. The two class member functions `__init__` and `resolve` perform the following tasks:

- The constructor `__init__` uses the `numpy` and `itertools` to build a cross product of the chosen values of ϕ and π^*. The specific values are generated using `linspace` and `geomspace` classes from the `numpy` library to construct evenly-spaced values and values with a geometric progression. The cross product of generated values (built with the `product` function from the `itertools` library) is then represented as discrete action space that can be used by a reinforcement learning agent to choose actions.
- The member function `resolve` translates actions (from `geomspace`) back to their corresponding parameter value combinations. The parameter values can then be used as (real-valued) thresholds for Aggregate Monitoring and Filter Rule Refinement.

```python
import numpy as np

from itertools import product
from gym.spaces import Discrete


class ActionsDiscrete32x21(object):

    def __init__(self):
        """ The constructor creates a discrete action space
            that represents combinations of values for the
            HHH detection threshold (phi) and the minimum
            required filter rule precision (pi).
        """
        # 32 geometrically progressing values of phi
        # (values are 2^-i for i = 1, ..., 32).
        phi = np.geomspace(2 ** -1, 2 ** -32, 32)
        # 21 evenly distributed values of pi
        # (values are 0.5, 0.525, ...,  1.0).
        pi = np.linspace(0.5, 1.0, 21)
        # The product function (from the itertools library)
        # builds the cross-product of the phi- and pi-values.
        self.actions = np.array(list(product(phi, pi)))
        # The gymspace and shape class members are read by
        # the reinforcement learning agent. They inform the
        # agent of the selectable actions and the total
        # number of actions.
        self.gymspace = Discrete(len(self.actions))
        self.shape = self.actions.shape

    def resolve(self, action):
        """ Resolves an action (from self.gymspace) to the
            specific (phi, pi)-parameter value combination.
        """
        return self.actions[action]
```

Listing A.4: The Python class `ActionsDiscrete32x21` that implements the (discrete) parameter action space $\mathbf{A}^{32}_{0.5,1.0,21} = \mathbf{G}_{32} \times \mathbf{L}_{0.5,1,21}$. The action space contains all choosable values of the HHH detection threshold ϕ and the minimum required (discounted) filter rule precision π^* (denoted as `phi` and `pi` in the code).

Bibliography

[ABF19] Y. Afek, A. Bremler-Barr, and S. L. Feibish. "Zero-Day Signature Extraction for High-Volume Attacks." In: *IEEE/ACM Transactions on Networking* 27.2 (2019), pp. 691–706. DOI: 10.1109/TNET.2019.2899124.

[Abu+20] A. Abusnaina et al. "Insights into Attacks' Progression: Prediction of Spatio-Temporal Behavior of DDoS Attacks." In: *Information Security Applications*. Ed. by I. You. Cham: Springer International Publishing, 2020, pp. 362–374.

[Agg18] C. C. Aggarwal. *Neural networks and deep learning. A Textbook*. Springer Cham, 2018. DOI: 10.1007/978-3-319-94463-0.

[Agr+11] P. K. Agrawal et al. "Estimating Strength of a DDoS Attack in Real Time Using ANN Based Scheme." In: *Computer Networks and Intelligent Computing*. Ed. by K. R. Venugopal and L. M. Patnaik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 301–310.

[Alb+99] S. Albright et al. *Simple Service Discovery Protocol/1.0*. Internet-Draft draft-cai-ssdp-v1-03. Internet Engineering Task Force, Nov. 1999.

[Alc+22] A. G. Alcoz et al. "Aggregate-based congestion control for pulse-wave DDoS defense." In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, pp. 693–706. DOI: 10.1145/3544216.3544263.

[Alk+21] R. Alkhadra et al. “Solar Winds Hack: In-Depth Analysis and Countermeasures.” In: *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 2021, pp. 1–7. DOI: 10.1109/ICCCNT51525.2021.9579611.

[ALK22] M. Anagnostopoulos, S. Lagos, and G. Kambourakis. “Large-scale empirical evaluation of DNS and SSDP amplification attacks.” In: *Journal of Information Security and Applications* 66 (2022). DOI: https://doi.org/10.1016/j.jisa.2022.103168.

[AMA19] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey. “Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey.” In: *IEEE Access* 7 (2019), pp. 107346–107379. DOI: 10.1109/ACCESS.2019.2932422.

[Ant+17] M. Antonakakis et al. “Understanding the Mirai Botnet.” In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110.

[AÖ19] B. G. Assefa and Ö. Özkasap. “A survey of energy efficiency in SDN: Software-based methods and optimization models.” In: *Journal of Network and Computer Applications* 137 (2019), pp. 127–143. DOI: https://doi.org/10.1016/j.jnca.2019.04.001.

[Api+21] A. Apicella et al. “A survey on modern trainable activation functions.” In: *Neural Networks* 138 (2021), pp. 14–32. DOI: https://doi.org/10.1016/j.neunet.2021.01.026.

[ARN19] A. Abhishta, R. van Rijswijk-Deij, and L. J. M. Nieuwenhuis. “Measuring the impact of a successful DDoS attack on the customer behaviour of managed DNS service providers.” In: *SIGCOMM Comput. Commun. Rev.* 48.5 (Jan. 2019), pp. 70–76. DOI: 10.1145/3310165.3310175.

[Bar94] A. R. Barron. “Approximation and estimation bounds for artificial neural networks.” In: *Machine learning* 14 (1994), pp. 115–133.

[Bau20] R. Bauer. “Flow Delegation: Flow Table Capacity Bottleneck Mitigation for Software-defined Networks.” PhD thesis. Karlsruher Institut für Technologie (KIT), 2020. 418 pp. DOI: 10.5445/IR/1000122318.

[BB16] E. Barker and W. Barker. *Guideline for Using Cryptographic Standards in the Federal Government: Directives, Mandates and Policies*. Special Publication (NIST SP), National Institute of Standards and Technology. 2016. DOI: https://doi.org/10.6028/NIST.SP.800-175A.

[Ben+20] R. Ben Basat et al. “Designing Heavy-Hitter Detection Algorithms for Programmable Switches.” In: *IEEE/ACM Transactions on Networking* 28.3 (2020), pp. 1172–1185. DOI: 10.1109/TNET.2020.2982739.

[BKH16] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer Normalization*. 2016. URL: https://arxiv.org/abs/1607.06450.

[Boc+21] K. Bock et al. “Weaponizing Middleboxes for TCP Reflected Amplification.” In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3345–3361.

[Bre01] L. Breiman. “Random forests.” In: *Machine learning* 45 (2001), pp. 5–32. DOI: https://doi.org/10.1023/A:1010933404324.

[BZ16] R. Bauer and M. Zitterbart. “Port Based Capacity Extensions (PBCEs): Improving SDNs Flow Table Scalability.” In: *2016 28th International Teletraffic Congress (ITC 28)*. Vol. 01. 2016, pp. 225–233. DOI: 10.1109/ITC-28.2016.139.

[CCF02] M. Charikar, K. Chen, and M. Farach-Colton. “Finding Frequent Items in Data Streams.” In: *Automata, Languages and Programming*. Ed. by P. Widmayer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[Cha+15] W. Chang et al. “Measuring Botnets in the Wild: Some New Trends.” In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS ’15. Singapore, Republic of Singapore: Association for Computing Machinery, 2015, pp. 645–650. DOI: 10.1145/2714576.2714637.

[Cha02] R. Chang. “Defending against flooding-based distributed denial-of-service attacks: a tutorial.” In: *IEEE Communications Magazine* 40.10 (2002), pp. 42–51. DOI: 10.1109/MCOM.2002.1039856.

[CM05] G. Cormode and S. Muthukrishnan. “An improved data stream summary: the count-min sketch and its applications.” In: *Journal of Algorithms* 55.1 (2005), pp. 58–75. DOI: https://doi.org/10.1016/j.jalgor.2003.12.001.

[Cor+03] G. Cormode et al. “Finding Hierarchical Heavy Hitters in Data Streams.” In: *Proceedings 2003 VLDB Conference*. Ed. by J.-C. Freytag et al. San Francisco: Morgan Kaufmann, 2003, pp. 464–475. DOI: https://doi.org/10.1016/B978-012722442-8/50048-3.

[Cor+08] G. Cormode et al. “Finding hierarchical heavy hitters in streaming data.” In: *ACM Trans. Knowl. Discov. Data* 1.4 (Feb. 2008). DOI: 10.1145/1324172.1324174.

[CS14] G. Chandrashekar and F. Sahin. “A survey on feature selection methods.” In: *Computers & Electrical Engineering* 40.1 (2014), pp. 16–28. DOI: 10.1016/j.compeleceng.2013.11.024.

[Cyb19] Cybersecurity & Infrastructure Security Agency (CISA). *UDP-Based Amplification Attacks*. 2019. URL: https://www.cisa.gov/news-events/alerts/2014/01/17/udp-based-amplification-attacks (visited on 10/12/2023).

[Cyb89] G. Cybenko. “Approximation by superpositions of a sigmoidal function.” In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314. DOI: https://doi.org/10.1007/BF02551274.

[Czy+14] J. Czyz et al. “Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks.” In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. IMC ’14. Vancouver, BC, Canada: Association for Computing Machinery, 2014. DOI: 10.1145/2663716.2663717.

[DA22] P. Dhal and C. Azad. “A comprehensive survey on feature selection in the various fields of machine learning.” In: *Applied Intelligence* (2022), pp. 1–39. DOI: 10.1007/s10489-021-02550-9.

[DQZ18] B. Ding, H. Qian, and J. Zhou. “Activation functions and their characteristics in deep neural networks.” In: *2018 Chinese Control And Decision Conference (CCDC)*. 2018, pp. 1836–1841. DOI: 10.1109/CCDC.2018.8407425.

[DSC22] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. “Activation functions in deep learning: A comprehensive survey and benchmark.” In: *Neurocomputing* 503 (2022), pp. 92–108. DOI: https://doi.org/10.1016/j.neucom.2022.06.111.

[Fer31] W. F. Ferger. “The Nature and Use of the Harmonic Mean.” In: *Journal of the American Statistical Association* 26.173 (1931), pp. 36–40.

[FMC+11] N. Falliere, L. O. Murchu, E. Chien, et al. “W32.Stuxnet Dossier.” In: *White paper, symantec corp., security response* 5.6 (2011), p. 29.

[Fun89] K.-I. Funahashi. “On the approximate realization of continuous mappings by neural networks.” In: *Neural Networks* 2.3 (1989), pp. 183–192. DOI: https://doi.org/10.1016/0893-6080(89)90003-8.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[Gha+17] S. Ghafur et al. "A retrospective impact analysis of the WannaCry cyberattack on the NHS." In: *npj Digital Medicine* 0.-1 (2017), p. 96.

[GJM12] B. B. Gupta, R. C. Joshi, and M. Misra. *Estimating strength of DDoS attack using various regression models*. 2012. arXiv: 1203.2399 [cs.CR]. URL: https://arxiv.org/abs/1203.2399.

[Gri+21] H. Griffioen et al. "Scan, Test, Execute: Adversarial Tactics in Amplification DDoS Attacks." In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 940–954. DOI: 10.1145/3460120.3484747.

[Gup+11a] B. B. Gupta et al. "Estimating Strength of a DDoS Attack Using Multiple Regression Analysis." In: *Advanced Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 280–289. DOI: 10.1007/978-3-642-17881-8_27.

[Gup+11b] B. B. Gupta et al. "On Estimating Strength of a DDoS Attack Using Polynomial Regression Model." In: *Advances in Computing and Communications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 244–249. DOI: h10.1007/978-3-642-22726-4_26.

[Haa+18] T. Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018.

[Hes+23] H. Heseding et al. "SAFFIRRE: Selective Aggregate Filtering Through Filter Rule Refinement." In: *2023 14th International Conference on Network of the Future (NoF)*. 2023, pp. 42–46. DOI: 10.1109/NoF58724.2023.10302813.

[Hes22] H. Heseding. *Reinforcement Learning-Controlled Mitigation of Volumetric DDoS Attacks*. GI SICHERHEIT 2022. 2022. DOI: 10.18420/sicherheit2022_20.

[Hin+12] G. E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580. URL: https://arxiv.org/abs/1207.0580.

[HKZ23] H. Heseding, T. Krack, and M. Zitterbart. “Reducing Memory Footprints in Purity Estimations of Volumetric DDoS Traffic Aggregates.” In: *2nd Workshop on Machine Learning & Netwoking (MaLeNe) Proceedings. Co-located with the 5th International Conference on Networked Systems (NetSys 2023)*. Universität Augsburg, 2023.

[Hor91] K. Hornik. “Approximation capabilities of multilayer feedforward networks.” In: *Neural Networks* 4.2 (1991), pp. 251–257. DOI: https://doi.org/10.1016/0893-6080(91)90009-T.

[HR24] G. Harris and M. Richardson. *PCAP Capture File Format*. Internet-Draft draft-ietf-opsawg-pcap-04. Work in Progress. Internet Engineering Task Force, Aug. 2024. 10 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-opsawg-pcap/04/.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators.” In: *Neural Networks* 2.5 (1989). DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

[Hub64] P. J. Huber. “Robust Estimation of a Location Parameter.” In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. DOI: 10.1214/aoms/1177703732.

[HWH21] Z. Hang, Y. Wang, and S. Huang. “ARMHH: Accurate, Rapid and Memory-Efficient Heavy Hitter Detection with Sliding Window in the Software-Defined Network Context.” In: *2021 IFIP Networking Conference (IFIP Networking)*. 2021, pp. 1–9.

[HZ22] H. Heseding and M. Zitterbart. “ReCEIF: Reinforcement Learning-Controlled Effective Ingress Filtering.” In: *2022 IEEE 47th Conference on Local Computer Networks (LCN)*. 2022, pp. 106–113. DOI: 10.1109/LCN53696.2022.9843478.

[IS15] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167. URL: https://arxiv.org/abs/1502.03167.

[Isy+20] B. Isyaku et al. “Software defined networking flow table management of openflow switches performance and security challenges: A survey.” In: *Future Internet* 12.9 (2020), p. 147.

[Jad20] S. Jadon. “A survey of loss functions for semantic segmentation.” In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. 2020, pp. 1–7. DOI: 10.1109/CIBCB48159.2020.9277638.

[Jar+14] M. Jarschel et al. “Interfaces, attributes, and use cases: A compass for SDN.” In: *IEEE Communications Magazine* 52.6 (2014), pp. 210–217. DOI: 10.1109/MCOM.2014.6829966.

[JC16] I. T. Jolliffe and J. Cadima. “Principal component analysis: a review and recent developments.” In: *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences* (2016).

[Jin+15] X. Jin et al. “CoVisor: A Compositional Hypervisor for Software-Defined Networks.” In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 87–101.

[Jon+17] M. Jonker et al. “Millions of Targets under Attack: A Macroscopic Characterization of the DoS Ecosystem.” In: *Proceedings of the 2017 Internet Measurement Conference*. IMC ’17. London, United Kingdom: Association for Computing Machinery, 2017, pp. 100–113. DOI: 10.1145/3131365.3131383.

[Kan+13] N. Kang et al. “Optimizing the "One Big Switch" Abstraction in Software-Defined Networks.” In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT ’13. Santa Barbara, California, USA: Association for Computing Machinery, 2013, pp. 13–24. DOI: 10.1145/2535372.2535373.

[KB17] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980.

[KDH21] D. Kopp, C. Dietzel, and O. Hohlfeld. “DDoS Never Dies? An IXP Perspective on DDoS Amplification Attacks.” In: *Passive and Active Measurement*. Cham: Springer International Publishing, 2021.

[Kin+16] T. King et al. *BLACKHOLE Community*. RFC 7999. Oct. 2016. DOI: 10.17487/RFC7999.

[KM09] W. Kumari and D. R. McPherson. *Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF)*. RFC 5635. Aug. 2009. DOI: 10.17487/RFC5635.

[Kon+22] M. Kondo et al. “Amplification Chamber: Dissecting the Attack Infrastructure of Memcached DRDoS Attacks.” In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Cham: Springer International Publishing, 2022, pp. 178–196.

[Kre+15] D. Kreutz et al. “Software-Defined Networking: A Comprehensive Survey.” In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76. DOI: 10.1109/JPROC.2014.2371999.

[LBH15] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning.” In: *nature* 521.7553 (2015), pp. 436–444.

[LeC89] Y. LeCun. “Generalization and network design strategies.” In: *Connections in Perspective* (1989).

[Les07] M. Lesk. “The New Front Line: Estonia under Cyberassault.” In: *IEEE Security & Privacy* 5.4 (2007), pp. 76–79. DOI: 10.1109/MSP.2007.98.

[Li+17] J. Li et al. “Feature Selection: A Data Perspective.” In: *ACM Computing Surveys* 50.6 (Dec. 2017). DOI: 10.1145/3136625.

[Li+23] J. Li et al. “Toward Adaptive DDoS-Filtering Rule Generation.” In: *2023 IEEE Conference on Communications and Network Security (CNS)*. 2023, pp. 1–9. DOI: 10.1109/CNS59707.2023.10288699.

[Lil+19] T. P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971. URL: https://arxiv.org/abs/1509.02971.

[Lin92] L.-J. Lin. “Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching.” In: *Mach. Learn.* 8.3–4 (May 1992), pp. 293–321. DOI: 10.1007/BF00992699.

[Liu+19] Z. Liu et al. “Nitrosketch: robust and general sketch-based monitoring in software switches.” In: *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM ’19. Beijing, China: Association for Computing Machinery, 2019. DOI: 10.1145/3341302.3342076.

[LTR17] H. W. Lin, M. Tegmark, and D. Rolnick. “Why Does Deep and Cheap Learning Work So Well?” In: *Journal of Statistical Physics* 168.6 (July 2017), pp. 1223–1247. DOI: 10.1007/s10955-017-1836-5.

[Lu+17] Z. Lu et al. “The Expressive Power of Neural Networks: A View from the Width.” In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[MAE05] A. Metwally, D. Agrawal, and A. El Abbadi. “Efficient Computation of Frequent and Top-k Elements in Data Streams.” In: *Database Theory - ICDT 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[Mar+10] J. Martin et al. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. June 2010. DOI: 10.17487/RFC5905.

[Mar+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[MC89] M. McCloskey and N. J. Cohen. "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem." In: vol. 24. Psychology of Learning and Motivation. Academic Press, 1989, pp. 109–165. DOI: https://doi.org/10.1016/S0079-7421(08)60536-8.

[Mis+24] C. Misa et al. "Leveraging Prefix Structure to Detect Volumetric DDoS Attack Signatures with Programmable Switches." In: *2024 IEEE Symposium on Security and Privacy (SP)*. 2024, pp. 4535–4553. DOI: 10.1109/SP54263.2024.00267.

[MMW22] Y. Mi, D. Mohaisen, and A. Wang. "AutoDefense: Reinforcement Learning Based Autoreactive Defense Against Network Attacks." In: *2022 IEEE Conference on Communications and Network Security (CNS)*. 2022, pp. 163–171. DOI: 10.1109/CNS56114.2022.9947232.

[Mni+13] V. Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602. URL: https://arxiv.org/abs/1312.5602.

[Mni+15] V. Mnih et al. "Human-level control through deep reinforcement learning." In: *nature* 518.7540 (2015), pp. 529–533.

[Moc87] P. Mockapetris. *Domain names - implementation and specification*. RFC 1035. Nov. 1987. DOI: 10.17487/RFC1035.

[Mon14] G. F. Montúfar. "Universal Approximation Depth and Errors of Narrow Belief Networks with Discrete Units." In: *Neural Computation* 26.7 (2014), pp. 1386–1407. DOI: 10.1162/NECO_a_00601.

[MP17] S. Mohurle and M. Patil. "A brief study of wannacry threat: Ransomware attack 2017." In: *International journal of advanced research in computer science* 8.5 (2017), pp. 1938–1940.

[MP43] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133. DOI: 10.1007/BF02478259.

[MRM19] A. Mortensen, T. Reddy, and R. Moskowitz. *DDoS Open Threat Signaling (DOTS) Requirements*. RFC 8612. May 2019. DOI: 10.17487/RFC8612.

[Naw+21] M. Nawrocki et al. “The Far Side of DNS Amplification: Tracing the DDoS Attack Ecosystem from the Internet Core.” In: *Proceedings of the 21st ACM Internet Measurement Conference*. IMC ’21. Virtual Event: Association for Computing Machinery, 2021, pp. 419–434. DOI: 10.1145/3487552.3487835.

[NET24] NETSCOUT. *NETSCOUT DDoS Threat Intelligence Report Issue 13: An Era of DDoS Hacktivism*. 2024. URL: https://www.netscout.com/threatreport/ (visited on 12/05/2024).

[NIS18] NIST National Vulnerability Database. *CVE-2018-1000115 Detail*. 2018. URL: https://nvd.nist.gov/vuln/detail/CVE-2018-1000115 (visited on 10/18/2023).

[Pas+19] A. Paszke et al. “PyTorch: an imperative style, high-performance deep learning library.” In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[Pax01a] V. Paxson. “An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks.” In: *SIGCOMM Comput. Commun. Rev.* 31.3 (July 2001), pp. 38–47. DOI: 10.1145/505659.505664.

[Pax01b] V. Paxson. “An analysis of using reflectors for distributed denial-of-service attacks.” In: *SIGCOMM Comput. Commun. Rev.* 31.3 (July 2001), pp. 38–47. DOI: 10.1145/505659.505664.

[Pos83] J. Postel. *Character Generator Protocol*. RFC 864. May 1983. DOI: 10.17487/RFC0864.

[Pre02] R. Presuhn. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. RFC 3416. Dec. 2002. DOI: 10.17487/RFC3416.

[Raf+21] A. Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations.” In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[Rat90] R. Ratcliff. “Connectionist models of recognition memory: constraints imposed by learning and forgetting functions.” In: *Psychological review* 97(2) (1990), pp. 285–308.

[Ros14] C. Rossow. “Amplification Hell: Revisiting Network Protocols for DDoS Abuse.” In: *NDSS*. 2014, pp. 1–15.

[Ros58] F. Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386. DOI: 10.1037/h0042519.

[RSP14] R. van Rijswijk-Deij, A. Sperotto, and A. Pras. “DNSSEC and Its Potential for DDoS Attacks: A Comprehensive Measurement Study.” In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. IMC ’14. Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 449–460. DOI: 10.1145/2663716.2663731.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[SC16] J.-P. Sheu and Y.-C. Chuo. “Wildcard Rules Caching and Cache Replacement Algorithms in Software-Defined Networking.” In: *IEEE Transactions on Network and Service Management* 13.1 (2016), pp. 19–29. DOI: 10.1109/TNSM.2016.2530687.

[Sch+17] J. Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347. URL: https://arxiv.org/abs/1707.06347.

[SGV13] J. da Silva Damas, M. Graff, and P. A. Vixie. *Extension Mechanisms for DNS (EDNS(0))*. RFC 6891. Apr. 2013. DOI: 10.17487/RFC6891.

[SRK15] D. Storcheus, A. Rostamizadeh, and S. Kumar. “A Survey of Modern Questions and Challenges in Feature Extraction.” In: *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*. Vol. 44. Proceedings of Machine Learning Research. Montreal, Canada: PMLR, Nov. 2015, pp. 1–18.

[SS18] K. Singh and A. Singh. “Memcached DDoS Exploits: Operations, Vulnerabilities, Preventions and Mitigations.” In: *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*. 2018, pp. 171–179. DOI: 10.1109/CCCS.2018.8586810.

[Too+21] O. van der Toorn et al. “ANYway: Measuring the Amplification DDoS Potential of Domains.” In: *2021 17th International Conference on Network and Service Management (CNSM)*. 2021, pp. 500–508. DOI: 10.23919/CNSM52442.2021.9615596.

[Tur04] D. Turk. *Configuring BGP to Block Denial-of-Service Attacks*. RFC 3882. Oct. 2004. DOI: 10.17487/RFC3882.

[Wan+18] A. Wang et al. “Delving Into Internet DDoS Attacks by Botnets: Characterization and Analysis.” In: *IEEE/ACM Transactions on Networking* 26.6 (2018), pp. 2843–2855. DOI: 10.1109/TNET.2018.2874896.

[Wan+20] A. Wang et al. “A Data-Driven Study of DDoS Attacks and Their Dynamics.” In: *IEEE Transactions on Dependable and Secure Computing* 17.3 (2020), pp. 648–661. DOI: 10.1109/TDSC.2018.2808344.

[Wan+22] Q. Wang et al. “A Comprehensive Survey of Loss Functions in Machine Learning.” In: *Annals of Data Science* 9.2 (2022), pp. 187–212.

[WD92] C. J. C. H. Watkins and P. Dayan. “Q-learning.” In: *Machine learning* 8 (1992), pp. 279–292.

[Wie23] S. Wiedemar. “NATO and Article 5 in Cyberspace.” In: *CSS Analyses in Security Policy* (2023). DOI: https://doi.org/10.3929/ethz-b-000610328.

[WLS15] P. Wei, Z. Lu, and J. Song. “Variable importance analysis: A comprehensive review.” In: *Reliability Engineering & System Safety* 142 (2015). DOI: https://doi.org/10.1016/j.ress.2015.05.018.

[Wu+21] Z. Wu et al. “A Comprehensive Survey on Graph Neural Networks.” In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386.

[Yaz+22] R. Yazdani et al. “A Matter of Degree: Characterizing the Amplification Power of Open DNS Resolvers.” In: *Passive and Active Measurement*. Cham: Springer International Publishing, 2022, pp. 293–318.

[YXC18] B. Yan, Y. Xu, and H. J. Chao. “Adaptive Wildcard Rule Cache Management for Software-Defined Networks.” In: *IEEE/ACM Transactions on Networking* 26.2 (2018), pp. 962–975. DOI: 10.1109/TNET.2018.2815983.

[Zha+20] C. Zhang et al. “RETCAM: An efficient TCAM compression model for flow table of OpenFlow.” In: *Journal of Communications and Networks* 22.6 (2020), pp. 484–492. DOI: 10.23919/JCN.2020.000033.

[Zha+21] Y. Zhang et al. “CocoSketch: high-performance sketch-based measurement over arbitrary partial key query.” In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. SIGCOMM ’21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 207–222. DOI: 10.1145/3452296.3472892.

[Zhe+18] J. Zheng et al. “Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis.” In: *IEEE Transactions on Information Forensics and Security* 13.7 (2018), pp. 1838–1853. DOI: 10.1109/TIFS.2018.2805600.

[Zho+22a] Y. Zhou et al. “Raze policy conflicts in SDN.” In: *Journal of Network and Computer Applications* 199 (2022), p. 103307. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2021.103307.

[Zho+22b] Y. Zhou et al. “Raze policy conflicts in SDN.” In: *Journal of Network and Computer Applications* 199 (2022). DOI: https://doi.org/10.1016/j.jnca.2021.103307.

List of Publications

Conference articles

1. Hauke Heseding, Johannes Glöckle, Robert Bauer, and Martina Zitterbart. Balancing Performance Characteristics of Hierarchical Heavy Hitters. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2019

2. Hauke Heseding. Reinforcement Learning-Controlled Mitigation of Volumetric DDoS Attacks. GI SICHERHEIT 2022, 2022

3. Hauke Heseding and Martina Zitterbart. ReCEIF: Reinforcement Learning-Controlled Effective Ingress Filtering. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 106–113, 2022

4. Hauke Heseding, Moritz Dieing, Ankush Meshram, and Martina Zitterbart. Protocol-Agnostic Detection of Stealth Attacks on Networked Control Systems. In *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*, pages 1–7, 2023

5. Hauke Heseding, Timon Krack, and Martina Zitterbart. Reducing Memory Footprints in Purity Estimations of Volumetric DDoS Traffic Aggregates. In *2nd Workshop on Machine Learning & Netwoking (MaLeNe) Proceedings. Co-located with the 5th International Conference on Networked Systems (NetSys 2023)*. Universität Augsburg, 2023

6. Hauke Heseding, Felix Bachmann, Philipp Sebastian Bien, and Martina Zitterbart. SAFFIRRE: Selective Aggregate Filtering Through Filter Rule Refinement. In *2023 14th International Conference on Network of the Future (NoF)*, pages 42–46, 2023