Resource-predictable Machine Learning-based Detection of 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

Samuel Kopmann

Tag der mündlichen Prüfung: 12. Juni 2026

1. Referentin: Prof. Dr. Martina Zitterbart
   Karlsruher Institut für Technologie

2. Referentin: Prof. Dr. Tanja Zseby
   Technische Universität Wien

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Professor Dr. Martina Zitterbart for giving me the opportunity to pursue my doctoral studies through my position at the Institute of Telematics and for supervising this dissertation. I greatly enjoyed our insightful discussions and deeply appreciated the freedom and trust she granted me in shaping the direction of my research. I would also like to thank Professor Dr. Tanja Zseby for agreeing to review my dissertation despite her many responsibilities.

My deepest gratitude goes to Hauke Heseding. His collegiality, continued support, and countless engaging conversations have been invaluable throughout my doctoral journey. I am truly grateful for our friendship and look forward to many more years of collaboration, inspiring ideas, and shared adventures.

I would also like to thank my colleagues and fellow researchers at the Institute of Telematics, especially Roland Bless, Michael König, Felix Neumeister, Hendrik Mahrt, Paul Seehofer, and Frank Winter. I deeply value your friendship and support, both professionally and personally. Your encouragement, companionship, and the many memorable moments we shared made my time at the institute both enjoyable and rewarding.

Working with many talented students has been one of the most rewarding aspects of my time at the Institute of Telematics. Their curiosity, fresh perspectives, and research contributions enriched this dissertation and broadened my own thinking. I would especially like to acknowledge Timon Krack, whose outstanding commitment and contributions had a lasting impact on this research.

Finally, I would like to thank my parents. Your unconditional love, encouragement, and support made both my studies and this dissertation possible. This achievement would not have been possible without you.

Karlsruhe, July 2026

Samuel Kopmann

Contents

List of Figures

List of Tables

List of Variables

IP^{Δ} The destination IP address space.

IP^{Σ} The source IP address space.

IP_A^{Δ} Lower bound of the destination IP address space of a traffic image.

IP_A^{Σ} Lower bound of the source IP address space of a traffic image.

IP_{Ω}^{Δ} Upper bound of the destination IP address space of a traffic image.

IP_{Ω}^{Σ} Upper bound of the source IP address space of a traffic image.

L Traffic image series length.

N^{Δ} The number of destination IP subnets in a traffic image.

N^{Σ} The number of source IP subnets in a traffic image.

Q_1 First quartile of a set of values.

Q_3 Third quartile of a set of values.

Γ Set of color channels in multi-channel traffic images.

α_t The attack intensity compared to the legitimate traffic in time window δ_t.

χ The traffic image resolution.

δ_t The t-th time window in a sequence of time windows.

δ Time window duration for time window-based traffic monitoring.

μ Mean of a set of values.

ρ^{Δ} The destination IP subnet size of a traffic image.

ρ^{Σ} The source IP subnet size of a traffic image.

List of Abbreviations

AS Autonomous System.

CAIDA Center for Applied Internet Data Analysis.

CNN Convolutional Neural Network.

DDoS Distributed Denial-of-Service.

DoS Denial-of-Service.

DT Decision Tree.

FN False Negative.

FP False Positive.

ICMP Internet Control Message Protocol.

IDS Intrusion Detection System.

IPS Intrusion Prevention System.

ISP Internet Service Provider.

LSTM Long Short-Term Memory.

MAWI Measurement and Analysis on the WIDE Internet.

ML Machine Learning.

MLP Multi-Layer Perceptron.

NAT Network Address Translation.

NICs Network Interface Cards.

NTP Network Time Protocol.

PCAP Packet Capture.

ReLU Rectified Linear Unit.

RF Random Forest.

SVM Support Vector Machine.

TCAM Ternary Content Addressable Memory.

TCP Transmission Control Protocol.

TN True Negative.

TP True Positive.

UDP User Datagram Protocol.

Chapter 1

Introduction

Volumetric Distributed Denial-of-Service (DDoS) attacks pose a significant and ongoing threat to the availability of Internet services. By sending large amounts of network traffic from numerous sources towards a common target, attackers overload the target and its surrounding network infrastructure to make it unavailable for legitimate users, which causes considerable damage for the users and the service providers.

The volume and frequency of DDoS attacks continue to increase continuously. For instance, Cloudflare reported a 53 % year-over-year increase in the total number of DDoS attacks between 2023 and 2024 [Clo24]. Furthermore, the peak data rate of the largest recorded attack rose by approximately 430 % within a nine-month period, from 5.6 Tbit/s at the end of 2024 [Clo24] to 29.7 Tbit/s by the third quarter of 2025 [Clo25b]. Considering this trend over last years [Clo] while facing more and more devices and critical infrastructure connected to the Internet, the need for effective detection and mitigation mechanisms is crucial.

Not only do the DDoS attacks' traffic volume and frequency of occurrence increase, but also the attack vectors shift, which is observable through Cloudflare's report about emerging threats [Clo24; Clo25a]. To avoid permanent traffic evaluation and hand-crafting detection rules by network experts, which is prohibitively expensive and infeasible in large-scale networks, supervised Machine Learning (ML) has emerged as a promising approach to detect and mitigate DDoS attacks [Ala+23; NZM23; ACM23; Bah+23].

However, the success of supervised ML for DDoS detection depends on three main requirements. First, network traffic monitoring must capture the detection-relevant traffic characteristics. Second, the reaction time from the start of an attack until its detection must be short to establish countermeasures fast and prevent the target from being overloaded. Given that the largest recorded attack (according to Cloudflare) of

29.7 Tbit/s lasted only 69 seconds [Clo25b], detection systems must react within seconds to be effective. And third, it requires access to labeled network traffic data for training.

1.1 Problem Statement

State-of-the-art network traffic monitoring is based on flow observation, e.g., NetFlow [Cla04] and IPFIX [CTA13]. A flow is defined as a packet sequence monitored at the same observation point sharing a common set of attributes, such as packet headers. The most common flow monitoring approach is microflow-based monitoring, where the packets' common attribute set is defined by the five-tuple of source and destination IP addresses, port numbers, and the transport protocol. As the microflow-based monitoring paradigm is widely adopted, it is the basis for many ML-based DDoS detection approaches. This paradigm also manifests in publicly available datasets [SHG18; Sha+19], as they provide labeled data on a per-microflow basis. The accessibility of labeled microflows through the publicly available datasets further encourages the development of microflow-based detection approaches because the microflows with the corresponding labels can directly by used for supervised ML model training. However, microflow-based detection approaches suffer from the following problems (P1) to (P3), which are particularly severe in the scenario of volumetric DDoS attacks. Furthermore, all supervised ML-based detection approaches face the challenge of labeling traffic data in real-world networks (P4).

(P1) Required CPU and memory utilization depend on the microflow count.

Maintaining microflow-based monitoring requires storing the state of active microflows in the so-called flow table and updating corresponding entries for each arriving packet. This results in a linear growth of memory utilization according to the number of active microflows. To not waste memory in the flow table, it must be periodically checked to remove expired microflows, which demands additional computational resources. In the scenario of volumetric DDoS attacks, with additional thousands of attack flows and millions of packets per second, microflow-based monitoring can become the bottleneck of the detection itself.

(P2) Number of ML classifications and reaction time depend on the microflow count.

Microflow-based detection approaches require one ML classification per microflow. Therefore, the required computational resources scale with the number of active microflows. It also leads to an unpredictable reaction time as the number of active microflows varies over time. This effect becomes even more severe in the scenario of DDoS attacks, as the number of active microflows is significantly larger compared to the legitimate traffic due to the distributed attack sources simultaneously sending traffic towards the victim. Maintaining a short reaction time with microflow-based approaches therefore requires

fast ML classifications, which can be either achieved by restricting the ML model size (to reach short classification durations) or by using expensive hardware, such as GPUs for ML acceleration and parallel processing, to provide sufficient computational resources.

(P3) Microflow-based classification misses detection-relevant features, such as traffic distributions, causing detection performance to suffer.

Microflow-based classification only considers traffic characteristics of an individual microflow, neglecting microflow-overarching features, such as IP address, protocol, and port distributions, which are key characteristics of DDoS attacks.

(P4) Expert-driven labeling is error-prone and expensive in large-scale deployments.

A further significant challenge lies in labeling the large volumes of traffic data generated in today's networks. Although supervised ML shifts the network expert's effort from detection rule creation to label assignment, manual labeling is still prohibitively expensive and impractical at the scale of Internet Service Provider (ISP) networks.

1.2 Research Questions and Main Contributions

This dissertation addresses the above problems (P1) to (P4) by answering the following research questions:

RQ1 Can volumetric DDoS attacks be detected with microflow-independent monitoring and classification, while maintaining a small and fixed computational and memory resource footprint?

RQ2 Beyond binary attack detection with microflow-independent monitoring, can more detailed attack traffic characteristics be derived, such as the attack destination IP subnet or source IP subnets?

RQ3 Can labeled network traffic data for DDoS detection be created autonomously to avoid laborious investigation and hand-crafting rules by network experts?

The answers to the research questions are provided through individual ML-based approaches (Chapters 3-5). An extensive analysis of their benefits and drawbacks is conducted with tailored ML models and real-world datasets. In addition, synthesized datasets are used to evaluate the limitations and trade-offs of the proposed approaches.

Resource-predictable Time Window-based DDoS Detection (Chapter 3 answering RQ1)

This dissertation comprehensively evaluates the **e**fficient **M**icroflow-**in**dependent **D**DoS detection pipeline **eMinD** [KZ23] with fixed computational and memory resource requirements. Compared to microflow-based approaches, eMinD maintains a **small resource footprint**, e.g., 4.12 kB of RAM in an Internet backbone scenario, which constitutes a reduction in memory utilization of 99.98 percent compared to microflow-based monitoring. It reduces the number of ML classifications by 99.99 percent compared to microflow-based classification and further enables a **fixed and subsecond reaction time**.

This is achieved by changing the monitoring paradigm from per-flow monitoring, which requires microflow segregation, to time window-based monitoring of the whole arriving network traffic. Instead of extracting one feature set per microflow, e.g., the mean packet size and inter-arrival time, time window-based monitoring extracts **one feature set per time window** δ (e.g., one second). This monitoring paradigm change consequently requires only **one ML classification per time window** instead of one classification per microflow (potentially thousands of classifications).

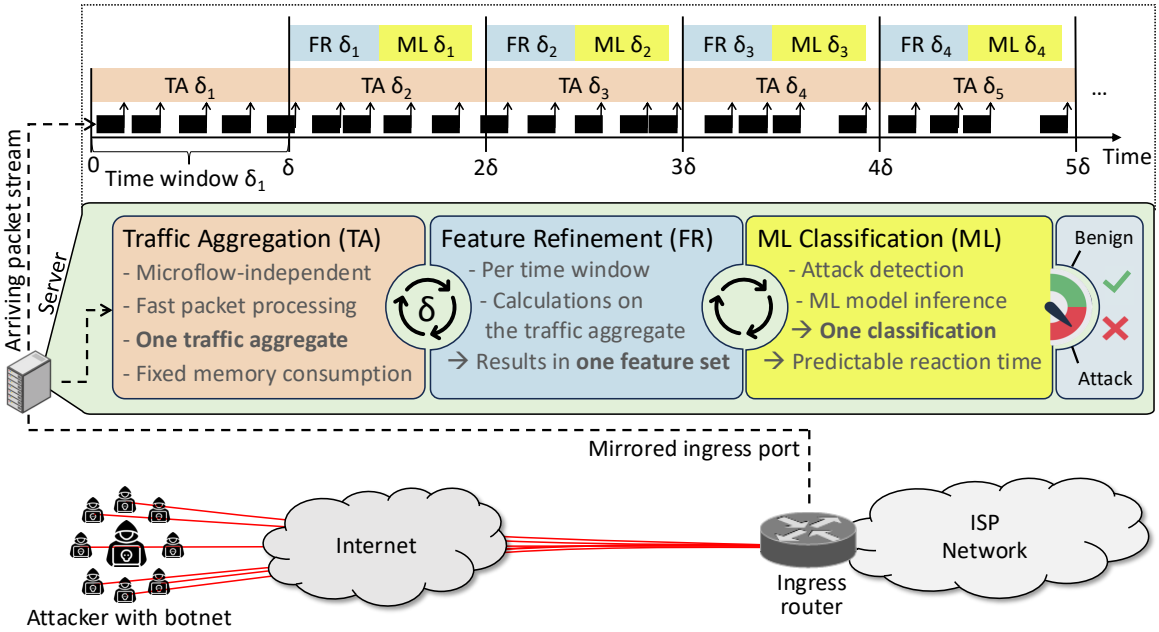


Figure 1.1: *Illustration of the eMinD pipeline with three consecutive stages: traffic aggregation, feature refinement, and ML classification.*

eMinD (see Figure 1.1) is deployed at a server collocated with an ingress router of an ISP network and receives mirrored ingress traffic, i.e., the arriving packet stream. It employs three consecutive and time window-based (duration δ) pipeline stages, namely traffic aggregation, feature refinement, and ML classification.

During the traffic aggregation stage, the traffic aggregate is calculated maintaining a small memory footprint in the order of kilobytes. This is achieved through a fixed set of monitored traffic attributes, such as the total number of packets and the sum of all packet sizes, that are incrementally updated for each arriving packet. Updates facilitate fast computational operations, such as bitwise operations and integer addition. After each time window δ_t, i.e., during the following time window δ_{t+1}, the feature refinement reads (and resets) the traffic aggregate of δ_t and refines it to one feature set with derived statistical measures for subsequent ML classification, e.g., divide the sum of all packet sizes by the number of packets to calculate the mean packet size.
Time window-based monitoring of all arriving packets enables the use of microflow-overarching features, such as IP address and port distributions, which are crucial for DDoS detection [KZ24]. Furthermore, eMinD's design makes flow table maintenance obsolete, saving additional computational resources. eMinD reduces the required computational and memory resources for monitoring, feature refinement, and ML classification in a hierarchical manner: For each time window δ_t, **N arriving packets** are aggregated into **10 aggregated traffic attributes** (traffic aggregate), which are refined into **one feature set** resulting in **one ML classification**.

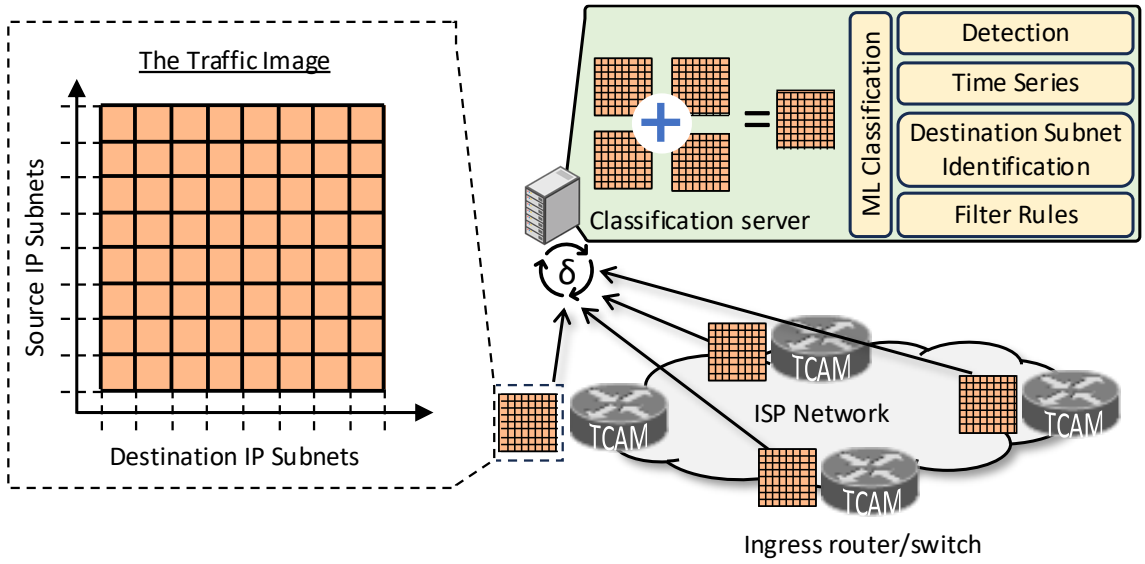


Figure 1.2: *Illustration of the traffic image, an IP prefix-preserving traffic aggregate, with an IP subnet resolution of 8x8 and three color channels.*

IP Distribution-preserving Time Window-based DDoS Detection (Chapter 4 answering RQ2)
Building upon the eMinD pipeline, this dissertation introduces the traffic image [KHZ22b] (see Figure 1.2), an IP prefix-preserving traffic aggregate preserving the microflow-independent, time-window-based monitoring paradigm of eMinD. In contrast to eMinD,

which relies on a server for traffic aggregation collocated with an ingress router, the packet-to-pixel mapping can be implemented in hardware with Ternary Content Addressable Memory (TCAM). Therefore, traffic images can be monitored in and fetched from routers and switches equipped with TCAM, which avoids the cost of additional monitoring hardware. Every time window δ_t, the classification server collects one traffic image from each ingress node (routers or switches), refines it, and classifies it with a Machine Learning (ML) model.
Traffic images consist of pixels that represent source-to-destination IP subnet pairs. Each pixel holds a counter that is incremented for every matching packet arriving during a time window δ_t. Due to the IP prefix-preserving pixel design, traffic images capture DDoS attacks column-wise, as the attacks are distributed over the source IP address space and focus on a specific target in the destination IP address space. Through traffic image classification approaches that are inspired by the research area of computer vision, conclusions about the attack's source IP subnets and the target IP subnet can be drawn. As traffic images only consist of counters, multiple traffic images fetched from different ingress routers can be aggregated (through summation) at the classification server, which enables a global view of the monitored ingress traffic, e.g., fetching traffic images from all ingress routers of an ISP network. Furthermore, TCAM rule evaluation (triggered per packet arrival) requires only one clock cycle, which enables fast and packet-sampling-free monitoring.
Despite the fixed state size of traffic images, they enable fine-grained DDoS detection through destination IP subnet identification [KKZ24a; KKZ24b] and DDoS mitigation through filter rules [KHZ23].
Traffic images also facilitate time-series classification [KHZ22a], which enables to capture and learn the traffic dynamics of DDoS attacks over time, such as the development of the source IP address distribution. This is especially important for fast detection of an attack's start and end, as attack sources in large botnets are joining and leaving the attack step-by-step. Being able to observe this gradual step-by-step change of the traffic volume and the IP address distribution in a time series of traffic images enables a faster reaction time and improved detection performance.

Feedback-driven Autonomous Denial-of-Service Traffic Data Labeling (Chapter 5 answering RQ3)
As supervised ML relies on labeled training data and expert-driven label assignment is infeasible considering ISP-scale networks, this dissertation outlines a novel feedback-driven, **autonomous labeling** approach *FeADable* for DDoS traffic [KLZ25]. FeADable complements the online classification pipeline of eMinD with offline training dataset creation and model training. However, it is not constrained to feature sets of traffic aggregates, such as traffic images, but can also be applied to microflow-based feature sets, and achieves low **false-positive rates below 0.001**.

FeADable reduces the required knowledge for label assignment to only the start and the end time of a DDoS attack, i.e., the attack period, which can be actively polled or be reported by an affected end-system. According to registered attack periods, the monitored traffic is divided into legitimate traffic (before and after the attack period) and unknown traffic (during the attack period). Legitimate traffic, monitored immediately before and after individual attacks, serves as training data for an autoencoder that subsequently classifies unknown traffic during attack periods into the classes legitimate or attack.

This attack period-specific labeling approach considers the current legitimate traffic characteristics (before and after the attack) to separate legitimate from attack traffic during the attack period, and consequently avoid false-positives. It is fully autonomous, does not require a human in the loop, and therefore provides a scalable approach for labeling DoS attack data in large real-world networks.

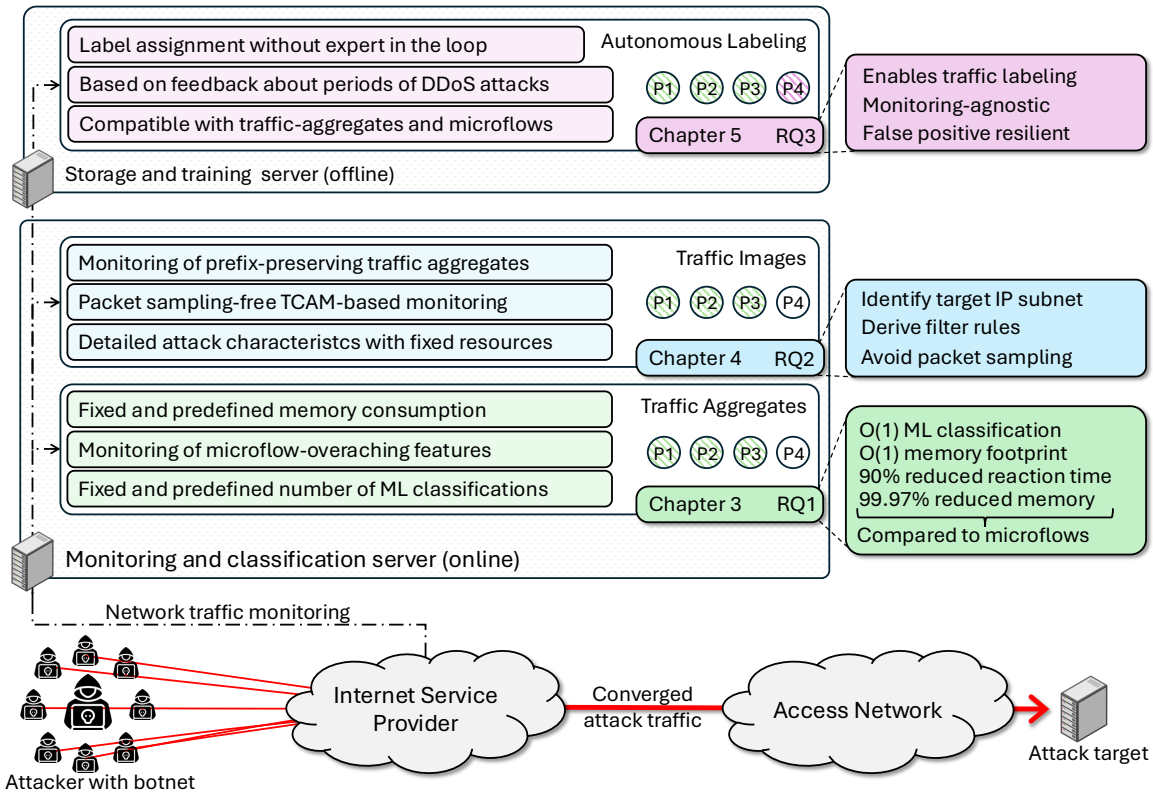


Figure 1.3: *Dissertation overview with the contribution of individual chapters, the addressed research questions RQ1-RQ3, and the solved problems (P1) to (P4)*

1.3 Outline

Figure 1.3 illustrates the contributions of individual chapters and their connection. The remainder of this dissertation is organized as follows.

Chapter 2: Background and Related Work
This chapter provides the required background on volumetric DDoS attacks, the used real-world datasets, and the ML methodology. It also discusses related work on ML-based DDoS detection and the limitations of existing labeling approaches.

Chapter 3: Resource-predictable Time Window-based DDoS Detection
This chapter introduces a novel, microflow-independent traffic monitoring approach based on time-window-based traffic aggregates. It presents the design with its goals and the implementation of an ML-based detection pipeline with fixed resource requirements. Detailed evaluation and trade-off analysis show that the proposed approach addresses RQ1 by enabling scalable DDoS detection regarding computational and memory resources while maintaining low reaction times and improving detection accuracy.

Chapter 4: IP Distribution-preserving Time Window-based DDoS Detection
Building on traffic aggregates, this chapter introduces and evaluates traffic images, IP prefix-preserving traffic aggregates preserving a fixed and predictable state size. Designed for hardware-based monitoring on routers and switches equipped with TCAM, traffic images enable fast and packet sampling-free monitoring. Traffic images support time-series classification, capturing the temporal development of source and destination IP distributions. Furthermore, a hierarchical IP drill-down (iterative traffic image zoom) allows fine-grained destination identification and mitigation with fixed computational and memory resources, thus addressing RQ2.

Chapter 5: Feedback-driven Autonomous Denial-of-Service Traffic Data Labeling
This chapter introduces and evaluates the fully autonomous, feedback-driven labeling approach FeADable that overcomes the limitations of manual traffic labeling in large-scale networks. The approach requires only feedback on the start times, the end times, and the target IP addresses of DDoS attacks to distinguish between legitimate and potentially malicious traffic with attack-specific trained autoencoders. This enables precise labeling, which avoids false positives, and eliminates the need for expert involvement. FeADable supports scalable training dataset creation and addresses RQ3.

Chapter 6: Conclusion
The final chapter summarizes the main contributions of the dissertation, reviews the research questions and how they were addressed.

Chapter 2

Background and Related Work

This chapter introduces the required background information for this dissertation, such as volumetric Distributed Denial-of-Service (DDoS) attacks and the applied Machine Learning (ML) methodology. It also outlines the real-world datasets used for evaluation. Furthermore, it discusses related work in the areas of DDoS detection, image-based network traffic classification, and DDoS attack traffic labeling.

2.1 Volumetric Distributed Denial-of-Service Attacks

Volumetric DDoS attacks aim at limiting the availability of a targeted service for legitimate users. The primary objective is to saturate the available bandwidth of the target and its surrounding network infrastructure, preventing legitimate packets from reaching the target and therefore rendering the service unavailable. This objective is achieved by sending network traffic from thousands to hundreds of thousands of distributed sources, e.g., a botnet consisting of compromised devices, towards the common target. A famous example of such a botnet is the Mirai botnet [Ant+17], which consists of compromised Internet of Things (IoT) devices (e.g., through hard-coded credentials) and is ”specifically designed for performing Distributed Denial of Service (DDoS) attacks” [Mar+17]. The attacker steers the botnet through a command-and-control infrastructure and orchestrates the attack by sending attack vector-specific commands to the bots.

Although there exists a plethora of different DDoS attack vectors [Sha+19], they can be coarsely categorized into two main types: direct DDoS attacks and reflection DDoS attacks.

2.1.1 Direct DDoS Attacks

During direct DDoS attacks, attack traffic is sent directly from the attack sources to the target. Therefore, the attack traffic volume depends on the number of attack sources and their individual traffic generation capabilities, e.g., available bandwidth. Direct DDoS attack vectors relevant for this dissertation (as they are included in the used datasets) are UDP flooding, TCP SYN flooding, and ICMP flooding attacks.

UDP Flooding Attack
User Datagram Protocol (UDP) flooding attacks involve overwhelming the victim with a high volume of UDP datagrams. Besides consuming bandwidth, these datagrams are also processed by the victim's network stack, which leads to computational overhead on the victim. Therefore, when performed at scale, UDP flooding attacks consume both bandwidth and CPU resources, leading to severe performance degradation or complete service disruption.

TCP SYN Flooding Attack
Transmission Control Protocol (TCP) SYN flooding attacks abuse the TCP connection establishment procedure. The attack sources only send TCP SYN segments to initiate TCP connections with the victim but do not complete the three-way handshake. The victim allocates state for each TCP connection request and responds with SYN-ACK segments, waiting for the final ACK that never arrives. As the number of incomplete connection establishments grows, the victim's available resources for TCP connection establishment become saturated, preventing it from accepting new legitimate TCP connections. This attack primarily targets server-side resources for connection management, in addition to consuming network bandwidth.

ICMP Flooding Attack
Internet Control Message Protocol (ICMP) flooding attacks overwhelm the victim by sending large volumes of ICMP echo requests. For each received request, the victim processes the packet and sends a corresponding ICMP echo reply. At high request rates, the processing and transmission overhead saturates the victim's CPU and network resources.

2.1.2 Reflection DDoS Attacks

Although the datasets used in this dissertation do not contain reflection DDoS attack traffic, this section briefly introduces them for completeness. Reflection DDoS attacks leverage publicly accessible third-party servers, e.g., Network Time Protocol (NTP) servers [Czy+14], to multiply attack traffic and redirect it towards the victim. Instead of sending traffic directly to the target (direct DDoS attacks), the attack sources send

requests to the third-party servers, forging the source IP address to the victim's IP address, i.e., source IP address spoofing. As a result, the third-party servers send their responses, which are larger in size than the original requests, to the victim, which leads to an amplification of the original attack traffic volume. This amplification can reach factors of thousands, e.g., 4960 for NTP reflection attacks [Ros14].

2.2 Cardinality Estimation with Bloom Filters

A *bloom filter* is a space-efficient probabilistic data structure designed to test set membership [Blo70]. It supports insertions and membership queries but does not allow deletions or storage of actual items. A standard bloom filter consists of a **bit array of length m** and k independent hash functions. To insert an element, it is hashed k times, and the corresponding bit positions (hash values) are set to 1. To check membership for an element e, the same k hash functions are computed for e and the corresponding bits are examined. If all bits are 1, e is *potentially* in the set, otherwise, it is definitely not.
Bloom filters are not inherently designed to estimate the number of distinct elements in a stream. However, their structure can be exploited to derive approximate cardinalities by using $k = 1$ and examining the number of set bits after a sequence of insertions, i.e., the simple bit counting approach.
However, the simple bit counting approach does not account for hash collisions. If multiple elements hash to the same bit array position, the bit array's state does not reflect the true number of distinct elements. This can be mitigated by using the *linear counting formula* [WVZT90], which estimates the number of distinct elements $\hat{n}$ based on the number of unset bits, i.e., bits that remain 0 after a sequence of insertions.
Let n distinct elements be inserted into a bit array of size m using a single hash function. Under the assumption of uniform and independent hash outputs (a bit position is hit with probability $1/m$), the probability for any specific bit to remain unset after n insertions is:

$$p_0 = \left(1 - \frac{1}{m}\right)^n \tag{2.1}$$

Therefore, the expected number of unset bits $E[X]$ in the bit array is:

$$E[X] = m \cdot p_0 = m \cdot \left(1 - \frac{1}{m}\right)^n \tag{2.2}$$

Replacing the expected number of unset bits $E[X]$ with the actually observed number of unset bits X, the equation can be rearranged to obtain an estimate of the number of distinct elements $\hat{n}$:

$$\frac{X}{m} \approx \left(1 - \frac{1}{m}\right)^{\hat{n}} \implies \hat{n} \approx \frac{\ln(X/m)}{\ln(1 - 1/m)} \tag{2.3}$$

Applying the Maclaurin series expansion for $\ln(1-\frac{1}{m})$ leads to the infinite series:

$$\ln(1-\frac{1}{m}) = -\sum_{i=1}^{\infty}\frac{(1/m)^i}{i} = -\frac{1}{m} - \frac{1}{2m^2} - \frac{1}{3m^3} - \dots \tag{2.4}$$

For large m, higher-order terms become negligible, and the series can be approximated by $-\frac{1}{m}$, which leads to the *linear counting formula* for the number of distinct elements $\hat{n}$, where X is the number of unset bits in a bit array of size m:

$$\hat{n} \approx -m \cdot \ln\left(\frac{X}{m}\right) \tag{2.5}$$

2.3 Datasets

This section introduces the real-world datasets used for the evaluation in this dissertation. First, a dataset from a major tier-1 ISP (Section 2.3.1) is presented, which contains network traffic monitored at the ingress routers of the ISP network. This section further covers the CIC-IDS 2017 dataset (Section 2.3.2), which is used to provide comparability to related work. Furthermore, it introduces the CAIDA DDoS 2007 dataset (Section 2.3.3), the MAWI datasets (Section 2.3.4), and the self-crafted MAWI+CAIDA dataset (Section 2.3.5), which are used to evaluate the proposed approaches in realistic Internet-scale (backbone) DDoS attack scenarios. Finally, the BelWue dataset (Section 2.3.6) is presented, which contains authentic network traffic (legitimate and attack traffic recorded in April 2025) of a regional ISP network. All datasets except the ISP datasets (BelWue and the major tier-1 ISP dataset) are publicly available and therefore enable reproducibility.

2.3.1 Major Tier-1 ISP Dataset

The dataset comprises authentic, real-world network traffic captured at the ingress routers of a major tier-1 ISP. For evaluation, one hour of recorded network traffic was used on each of ten consecutive days in February 2025 [KLZ25]. For traffic image-based DDoS detection (Chapter 4), the traffic image monitoring was integrated into the existing operational network monitoring infrastructure to assess its applicability with respect to scalability under large traffic volumes and high data rates. Due to privacy and legal constraints, the dataset cannot be publicly released. Nevertheless, it enables the evaluation of the proposed approaches under realistic operational conditions and provides insights into their applicability. No statements regarding traffic distributions, protocol compositions, or traffic volumes can be disclosed. Despite these limitations, the dataset is highly valuable because it reflects authentic traffic from a large-scale tier-1 ISP network.

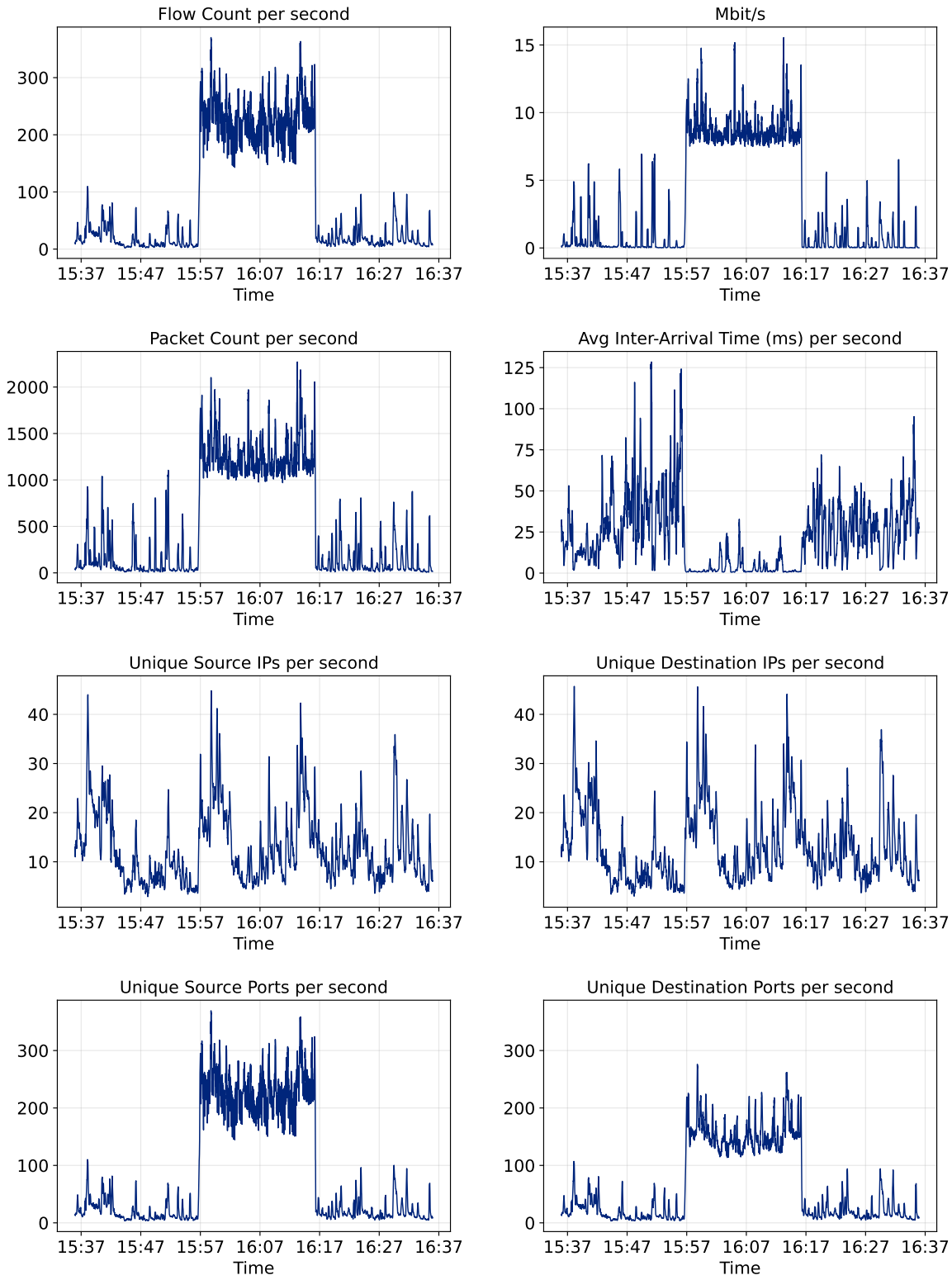


Figure 2.1: *Traffic statistic insights into the CIC-IDS 2017 dataset.*

2.3.2 The CIC-IDS 2017 Dataset

The CIC-IDS2017 dataset [SHG18] was designed to facilitate the evaluation of Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) by providing realistic and contemporary network traffic traces. It has been widely adopted within the research community due to its comprehensive coverage of various attack vectors, such as port scans, Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks. The dataset includes labeled microflows that are generated using the custom tool CICFlowMeter and contain metadata such as timestamps, protocol information, and flow identifiers. In addition, it provides raw packet capture (PCAP) traces, which are particularly relevant for the approaches evaluated in this dissertation, as traffic processing is performed on packet streams. Although certain limitations are known, such as ”missing and redundant data records” [TL20], ”the misimplementation of the DoS Hulk attack, the misunderstanding of the TCP protocol in flow construction and errors in feature extraction by the flawed CICFlowMeter tool” [ERJ21; Ros+22], these issues do not impact the processing of the raw PCAP traces. Consequently, the dataset remains a valuable resource for the evaluation regarding the comparability to other approaches.
Figure 2.1 presents statistical characteristics of the subset of the CIC-IDS 2017 dataset that contains a DDoS attack. This attack was executed on Friday, July 7, 2017, between 15:56 and 16:16 o’clock, and corresponds to a volumetric DDoS attack originating from three distinct sources targeting a single victim. However, due to the application of Network Address Translation (NAT) within the dataset, only a single attack source IP address appears in the recorded trace. As a result, the attack is not evident when observing unique source IP counts but becomes clearly distinguishable through other metrics, such as the data rate and the packet count. This scenario exemplifies cases where the number of attack sources is relatively small compared to the number of legitimate sources in the background traffic, a situation that can be encountered in ISP networks.
For the experiments conducted in this work, only the aforementioned subset is used, comprising 20 minutes of legitimate traffic preceding the attack, the traffic during the attack, and 20 minutes of legitimate traffic succeeding the attack. The following list summarizes why the CIC-IDS 2017 dataset is particularly suitable for evaluation:

- Widely adopted by the research community ensuring comparability
- Includes raw PCAP traces, which are essential for this dissertation’s traffic processing that is performed on packet streams
- Contains detailed attack metadata (start/end times, source and victim IPs), enabling precise labeling without relying on flow data of the custom CICFlowMeter tool
- Despite known issues such as incorrect timestamps and duplicate flow identifiers, there is no impact on PCAP-based processing.

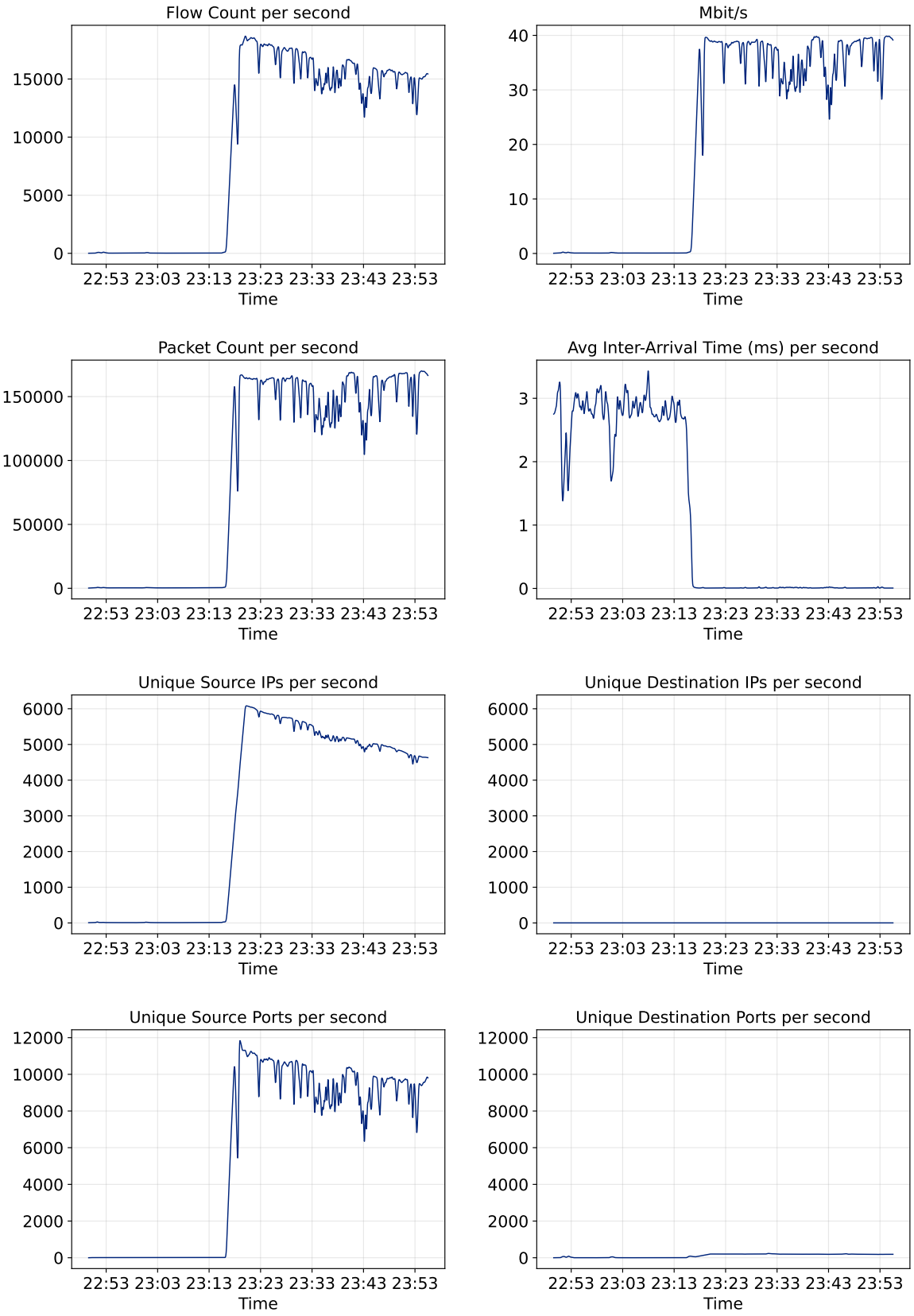


Figure 2.2: *Traffic statistic insights into the CAIDA dataset.*

2.3.3 The CAIDA Dataset

”The Center for Applied Internet Data Analysis (CAIDA) conducts network research and builds research infrastructure to support large-scale data collection, curation, and data distribution to the scientific research community.” [Cen25]

The CAIDA DDoS 2007 dataset [Cen07] captures a one-hour traffic trace of a large-scale, real-world DDoS attack directed against a single victim. Figure 2.2 illustrates the statistical properties of the dataset relevant for this evaluation. The attack originates from thousands of distributed sources, with the primary attack vector being an ICMP flood, resulting in a high packet rate toward the targeted host. Note that the flow count per second does not only comprise quintuple microflows but also includes source-to-destination IP flows in the case of ICMP traffic.

The DDoS attack begins after 20 minutes of the recording and persists for the remaining 40 minutes of the trace. During the attack phase, the traffic directed toward the victim exceeds 150,000 packets per second and has a peak unique source IP address count of 6,000 per second. For the analysis in this work, only the victim-directed traffic is considered.

This dataset is particularly valuable because it represents Internet-scale DDoS traffic, which is otherwise difficult to obtain due to privacy, legal, and operational constraints. Moreover, the dataset is available in raw PCAP format, allowing custom preprocessing tailored for this dissertation’s evaluation needs.
An additional advantage of the dataset is its flexibility for integration into other traffic recordings. Since the attack traffic is provided as raw PCAP traces, it can be seamlessly injected into existing datasets also available in PCAP format, enabling the creation of mixed traffic scenarios that combine legitimate and attack traffic. This allows for the controlled evaluation of detection approaches under more complex and realistic conditions, such as varying background traffic over time (e.g., multiple days).

The CAIDA dataset is suitable for use in the evaluation for the following reasons:

- Contains an authentic large-scale DDoS attack captured in the Internet backbone
- Features a high number of distributed attack sources, representative of Internet-scale threats
- Provides a long attack period (40-minutes sustained attack traffic) with attack traffic dynamics
- Available as raw PCAP traces, enabling direct packet-level analysis and processing

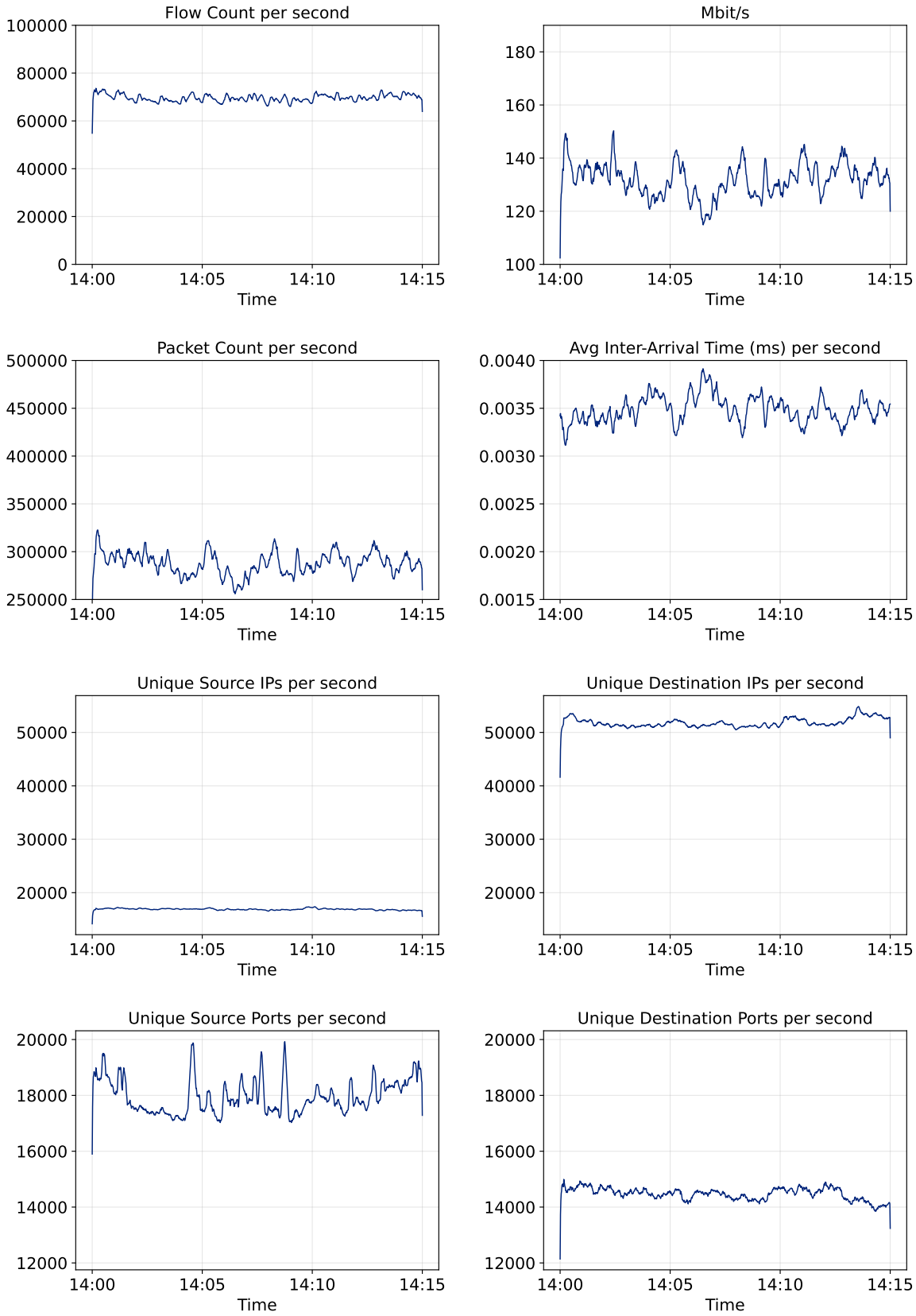


Figure 2.3: *Traffic statistic insights into the MAWI dataset.*

2.3.4 The MAWI Datasets

”The Measurement and Analysis on the WIDE Internet (MAWI) Working Group is a working group that has carried out network traffic measurement, analysis, evaluation, and verification from the beginning of the WIDE Project.” [Mea25]

The MAWI traffic archive provides daily PCAP traces collected from an Internet backbone link in Japan. Each trace covers a 15-minute interval between 14:00 and 14:15 and has been recorded continuously since 2006. In this work, the daily traces from July 1, 2025, to July 10, 2025, are used and are assumed to contain only legitimate background traffic. Figure 2.3 illustrates the statistical properties of the trace from July 1, 2025, which contains approximately unique 70,000 microflows per second and exhibits an average packet rate of around 300,000 packets per second.

The MAWI datasets serve as legitimate background traffic for constructing Internet-scale attack scenarios. Specifically, they are used as the legitimate traffic component into which the CAIDA DDoS 2007 attack trace is injected, thereby creating a realistic mixed traffic trace suitable for large-scale detection evaluation.

Using consecutive days of traffic collected from the same monitoring location is particularly beneficial for assessing the generalization capability of ML approaches in DDoS detection. By injecting the same attack trace into multiple daily snapshots of legitimate traffic, it is possible to evaluate whether the detection performance remains consistent despite natural variations in background traffic patterns. This approach enables testing the robustness of a trained ML model across different but related traffic conditions, providing stronger evidence of its applicability in real-world ISP environments.

The MAWI datasets are particularly suitable for evaluation for the following reasons:

- Provide real Internet backbone traffic as PCAP traces enabling traffic aggregation and injection of the CAIDA DDoS attack trace.
- The datasets’ scale and diversity reflect Internet-grade conditions, offering a realistic background for validating large-scale detection mechanisms.
- The continuous availability of daily traces enables the evaluation of generalization capabilities across different traffic conditions.

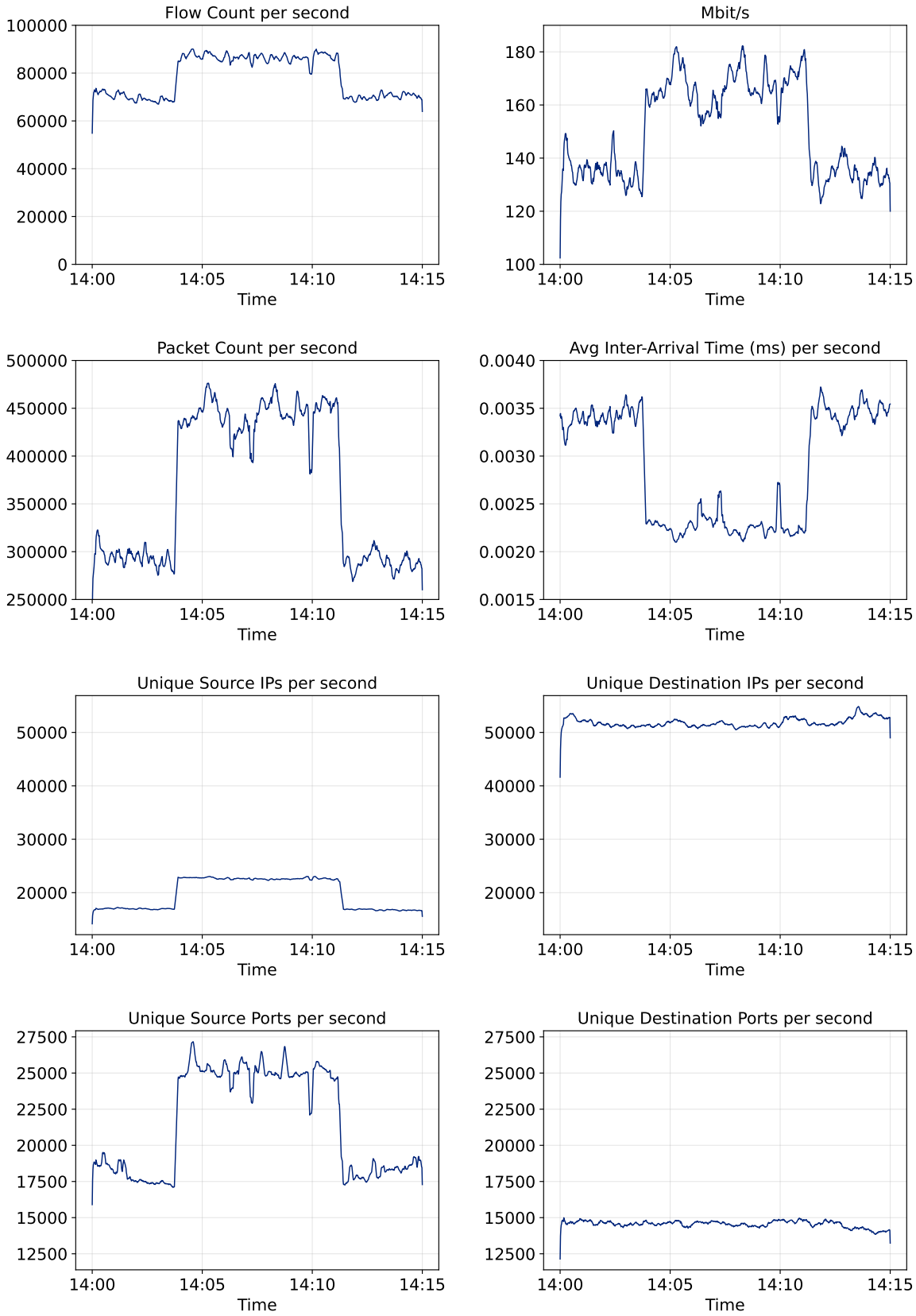


Figure 2.4: *Traffic statistic insights into the MAWI+CAIDA dataset.*

2.3.5 The MAWI+CAIDA Dataset

A self-crafted evaluation dataset is created by injecting the CAIDA DDoS 2007 traffic trace into MAWI backbone traffic. This combined dataset merges the previously introduced CAIDA DDoS attack with the MAWI traffic, resulting in a realistic Internet-grade traffic scenario. Figure 2.4 illustrates the statistical properties of the resulting MAWI+CAIDA dataset exemplarily for July 1st.

15 minutes of MAWI backbone traffic are used as the legitimate baseline. In the middle of each trace, a 7.5-minute segment cut from the CAIDA DDoS attack is injected, thereby preserving both the temporal structure of the legitimate background and the intensity characteristics of the original attack. This approach ensures that the resulting mixed traffic closely resembles operational network conditions. Furthermore, the injection of attack traffic for half of the trace's duration allows the creation of a balanced dataset for later evaluation.

This MAWI+CAIDA dataset is particularly suitable for evaluation due to the following reasons:

- It combines real-world backbone traffic with an authentic, large-scale DDoS attack, maintaining realistic Internet traffic dynamics.
- The injection of the same attack into different MAWI traces enables controlled experiments with naturally varying background traffic.
- It allows reproducible testing of detection generalization across multiple days from the same monitoring location.
- The dataset preserves packet-level fidelity through PCAP traces, enabling precise temporal and volumetric analysis.
- The resulting traffic mix reflects ISP-scale challenges, such as high traffic diversity and variability in legitimate flow patterns.

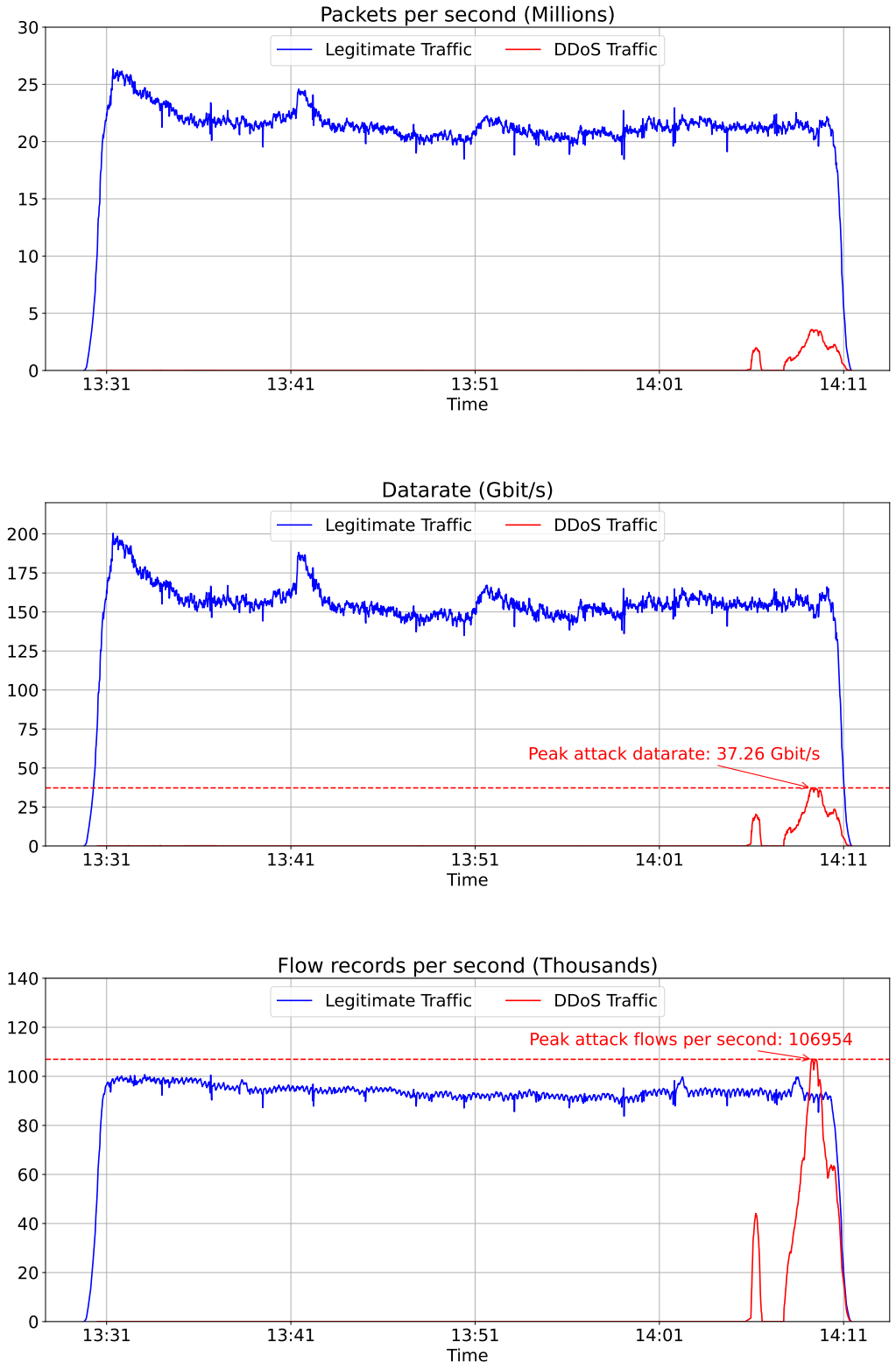


Figure 2.5: *Traffic statistic insights into the BelWue dataset.*

2.3.6 The BelWue Dataset

The BelWue is ”Baden-Württembergs extended LAN” [Bel], which is the data network of the scientific institutions of the State of Baden-Württemberg (Germany). It connects universities, universities of applied sciences, as well as other scientific and public institutions, schools, and public libraries with one another.

The BelWue dataset is a 42-minute export of NetFlow records from April 13th, 2025. The netflow records were collected from the border routers of the BelWue network and anonymized using **Crypto**graphy-based **P**refix-preserving **An**onymization (CryptoPAn) [Xu+01; Xu+02]. During the recording period, a volumetric DDoS attack (UDP flood) targeting a single victim within the network occurred. The attack was detected through a traffic volume threshold-based alerting system and confirmed by the network operators of BelWue. According to the identified destination IP address of the attack, the flow records are labeled as legitimate or attack traffic during the attack period.

Figure 2.5 illustrates the packets per second, the data rate per second, and number of flows records per second. The blue lines indicate the statistics for the legitimate traffic, while the red lines represent the traffic directed toward the victim during the attack period. The legitimate traffic comprises over 20 million packets per second in average, over 150 Gbit/s data rate in average, and over 90 thousand flow records per second in average. The attack traffic is present during the last five minutes of the recording and consists of two attack waves. The first attack wave peaks at an approximate data rate of 20 Gbit/s comprising more than 40 thousand flow records per second. The second attack wave peaks at an approximate data rate of 37 Gbit/s comprising more than 100 thousand flow records per second.

The BelWue dataset is particularly valuable for the evaluation as it provides authentic network traffic from a regional ISP network, including both legitimate and attack traffic. In contrast to the MAWI+CAIDA dataset, where the attack traffic and the legitimate traffic originate from different networks and time periods, the legitimate and attack traffic of the BelWue dataset were recorded simultaneously in a real-world scenario.

Furthermore, the attack recorded in the BelWue dataset is highly distributed attack, e.g., in contrast to the CIC-IDS 2017 dataset, and contemporary as it occurred in 2025, e.g., in contrast to the CAIDA DDoS 2007 dataset.

2.4 Methodology

This section presents the experimental methodology employed in this dissertation. It provides a detailed description of the ML model training and hardware configuration used in the evaluation.

2.4.1 ML Model Training

This section covers the training methodology for the ML models employed for supervised learning tasks throughout this dissertation. It outlines data processing steps, such as dataset splitting and normalization, and the training procedure.

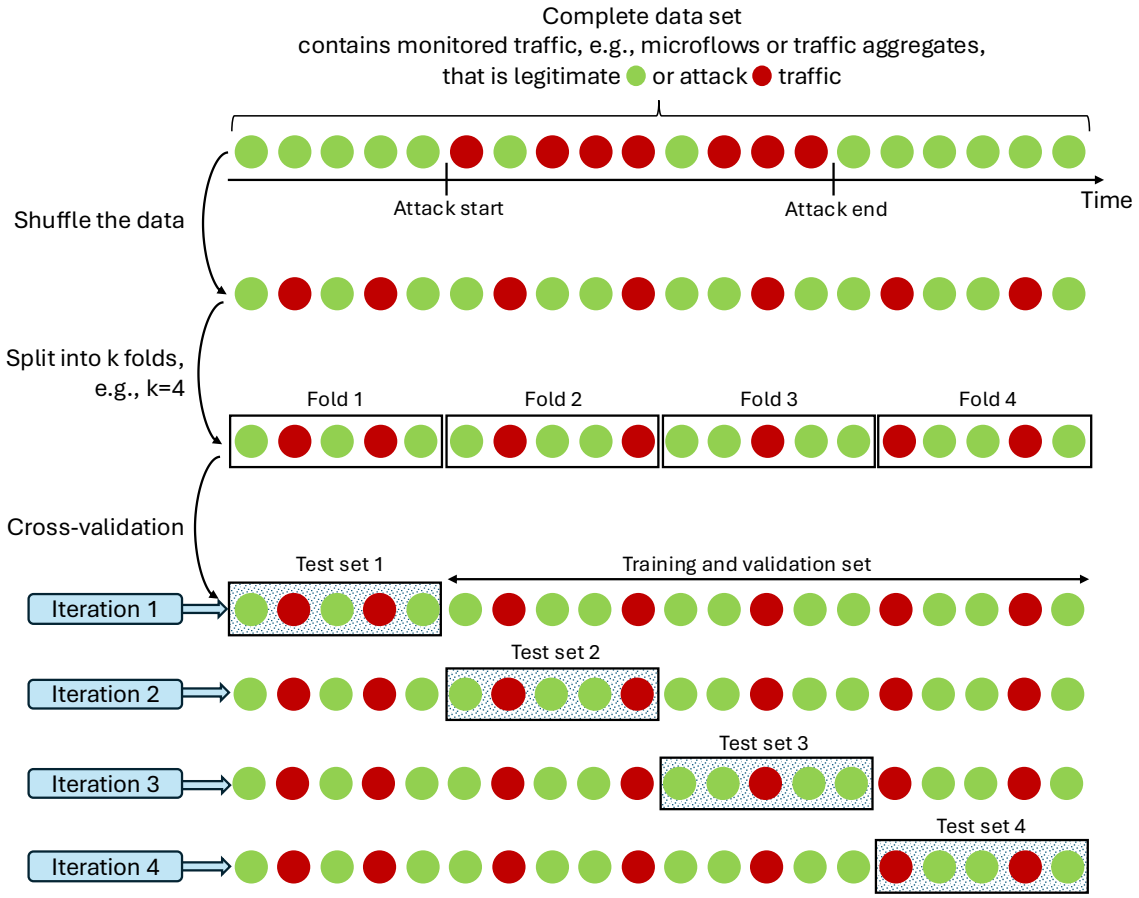


Figure 2.6: *K-fold cross-validation procedure.*

Training Procedure
For supervised ML, it is important to avoid that the data used for model evaluation has been used for training the model. During the training the model learns patterns of the

training data. If the same data is used for evaluation, the model might simply memorize the training data instead of learning generalizable patterns. Therefore, the dataset is split into three disjoint subsets, i.e., the training, validation, and test set. The training set is exclusively used for model training, while the validation set is used for hyperparameter tuning and stop criteria, such as early stopping, i.e., stopping training when the validation performance does not improve for a certain number of training steps. The test set serves as an independent evaluation set of the resulting trained model. Furthermore, the training data is normalized using z-score normalization, i.e., each feature is transformed to have a mean of 0 and a standard deviation of 1. This is achieved by subtracting the mean μ of the data (feature-wise) and dividing by the standard deviation σ:

$$z = \frac{(x - \mu)}{\sigma} \tag{2.6}$$

This avoids that features with larger absolute values dominate the training process. The normalization parameters μ and σ are computed solely on the training set and applied to the validation and test set, which avoids that the model has implicit knowledge about the validation and test set through normalization.
To ensure that every sample of the dataset is used for evaluation and to avoid that easily classifiable samples form the majority of the test data, k-fold cross-validation is employed (see Figure 2.6) per default, if not stated otherwise, e.g., for time-series classification, where shuffling the data breaks the temporal order. The dataset is split into k equally sized folds (disjoint subsets) and the training follows k iterations. In each iteration, one of the k folds is used as the test set, while the remaining k-1 folds build the training and validation set. In this work, 10-fold cross-validation is applied and the training and validation set follows an 90/10 split. The 90/10 split is chosen as the utilized real-world datasets are small, i.e., contain at most a few thousand samples, ensuring that the model receives enough training data (e.g., compared to an 80/20 split) while still maintaining a validation set.

		Predicted Class	
	All samples	Attack	Legitimate
Actual Class	Attack	True Positive (TP)	False Negative (FN)
	Legitimate	False Positive (FP)	True Negative (TN)

Figure 2.7: *Confusion matrix for binary classification.*

2.4.2 Model Evaluation

This section covers the evaluation methodology for trained ML models throughout this dissertation. The ML models that perform binary classification of data samples, i.e., microflows or traffic aggregates, classify them as either legitimate or attack. Therefore, the data has two classes annotated with a binary label, i.e., 0 for legitimate and 1 for attack traffic. Classification results can be categorized into four outcomes:

- True Positive (TP): An attack sample correctly classified as attack.
- True Negative (TN): A legitimate sample correctly classified as legitimate.
- False Positive (FP): A legitimate sample incorrectly classified as attack.
- False Negative (FN): An attack sample incorrectly classified as legitimate.

From these outcomes, illustrated in Figure 2.7 as a *confusion matrix*, three evaluation metrics are derived, namely precision, recall, and accuracy. The precision indicates the proportion of correctly identified attack samples among all samples classified as attack. For example, the test set contains 100 attack samples and 100 legitimate samples. If the model classifies 90 attack samples correctly as attack (TP) and all other samples (10 attack and 100 legitimate) as legitimate (FN and TN), the achieved precision is 100%, as all samples classified as attack are indeed attack samples.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.7}$$

The recall indicates the fraction of identified attack samples among all attack samples. For example, the test set contains 100 attack samples and 100 legitimate samples, and the model classifies all samples as attack. The achieved recall is 100%, as all attack samples are correctly identified. However, the precision is only 50%, as only half of the samples classified as attack are indeed attack samples.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.8}$$

Therefore, the goal for an ML model is to achieve high precision and high recall.
The accuracy indicates the proportion of correctly classified samples among all samples, independent of their class.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.9}$$

However, the accuracy is not a suitable metric for imbalanced datasets, i.e., when one class is significantly more frequent than the other. For example, the test set contains 99

legitimate samples and 1 attack sample. If the model classifies all samples as legitimate, the achieved accuracy is 99%, although the model fails to identify any attack sample, resulting in a recall of 0%. As most of the used datasets in this work are balanced, the accuracy is reported in addition to precision and recall. The F1 score, which is the harmonic mean of precision and recall, is not reported separately as it can be calculated directly from these two metrics.

Algorithm 1: Permutation Feature Importance

Input: Trained model M, test set T with m features, evaluation metric E
Output: $\text{Importance}_1, \ldots, \text{Importance}_m$

$S_{base} \leftarrow E(M, T)$ `// Compute baseline score`
for $a = 1$ *to* m **do**
  $T_{perm} \leftarrow$ copy of T ;
  Randomly permute feature F_a in T_{perm} ;
  $S_{perm} \leftarrow E(M, T_{perm})$;
  $\text{Importance}_a \leftarrow S_{base} - S_{perm}$;
return $Importance_1, \ldots, Importance_m$

2.4.3 Permutation Feature Importance

Each sample of the training, validation, and test set is represented by a feature set, i.e., a vector of numerical values, e.g., average packet size or average packet inter-arrival time. These features are used by the ML model to learn patterns that distinguish between legitimate and attack samples. To assess, which features contribute most to the model's decision (the classification result), the permutation feature importance method is applied. This method takes the model's performance on the test set as a baseline. Then, for each feature, its values are randomly shuffled across all samples in the test set, thereby destroying any relationship between that feature, the other features, and the target label. The model's performance is then re-evaluated on this modified test set and the decrease in performance is measured compared to the baseline. In this dissertation, the permutation feature importance is only calculated for balanced datasets and the decrease in accuracy is used as the performance metric. For example, an ML model achieves perfect classification on the test set with an accuracy of 100%. If the values of a specific feature F_a are randomly shuffled and the model's accuracy drops to 50%, the feature importance of F_a is 50%. This means, that feature F_a is essential for the classification result, as destroying its relationship to the target label decreases the previously perfect performance to random guessing. If the permutation feature importance of a feature F_b is only 1%, permuting it only decreases the accuracy by one percent, indicating that F_b is not very relevant for the classification result.

However, the permutation feature importance goes along with a significant problem. If a feature has a permutation feature importance of 0%, it does not necessarily imply that the feature is irrelevant for the given classification problem in general. It means, that the evaluated ML model does not rely on that feature for its decision, because other features are sufficient for classification. Removing such features from the feature set based on their permutation feature importance can be fatal considering different scenarios, in which the feature might be essential for classification.

Algorithm 2: Iterative Feature Ranking

Input: Trained model M, test set T with m features, evaluation metric E
Output: Features ordered by importance with associated evaluation metric

$F \leftarrow$ list of all features in T ;
$R \leftarrow$ empty list ; `// Stores ranking and accuracy`
while F *is not empty* **do**
  M is trained with features in F ;
  $acc \leftarrow E(M, T)$; `// Evaluate model with` F
  Compute permutation feature importance for features in F using Algorithm 1 ;
  $f^* \leftarrow$ feature in F with highest importance ;
  Append (f^*, acc) to R ;
  Remove f^* from F ;
  Remove f^* from T ;
return R

Feature Ranking
To address the aforementioned problem of permutation feature importance, an iterative feature ranking approach (see Algorithm 2) is employed in this dissertation. It is based on the permutation feature importance and starts with the complete feature set. For each iteration, the permutation feature importance is calculated for all features in the current feature set. The feature with the highest permutation feature importance (accuracy decrease) is removed from the feature set and stored with the overall achieved accuracy. After removing the most important feature, the model is retrained with the remaining features. This process is repeated until no features remain in the feature set.

The feature ranking provides insights into the importance of all features, even if some features show a permutation feature importance of 0%. By removing the most important feature iteratively, the ML model is forced to use the remaining feature set for classification when retrained. As long as the ML model does not degrade in performance, the remaining features are sufficient for the classification task.

The permutation feature importance and the feature ranking are conducted on the test set of each fold during k-fold cross-validation. This results in k results, whose median and interquartile range are reported in the evaluation.

Table 2.1: *Hardware specifications of the experimental server used for evaluation.*

Component	Specification
Processor	AMD EPYC 7413, 24 cores (1 thread/core), base 1.50 GHz, boost up to 3.63 GHz, 128 MB L3 cache, AMD-V virtualization
Memory	256 GB DDR4 ECC Registered, 3200 MT/s (8×32 GB modules)
GPU Accelerators	4× NVIDIA A30 (24 GB VRAM each, 96 GB total), CUDA 12.6, driver 560.35.03
Motherboard	GIGABYTE MZ22-G20-00, BIOS vM14 (2023-08-02)
NUMA Topology	Single NUMA node (24 CPUs)
Operating System	Debian GNU/Linux 11, kernel 5.10.226-1 (x86_64)

2.4.4 Hardware Setup

All experiments were conducted on a high-performance computing server equipped with an AMD EPYC 7413 24-core processor, 256 GB of DDR4 ECC registered memory, and four NVIDIA A30 GPUs with a total of 96 GB of GPU memory. This configuration provides enough computational capacity for training ML models and processing large-scale network traffic datasets efficiently. The use of dedicated, server-grade hardware minimizes performance variability, ensuring consistent experimental conditions. A detailed overview of the hardware configuration is provided in Table 2.1.

2.5 Related Work

This section covers related work in the area of DDoS attack detection. As this dissertation facilitates resource-predictable DDoS detection in ISP networks regarding the required memory utilization for monitoring and the number of required ML classifications for attack detection (independent on the traffic volume or the microflow count), the related work is primarily discussed based on their monitoring paradigms and the number of required ML classifications for attack detection. In addition, problems with existing data labeling approaches for DDoS attack detection are discussed and the suitability of the autoencoder-based labeling approach FeADable is explained.

State-of-the-art DDoS detection and mitigation approaches are Jaqen [Liu+21], Poseidon [Zha+20], ACC-Turbo [Alc+22], and IXPScrubber [Wic+22]. Jaqen and Poseidon (non-ML approaches) are designed for programmable switches and rely on counter- and sketch-based monitoring to collect network traffic statistics. Both approaches enforce mitigation policies derived from known attack signatures, which are expressed as manually predefined rules. For instance, Poseidon mitigates SYN flooding attacks by tracking the number of open TCP connections per source IP at five-second intervals. This is implemented using a count primitive that records the number of observed SYN and ACK packets. Based on the statistics collected during the previous interval, a source IP that exhibits a significantly higher number of SYNs than ACKs is classified as malicious, and its subsequent packets are marked as attack traffic and dropped. Such an approach requires prior knowledge of the attack vector and manual rule specification, making it difficult to maintain in the presence of evolving attack patterns. To alleviate this limitation, Jaqen employs universal sketches, which "make it possible to track a broad range of current and unforeseen metrics with a single algorithm" [Liu+21]. Nevertheless, if newly relevant metrics emerge for attack detection due to novel attack vectors, the corresponding mitigation policies must still be manually designed utilizing the new metrics. This paradigm contradicts ML-based DDoS detection, which aims to avoid manually handcrafted detection rules but learn attack patterns from monitored data instead. However, similarities between Jaqen, Poseidon, and the presented approaches in this dissertation exist:

- Poseidon and Jaqen utilize switch hardware for monitoring, which is also done by the traffic image-based DDoS detection approach (Chapter 4) to match packets at line speed.
- Poseidon uses time windows similar to eMinD (Chapter 3) and the traffic image-based detection (Chapter 4) for traffic monitoring and decision-making, e.g., five-second intervals for SYN flooding defense.

In contrast to Jaqen and Poseidon, the traffic image-based DDoS detection approach (Chapter 4) considers the distribution (prefix-preserving aggregation) of packets in the IP address space to differentiate between legitimate and attack traffic.

ACC-Turbo [Alc+22] introduces the unsupervised machine learning paradigm for DDoS mitigation on programmable switches. Arriving packets are clustered online based on packet header attributes directly on the switch. Depending on the cluster assignment, packets are queued for forwarding with different priorities, where low-priority queues are rate-limited to mitigate potential DDoS attacks. However, the decision of which clusters potentially contain attack traffic is made by a network expert-defined metric, e.g., the throughput or the packet count of the traffic in the cluster. Therefore, ACC-Turbo requires

human expertise in the loop for cluster evaluation.

IXPScrubber [Wic+22] presents a DDoS mitigation service designed for Internet Exchange Points (IXPs). The approach utilizes supervised machine learning and has been trained and evaluated with three months of real-world traffic from five IXPs. IXPScrubber utilizes microflow-based monitoring. However, the monitored microflows are refined into a "feature set that aggregates individual flows to a macroscopic per-target IP perspective" [Wic+22], which is similar to the traffic image columns of the traffic image-based DDoS detection approach. However, microflow-based monitoring requires stored state for each active microflow, which inherently leads to unpredictable memory utilization depending on the microflow count (especially in DDoS attack scenarios with many additional attack sources). Furthermore, one classification is required per target IP address, whose count is also not predictable and results in many ML classifications in an ISP/IXP scenario (more than $100,000$ in the BelWue dataset). In contrast, eMinD and the traffic image-based DDoS detection approach utilize fixed-state monitoring and require only one ML classification per time window, independent of the microflow count. Furthermore, the traffic image-based approach enables the derivation of the destination IP through an iterative multi-class classification [KKZ24b], while still maintaining fixed memory utilization and a fixed number of classifications per time window.

With the rising popularity of the CIC-IDS 2017 dataset [SHG18], numerous ML-based approaches for network attack detection have emerged that utilize this dataset for training and evaluation, e.g., decision tree-based and random forest-based approaches [IJJ24; KJ22], fully connected neural network-based approaches [CNN22; WK18], CNN-based approaches [RS24; Hal+22], hybrid approaches combining multiple ML models for detection [ZKA23; Saj+24], as well as work comparing a plethora of different ML approaches [Mas+21]. However, many of these approaches focus on multi-class microflow classification, i.e., classifying microflows into multiple attack classes (different attack vectors) and legitimate traffic. As the approaches in this dissertation focus on volumetric DDoS attack detection, i.e., binary classification of time windows into attack and legitimate traffic, these multi-class approaches are not discussed. However, Choobdar et al. [CNN22] not only evaluate their ML model regarding multi-class classification but also regarding binary DDoS detection on the CIC-IDS 2017 dataset. In addition, they employ a fully connected neural network for the classification task, which makes their approach suitable for a comparison to eMinD (Section 3.3).

In general, approaches that rely on microflow-based monitoring and classification suffer from the following problems regarding the memory resources, the computational resources, the reaction time, and the available traffic characteristics provided to the ML model: First, microflow-based monitoring leads to unpredictable memory utilization (see

IXPScrubber discussion above). Second, microflow-based approaches require the ML model to perform a separate classification for each active microflow in order to detect ongoing attacks. Consequently, the reaction time of the detector is directly dependent on both the number of active microflows and the classification duration. In DDoS attack scenarios, this dependency can result in unpredictably long reaction times. Third, microflow-level detectors are inherently restricted to five-tuple–based inspection and therefore do not consider distributions of IP addresses or port numbers. As demonstrated in this work, these microflow-overarching traffic attribute distributions constitute key characteristics of distributed attacks.

In contrast, the detection approaches presented in this dissertation perform only a single classification per time window, entirely independent of the number of microflows. By highly aggregating incoming network traffic and expanding the observation scope beyond individual microflows, IP address and port distributions can be monitored and leveraged for model training and inference. This broader visibility of traffic characteristics leads to improved DDoS detection performance, while reducing the average reaction time due to the fixed number of classifications.

2.5.1 Image-based Network Traffic Classification

This section covers related work utilizing image representations for network traffic classification and discusses their applicability to DDoS detection.

Seq2Img [Che+17] transforms packet sequences into 6-channel images using Reproducing Hilbert Kernel Space Embeddings (RHKS). These images are then classified using Convolutional Neural Networks (CNNs) to distinguish between different application-layer protocols, e.g., FTP, HTTP, and SSH. Seq2Img utilizes the first 10 packets of a microflow to obtain features, such as packet size differences and inter-arrival time differences between subsequent packets. This requires unsampled microflow-based monitoring, which contradicts the fixed-state monitoring paradigm followed in this dissertation.

FlowPic [SS19] aims at classifying encrypted network traffic with a CNN into different classes, such as browsing, chat, video, and voice-over-IP. FlowPic creates single-channel images per microflow by mapping individual packets to image pixels based on their size and arrival time, where the x-axis represents the flow duration and the y-axis represents the packet size. This approach also requires unsampled microflow-based monitoring, as it relies on individual packet information. Although the CNN architecture is similar to the architecture employed by this dissertation's traffic image-based DDoS detection approach (Chapter 4), FlowPic is, such as Seq2Img, not applicable to DDoS detection with fixed-state monitoring and a fixed number of classifications.

Agrafiotis et al. [Agr+22] present an approach that converts packet capture (PCAP) files into single-channel images by converting raw packet bytes of individual microflows to images. The CIC-IDS 2017 dataset with its corresponding classes is used for evaluation. The PCAP files are first separated into individual microflows based on the five-tuple definition. Each microflow is then preprocessed by removing IP and MAC addresses to avoid overfitting to specific addresses. After preprocessing, the packets' bytes of each microflow are concatenated and trimmed to a fixed length of 784 bytes (28x28 pixels consuming 1 byte per pixel in grayscale representation). If the microflow contains fewer than 784 bytes, it is padded with zeros. If it contains more, it is truncated. A CNN is then used to classify these images into different classes. Although this approach achieves an accuracy of 99.9% on the complete CIC-IDS 2017 dataset (including the DDoS traffic) performing multi-class classification, it relies on storing all packet bytes (up to 784 bytes) separately per microflow, which also contradicts the fixed-state monitoring paradigm of this dissertation. Furthermore, it is infeasible for deployment in ISP networks due to the large monitored traffic volume (especially during DDoS attacks).

2.5.2 Problems with Existing Data Labeling Approaches

This section discusses problems with existing data labeling approaches for DDoS attack detection and outlines why the autoencoder-based labeling approach FeADable (Chapter 5) is suitable, in contrast to other approaches, for creating labeled datasets for DDoS attack detection.

The first and most straightforward approach for data set labeling is **knowledge-based labeling**. When an attack occurs and the associated traffic is monitored, e.g., the CAIDA DDoS attack recorded in 2007 [Cen07], a human expert analyzes the overall recorded traffic and labels the attack traffic. This typically involves identifying the attack destination based on traffic volume thresholds and manually classifying the attack vector. However, knowledge-based labeling is **prohibitively laborious at large scale deployments**, such as ISP networks, and inherently limited by the expertise of the network expert probably resulting in labeling errors. The IXPScrubber circumvents this problem by utilizing traffic matching blackhole announcements, where "the vast majority of blackholing traffic is DDoS" [Wic+22]. This is a feasible way to create labeled training data, if the possibility of obtaining blackholing traffic exists for the ISP/IXP.

In contrast to knowledge-based labeling, which requires human expertise and manual effort, semi-supervised and unsupervised labeling approaches have been proposed to automate the labeling process.

Semi-supervised labeling approaches [Gu+19; Hou+22; AZ21] assume that labeled training data exists. They utilize the already labeled training data to train an initial ML model in a supervisory manner, which is then "used to classify the unlabeled data" [Hou+22]. The unlabeled data samples that are classified with high confidence are added to the training set, and the model is retrained iteratively until convergence. Assuming initially labeled training data, even if it is only a "small amount of labeled data" [Hou+22], implies that the labeling process is not fully autonomous and requires human expertise.

Unsupervised labeling approaches, such as clustering-based approaches [Bae+17] and novelty detection-based approaches [CBG12], do not require labeled data initially. However, existing unsupervised labeling goes along with crucial limitations. Unsupervised clustering-based approaches [Bae+17; Alc+22] generate groups of data points without guaranteeing cluster purity or identifying the class of the clusters. As a result, human expertise is required for cluster evaluation, e.g., "if a cluster is extremely dense, label it as anomalous" [Bae+17]. Similar cluster evaluation metrics can be observed in ACC-Turbo [Alc+22], such as the throughput and the number of packets per cluster. Requiring human expertise for cluster evaluation contradicts the goal of fully autonomous labeling. Furthermore, clustering is hard in high-dimensional feature spaces. Therefore, dimensionality reduction techniques are often applied before clustering, such as Principal Component Analysis (PCA) or autoencoders. FeADable (Chapter 5) overcomes these limitations by directly employing autoencoder-based anomaly detection. On the other hand, FeADable can only distinguish between legitimate and attack traffic, while clustering-based approaches can potentially identify more than two classes if the resulting clusters are correctly interpreted. If the goal of the labeling is assigning different attack vector classes to the traffic data, FeADable is not suitable.

Catania et al. [CBG12] introduce an unsupervised and autonomous labeling approach based on novelty detection with One-Class Support Vector Machines (OCSVMs). SVM-based novelty detection approaches require the attack traffic to be a small fraction of the overall traffic, as "these algorithms seem to be accurate only when the normal traffic vastly outnumbers the number of attacks present in the dataset." [CBG12]. This requirement is specifically difficult to meet in the context of volumetric DDoS attacks, where the attack traffic volume is large and can exceed the legitimate traffic volume. To overcome this problem, Catania et al. [CBG12] exclude attack traffic from the dataset used for training a OCSVM by removing known attack traffic with the signature-based intrusion

detection system SNORT [Roe+99]. They utilize SNORT to remove attack traffic from the dataset because relying ”on experts for removing known attacks from the training set [...] would be an extremely expensive and tedious task” [CBG12]. However, SNORT’s attack signatures must also be created by network experts suffering from the same problems as knowledge-based labeling, i.e., manual attack signature creation is expensive and novel attack vectors might not be captured. Furthermore, to utilize SNORT for attack traffic removal, the traffic monitoring approach must be compatible with the employed SNORT rules. In contrast, FeADable does not make any assumptions about the traffic monitoring approach and can be applied to different traffic monitoring approaches, such as microflow-based monitoring or traffic image-based monitoring.

To summarize, the autoencoder-based labeling approach FeADable overcomes certain limitations of existing labeling approaches contributing the following benefits:

- No labeled data is required for bootstrapping the labeling process
- The attack traffic is not required to be a small fraction of the overall traffic
- The applied traffic monitoring approach is adaptable, e.g., microflows or traffic images, and not predetermined by the labeling approach itself

Chapter 3

Resource-predictable Time Window-based DDoS Detection

This chapter presents and evaluates *eMinD*, an **e**fficient **M**icroflow-**in**dependent **D**etection pipeline for volumetric DDoS attacks [KZ23; KZ24].

eMinD utilizes a **time window-based monitoring** paradigm (microflow-agnostic) that eliminates per-microflow processing, i.e., microflow segregation, feature extraction, and microflow state maintenance. Time windows represent time periods, e.g., one second each, that condense all packets arriving at the monitoring point into a single fixed set of aggregated packet attributes, i.e., the **traffic aggregate**. Time window-based network traffic monitoring enables constant-time packet processing with fixed memory utilization, independent of traffic volume or concurrent microflow count.

The **eMinD pipeline** refines and classifies traffic aggregates per time window. This design results in one classification per time window and ensures a fixed reaction time.

ML-based classification is employed to **avoid manually handcrafted detection rules**, e.g., threshold-based or signature-based rules, which are expensive to evaluate, and difficult to maintain in network environments with dynamically developing traffic patterns, such as Internet Service Provider (ISP) networks. Instead, manual work is only required for label assignment over time windows. Moreover, the use of ML enables eMinD to identify attack patterns that might not be covered by expert-defined rules.

This chapter outlines that ML-based detection is particularly relevant for volumetric DDoS attack traffic, where precise attack detection emerges from complex feature in-

teractions. For example, combinations of cardinality estimates of IP addresses and port numbers, the transport protocol composition, and inter-arrival time statistics can jointly characterize subtle attacks that are not detectable through thresholding single features. Crafting explicit detection rules that capture the nuanced interplay of traffic patterns in a dynamic, high-volume environment, such as ISP networks, is impractical and error-prone. In contrast, ML models can efficiently learn and generalize from multi-dimensional feature relationships, enabling reliable detection of attacks in scenarios where traditional rule-based systems may fail.

This chapter addresses **Research Question RQ1**, namely whether volumetric DDoS attacks can be detected with microflow-independent monitoring and classification while maintaining fixed computational and memory resources. The proposed approach resolves two key limitations of traditional microflow-based detection. First, it overcomes the issue of **unpredictable and potentially long reaction times (P2)** caused by the need to classify thousands of individual microflows before an attack can be recognized. Second, it mitigates the loss of **detection-relevant information (P3)** by incorporating microflow-overarching traffic characteristics, such as IP address, protocol, and port distributions, which are essential for identifying DDoS attack patterns. By addressing these problems, eMinD demonstrates that resource-efficient detection (regarding the required memory and the number of ML classifications) is achievable without sacrificing classification performance.
To summarize, this chapter contains the following key contributions:

- **Time window-based traffic monitoring** that eliminates per-microflow processing
- Fixed-state **traffic aggregates** facilitating microflow-overarching feature sets
- **eMinD pipeline** with fixed reaction time
- **Detection performance** and **generalization** evaluation with real-world datasets
- A **comparison to microflow-based approaches** with respect to memory utilization, the classification performance, and the reaction time
- **Feature importance analysis** outlining the benefit of microflow-agnostic features for DDoS detection and the benefit of ML for capturing complex feature interactions

The remainder of this chapter is structured as follows:
Section 3.1 describes the **functionality of eMinD**, outlining its core architectural components and the time window-based (microflow-independent) monitoring approach. Section 3.2 discusses the **impact of eMinD's system parameters** on the classification performance, the resource consumption, and the reaction time. Section 3.3 provides a

comprehensive **performance evaluation and analysis of trade-offs**, including memory utilization, reaction time, and classification accuracy. It further compares eMinD to a microflow-based approach, shows the importance of ML for DDoS detection through **feature importance analysis**, and eMinD's compatibility with different ML models. Finally, Section 3.4 **concludes the chapter** with a summary of findings.

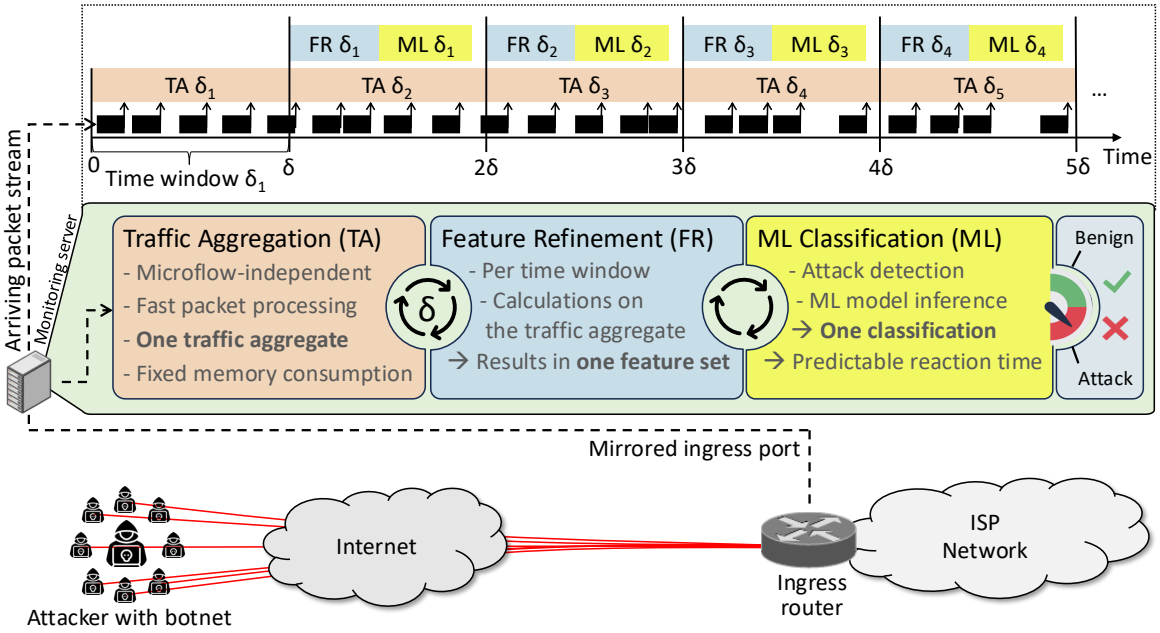


Figure 3.1: *High-level overview of eMinD's architecture and individual pipeline steps (duplicate of Figure 1.1).*

3.1 The eMinD Pipeline

This section introduces the functionality of *eMinD* [KZ23; KZ24], an **e**fficient **M**icroflow-**in**dependent **D**etection pipeline for volumetric DDoS attacks. The pipeline (see Figure 3.1) operates on time windows of duration δ, e.g., 1 second.

Definition 3.1: Time Window

A time window δ_t is defined as a fixed-duration time interval $[t \cdot \delta - \delta, t \cdot \delta)$, where $t \in \mathbb{N}$ enumerates time windows from the monitoring start and δ represents the time window's duration.
Time windows are consecutive and non-overlapping, such that δ_{t+1} is the fixed-duration interval immediately following δ_t, i.e., $\delta_{t+1} = [t \cdot \delta, (t+1) \cdot \delta)$.

Based on time windows, eMinD comprises three stages:

(1) **Traffic aggregation** (per packet) – Continuous monitoring of the entire packet stream to condense all packets arriving within a time window into a fixed-size set of aggregated packet attributes, i.e., the *traffic aggregate*. This stage outputs one traffic aggregate per time window.

(2) **Feature refinement** (per time window) – Transformation of traffic aggregates from the previous stage into a fixed-size feature vector, i.e., one feature set, suitable for ML classification. This stage outputs one feature set per time window.

(3) **Classification** (per time window) – Application of ML models to the refined feature sets for binary classification, i.e., determining whether a DDoS attack is ongoing or not. This results in one classification per time window and a fixed reaction time, i.e., the time from the start of an attack until its detection, bounded by 2δ.

eMinD is designed for deployment on a *server collocated with an ingress router of an ISP network*, where it processes a *mirrored copy of ingress traffic*. This requires installed hardware at each desired ingress router.

Table 3.1: *Overview of eMinD's monitored packet attributes, their aggregation method, and the resulting feature set.*

Traffic Aggregation		**Feature Refinement**	**Feature Set**
Method	**Packet Attributes**		
Hash	Source IP	Cardinality Estimation	Source IP cardinality
	Destination IP		Destination IP cardinality
	Source port		Source port cardinality
	Destination port		Destination port cardinality
Count	TCP packets	Identity	TCP packet count
	UDP packets		UDP packet count
	Total packets		Total packet count
	TCP SYN flags		TCP SYN flag count
Sum	Packet size	Mean	Mean packet size
	Inter-arrival time		Mean inter-arrival time
			Feature set size: 10

3.1.1 Traffic Aggregation and Feature Refinement

During the traffic aggregation stage (compare Figure 3.1), **eMinD** monitors a fixed set of *packet attributes* (see Table 3.1) across the entire ingress packet stream. Upon packet arrival, each packet attribute is processed according to one of three aggregation methods: Hash aggregation, count aggregation, or sum aggregation. Respectively, the results of these aggregation methods after each time window of duration δ are hash-aggregates, count-aggregates, and sum-aggregates. They collectively form the **traffic aggregate** for a time window, consisting of four hash-aggregates, four count-aggregates, and two sum-aggregates. Aggregation operations are implemented using computationally lightweight primitives, including integer addition, cryptographic hashing, and bitwise manipulations, thereby achieving **constant-time update performance** per packet. This high computational efficiency is crucial for sustaining per-packet processing in network environments with large traffic volumes, particularly under *DDoS attack conditions*, where packet arrival rates may exceed millions of packets per second. Furthermore, eMinD guarantees **constant memory utilization**, ensuring that memory utilization remains independent of the overall traffic volume, total packet count, and the number of concurrently active microflows.
During feature refinement, three refinement methods are applied: Hash-aggregates are transformed into cardinality estimation features, count-aggregates into identity features, and sum-aggregates into mean features, which results in a fixed-size set of 10 features per time window. These three pairings of aggregation and refinement methods (see Table 3.1) are explained in the following.

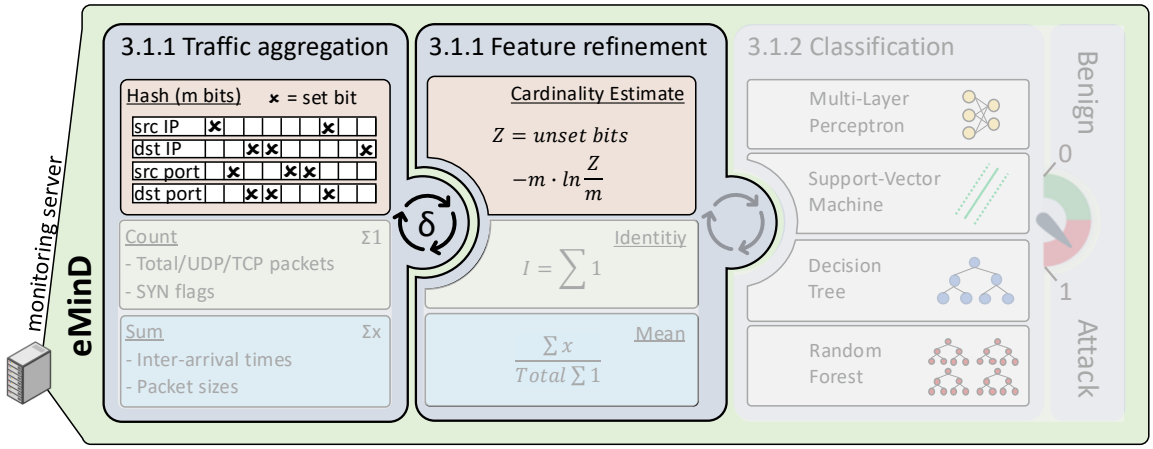


Figure 3.2: *Overview of the interplay between hash-aggregates and cardinality estimation feature refinement, whose calculation is triggered after each time window.*

Hash-Aggregates and Cardinality Estimation Features. The interplay between the hash aggregation method and the cardinality estimation feature refinement method

is illustrated in Figure 3.2. Hash-aggregates provide data for probabilistic cardinality estimates of distinct packet attributes within a packet stream. In eMinD, hash-aggregates are employed for quantifying the number of distinct *source and destination IP addresses and port numbers*. Hash-aggregates employ a uniform hash function to map the packet attributes onto a bit array of size m, which is updated upon each packet arrival by setting the array to 1 at the index of corresponding bit, i.e., the hash value. This facilitates constant-time updates and memory-efficient storage.

Definition 3.2: Hash-Aggregates

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a sequence of packets from the observed packet stream, and let $\mathscr{A}$ be the set of all possible packet attributes.
For a specific packet attribute type $a \in \mathscr{A}$, let $S_a = \{a(p_1), a(p_2), \ldots, a(p_n)\}$ be the set of attribute values extracted from the packet sequence.
Let $h : \mathscr{A} \to \{0, 1, \ldots, m-1\}$ be a uniform hash function. Define a bit array $B \in \{0,1\}^m$ of size m where each attribute value $a(p_i) \in S_a$ sets $B[h(a(p_i))] \leftarrow 1$.
The hash-aggregate for packet attribute a is the resulting bit array B after processing all packets in the stream for a given time window.

The design of hash-aggregates extends the probabilistic membership testing capabilities of Bloom Filters [Blo70] (details in Section 2.2) to support fast cardinality estimation for network traffic analysis. Bloom filters have been successfully deployed in time-critical networking applications such as longest prefix matching [DKT03], demonstrating their suitability for high-performance packet processing tasks.

Cardinality estimation feature refinement transforms the probabilistic bit arrays generated through hash aggregation into quantitative cardinality estimates of packet attributes. It applies the Linear Counting formula [WVZT90] (see Sec. 2.2) to convert the binary occupancy patterns in hash-aggregate bit arrays into numerical cardinality approximations.

Definition 3.3: Cardinality Estimation Features

Let $B \in \{0,1\}^m$ be the bit array resulting from a hash-aggregate over a packet sequence $P = \{p_1, p_2, \ldots, p_n\}$. Let X denote the number of bits in B that remain unset (i.e., equal to 0) after processing all attribute values in P. The estimated cardinality of distinct attribute values $\hat{n}$ in P using the Linear Counting formula is:

$$\hat{n} = -m \cdot \ln\left(\frac{X}{m}\right)$$

where m is the total size of the bit array and $\frac{X}{m}$ represents the fraction of unset bits.

The Linear Counting formula provides significantly improved accuracy compared to the simple bit-counting approach (introduced in Section 2.2). While the simple approach [KZ23; KZ24] would approximate cardinality as the number of set bits in the array, this approach suffers from substantial bias and inaccuracy due to hash collisions. Linear Counting mitigates these issues by leveraging the statistical relationship between the fraction of unset bits and the true cardinality, under the assumption of a uniform hash distribution.
Applying the linear counting formula is *not computationally expensive* in this context. Feature refinement is not performed per packet but per time window. Consequently, the logarithm is evaluated only once per time window, resulting in negligible computational overhead compared to the cost of per-packet processing.

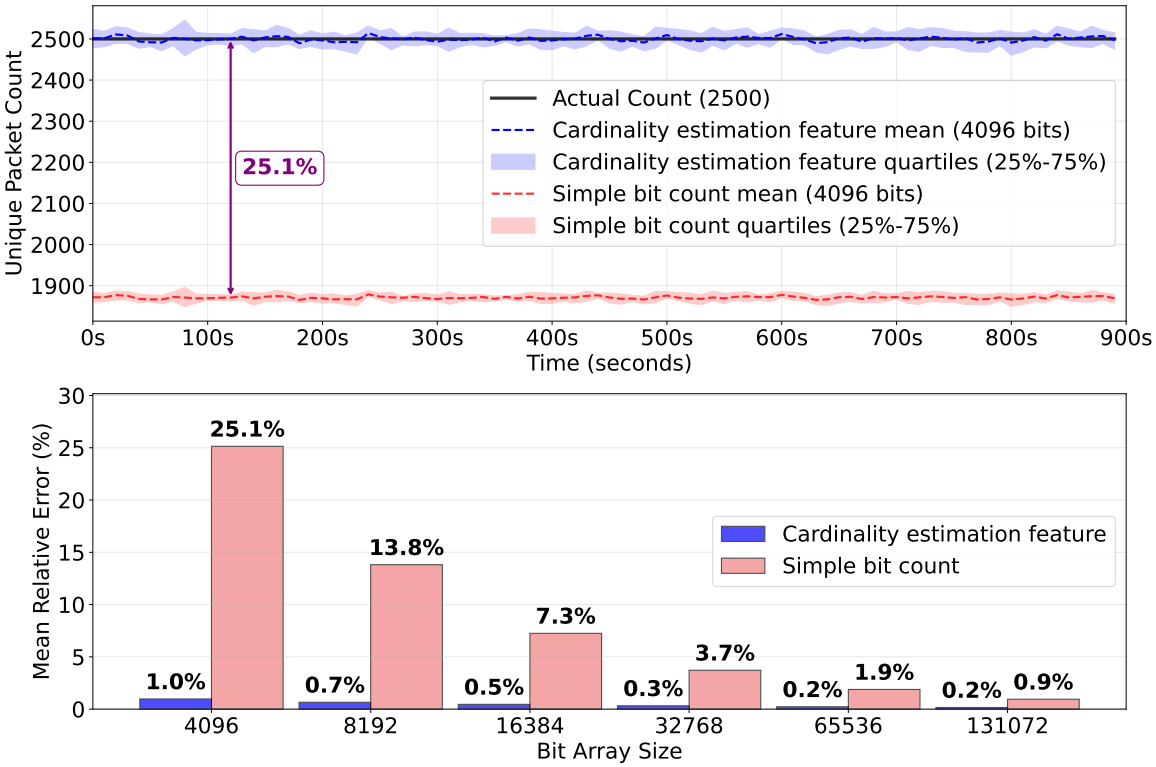


Figure 3.3: *Cardinality estimation visualization and comparison to simple bit counting.*

Figure 3.3 compares simple bit counting with the Linear Counting approach for cardinality estimation. The illustrated results stem from an experiment that simulated 900 seconds of traffic, generating 2,500 random source IP addresses per second. The top part of the figure shows the actual cardinality over time alongside estimates from simple bit counting and the cardinality estimation feature (Linear Counting), both based on a bit array of size 4,096. Each experiment was repeated 100 times, and the mean with interquartile range of the estimations is shown.

For a bit array size of 4,096, larger than the expected number of distinct source IP addresses, Linear Counting yields highly accurate cardinality estimates, whereas simple bit counting underestimates the true cardinality by 25.1%.
The bottom part of the figure illustrates estimation accuracy for different bit array sizes. While both methods improve with larger arrays, simple bit counting requires 131,072 bits, *32 times more memory* than Linear Counting with a 4,096-bit array, to achieve a comparable level of accuracy.
The choice of the bit array size m is a crucial design decision that impacts the accuracy of cardinality estimates and memory efficiency of hash-aggregates. A larger bit array reduces the probability of hash collisions, thereby improving the accuracy of cardinality estimates. However, it also increases memory utilization, necessitating a trade-off between accuracy and resource utilization. The choice depends on the expected number of distinct attribute values and the desired estimation accuracy, which is mainly influenced by the location of deployment and the expected traffic patterns (later shown in the evaluation).

Relevance of Cardinality Estimation Features for DDoS Detection.
The accurate characterization of IP address and port number distributions constitutes a fundamental requirement for effective DDoS detection, given that DDoS attacks exhibit characteristic patterns of high source diversity coupled with concentrated destination targeting. Consequently, the detection of anomalous cardinality increases in source IP addresses or port numbers while destination cardinalities remain stable serves as a reliable indicator for identifying potential DDoS attacks.

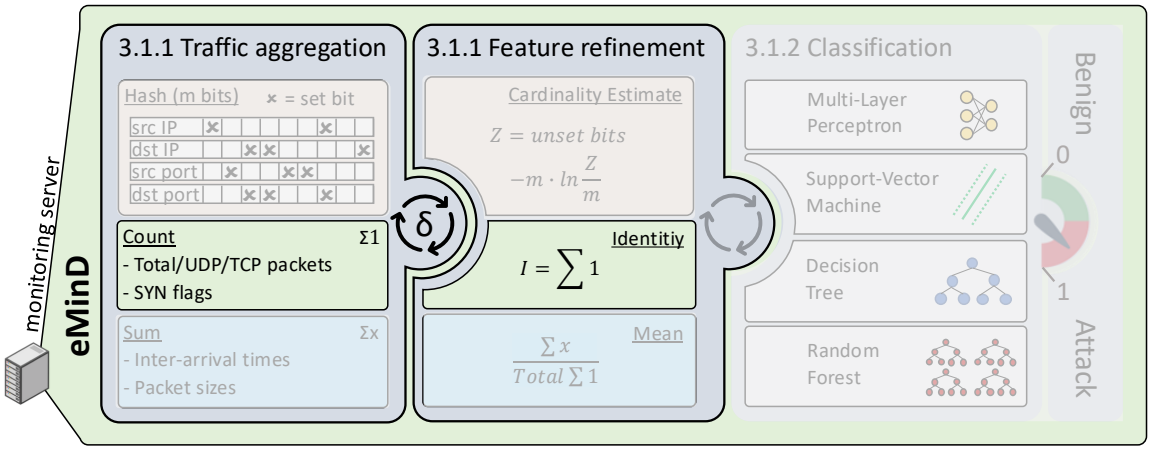


Figure 3.4: *Overview of the interplay between count-aggregates and identity feature refinement, whose calculation is triggered after each time window.*

Count Aggregates and Identity Features. Count aggregates measure the frequency of packet attributes over a sequence of packets from the observed packet stream. The measured attributes are the TCP and UDP packets, the total packet count, and TCP packets

with the SYN control flag. Each count-aggregate is defined by an indicator applied to its corresponding packet attribute, i.e., a Boolean-valued function that indicates whether a packet satisfies the measured attribute condition, e.g., *is TCP?* or *is SYN flag set?*. The predicate for the total packet count always evaluates to true.

Definition 3.4: Count-Aggregates

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a sequence of packets from the observed packet stream, and let $f_i : p \rightarrow \{0,1\}$ be a Boolean-valued indicator function for the count-aggregate i, that evaluates whether packet p satisfies the corresponding attribute condition. The result c_i of count-aggregate i with the indicator f_i is defined as:

$$c_i = \sum_{j=1}^{n} f_i(p_j)$$

where $f_i(p_j)$ evaluates to 1 if packet p_j satisfies the condition for count-aggregate i, and 0 otherwise.

Identity Features represent the absolute count of selected packet attributes within a given time window, i.e., the result of the corresponding count-aggregates (see Figure 3.4). They perform a *pass through* of the count-aggregate results to the classification stage without further transformation. Importantly, the packet attributes used in identity features are not necessarily mutually exclusive. For instance, the total packet count includes both TCP and UDP packets, leading to overlapping counts among features.

Definition 3.5: Identity Features

Let $\{c_1, c_2, c_3, c_4\}$ be the result of the count aggregates for the packet attributes. The identity feature associated with each attribute is defined directly as:

$$\phi(c_i) = c_i$$

where c_i is the count aggregate result.

Relevance of Identity Features (Count-Aggregates) for DDoS Detection.
Identity features, i.e., the values of count-aggregates, complement cardinality estimation features by capturing packet attribute distributions crucial for detecting deviations of DDoS attack traffic from legitimate traffic. In large-scale networks with high legitimate traffic volume, volumetric DDoS attacks can represent only a small fraction of the overall traffic. Therefore, volumetric DDoS attacks cannot be reliably identified solely through cardinality estimation features. Count-based metrics are essential to distinguish attack-

induced traffic volume spikes (total count) or protocol distribution shifts, e.g., through a UDP reflection attack.

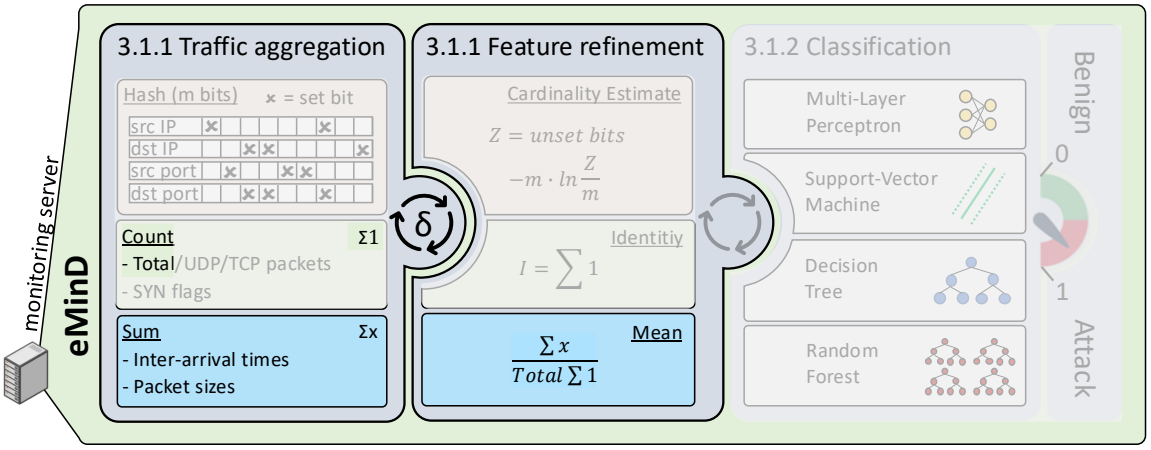


Figure 3.5: *Overview of the interplay between sum-aggregates and mean feature refinement, whose calculation is triggered after each time window.*

Sum-Aggregates and Mean Features. Sum-aggregates add up packet attributes of the observed packet stream, i.e., the packet sizes and the inter-arrival times. This enables the computation of mean features during feature refinement without storing individual packet attribute values (see Figure 3.5).

Definition 3.6: Sum-Aggregates

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a sequence of packets from the observed packet stream, and let $g : P \to \mathbb{R}$ be a function that extracts an attribute from each packet. The sum-aggregate for attribute g is defined as:

$$s = \sum_{j=1}^{n} g(p_j)$$

where $g(p_j)$ represents the attribute's numeric value extracted from packet p_j.

Mean Features calculate the average values of the packet sizes and the inter-arrival times based on the corresponding sum-aggregates and the total packet count from the count-aggregates. Mean features provide essential statistical characterization of packet size and inter-arrival time distributions, enabling the detection of traffic patterns that deviate from typical network behavior, e.g., sudden spikes in packet sizes due to amplification attacks or a drop of the inter-arrival times (see Figure 2.1 and Figure 2.4) indicating traffic bursts.

Definition 3.7: Mean Features

Let s be the sum of n observed packet attribute values and n be the total packet count, as defined in the associated sum-aggregate. The mean feature is then computed as:

$$\mu = \frac{s}{n}$$

where μ represents the average value of the continuous-valued packet attribute over the time window.

Relevance of Mean-Features for DDoS Detection

Packet size statistics expose volumetric attack patterns, such as the large response packets typical of amplification attacks. Inter-arrival time statistics capture traffic burstiness and packet arrival rate changes, which are indicative of sudden traffic volume increases characteristic of DDoS attacks. The mean-features provide essential temporal and volumetric signatures that complement cardinality and count-based metrics, enabling more accurate attack detection.

Feature Refinement Summary

The feature refinement process systematically transforms the raw traffic aggregates into meaningful input features for ML classification. The resulting 10-dimensional feature vector provides a comprehensive yet compact representation of network traffic characteristics, enabling efficient classification while maintaining the constant memory complexity of the underlying aggregation scheme. This feature set captures cardinality estimation features (4 features), packet attribute features (4 features), and statistical features (2 features), collectively spanning detection relevant dimensions for DDoS detection.

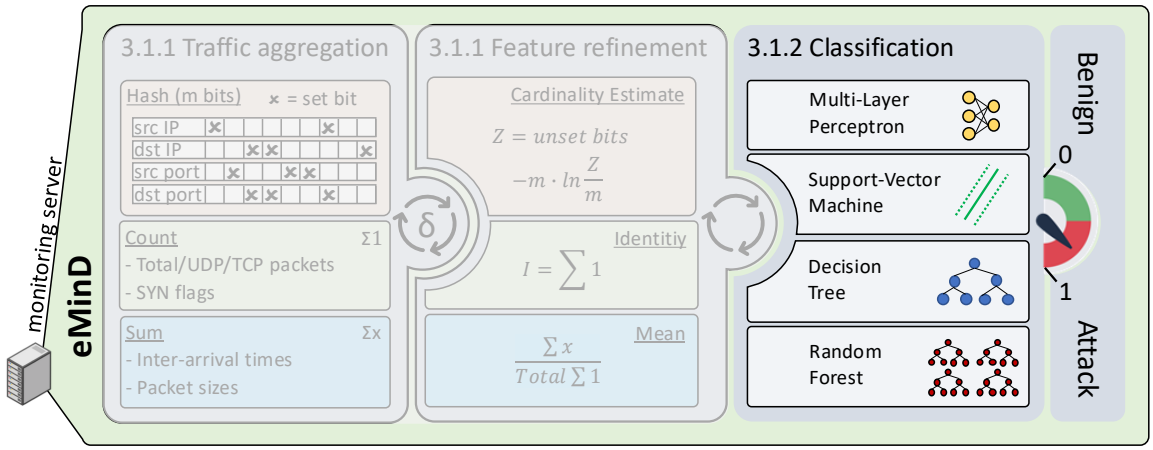


Figure 3.6: *Illustration of eMinD's classification pipeline stage.*

3.1.2 Traffic Classification

Following the traffic aggregation and feature refinement, the resulting feature vectors are utilized as input to an ML model designed for binary classification (Figure 3.6). The objective of this classification stage is to determine whether an attack is ongoing or not.

The employed ML model is trained offline in a supervised learning manner with labeled data. Importantly, the classification is performed at the granularity of a time window, irrespective of the number of packets, micro-flows, or the overall traffic volume. This design choice promotes scalability regarding large traffic volumes while accommodating the use of more computationally intensive ML models that may exhibit inference durations of up to one time window duration δ.

To demonstrate the generality and adaptability of the proposed methodology, as well as to enable more comparison possibilities for the research community, multiple classification algorithms are evaluated in the experimental study. Specifically, the considered models include Decision Trees (DT), Random Forests (RF), Support Vector Machines (SVM), and Multi-Layer Perceptrons (MLP) as they are popularly used in related work. The goal is to show that eMinD's refined features capture meaningful and detection relevant traffic characteristics that can be utilized across different ML approaches, in contrast to showing that a specific ML model outperforms others.

The overall reaction time of eMinD, the duration from the start of an attack until its detection, will later be empirically evaluated and compared against microflow-based detection approaches and comprises two key components: feature refinement and classification. To ensure fast reaction times, the cumulative processing time of these steps must not exceed the duration of a single time window. This constraint guarantees that the system maintains its responsiveness.

3.2 Examining eMinD's System Parameters

As outlined in the previous section, eMinD comprises two key system parameters: the time window duration δ and the bit array size m of the hash-aggregates. This section provides a systematic evaluation of their interplay based on real-world data and considerations for their choice in practice, e.g., that longer time windows require larger bit arrays to maintain accurate cardinality estimation.

3.2.1 The Time Window Duration

The time window duration δ is a critical parameter that defines the temporal scope of each traffic aggregate and the resulting feature set respectively. It directly influences both the detection accuracy and the reaction time of eMinD, i.e., the duration from the start of an attack until its detection, as larger time windows allow for more traffic to be observed and aggregated, which can potentially lead to a better detection accuracy, while shorter time windows lead to faster reaction times.

Definition 3.8: Reaction Time

The **reaction time** is defined as the duration from the start of an attack until its detection.

A fundamental lower bound on δ is imposed by the cumulative processing time required for feature refinement and classification, as these operations are executed once per time window. If δ is chosen to be shorter than this processing time, the system cannot complete the analysis before the next time window begins. This mismatch leads to a processing backlog, resulting in delayed analysis of subsequent traffic aggregates and potentially degrading overall system performance.

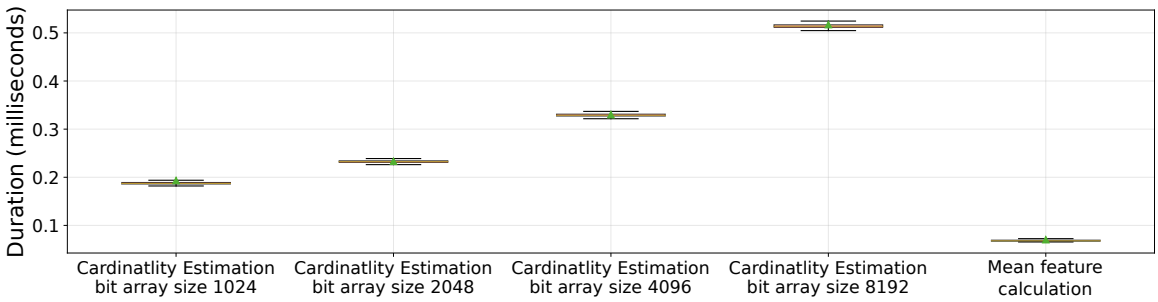


Figure 3.7: *Feature refinement durations across different hash-aggregate sizes.*

Figure 3.7 illustrates the processing duration required for computing cardinality estimation features, which correspond to bit-counting over the hash-aggregate's bit array and

the application of the linear counting formula. The results are shown across varying bit array sizes of the hash-aggregates. The results are derived from 1,000 feature refinement operations performed on traffic from the MAWI-CAIDA dataset and show the mean and median values, as well as the quartiles. The results seem to be represented by individual horizontal lines, as the variance of the processing duration is marginal across the different runs and configurations.

For comparison, the figure also reports the processing time for computing mean features (see Section 3.1.1), which remains constant and independent of the hash-aggregate size under the same experimental conditions. Identity features are omitted from the figure, as their retrieval does not incur any computational overhead.

The mean feature computation consistently remains below 0.1 ms, whereas the processing time for cardinality estimation features increases with the hash-aggregate size, reaching up to 0.53 ms for the largest evaluated size of 8192 entries. Despite this growth, the processing delay for all shown configurations remains below one millisecond, indicating negligible overhead.

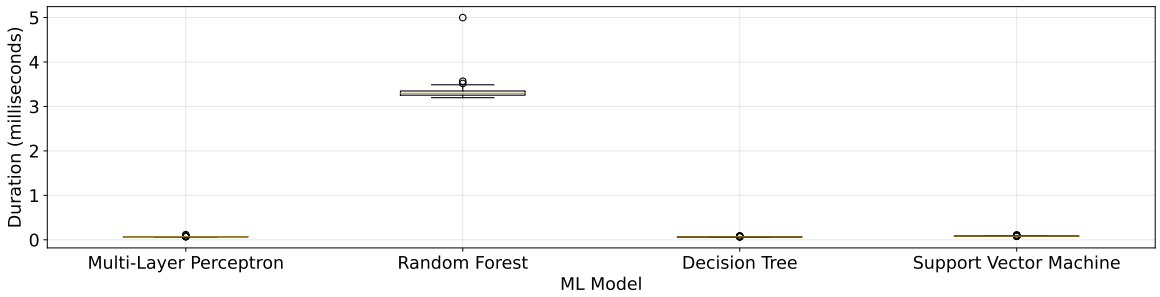


Figure 3.8: *Classification durations across different ML models.*

Figure 3.8 presents the classification durations, measured across multiple runs for statistical significance, for different machine learning models. For the *Multi-Layer Perceptron (MLP)*, *Decision Tree (DT)*, and *Support Vector Machine (SVM)* classifiers, the classification duration consistently remains below 0.1 ms with negligible variance. In contrast, the *Random Forest (RF)* classifier exhibits an average classification time of approximately 3.3 ms and shows a noticeably higher variance across repeated measurements.

When combined with feature refinement durations, the cumulative processing time suggests that the time window δ can be set to at least 5.5 ms, depending on the employed ML model. However, as later demonstrated in the evaluation, the time window must satisfy a minimum duration requirement to aggregate enough network traffic for reliable detection, which is larger than 5.5 ms. Therefore, **feature refinement and classification do not constitute a performance bottleneck of the eMinD pipeline**.

3.2.2 The Bit Array Size

The bit array size m is a critical parameter that directly determines the accuracy of monitoring the cardinality of packet-level attributes. A larger bit array enables a more accurate cardinality estimation of IP addresses and port numbers, potentially enhancing detection performance by capturing finer-grained traffic patterns.
However, increasing the bit array size also incurs higher memory overhead and prolongs feature refinement times (compare to Figure 3.7), which in turn affects the overall reaction time of the system. Consequently, there exists an inherent trade-off between memory utilization, reaction time, and detection accuracy that must be carefully balanced when selecting the bit array size.

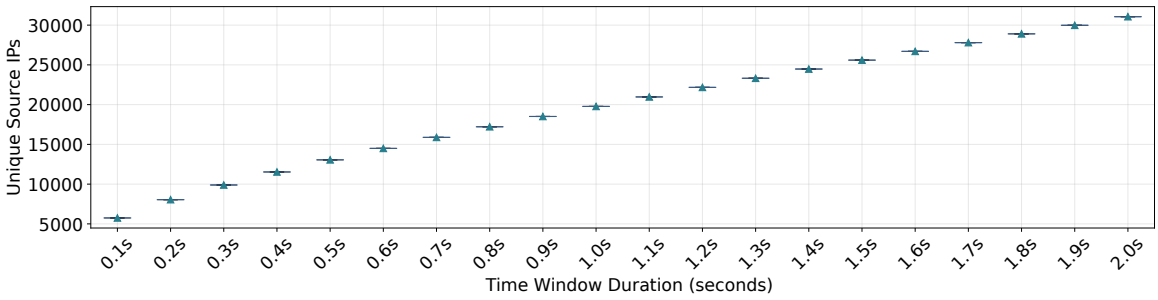


Figure 3.9: *Average unique source IP addresses for the MAWI+CAIDA dataset across different time window durations.*

The first relationship, independent of the bit array size, is that longer time window durations naturally aggregate a higher number of unique IP addresses and ports, as more traffic is observed over an extended period. Consequently, the bit array must be sufficiently large to accommodate the expected cardinality of unique IP addresses and ports within the chosen time window.
Figure 3.9 illustrates this relationship by depicting the number of unique source IP addresses across different time window durations. From this, it follows that longer time windows necessitate larger bit arrays to maintain accurate cardinality estimation.
Therefore, the bit array size must be selected such that it does not become saturated, i.e., mostly or entirely filled with ones, as this would significantly degrade the accuracy of the cardinality estimation. Figure 3.10 illustrates the fill ratio of the bit array for different time window durations and bit array sizes. The fill ratio is defined as the proportion of bits set to one in the bit array, indicating how much of the array is utilized. A fill ratio close to one indicates that the bit array is nearly full, while a fill ratio close to zero indicates that the array is mostly empty. The heatmap shows that as the time window duration increases, the fill ratio also increases. This is expected, as longer time windows allow for more traffic to be aggregated, leading to a higher number of unique IP addresses

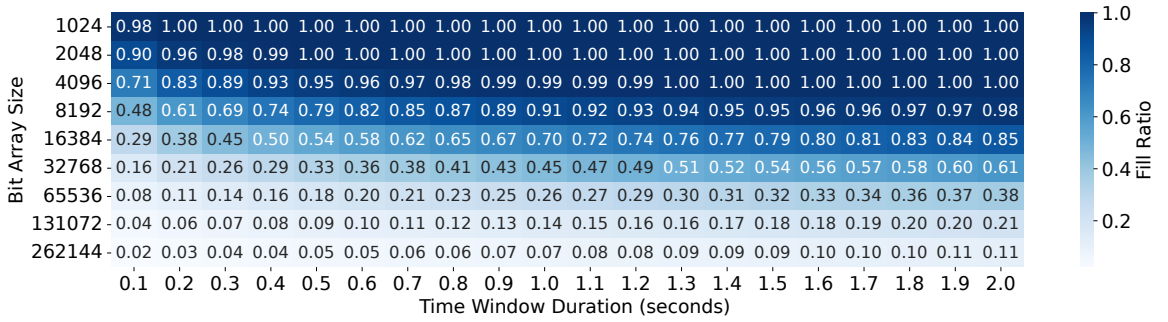


Figure 3.10: *Fill ratio of the bit array for the MAWI+CAIDA dataset, different time window durations and bit array sizes.*

observed. Larger bit array sizes result in lower fill ratios, as they can accommodate more unique IP addresses without becoming saturated.

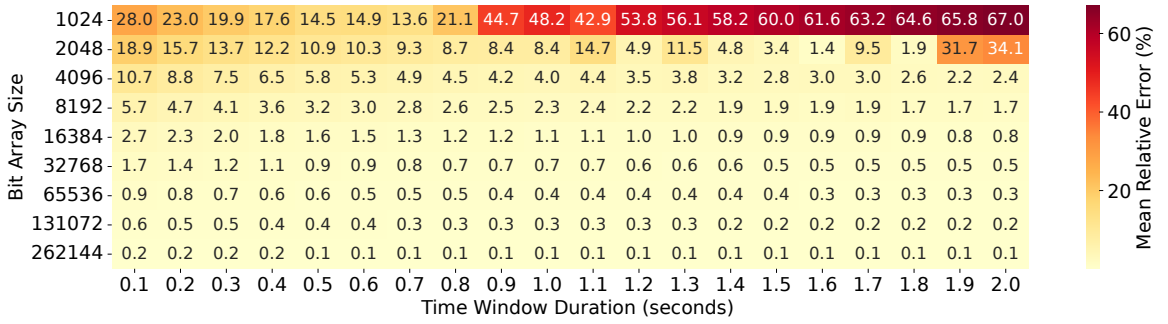


Figure 3.11: *Mean relative error of the cardinality estimation for the MAWI+CAIDA dataset, different time window durations and bit array sizes.*

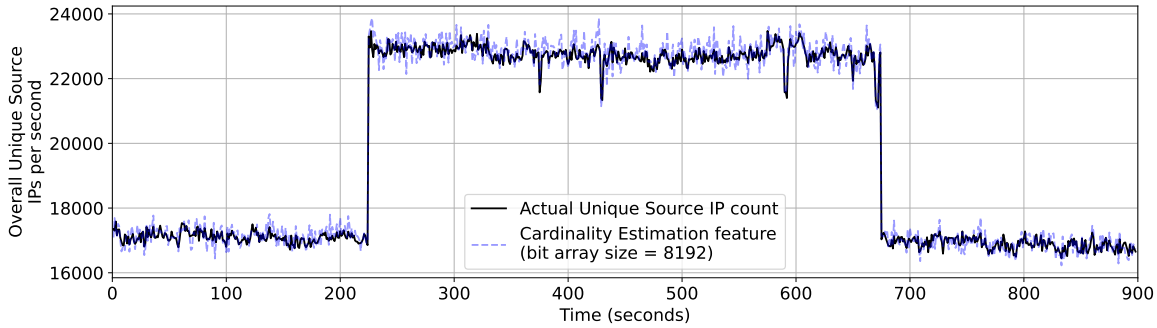


Figure 3.12: *Cardinality estimation over time for the MAWI+CAIDA dataset with a time window duration of 1 second and a bit array size of 8192 bits.*

Figure 3.11 illustrates the mean relative error of the cardinality estimation for different time window durations and bit array sizes. The mean relative error is defined as the

average absolute difference between the estimated cardinality and the true cardinality, normalized by the true cardinality. A lower mean relative error indicates a more accurate cardinality estimation. The heatmap shows that smaller bit array sizes lead to higher mean relative errors, as they are more prone to saturation and cannot accurately estimate the cardinality of unique IP addresses. Larger bit array sizes result in lower mean relative errors, as they can accommodate more unique IP addresses without becoming saturated. If the bit array is not saturated, the error of the cardinality estimation decreases with larger time window durations.
Figure 3.12 illustrates the cardinality estimation over time (MAWI+CAIDA dataset) for the time window duration $\delta = 1s$ and bit array size of 8192 bits. The x-axis represents the time in seconds, while the y-axis shows the estimated cardinality of unique source IP addresses. The plot also shows the true cardinality of unique source IP addresses as a reference. It demonstrates that the cardinality estimation closely follows the true cardinality, indicating that the bit array size of 8192 bits is sufficient to accurately (relative error smaller than 5.7%) estimate the cardinality of unique source IP addresses in the MAWI+CAIDA dataset. As later shown in the evaluation, perfect cardinality estimation is not required for DDoS detection, as the focus is on identifying deviations in traffic patterns rather than exact counts.

3.3 Evaluation and Trade-off Analysis

This section covers an evaluation of eMinD's DDoS detection performance. Experiments are based on the CIC-IDS 2017 dataset (Section 2.3.2) and the MAWI+CAIDA dataset (Section 2.3.5). For every experiment, ML model hyperparameters are optimized via grid search (see results for all models in Section A.1) and evaluated using 10-fold cross-validation (see Section 2.4), where each resulting partition serves as a test set once. The remaining nine partitions are used for training and validation with a 90%/10% split.

The overall detection performance, i.e., accuracy, precision, and recall, is evaluated for different time window durations (Section 3.3.1). For the commonly used CIC-IDS 2017 dataset, the performance results are compared to a microflow-based approach by Choobdar et al. [CNN22] that also utilizes a neural network for binary DDoS detection, which makes it a suitable benchmark for eMinD. Furthermore, a feature importance ranking analysis (see Section 2.4.3) is conducted, based on the permutation feature importance and the MAWI+CAIDA dataset, to show that traffic aggregates enable multiple decisive features in real-world attack scenarios (Section 3.3.2), which facilitate an effective detection of volumetric DDoS attacks.

To emphasize the importance of ML for DDoS detection, an attack scenario is presented, where the attack traffic is particularly subtle and blends into the legitimate background traffic (Section 3.3.3). This makes a simple threshold-based detection infeasible, as the attack traffic characteristics manifest in complex feature interactions spanning multiple features. A comparison of eMinD's detection performance with threshold-based detection is outlined.

In addition, the generalization capabilities of eMinD are evaluated (Section 3.3.4) with datasets from nine consecutive days of MAWI traffic in July 2025, while each day contains the injected CAIDA DDoS 2007 attack. The ML model is trained on the first day and evaluated on the remaining days to assess detection performance over time.

The resource consumption of eMinD is analyzed (Section 3.3.5) in terms of memory utilization and compared to microflow-based approaches. All experiments from Section 3.3.1 to Section 3.3.4 are conducted with an MLP model. Section 3.3.6 evaluates eMinD's compatibility with different ML models, including the Decision Tree, the Random Forest, and the Support Vector Machine. Different popularly used ML models are evaluated to show that eMinD's functionality is not limited to a specific ML model type, but instead its monitoring outputs meaningful features that can be utilized by different ML models. Furthermore, multiple ML models are evaluated to enable more possibilities for comparison through the research community.

3.3.1 Exposure Time vs. Detection Performance

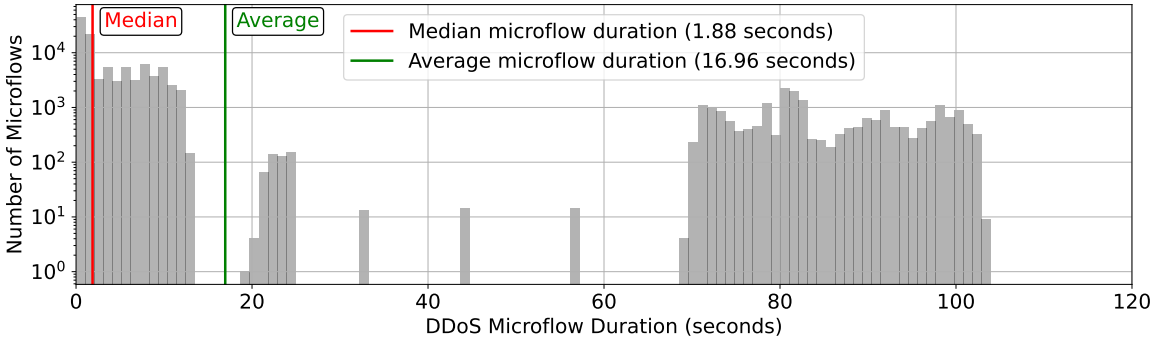


Figure 3.13: *Microflow duration histogram for the CIC-IDS 2017 DDoS data.*

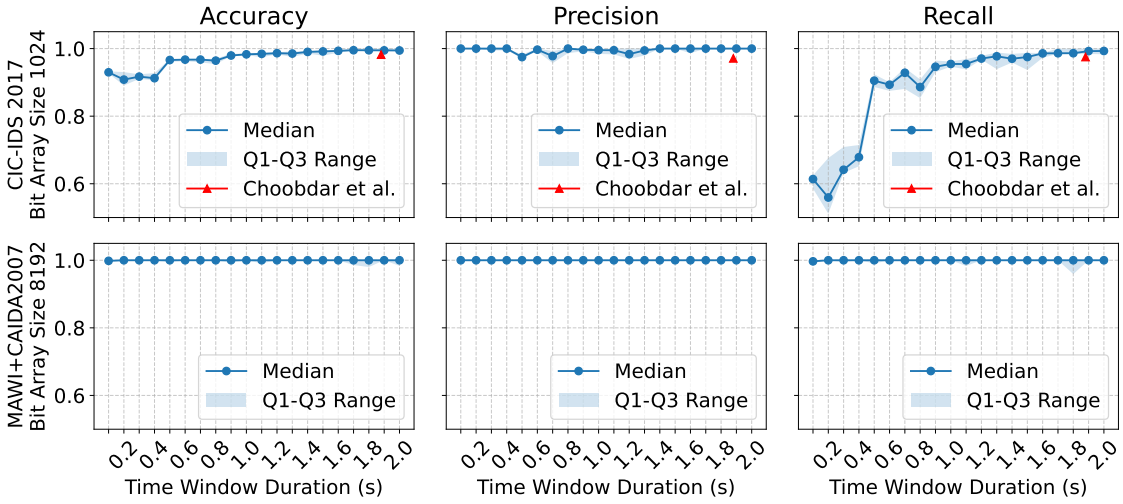


Figure 3.14: *Detection performance across different time window durations for the CIC-IDS 2017 and the MAWI+CAIDA datasets.*

This experiment evaluates the detection performance of eMinD for different time window durations, ranging from 0.1 s to 2.0 s. The lower boundary of 0.1 s is chosen to ensure that feature refinement and classification can certainly be completed within the time window duration (see Figure 3.7 and Figure 3.8).
Figure 3.13 shows the microflow duration histogram of the DDoS flows in the CIC-IDS 2017 dataset. It has a logarithmic y-axis and includes two vertical lines indicating the median (1.88s) and the average (16.96s) microflow duration. The upper boundary of 2.0 s is selected to contain the median flow duration of 1.88 s of the CIC-IDS 2017 DDoS dataset and therefore enable a comparison with microflow-based approaches regarding

the detection performance and the reaction time. To ensure meaningful cardinality estimates, experiments with the CIC-IDS 2017 dataset are conducted with a bit array size $m = 1024$, while experiments with the MAWI+CAIDA dataset are conducted with a bit array size $m = 8192$ (compare Figure 3.12). Choosing a larger bit array size for the MAWI+CAIDA dataset than for the CIC-IDS 2017 dataset is required as the MAWI+CAIDA dataset contains a significantly higher number of unique IP addresses.

Figure 3.14 presents the detection performance across the selected range of time window durations for the CIC-IDS 2017 (upper row) and the MAWI+CAIDA (bottom row) datasets. The three columns show the accuracy, precision, and recall, from left to right. As 10-fold cross-validation is employed, 10 results are obtained for each configuration. The median value and the interquartile range (Q1-Q3 range) are visualized. For the CIC-IDS 2017 dataset, also the respective detection performance results of the microflow-based approach by Choobdar et al. [CNN22] are shown as a reference for the median microflow duration of 1.88 s. The detection performance regarding the MAWI+CAIDA dataset is near perfect, representing an authentic backbone traffic scenario. All median values of accuracy, precision, and recall are 1.0 for all evaluated time window durations. However, the interquartile range shows a small deviation for the time window duration of 1.8 s, which is attributed to the limited amount of training data (2000 samples in total) and the random split into training and test sets. For the shortest time window duration of 0.1 s, eMinD aggregates 30,000 packets per time window at minimum, which is sufficient to capture and learn the traffic characteristics and maintain a short reaction time. In

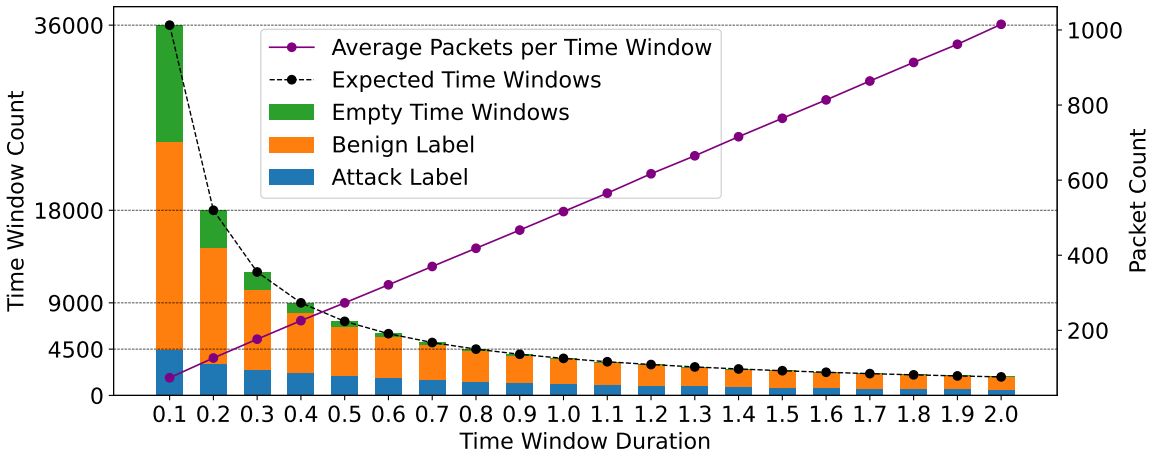


Figure 3.15: *Time window label distribution for the CIC-IDS 2017 dataset and different time window durations.*

contrast, the detection performance regarding the CIC-IDS 2017 dataset improves with longer time window durations and exhibits recall values smaller than 0.95 for time

window durations shorter than 1.0 s. The reasons for this effect are threefold. First, for time window durations $\delta < 1.0$ s, empty time windows exist, i.e., no packet arrives during the time window (green bars in Figure 3.15). As empty time windows are not used for training, the total amount of training data is reduced, which negatively impacts the detection performance. Second, as the CIC-IDS 2017 contains much less traffic than the MAWI+CAIDA dataset, causing fewer packets per time window (see Figure 3.15). Also for non-empty time windows with $\delta < 1.0$, the average number of packets is less than 550, while many time windows aggregate less than 10 packets. This leads to less significant feature values, and to a reduced detection performance respectively. Third, larger time window durations δ enable the aggregation of more packets per time window, but reduce the total amount of training data, as the total traffic trace duration is fixed (3600s). However, the results show that eMinD outperforms the microflow-based approach by

Table 3.2: *Detection performance comparison for time window duration $\delta = 1.8$ s.*

Approach	Accuracy	Precision	Recall
Choobdar et al. [CNN22]	0.982	0.971	0.975
eMinD ($\delta = 1.8s$)	**0.995**	**1.000**	**0.986**
eMinD ($\delta = 1.9s$)	0.995	1.000	0.9925

Choobdar et al. [CNN22] (see Table 3.2) for the time window duration $\delta = 1.80$ s (median flow duration rounded down) regarding accuracy, precision, and recall, while maintaining the same reaction time. The average reaction time of eMinD with $\delta = 1.8$ s is $\frac{16.96}{1.8} \approx 9.42$ times shorter than the microflow-based approach. Furthermore, eMinD achieves perfect precision for the CIC-IDS 2017 DDoS dataset and $\delta > 1.4$ s and for all configurations with the MAWI+CAIDA dataset, indicating no false positives.

3.3.2 Feature Importance Analysis

This experiment evaluates the importance of individual features for the DDoS detection with the MAWI+CAIDA dataset, a time window duration of $\delta = 0.1$ s, and a bit array size of $m = 8192$. The time frame duration of 0.1 s is chosen to obtain the maximum possible amount of time windows for training (from the available time window range). An iterative feature ranking is conducted, where in each iteration the permutation feature importance is computed and the most important feature is removed from the feature set. Figure 3.16 visualizes the feature importance ranking, that illustrates all iterations (from left to right) with the most important feature, its permutation feature importance (boxes) and the achieved accuracy (red line with area indicating median and interquartile range). The first result, i.e., the leftmost box with the x-label ”Src IP Cardinality”, represents the experiment that contains all features for model training and classification. The last

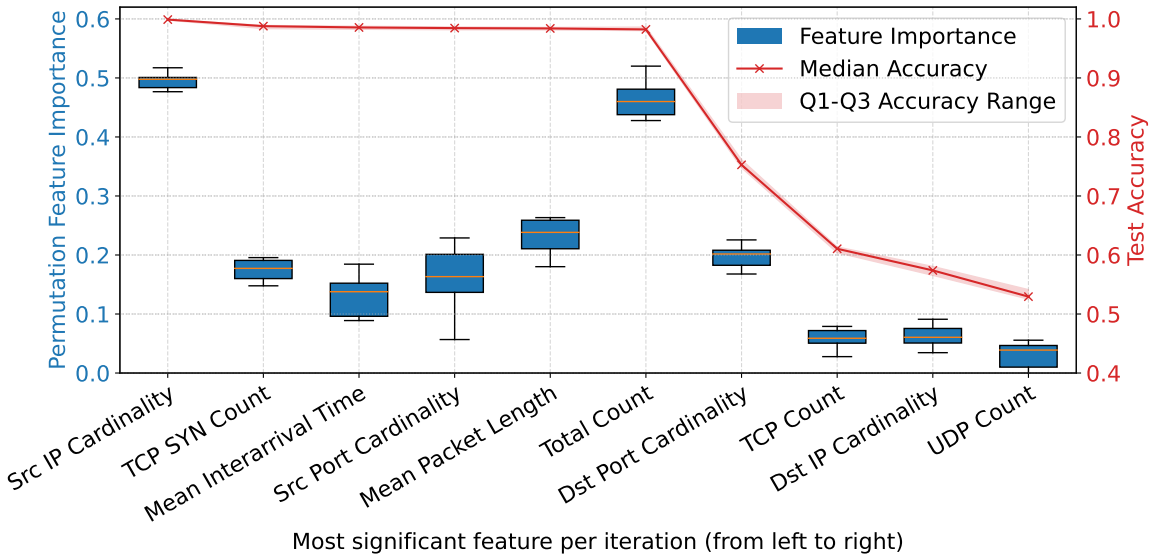


Figure 3.16: *Permutation feature importance ranking for the MAWI+CAIDA dataset with time window duration $\delta = 0.1s$ and bit array size $m = 8192$.*

result, i.e., the rightmost box with the x-label ”UDP Count”, represents the experiment that contains only the ”UDP Count” feature.

The results show that the trained ML model achieves a perfect detection performance with an accuracy of 1.0 when all features are employed. When training the ML model with all features, the most important feature is the ”Src IP Cardinality”, with a median permutation feature importance of approximately 0.49. This means that permuting this feature decreases the accuracy by 49%, resulting in a detection accuracy of approximately 51%, which is only one percent better than guessing. However, removing the feature completely from the feature set and retraining the ML model leads to an accuracy of approximately 0.98, as other features can compensate for the missing information. The second most important feature is the ”TCP Syn Count”. The results further show that the five most important features can be removed, while still achieving an accuracy of approximately 0.97 with the ”Total Count” feature as the new most important feature. However, removing the ”Total Count” feature leads to a significant accuracy drop to approximately 0.75, as no decisive features remain in the feature set. The four least important features are the ”Dst Port Cardinality”, ”TCP Count”, ”Dst IP Cardinality”, and ”UDP Count”, which is reasonable as the CAIDA DDoS 2007 attack contains an ICMP flood with a small TCP SYN flooding component, but no UDP traffic. Therefore, the destination ports and the TCP count do not increase significantly during the attack, while the destination IP cardinality is only increased by one for the DDoS target and the UDP count does not change at all (compare Figure 2.4).

eMinD thus demonstrates that microflow-agnostic features enable multiple decisive features for DDoS detection in real-world attack scenarios, which facilitate an effective detection of volumetric DDoS attacks. However, the question arises why ML is necessary for DDoS detection with eMinD, if decisive features exist that could also be employed for simple threshold-based detection.

3.3.3 Why Machine Learning is Beneficial for DDoS Detection

To address the question, why ML is beneficial for DDoS detection, this experiment presents a subtle attack scenario, where the attack traffic characteristics manifest in complex feature interactions spanning multiple features. A simple threshold-based detection is infeasible, as the attack traffic blends into the legitimate background traffic.

The legitimate traffic has the following characteristics for each time window:

- The TCP and UDP packet count is randomly drawn between 900k and 1100k.
- The total packet count is the sum of the TCP and UDP packet count.
- The source IP cardinality is a fifth of the total packet count.
- The source port cardinality is one five-thousandth of the total packet count.
- The destination IP cardinality is one tenth of the total packet count.
- The destination port cardinality is one ten-thousandth of the total packet count.
- The TCP SYN count is three quarters of the TCP count.
- The mean packet length is randomly drawn between 900 and 1100 bytes.
- The mean inter-arrival time is 1s divided by the total packet count.

The attack traffic represents a TCP SYN flooding attack with a small overall volume of 15 thousand packets per second compared to the legitimate traffic, which has a volume of approximately two million packets per second, which constitutes a fraction of 0.75%. It has the following characteristics per time window:

- The attack is originated from 15 thousand unique source IP addresses and ports.
- The attack has one destination IP address (the target) and one destination port.
- The TCP SYN count is 15 thousand.

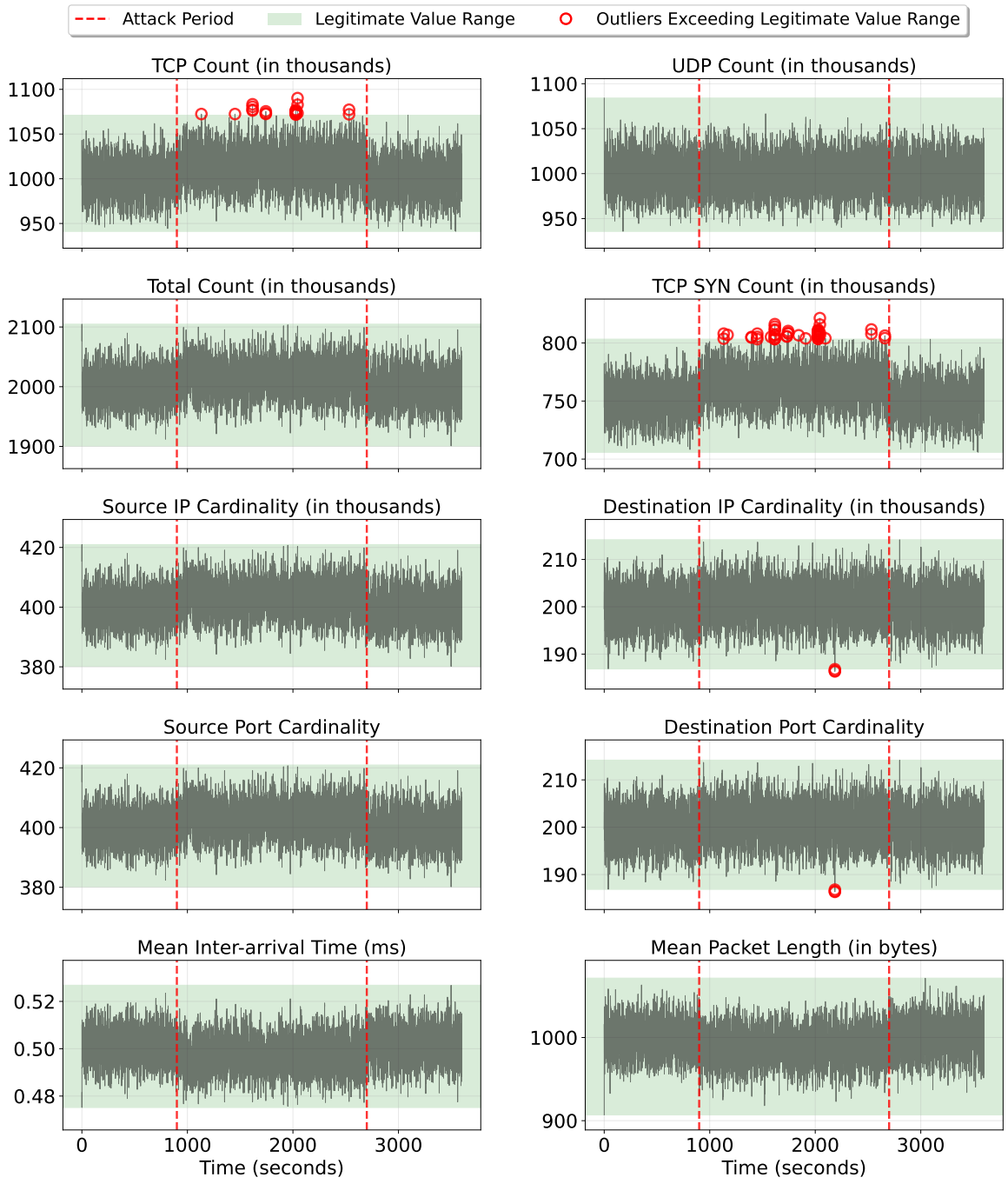


Figure 3.17: *Visualization of synthesized legitimate traffic with a subtle TCP SYN flooding attack.*

Figure 3.17 visualizes all of eMinD's 10 features in the subtle TCP SYN flooding attack scenario with 3600s of synthesized time windows. The attack traffic is injected into the legitimate traffic between 900s and 2700s (indicated by the dashed red lines), therefore

spanning half of the time windows and resulting in a balanced dataset. The mean inter-arrival time and the mean packet length decrease during the attack, as small TCP SYN packets are injected into the legitimate traffic. The green areas indicate the range between the minimum and the maximum value of the legitimate traffic for the respective feature. The red circles indicate values that are outside of this range during the attack. There are only four features with values outside of the green area during the attack: The ”TCP SYN Count”, the ”TCP Count”, the ”Destination IP Cardinality”, and the ”Destination Port Cardinality”. However, only the ”TCP SYN Count” and the ”TCP Count” exceed the legitimate value range.

Table 3.3: *Detection performance of threshold-based detection for the subtle TCP SYN flooding attack scenario.*

Approach	Accuracy	Precision	Recall
eMinD	**1.00**	**1.00**	**1.00**
Threshold - TCP SYN Count	0.5013	1.00	0.0027
Threshold - TCP Count	0.5006	1.00	0.0013

If threshold-based detection is applied to these two features, i.e., raising an alarm if the respective feature exceeds the legitimate values range, the overall accuracy is close to 0.5 (compare Table 3.3), which is equivalent to guessing, while the precision is 1.0 as no false positives occur. However, the recall is only 0.0027 for the ”TCP SYN Count” and 0.0013 for the ”TCP Count”, indicating that only a negligible fraction of the attack time windows is detected. In contrast, when using eMinD, the ML model achieves a perfect detection performance with an accuracy, precision, and recall of 1.0. The ML model is able to learn feature interactions of all provided features, which enables complex decision boundaries that exceed the capabilities of threshold-based detection. Furthermore, only binary labels for time windows are required for training, while hand-crafted rules require deep domain knowledge and the inspection of the network traffic.
Figure 3.18 visualizes the permutation feature importance using all available features for training and classification. The results show that seven features have a permutation feature importance larger than 0.35. This means that permuting any of these features significantly decreases the overall accuracy. Only all of these features combined enable perfect detection performance.
Figure 3.19 visualizes the feature importance ranking for the subtle TCP SYN flooding attack scenario. The results show that the trained ML model achieves a perfect detection performance with an accuracy of 1.0 when all features are employed. However, when removing the most important feature, i.e., the ”TCP SYN Count”, the accuracy drops to approximately 0.58, and shows that no decisive features remain in the feature set. This emphasizes that the required *detection rule* comprises multiple features and their

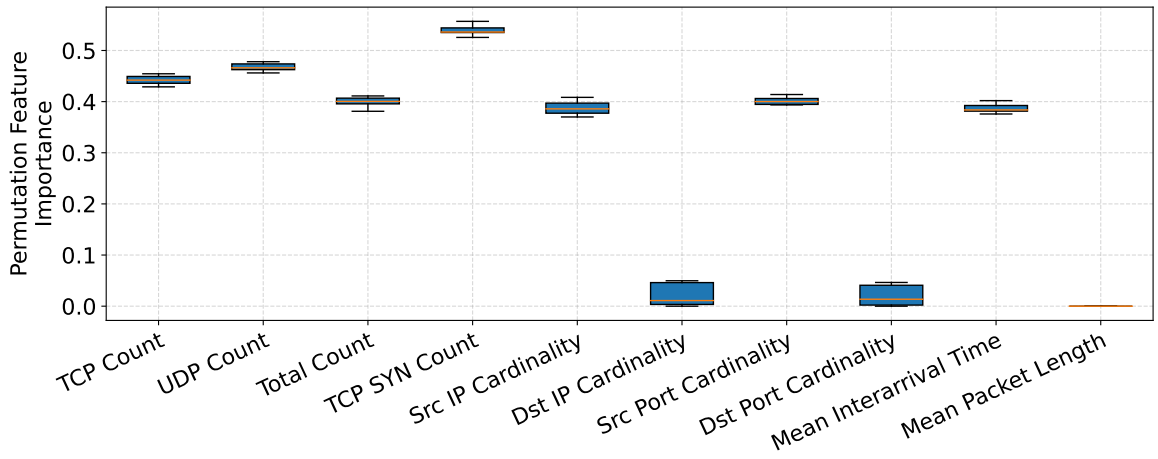


Figure 3.18: *The permutation feature importance with all features for the subtle TCP SYN flooding attack scenario achieving a perfect detection performance.*

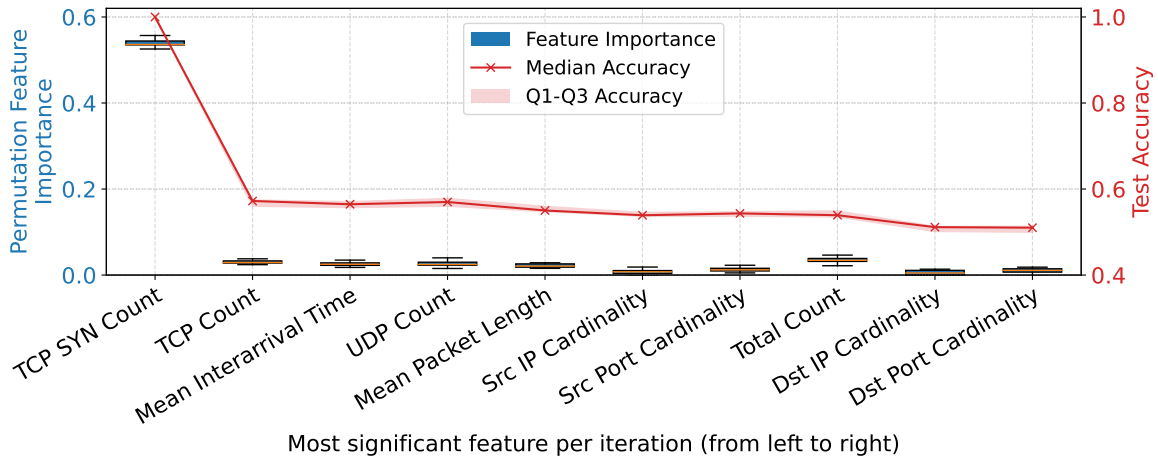


Figure 3.19: *Feature importance ranking for the subtle TCP SYN flooding attack.*

interplay is crucial for attack detection. No other individual feature or feature combination can compensate for the missing information.

3.3.4 Generalization Performance

The previous experiments are based on a single traffic trace, i.e., training, validation and testing is conducted with disjoint data of the same day. To evaluate the generalization capabilities of eMinD, this experiment utilizes data from nine consecutive days of MAWI traffic in July 2025 (15 minute traces), i.e., from July 1st to July 9th, while each day contains the injected CAIDA DDoS 2007 attack. The ML model is trained on the first day and evaluated on the remaining days to assess detection performance over time.

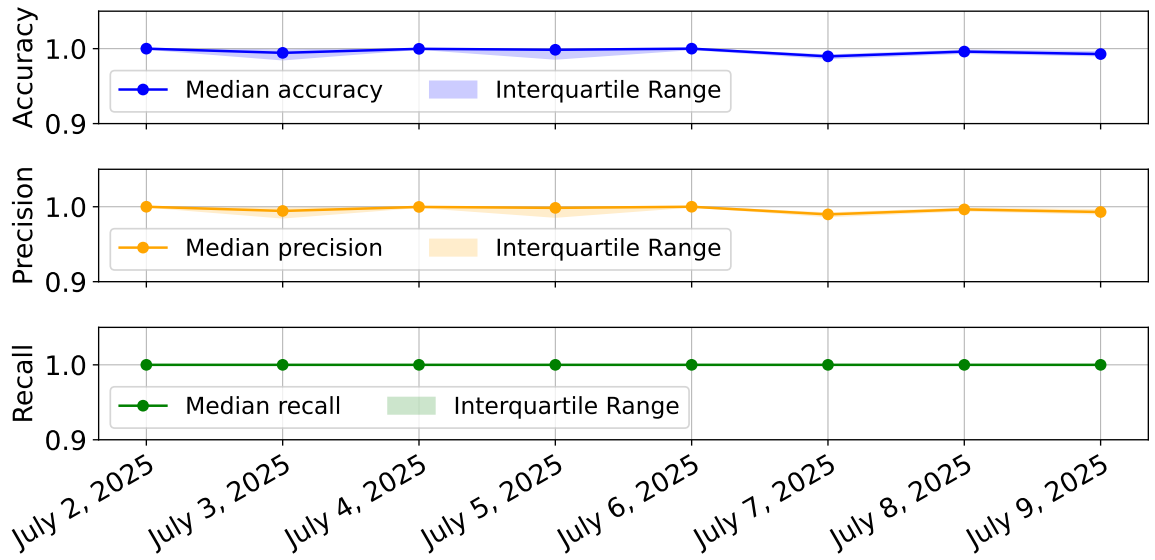


Figure 3.20: *eMinD's generalization performance results with one day of training.*

Figure 3.20 visualizes the detection performance, i.e., accuracy, precision, and recall, for the time window duration $\delta = 0.1$ s and a bit array size of $m = 8192$. The results show that the ML model achieves perfect recall over all eight days of testing. However, the median precision is degraded on the second and sixth day of testing to 99% and 98%, leading to a reduced accuracy. A degraded precision implies that false positives occur, i.e., legitimate traffic is misclassified as attack traffic. One reason for this effect is that the legitimate traffic characteristics change over time, and these changes are not reflected in the training data of the first day. Another reason can be the presence of additional DDoS attacks in the MAWI traffic, which is assumed to be legitimate traffic. To compensate for changes in the legitimate traffic characteristics, another experiment is conducted: The ML model is trained on three consecutive days (July 1st to July 3rd) and evaluated on the remaining six days (July 4th to July 9th). Figure 3.21 visualizes the detection performance. The results show that the median performance across all metrics is 1.0, implying a perfect detection performance.

The results lead to two conclusions:

(1) eMinD generalizes well over time, even when training and testing is conducted on different days, which makes it suitable for real-world deployment.

(2) eMinD requires enough training data that reflects the legitimate traffic characteristics to achieve a high precision.

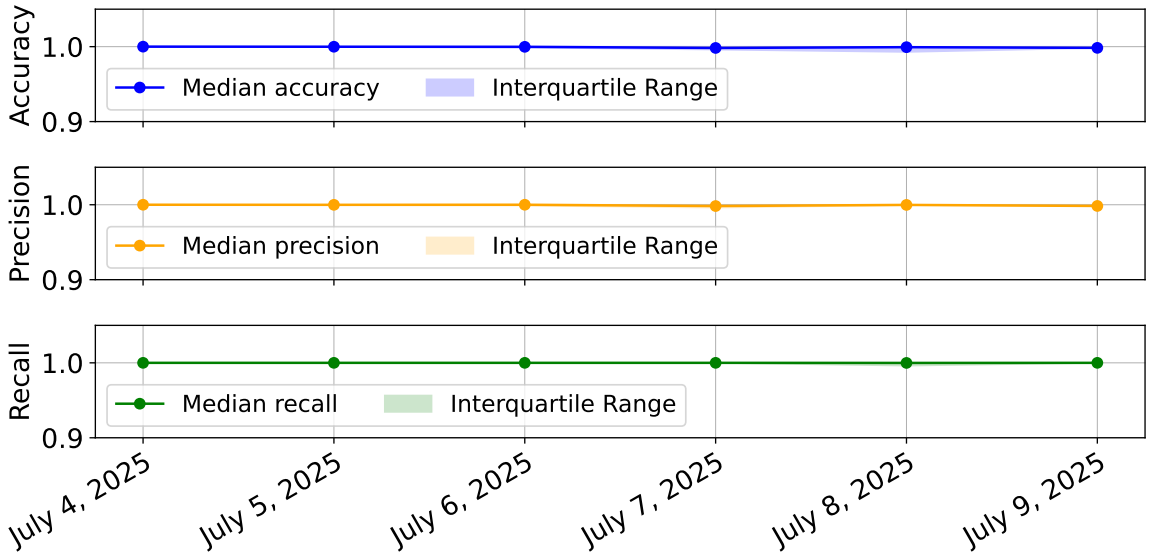


Figure 3.21: *eMinD's generalization performance results with three days of training.*

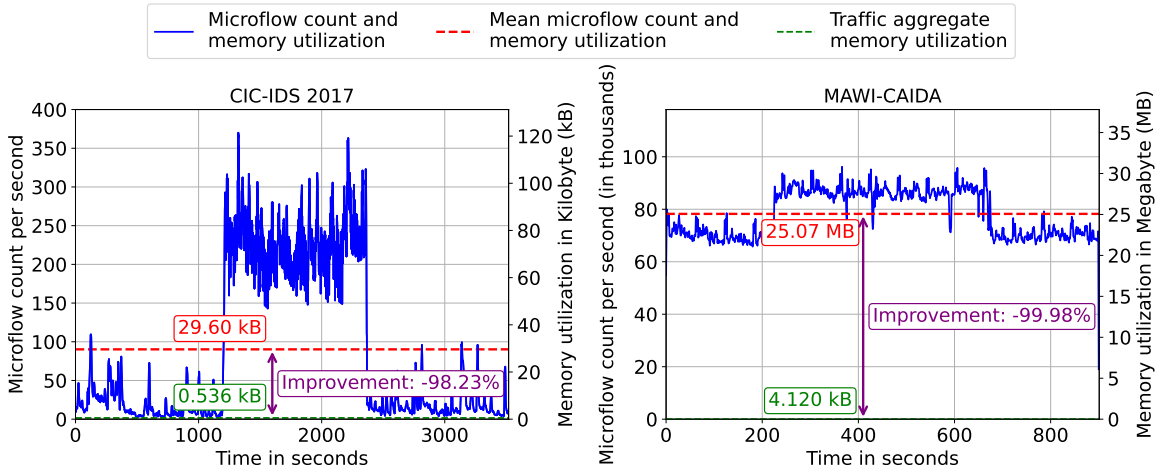


Figure 3.22: *Memory utilization comparison of eMinD with microflow-based approaches.*

3.3.5 Comparing Resource Consumption to Microflow-based Approaches

This evaluation compares the memory utilization and the required number of ML classifications between eMinD and microflow-based approaches. For the microflow-based approach, a memory utilization of 4 Bytes per feature is assumed, as well as 84 features per microflow, based on the CICFlowMeter tool [SHG18] and the work by Choobdar et al. [CNN22], which was also used for the performance comparison. This results in a memory utilization of 336 Bytes per microflow. In contrast, eMinD's memory utilization is fixed, i.e., 0.536 Kilobytes for bit array size $m = 1024$ and 4.120 Kilobytes for bit array

size $m = 8192$, as there are 4 hash-aggregates requiring a bit array and 6 features that require 4 Bytes each. For the CIC-IDS 2017 dataset, a bit array size of $m = 1024$ is chosen, while for the MAWI+CAIDA dataset, a bit array size of $m = 8192$ is selected to ensure meaningful cardinality estimates as the MAWI+CAIDA dataset contains much more traffic.

Figure 3.22 visualizes the microflows per second and the memory utilization comparison between eMinD and microflow-based approaches for the CIC-IDS 2017 (left) and the MAWI+CAIDA (right) datasets. The blue lines indicate the microflows per second over the whole duration of the trace, and the required memory utilization at the same time. The dashed red line shows the mean microflow count and the mean memory utilization over the whole trace. The dashed green line indicates eMinD's fixed memory utilization for bit array size $m = 1024$ (left) and $m = 8192$ (right).

The results show that eMinD saves 98.23% of memory for the CIC-IDS 2017 dataset in average. This effect is more pronounced during the DDoS attack period. For the authentic Internet backbone scenario with the MAWI+CAIDA dataset, eMinD saves 99.98% of memory in average. Furthermore, eMinD requires only one ML classification per time window, while microflow-based approaches require one ML classification per microflow. This saves approximately 90 classifications per second in average for the CIC-IDS 2017 scenario and approximately 78,000 classifications per second in average for the MAWI+CAIDA scenario.

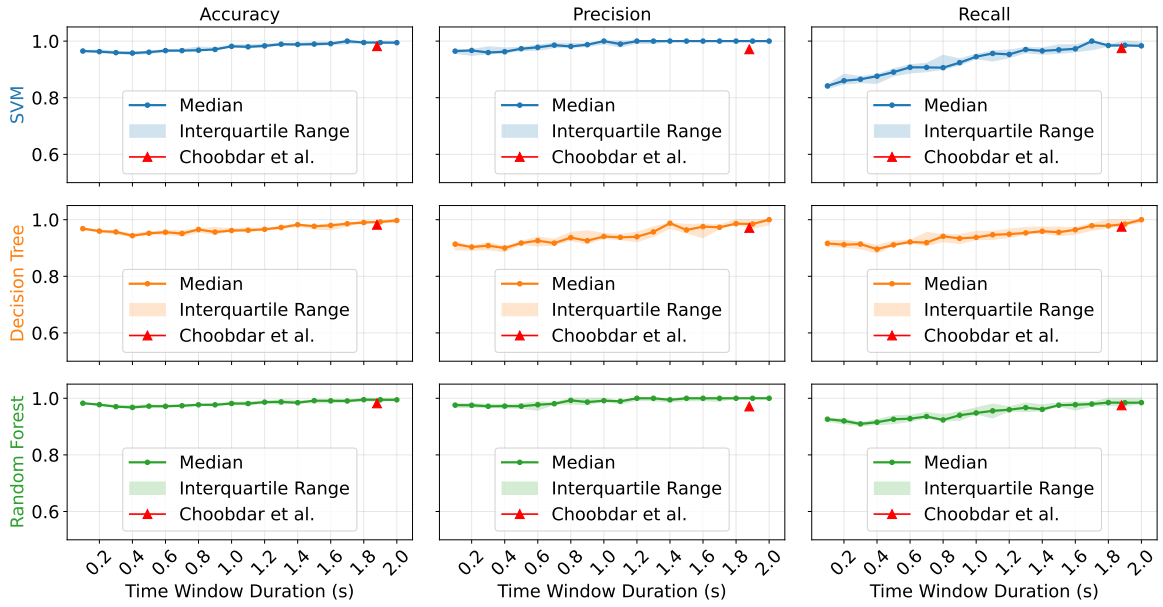


Figure 3.23: *Detection performance comparison of different ML models for the CIC-IDS 2017 dataset across different time window durations and bit array size $m = 1024$.*

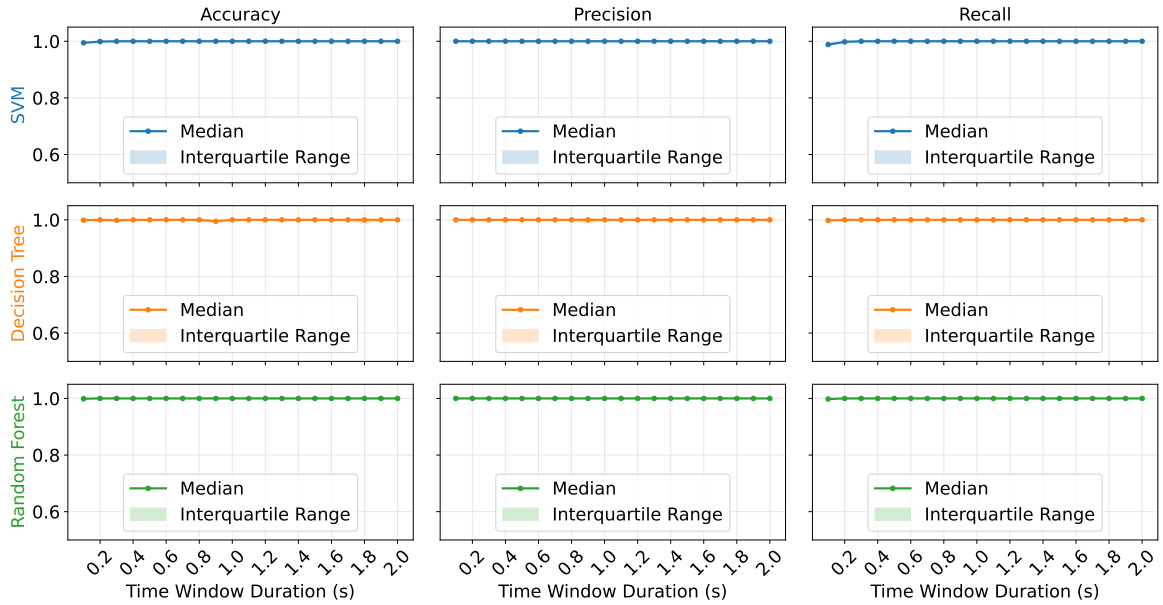


Figure 3.24: *Detection performance comparison of different ML models for the MAWI+CAIDA dataset across different time window durations and bit array size* $m = 8192$.

3.3.6 Different ML Models

This section contains additional experiments with different ML model types, i.e., Decision Tree (DT), Support Vector Machine (SVM), and Random Forest (RF), to show eMinD's time window-based monitoring approach is compatible with different ML models.

The detection performance of the three ML models is evaluated for the CIC-IDS 2017 dataset. The same parametrization as used for the experiment with the MLP is applied, i.e., bit array size $m = 1024$ and time window durations from 0.1 s to 2.0 s, to maintain comparability. Figure 3.23 visualizes the detection performance results, namely accuracy, precision, and recall (from left to right), for the three ML models (from top to bottom) across different time window durations (x-axis).

Three main observations can be made: First, all three other ML models achieve higher performance metrics than the MLP for lower time window durations $\delta < 1.0$ s. Second, all three ML models' results hold the trend that the detection performance improves with longer time window durations, which is consistent with the experiments with the MLP (compare Figure 3.14). And third, in contrast to the MLP, the other three models consistently start with a higher recall for the shortest time window duration of 0.1 s, while lacking precision, i.e., return more false positives.

Although these results are not methodically comparable to the results of Choobdar et al. [CNN22], as they used an MLP, all three ML models, utilizing time-window-based

feature sets, outperform the microflow-based approach for time window durations larger than the median flow duration of 1.88 s.
The detection performance results of the three ML models achieved for the MAWI+CAIDA dataset are visualized in Figure 3.24. The same parametrization as used for the experiment with the MLP is applied, i.e., bit array size $m = 8192$ and time window durations from 0.1 s to 2.0 s, to maintain comparability. The results show that all three ML models achieve perfect detection performance with an accuracy, precision, and recall of 1.0 for all evaluated time window durations, except for the SVM ($\delta = 0.1$ s) and the DT ($\delta = 0.9$ s). These results are consistent with the experiments with the MLP (compare Figure 3.14), that also achieved perfect detection performance for all evaluated time window durations.

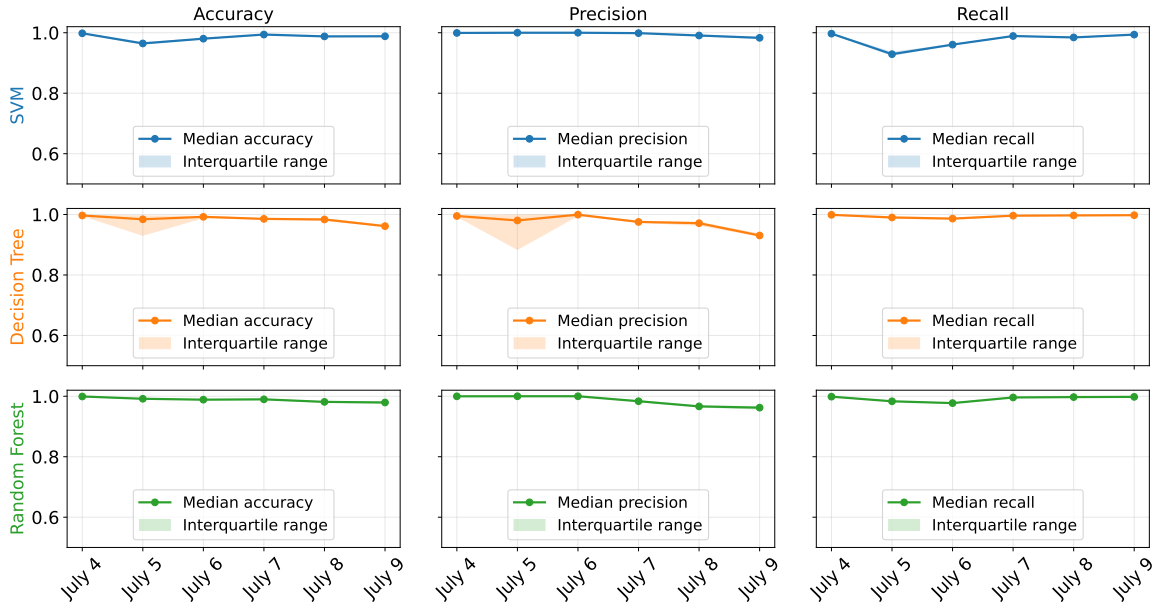


Figure 3.25: *Generalization performance comparison of different ML models for the MAWI+CAIDA dataset with three days of training, a bit array size $m = 8192$ and time frame duration $\delta = 0.1$ s.*

Next, the generalization capabilities of the three ML models are evaluated with the MAWI+CAIDA datasets. As conducted with the MLP, the ML models are trained on three consecutive days (July 1st to July 3rd) and evaluated on the remaining six days (July 4th to July 9th). Figure 3.25 visualizes the detection performance results, i.e., accuracy, precision, and recall (from left to right), for the three ML models (from top to bottom). The results show that all three models achieve perfect detection performance only on the first day of testing, while the MLP achieves perfect detection performance across all six days of testing (compare Figure 3.21). This leads to the conclusion that the MLP has the best generalization capability over time compared to the other three ML models.

3.4 Conclusion

This chapter presented eMinD, a traffic aggregate-based DDoS detection approach that leverages fixed-state monitoring with a low memory footprint and microflow-independent features to enable scalable and resource-efficient detection. This **saves 99.98% of utilized memory** in average compared to a traditional microflow-based approach in a realistic Internet backbone scenario. It also **saves 78,000 ML classifications per second** in average, reducing the required computational resources and enabling predictable reaction times. Furthermore, the average **reaction time is 9.42 times shorter** than microflow-based detection in the CIC-IDS 2017 DDoS scenario.
The experiments highlighted several important findings:

- eMinD outperforms microflow-based approaches in terms of memory utilization, and the number of ML classifications. It also outperforms a specific microflow-based approach [CNN22] in terms of detection accuracy.
- Depending on the overall traffic volume, the selection of the time window duration constitutes a trade-off between the detection accuracy and the reaction time.
- eMinD shows strong generalization capabilities over time in a real-world Internet backbone scenario.
- eMinD leverages ML to capture complex feature interactions for DDoS detection and reduces the detection effort from rule creation to label assignment over time windows.

3.4.1 Discussion

Despite the achieved improvements in the detection performance, the average reaction time, and the required memory utilization compared to microflow-based approaches, eMinD still encounters three main limitations: First, it facilitates **only binary attack detection**, signaling the presence or absence of an attack, but does not provide detailed attack information, e.g., the destination IP subnet, which can be used to derive network traffic filter rules or narrow down the traffic volume for mitigation functions. Second, eMinD is designed for software-based monitoring that runs on a server collocated with an ingress router. This design requires **installed hardware for each ingress router**, which limits eMinD's scalability to large networks with many ingress routers. And third, software-based monitoring restricts the maximum achievable monitoring data rate through the servers' hardware capabilities, which **inherently causes packet sampling** if the incoming data rate exceeds the servers' processing capabilities.

Chapter 4

IP Distribution-preserving Time Window-based DDoS Detection

To solve the limitations of eMinD (see Section 3.4.1) and to maintain the benefits of the **time window-based detection pipeline**, this chapter outlines and evaluates the traffic image-based monitoring and classification pipeline. **The traffic image** [KHZ22b; KHZ22a] is an IP prefix-preserving traffic aggregation scheme, that **counts packets** of a network's ingress traffic **per source-to-destination IP subnet pair** (pixel). This counting mechanism is implemented in the Ternary Content Addressable Memory (TCAM) of ingress routers through a fixed set of TCAM rules, which comes with two major benefits: First, TCAM rules are evaluated within one clock cycle, which enables **packet sampling-free monitoring at line speed**. Second, the monitoring is performed by the ingress routers themselves, which **eliminates the need for dedicated monitoring servers** per ingress router. The number of required TCAM rules for traffic image-based monitoring depends on the traffic image resolution, i.e., one TCAM rule per pixel. However, the traffic image resolution is fixed during the deployment and can by chosen according to the available TCAM resources of the ingress routers.
Traffic images also follow the **time window-based monitoring paradigm**, i.e., the TCAM rule counters are fetched periodically from the ingress routers by the remote classification server. Fetched traffic images are then classified through Machine Learning (ML)-based classification approaches inspired by the research area of computer vision. This chapter outlines and evaluates several ML-based traffic image classification approaches for DDoS detection and addresses **Research Question 2**, namely whether fine-grained attack information can be derived from microflow-independent network traffic monitoring, such as the attack's destination IP subnet or network filter rules.

This chapter contains the following contributions:

- The **traffic image**, a fixed-state and IP distribution-preserving traffic aggregate that can be monitored through ingress routers' TCAM
- The application of binary image classification to detect volumetric DDoS attacks with traffic images
- Utilizing **time series of traffic images** for improved attack detection
- The application of **multi-class traffic image classification** to identify the attack's destination IP subnet
- The derivation of **network filter rules through traffic image segmentation**
- An extensive **classification performance evaluation** with real-world and specifically tailored synthesized datasets
- A **generalization evaluation** over time to assess the applicability to real-world deployments
- A trade-off analysis regarding the required memory utilization, the required TCAM rule space, and the classification performance

The chapter's structure. Section 4.1 defines the **traffic image** as network traffic representation and explains the monitoring and classification pipeline. Section 4.2 applies **binary image classification** to traffic images for volumetric DDoS attack detection. Section 4.3 extends the binary classification to **time series traffic image classification** capturing temporal traffic development and improving the detection performance compared to single traffic image classification. Section 4.4 presents **multi-class traffic image classification** to identify the attack's destination IP subnet. Section 4.5 outlines **network filter rule derivation** through traffic image segmentation. Section 4.6 presents **extensions** of the traffic image-based detection pipeline, including its compatibility with software-based monitoring infrastructures, multi-color-channel traffic images, as well as its adaptability to detect port scans. Section 4.7 concludes the chapter.

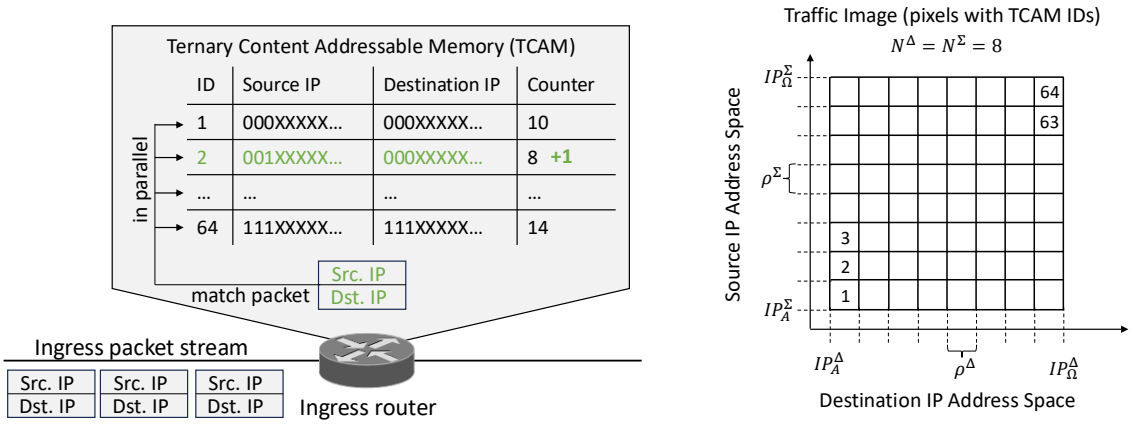


Figure 4.1: *A concept illustration of the traffic image with a resolution* $\chi = (8, 8)$. *The implemented TCAM rules and the two-dimensional pixel grid are shown.*

4.1 The Traffic Image

The **traffic image** (see Figure 4.1) serves as the central concept of this chapter. It represents a time window-based network traffic aggregate that preserves the distribution of source and destination IP addresses, as well as the overall packet count. The traffic image is designed for hardware-based monitoring in the Ternary Content Addressable Memory (TCAM) of ingress routers. It makes the installation of monitoring servers collocated with ingress routers obsolete at the cost of delay induced by fetching the traffic images periodically from a remote server (per time window δ).

The traffic image results from aggregating arriving packets in two dimensions, the source and destination IP address spaces. Each dimension is divided into equally sized and disjoint subnets according to their IP prefixes, spanning a **grid of subnet pairs**, where each **subnet pair corresponds to a pixel** in the traffic image. Each pixel holds the count of packets whose source and destination IP addresses match the corresponding subnet pair. This counting mechanism is implemented through installed TCAM rules tracking matched packets. **One TCAM rule is required per pixel**, which constrains the traffic image resolution based on the available TCAM size.

TCAM rules are matched in parallel during one clock cycle and therefore enable faster per-packet processing than software-based approaches, such as eMinD. However, the use of TCAM has several limitations, such as limited size, higher power consumption, and increased cost compared to RAM. As traffic images provide fixed memory utilization based on the resolution, they are suitable for TCAM-based monitoring, as the resolution can be adapted to the available TCAM size.

Definition 4.1: Traffic Image

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a sequence of packets from the observed packet stream. A **traffic image** is the result of aggregating packets in P into a grid of subnet pairs spanning **two dimensions**:

(1) The **source IP address space** IP^{Σ}, constrained to the range $[\mathrm{IP}^{\Sigma}_{A}, \mathrm{IP}^{\Sigma}_{\Omega}]$.

(2) The **destination IP address space** IP^{Δ}, constrained to the range $[\mathrm{IP}^{\Delta}_{A}, \mathrm{IP}^{\Delta}_{\Omega}]$.

Each dimension is divided into equally sized and disjoint subnets, N^{Σ} subnets for IP^{Σ} and N^{Δ} subnets for IP^{Δ}. Each subnet has a size of ρ^{Σ} for IP^{Σ} and ρ^{Δ} for IP^{Δ}. Let these subnets be indexed by i and j for each dimension, such that:

$$\mathrm{IP}^{\Sigma}_{i} = [\mathrm{IP}^{\Sigma}_{A} + (i-1)\rho^{\Sigma}, \mathrm{IP}^{\Sigma}_{A} + i\rho^{\Sigma}], \quad i = 1, 2, \ldots, N^{\Sigma},$$

$$\mathrm{IP}^{\Delta}_{j} = [\mathrm{IP}^{\Delta}_{A} + (j-1)\rho^{\Delta}, \mathrm{IP}^{\Delta}_{A} + j\rho^{\Delta}], \quad j = 1, 2, \ldots, N^{\Delta},$$

The **traffic image resolution** χ is defined as the tuple: $\chi = (N_A, N_\Omega)$.
A **pixel** in the traffic image corresponds to a subnet pair $(\mathrm{IP}^{\Sigma}_{i}, \mathrm{IP}^{\Delta}_{j})$ in the grid. Each pixel holds the number of observed packets in P whose source and destination IP addresses belong to the subnets IP^{Σ}_{i} and IP^{Δ}_{j}. Let $p_i^{\text{src\_IP}}$ and $p_i^{\text{dst\_IP}}$ denote the source and destination IP addresses of packet p_i in P. Formally, the value of each pixel $(\mathrm{IP}^{\Sigma}_{i}, \mathrm{IP}^{\Delta}_{j})$ is given by:

$$\mathrm{Value}(i,j) = \left| \left\{ p \in P \mid p.\text{src\_IP} \in \mathrm{IP}^{\Sigma}_{i},\ p.\text{dst\_IP} \in \mathrm{IP}^{\Delta}_{j}, \right\} \right|$$

Suitability of Traffic Images for Volumetric DDoS detection
Volumetric DDoS attacks have three main characteristics: (1) large traffic volume through high packet arrival rates, (2) distributed attack sources, and (3) a specific target destination IP address. The traffic image preserves all three characteristics, while it maintains fixed memory utilization that is independent of the source IP address and the packet count. It further utilizes the TCAM of ingress routers, which enables monitoring at line speed and eliminates the need for packet sampling.

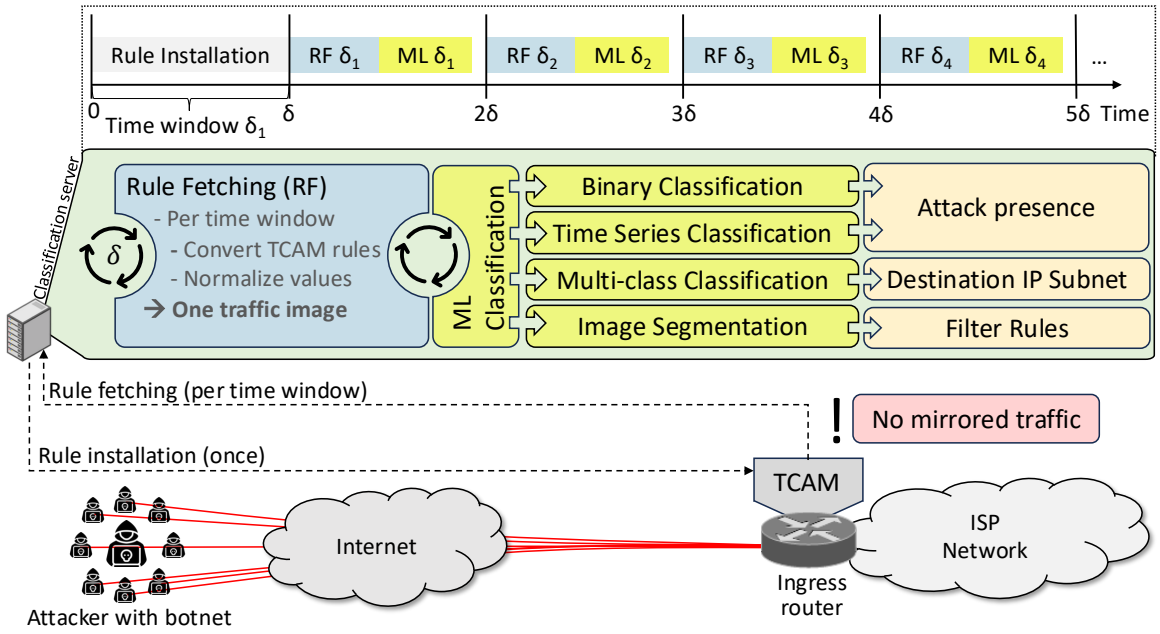


Figure 4.2: *Overview of the traffic image-based monitoring and classification pipeline.*

4.1.1 Monitoring and Classification Pipeline

The traffic image-based classification pipeline is similar to eMinD (compare Figure 3.1) and consists of three main steps:

(1) **Rule Installation** (once) – Calculation of TCAM rules depending on the traffic image resolution. Installation of calculated rules for traffic image monitoring on the ingress routers.

(2) **Rule Fetching** (per time window) – Fetch the installed TCAM rules' counters from the ingress routers and construct the traffic images.

(3) **ML Classification** (per time window) – Classification of the constructed traffic images through pre-trained ML models to derive fine-grained attack information beyond binary detection.

In contrast to eMinD, the traffic aggregation per packet is performed by the ingress router's TCAM through installed rules. The classification server (former monitoring server) **does not require expensive hardware**, such as Network Interface Cards (NICs), as **no per-packet processing is required** but only operations per time window. Lightweight servers with network connectivity to the ingress routers are sufficient to fetch the traffic images and perform the ML classification. The three steps of the classification pipeline are explained in detail in the following.

Rule Installation

The first step of the pipeline is the calculation and installation of TCAM rules on the ingress routers. Each pixel in the traffic image requires one TCAM rule that matches packets whose source and destination IP addresses belong to the corresponding subnet pair. The **number of required TCAM rules** $N_{\textbf{rules}}$ depends on the traffic image resolution $\chi = (N^{\Sigma}, N^{\Delta})$ and is given by:

$$N_{\text{rules}} = N^{\Sigma} \cdot N^{\Delta}.$$

N^{Σ} and N^{Δ} are restricted to powers of two as the subnet division follows IP prefixes, e.g., an IP prefix length of 3 bits results in $2^3 = 8$ subnets. The IP address ranges $[\text{IP}_A^{\Sigma}, \text{IP}_{\Omega}^{\Sigma}]$ and $[\text{IP}_A^{\Delta}, \text{IP}_{\Omega}^{\Delta}]$ cover the whole IP address space by default, i.e., $\text{IP}_A^{\Sigma} = \text{IP}_A^{\Delta} = 0$ and $\text{IP}_{\Omega}^{\Sigma} = \text{IP}_{\Omega}^{\Delta} = 2^{32} - 1$ for IPv4. However, if the monitoring is deployed in a stub Autonomous System (AS), the destination IP address range can be restricted to the AS's prefixes. If the monitoring is deployed in a transit AS, the default full IP address range is used to capture attacks targeting any destination IP address. The following algorithm outlines the default TCAM rule calculation for traffic image monitoring:

Algorithm 3: TCAM Rule Calculation for Traffic Image Monitoring

Input: Resolution $\chi = (N^{\Sigma}, N^{\Delta})$
Output: TCAM rule set $\mathscr{R}$

$\mathscr{R} \leftarrow \emptyset$, rule_id $\leftarrow 1$;
prefix_length$_{\text{src}} \leftarrow \log_2(N^{\Sigma})$;
prefix_length$_{\text{dst}} \leftarrow \log_2(N^{\Delta})$;
for $i \leftarrow 0$ **to** $N^{\Sigma} - 1$ **do**
  for $j \leftarrow 0$ **to** $N^{\Delta} - 1$ **do**
    src_ip_prefix $\leftarrow$ IntToBinary(i, prefix_length$_{\text{src}}$);
    dst_ip_prefix $\leftarrow$ IntToBinary(j, prefix_length$_{\text{dst}}$);
    src_pattern $\leftarrow$ src_ip_prefix + "X"$^{32-\text{prefix\_length}_{\text{src}}}$;
    dst_pattern $\leftarrow$ dst_ip_prefix + "X"$^{32-\text{prefix\_length}_{\text{dst}}}$;
    rule $\leftarrow$ CreateTCAMRule(src_pattern, dst_pattern, rule_id);
    $\mathscr{R} \leftarrow \mathscr{R} \cup \{\text{rule}\}$, rule_id $\leftarrow$ rule_id + 1;
return $\mathscr{R}$;

The calculated TCAM rules are then installed on the desired ingress routers, e.g., via NetConf or OpenFlow depending on the supported protocols. Although each rule is assigned an ID and therefore written to the TCAM in a specific order, all rules are mutually disjoint and can be matched in parallel without priority conflicts.

Table 4.1: *Example of installed TCAM rules for traffic images with resolution $\chi = (4, 4)$, indicating the four resulting traffic image columns.*

ID		Source IP Address (32 bits)				Destination IP Address (32 bits)				Count
		Byte 1	Byte 2	Byte 3	Byte 4	Byte 1	Byte 2	Byte 3	Byte 4	
Column 1	1	00XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	00XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	1045
	2	01XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	00XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	846
	3	10XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	00XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	591
	4	11XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	00XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	1388
Column 2	5	00XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	01XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	702
	6	01XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	01XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	1499
	7	10XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	01XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	1132
	8	11XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	01XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	517
Column 3	9	00XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	10XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	987
	10	01XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	10XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	1310
	11	10XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	10XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	742
	12	11XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	10XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	659
Column 4	13	00XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	11XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	1204
	14	01XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	11XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	1058
	15	10XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	11XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	880
	16	11XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	11XXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	1143

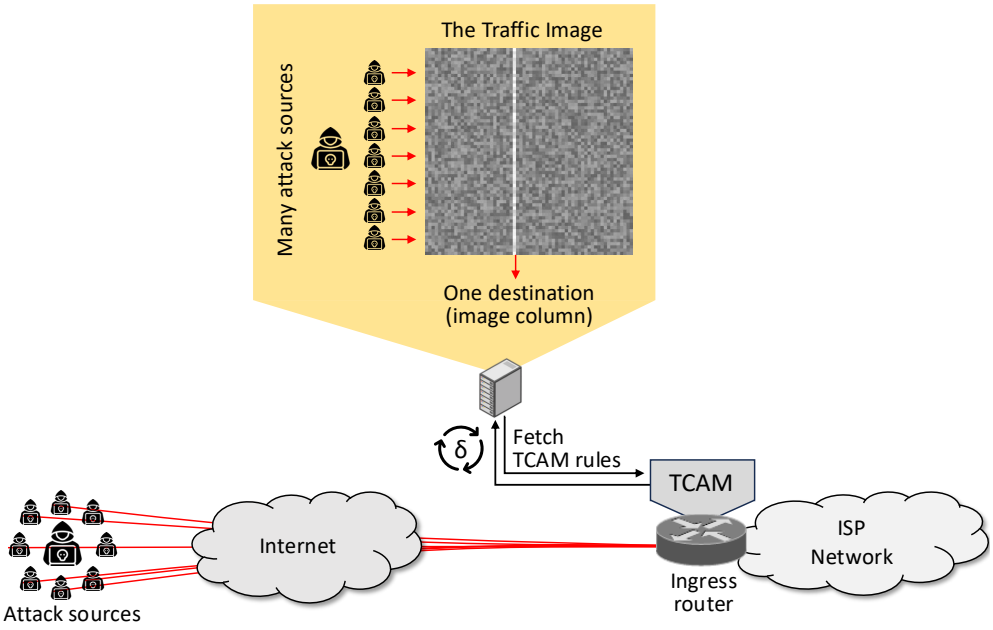


Figure 4.3: *Illustration of a DDoS attack manifesting in a traffic image column.*

As volumetric DDoS attacks originate from thousands of different attack sources (source IP addresses) and target a specific victim (destination IP address), these attacks manifest in individual **traffic image columns** (see Figure 4.3 that shows a traffic image with resolution $\chi = (256, 256)$ capturing an ongoing attack). A traffic image column corresponds to the set of pixels in the traffic image that share the same destination IP subnet, e.g., a traffic image with resolution $\chi = (8, 8)$ has eight traffic image columns (see Figure 4.1).

Definition 4.2: Traffic Image Column

Let ι be a **traffic image** according to Definition 4.1.
A **traffic image column** is a vertical slice of ι along a fixed **destination IP address subnet** IP_j^{Δ} for some $j \in \{1, 2, \ldots, N^{\Delta}\}$. Formally, a traffic image column X_j is the subset of a traffic image's pixels corresponding to

$$X_j = \left\{ (\mathrm{IP}_i^{\Sigma}, \mathrm{IP}_j^{\Delta}) \mid i = 1, 2, \ldots, N^{\Sigma} \right\}.$$

Table 4.1 shows an example of installed TCAM rules for a traffic image with resolution $\chi = (4, 4)$ and the resulting four traffic image columns.

Rule Fetching

Rule fetching is performed periodically per time window δ by the classification server. The server queries the installed TCAM rules' counters from the ingress routers and constructs the traffic image accordingly. For each fetched rule counter, the server maps the rule ID to the corresponding pixel in the traffic image and assigns the counter value to the pixel. To avoid resetting the TCAM rules' counters per time window after each rule fetching, the classification server stores the last fetched counter values. The current traffic image is then computed by calculating the difference between the current fetched counter values and the stored last fetched counter values. This requires the storage of two TCAM rules' counter values per ingress router.

Due to the additive nature of traffic images, the classification server can fetch TCAM rule counters from multiple ingress routers and aggregate them by adding up the corresponding pixel values. This enables the monitoring of traffic aggregated over multiple ingress points of the network.

The following algorithm outlines the rule fetching and the traffic image construction:

Algorithm 4: Rule Fetching and Traffic Image Construction

Input: Set of ingress routers $\mathscr{R} = \{r_1, \ldots, r_k\}$, Set of last counters per ingress router $\{\mathscr{C}_{last}^{r_1}, \ldots, \mathscr{C}_{last}^{r_k}\}$, Resolution $\chi = (N^{\Sigma}, N^{\Delta})$

Output: Aggregated traffic image I

```
1  I ← ZeroMatrix(N^Σ, N^Δ);
2  foreach router r in R do
3      C_r ← FetchCounters(r);
4      foreach counter c in C_r do
5          rule_id ← c.rule_id;
6          i ← ⌊(rule_id − 1)/N^Δ⌋;
7          j ← (rule_id − 1) mod N^Δ;
8          c_last ← GetLastCounter(C^r_last, rule_id);
9          I[i][j] ← I[i][j] + (c.count − c_last.count);
10     UpdateLastCounters(C^r_last, C_r);
11 return I;
```

This dissertation assumes that connectivity between the classification server and all ingress routers is established for rule fetching. Rule fetching introduces latency depending on the propagation delay between the classification server and the ingress routers. However, microflow-based monitoring approaches, e.g., utilizing NetFlow router exports, experience the same delay when receiving flow records from remote routers.

ML Classification
After the TCAM rule counters are fetched and the traffic image is constructed, it is passed to pre-trained ML models for classification. These ML models follow well-studied approaches from the area of computer vision to detect whether an attack is ongoing and to extract more detailed information about the attack, such as the destination IP subnet under attack. The following traffic image classification approaches are presented and evaluated in this chapter:

- **Binary Classification (Section 4.2)** – Assigning a binary label (attack or legitimate) to traffic images to detect ongoing attacks.
- **Time Series Binary Classification (Section 4.3)** – Assigning binary labels to sequences of traffic images to enable temporal correlation of consecutive traffic images for improved detection performance.
- **Multi-class Classification (Section 4.4)** – Assigning a label for each traffic image column to identify the destination IP subnet under attack.
- **Image Segmentation (Section 4.5)** – Assigning pixel-wise labels to traffic images to derive source and destination IP subnet pairs containing attack traffic for network filter rule creation.

The evaluation of the different traffic classification approaches is performed with the time window durations $\delta \in \{0.1s, \ldots, 1.0s\}$ and the traffic image resolutions $(N^{\Sigma}, N^{\Delta}) \in \{(8,8),(16,16),(32,32),(64,64),(128,128)\}$ when using real-world datasets (BelWue and MAWI+CAIDA). The lower bound of the time window duration is set to 0.1s to first ensure that enough packets are aggregated for attack detection and second to consider enough time for the rule fetching of the classification server from the ingress router's TCAM (see Section 4.1.1). The upper bound of $(128,128)$ for the traffic image resolution is chosen as the number of required TCAM rules scales quadratically with the traffic image resolution. A traffic image resolution of $(128,128)$ requires 16384 TCAM rules, which is feasible with modern hardware, while a traffic image resolution of $(256,256)$ would already require 65536 TCAM rules. As the authentic datasets stem from an Internet Service Provider (ISP) network (BelWue) and the Internet backbone (MAWI+CAIDA), the complete source and destination IP address spaces are monitored, i.e., $[\mathrm{IP}_A^{\Sigma}, \mathrm{IP}_{\Omega}^{\Sigma}] = [0, 2^{32}-1]$ and $[\mathrm{IP}_A^{\Delta}, \mathrm{IP}_{\Omega}^{\Delta}] = [0, 2^{32}-1]$.

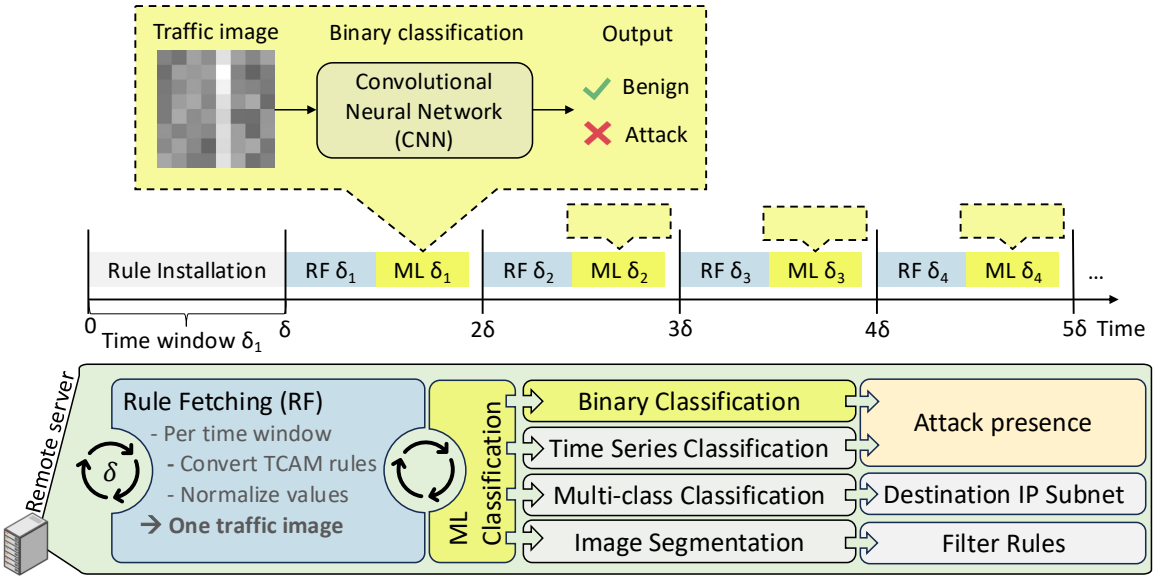


Figure 4.4: *Overview of the traffic image-based DDoS attack detection approach.*

4.2 Traffic Image-based DDoS Attack Detection

This section explains and evaluates the **binary classification** of traffic images for DDoS attack detection. The approach is outlined together with the traffic image labeling strategy and the employed ML model. The evaluation utilizes **real-world datasets** (MAWI+CAIDA and BelWü) as well as **synthesized datasets** to analyze trade-offs, such as the impact of the traffic image resolution on the detection performance. Furthermore, the generalization capability of the approach is evaluated over time.

Approach

The binary traffic image classification is outlined in Figure 4.4. During every time window δ_t, the resulting traffic image from rule fetching serves as input for a pre-trained Convolutional Neural Network (CNN). The CNN classifies the traffic image as either **legitimate** or **attack** depending on the presence of a DDoS attack in the corresponding time window. For training the CNN, the presence of a DDoS attack in a time window, i.e., the corresponding label, is defined based on the ground truth information of the utilized datasets. If at least one packet of the attack traffic arrived during the time window, the according traffic image is labeled as **attack**, otherwise as **legitimate**. Formally, the label y_t for a traffic image created during time window δ_t is defined as follows:

Definition 4.3: Binary Traffic Image Labeling

Let P_t be the set of arriving packets during time window δ_t, and $\mathscr{P}_{\text{attack}}$ be the set of packets belonging to a DDoS attack according to the ground truth. The label y_t for the traffic image created during time window δ_t is defined as

$$y_t = \begin{cases} 1, & \text{if } \exists p \in \mathscr{P}_{\text{attack}} : t_{\text{arrival}}(p) \in \delta_t \\ 0, & \text{otherwise} \end{cases}$$

where $t_{\text{arrival}}(p)$ denotes the arrival time of packet p.

Table 4.2: *CNN Architecture for Binary Traffic Image Classification*

Architecture	
Input Layer	Image input of shape $(N^{\Sigma}, N^{\Delta}, 1)$
Conv2D (32 filters, 3×3)	+ ReLU activation
Conv2D (32 filters, 3×3)	+ ReLU activation
MaxPooling2D (2×2)	+ Dropout(0.1)
Conv2D (32 filters, 3×3)	+ ReLU activation
Conv2D (32 filters, 3×3)	+ ReLU activation
MaxPooling2D (2×2)	+ Dropout(0.1)
Flatten	Fully connected layers
Dense (64)	Fully connected layer
Dense (32)	Fully connected layer
Output Dense (1)	Activation: `sigmoid`

The employed CNN architecture for binary traffic image classification is outlined in Table 4.2. It is inspired by VGGNet [SZ15], which demonstrated strong performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Rus+15], outperforming earlier architectures such as AlexNet [KSH12]. More recent CNN architectures that further improved over VGGNet regarding the ImageNet benchmark, such as GoogLeNet [Sze+14] and ResNet [He+15], are not considered as these models are substantially deeper and include more trainable paramters. Their effective training requires significantly larger datasets than are available for traffic image classification, in contrast to the scale of ImageNet. For this reason, the proposed architecture is a scaled-down version of VGGNet in terms of depth and the amount of trainable parameters.

The architecture consists of **two convolutional blocks**, each containing two convolutional layers followed by a max-pooling layer and a dropout layer. The convolutional layers utilize 32 filters with a kernel size of 3x3 and the Rectified Linear Unit (ReLU) activation function. After converting the output of the convolutional blocks into a one-dimensional vector (flattening), **two fully connected dense layers** with 64 and 32 neurons are used with the ReLU activation function. The output layer consists of a single neuron with a sigmoid activation function to produce a binary classification output between 0 and 1. The model is trained using the binary cross-entropy loss function and the Adam optimizer. All of the following experiments are conducted with this architecture. Each experiment run, i.e., for every combination of hyperparameters, is repeated twenty times. The datasets are shuffled and split into training and test sets with an 80/20 ratio with random seeds for each run, while the training set is further split into training and validation sets with a 80/20 ratio. The results present median metrics on the test set across all runs.

4.2.1 Evaluation with the MAWI+CAIDA Dataset

This section covers experiments with the MAWI+CAIDA dataset. The impact of the **time window duration** and the **traffic image resolution** on the detection performance is evaluated and explained. Furthermore, an experiment is conducted to evaluate the impact of the attack's data rate (compared to the data rate of the legitimate traffic) on its detectability to show potential limitations of the traffic image-based detection approach.

Time Window Duration and Traffic Image Resolution

The first experiment evaluates the impact of the time window duration δ and the traffic image resolution $\chi = (N^{\Sigma}, N^{\Delta})$ on the detection performance with the MAWI+CAIDA dataset. Figure 4.5 presents the evaluation results (accuracy, precision, and recall, from top to bottom) for all combinations of time window durations and traffic image resolutions. The binary classification of traffic images achieves **perfect accuracy, precision, and recall** for all parameter combinations, except for the time window duration of 0.4s, where the recall (and consequently the accuracy) drops below 1.0 across all traffic image resolutions, except for (64,64) that maintains a recall of 1.0. The reason for this anomaly is the traffic image labeling strategy that assigns the attack label to any traffic image containing at least one attack packet. This can lead to traffic images in the test set that contain only a few number of attack packets and can therefore not be differentiated from legitimate traffic images by the CNN, resulting in false negatives. This effect can be particularly pronounced for certain time window durations, where the start of the time window aligns right before the attack end or the end of the time window aligns right after the attack start.

Figure 4.6 illustrates why the detection performance is perfect across almost all parameter combinations. It presents the number of packets aggregated into traffic images with a time window duration of 0.1 for the MAWI+CAIDA dataset. It further shows the attack

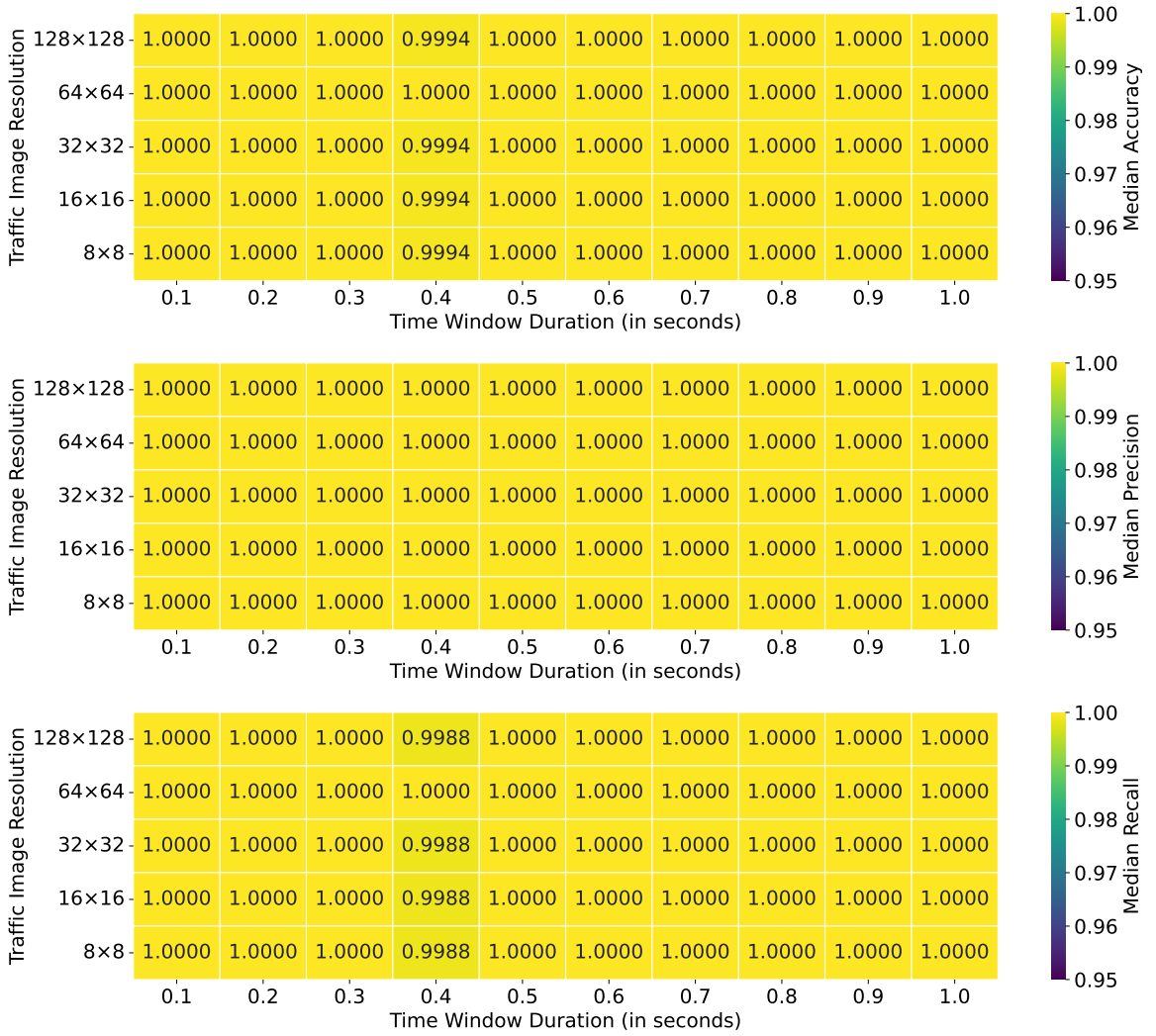


Figure 4.5: *Evaluation results for binary traffic image classification across different time window durations and traffic image resolutions with the MAWI+CAIDA dataset.*

period indicated by the red area. During the attack period, the packet count per time window is significantly larger than during periods with only legitimate traffic. Therefore, the detection decision can only be made on the **overall packet count** per time window, which is **contained in the traffic image** representation (packet counts per subnet pair). This characteristic makes the traffic image a suitable monitoring approach to detect high volume DDoS attacks. However, to avoid that the CNN learns to detect attacks solely based on the overall packet count, the following experiment is conducted.

Attack Intensity and Traffic Image Resolution

This experiment evaluates the detection performance with the MAWI+CAIDA dataset, but the CAIDA attack traffic is scaled down in its intensity, i.e., the fraction of attack

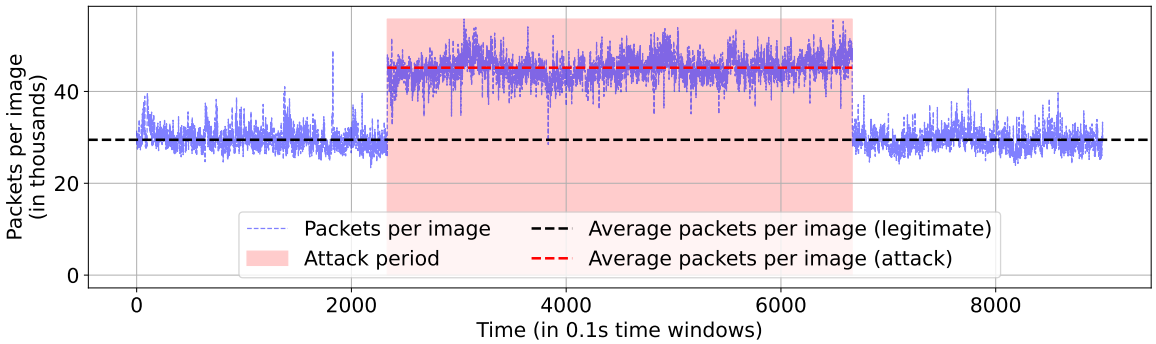


Figure 4.6: *Packet count per traffic image for the MAWI+CAIDA dataset and time window duration 0.1s.*

traffic compared to the legitimate traffic. For example, an attack intensity of 10% means that during a time window with 1000 legitimate packets arriving, 100 attack packets arrive. For each traffic image, the attack packet counts are linearly scaled over all pixels to achieve the desired attack intensity while maintaining the original source and destination IP distribution. The goal of this experiment is to evaluate the detection performance in a scenario where the CNN model cannot rely on the packet count alone for attack detection, but also needs to learn the IP address distribution of the traffic. To force this behavior, the time window duration is fixed to 0.1s (where the least packets are aggregated per time window) and the attack intensity is varied.

Definition 4.4: Attack Intensity

Let $P_t^{\text{legitimate}}$ be the set of legitimate packets arriving during time window δ_t, and P_t^{attack} be the set of attack packets arriving during time window δ_t. The attack intensity α_t for time window δ_t is defined as

$$\alpha_t = \frac{|P_t^{\text{attack}}|}{|P_t^{\text{legitimate}}|}$$

The evaluation results for the attack intensities $\alpha \in \{1\%, \ldots, 10\%\}$ are shown in Figure 4.7. The results indicate that the detection performance is 1.0 (all metrics) for resolutions $N^{\Sigma}, N^{\Delta} \geq 64$ and attack intensities $\alpha \geq 6\%$. The recall degrades for some attack intensities $\alpha < 6\%$ and $N^{\Sigma}, N^{\Delta} \leq 32$, which indicates that not all traffic images during the attack period are correctly classified as attack. This result is reasonable as the attack traffic blends more into the legitimate traffic the lower the attack intensity is, which leads to an indistinguishability between legitimate and attack traffic images. The precision degrades for some attack intensities $\alpha < 5\%$ and $N^{\Sigma}, N^{\Delta} \leq 16$ (lower ranges than for recall) for the same reason. The overall accuracy degrades for lower resolutions

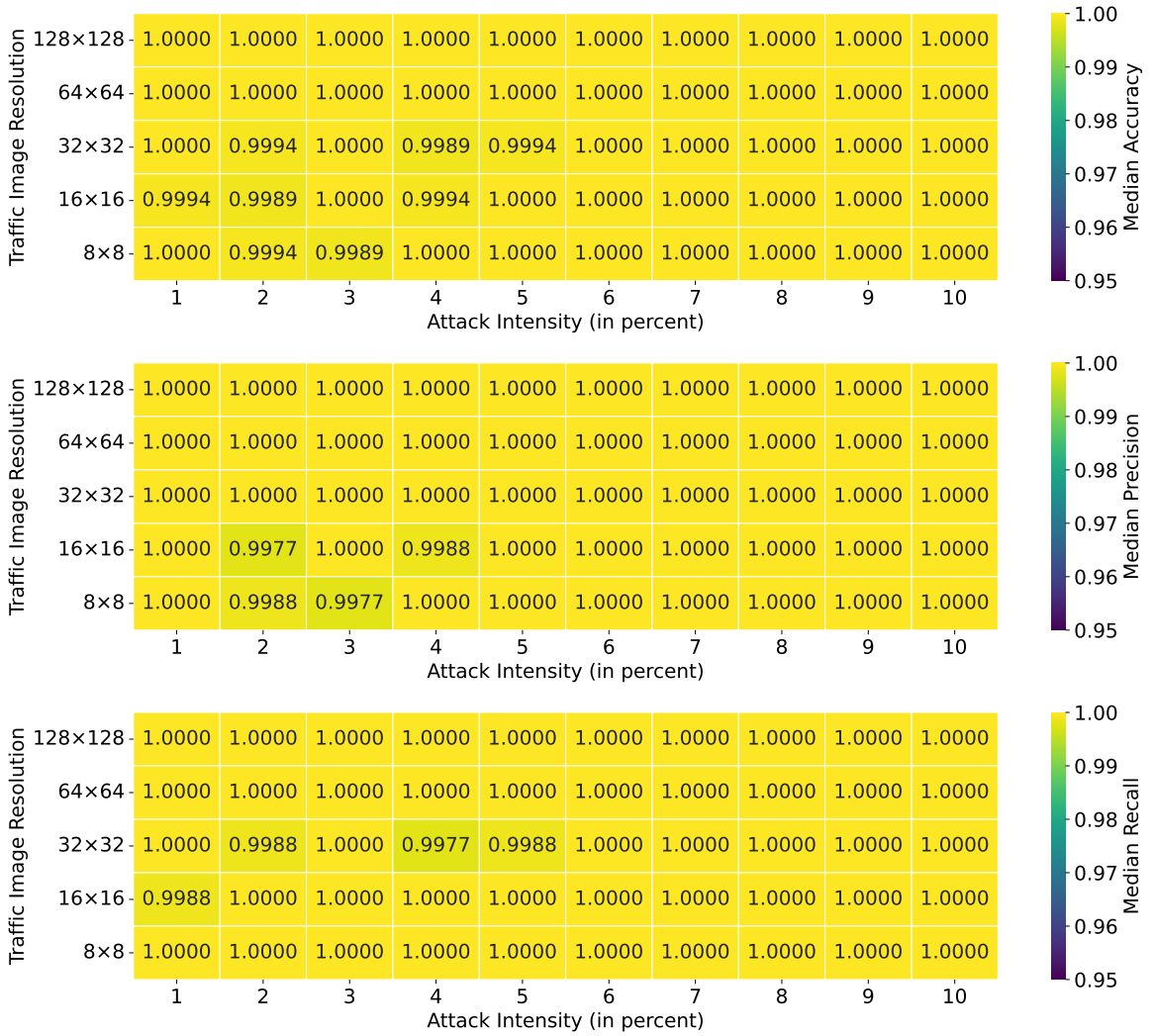


Figure 4.7: *Evaluation results for binary traffic image classification across different attack intensities and traffic image resolutions with the MAWI+CAIDA dataset.*

and lower attack intensities as information about the packet distribution across the source and destination IP address space is lost. Therefore, a trade-off between the required Ternary Content Addressable Memory (TCAM) rules (and the memory utilization at the classification server) and the detection performance exists. However, why does the DDoS detection perform well even for low attack intensities?

Figure 4.8 illustrates traffic images in RGB format for the different resolutions $(N^{\Sigma}, N^{\Delta}) \in \{(8,8),(16,16),(32,32),(64,64),(128,128)\}$ from left to right. The RGB representation is only for the purpose of illustration, as the actual traffic images are single-channel images with packet counts. Each image is created over the whole attack period and shows the aggregated attack traffic in the red channel, while the legitimate traffic is shown in the

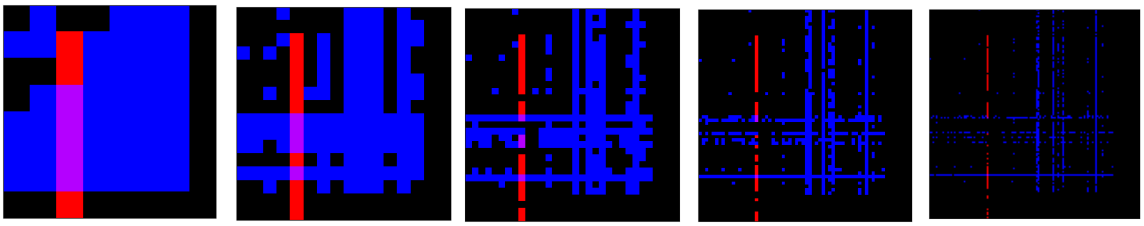


Figure 4.8: *IP address distributions in the MAWI+CAIDA dataset for attack (red) and legitimate (blue) traffic. Resolutions (8,8)... (128,128) from left to right.*

blue channel. The pixel channel values are either 0 (no traffic) or 255 (traffic present) and therefore only show the distribution and not the intensity of the traffic. Red pixels contain only attack traffic during the whole attack period, blue pixels contain only legitimate traffic, and purple pixels contain both traffic types. Black pixels contain no traffic at all. The detection performance with the MAWI+CAIDA dataset and low attack intensities can be attributed to the clear separation of attack and legitimate traffic (many red and blue pixels and few purple pixels), which is enabled through the two-dimensional traffic image. This separation becomes more pronounced with higher traffic image resolutions, which make differences between attack and legitimate traffic more visible. Therefore, the traffic image is a suitable traffic representation for DDoS attack detection even at low attack intensities because it captures the IP address distribution of the attack and the legitimate traffic and enables a differentiation.

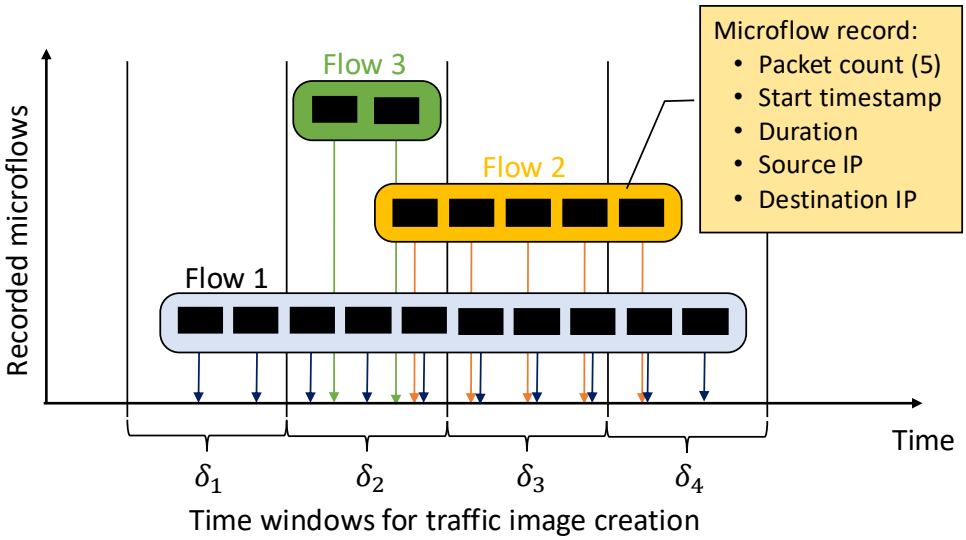


Figure 4.9: *Traffic image creation from microflow records.*

4.2.2 Evaluation with the BelWue Dataset

This section contains experiments with the BelWue dataset to evaluate the traffic image-based DDoS detection approach in a real-world Internet Service Provider (ISP) network scenario. Before the explanation and evaluation of the experiments, the dataset creation is outlined, followed by an overview of the IP address distribution of the attack and the legitimate traffic, as well as the attack intensity.

Dataset Creation – Adaptability to Microflow-based Monitoring
The previous experiments (MAWI+CAIDA) utilized traffic images created from recorded Packet Capture (PCAP) files, where all arriving packets were available. However, if a network administrator has already deployed a microflow-based monitoring solution, e.g., based on NetFlow or IPFIX exports from border routers, traffic images can also be created from the monitored microflows (see Figure 4.9) if their records contain at least the following information: the source and destination IP addresses, the packet count, the start timestamp, and the microflow's duration. From this information, traffic images at arbitrary resolutions can be constructed, as the complete IP addresses are available. Traffic images, in contrast to microflows, are time window-based traffic aggregates. To create traffic images from microflow records, all microflows need to be mapped to the corresponding time windows based on their start timestamps and durations. If a microflow is active during a single time window, all packets of the microflow (packet count) are aggregated into the corresponding pixel of the traffic image according to the microflows source and destination IP addresses. If a microflow spans multiple time windows, the flow's packet arrivals are uniformly distributed over the spanned duration and the according fraction of packets is aggregated into each affected time window's traffic image pixel.

This microflow-based traffic image creation is used in this section to evaluate the traffic image-based DDoS detection approach with the BelWue dataset and has been tested with NetFlow- and IPFIX-based monitoring of a major Tier-1 ISP [KLZ25] (also see results in Section 5.2.1). Although the traffic images are created from microflow records, the traffic image-based DDoS detection still benefits from a significantly reduced memory utilization and ML classification count at the classification server (see Section 4.2.6).

Overview of the IP Address distributions and the Packet Count per Traffic Image
Figure 4.10 presents the number of packets (in millions) aggregated per traffic image with a time window duration of 0.1s for the BelWue dataset. The attack period is indicated by the red area. The attack traffic blends into the legitimate traffic volume-wise as the average packet count per traffic image during the attack period is smaller than during certain periods with only legitimate traffic. Therefore, the detection decision cannot be made solely on the overall packet count per traffic image.

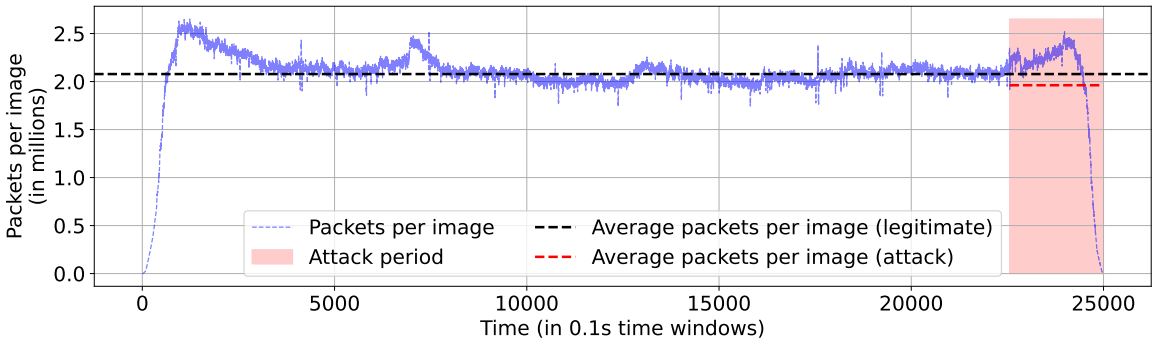


Figure 4.10: *Packet count per traffic image for the BelWue dataset and time window duration 0.1s.*

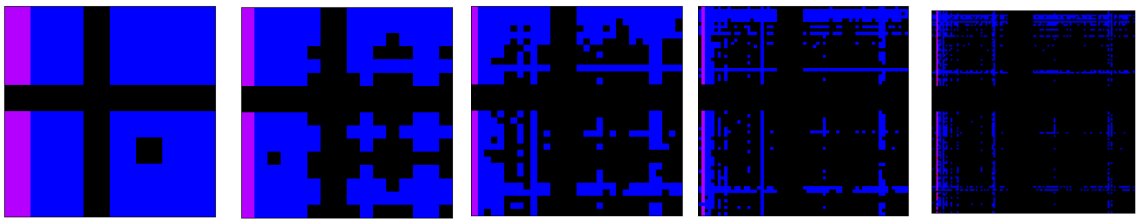


Figure 4.11: *IP address distributions in the BelWue dataset for attack (red) and legitimate (blue) traffic. Resolutions (8,8)... (128,128) from left to right.*

Figure 4.11 illustrates traffic images in RGB format (compare to Figure 4.8) for the different resolutions $(N^{\Sigma}, N^{\Delta}) \in \{(8,8),(16,16),(32,32),(64,64),(128,128)\}$ from left to right. Again, each image is created over the whole attack period and shows the aggregated attack traffic in the red channel, while the legitimate traffic is shown in the blue channel. The pixel channel values are either 0 (no traffic) or 255 (traffic present) and therefore only show the distribution and not the intensity of the traffic. Only purple pixels exist for every resolution (in the traffic image column with the attack traffic), indicating that attack and legitimate traffic share the same source and destination IP subnets. Therefore, the detection cannot be made solely on the IP address distribution, as for the MAWI+CAIDA dataset.

To summarize, the detection decision with the BelWue dataset can neither be made solely on the overall packet count per time window nor on the IP address distribution of attack and legitimate traffic. To evaluate the detection performance under these conditions, the following experiment is conducted.

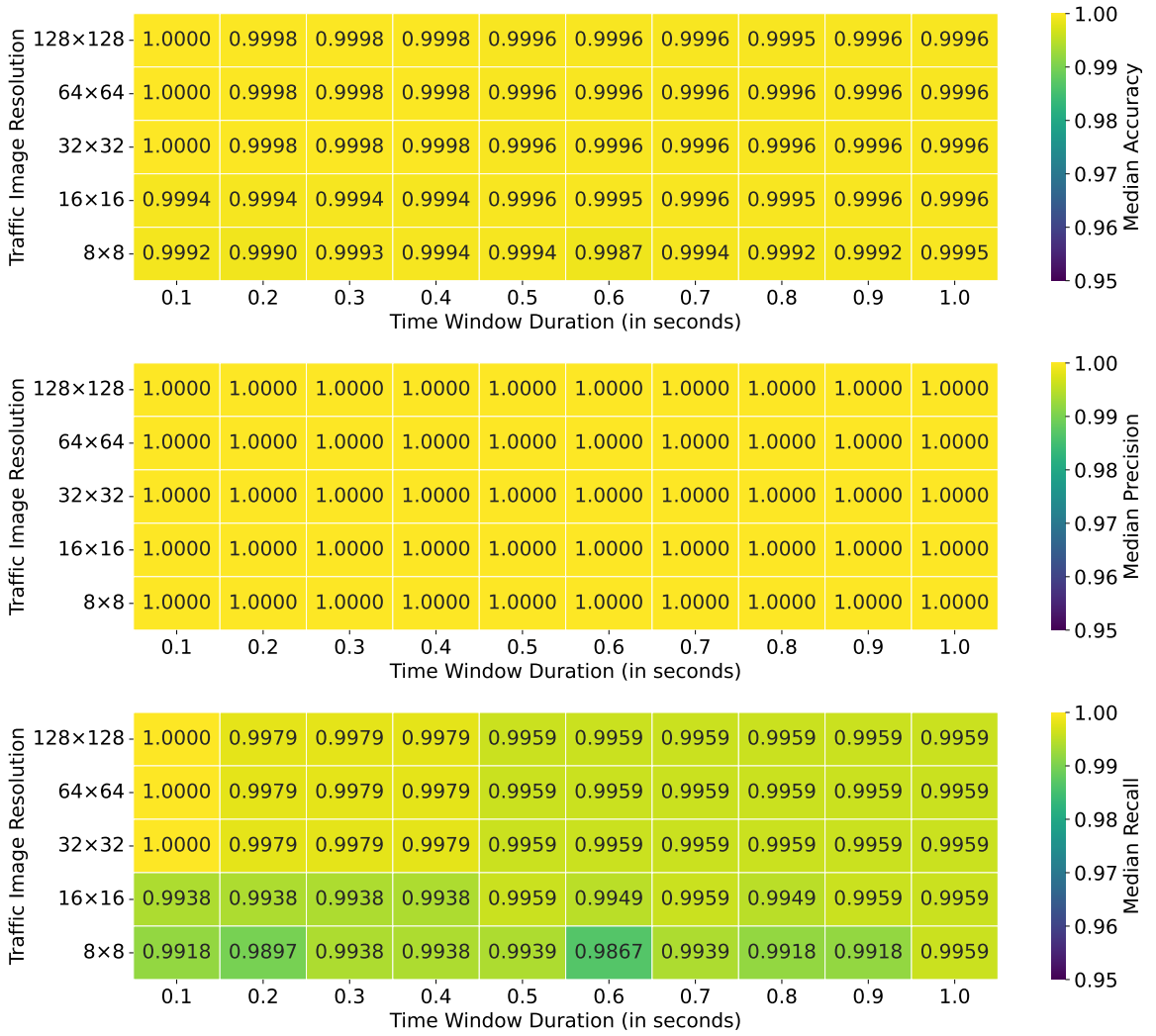


Figure 4.12: *Evaluation results for binary traffic image classification across different time window durations and traffic image resolutions with the BelWue dataset.*

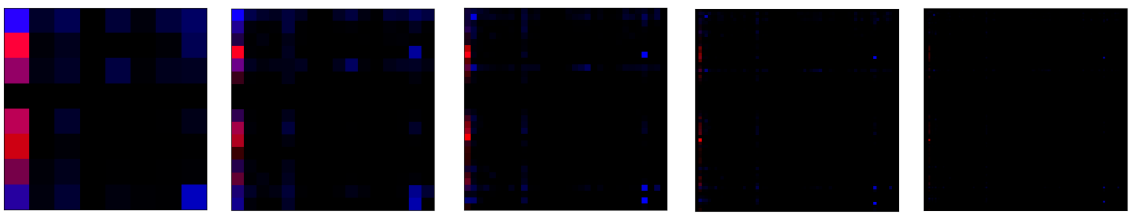


Figure 4.13: *Packet distributions in the IP address space (BelWue dataset) for attack (red) and legitimate (blue) traffic and the resolutions (8,8)...(128,128).*

Traffic Image Resolution and Time Window Duration
This experiment evaluates the impact of the time window duration δ and the traffic image resolution $\chi = (N^{\Sigma}, N^{\Delta})$ on the detection performance with the BelWue dataset. Figure 4.12 presents the evaluation results (accuracy, precision, and recall, from top to bottom) for all combinations of time window durations and traffic image resolutions. The first observation is that the precision is 1.0 independent of the time window duration and traffic image resolution. This means that no false positives occur, which is crucial for DDoS attack detection in a real-world scenario to avoid false alarms and the unnecessary deployment of mitigation mechanisms. The second observation is that perfect detection is only achieved for the time window duration $\delta = 0.1$s and the resolutions $N^{\Sigma}, N^{\Delta} \geq 32$. For all other parameter combinations, a trend of monotonically decreasing recall is visible with decreasing resolution. Therefore, a trade-off between the required TCAM rules (and the memory utilization at the classification server) and the recall exists also for the BelWue dataset.

The third observation is that the recall monotonically decreases with increasing time window durations for the resolutions $N^{\Sigma}, N^{\Delta} \geq 32$. This effect can be attributed to the amount of available training data per time window duration. Longer time window durations lead to fewer traffic images in total for training and testing, e.g., a time window duration of 1.0s leads to ten times fewer traffic images than a time window duration of 0.1s. For the BelWue dataset, this results in a decrease from approximately 25.000 traffic images in total for $\delta = 0.1$s to 2.500 traffic images in total for $\delta = 1.0$s. However, the worst achieved recall is 0.9867 for the time window duration $\delta = 0.6$s and the resolution $N^{\Sigma}, N^{\Delta} = 8$. Why is the detection performance still high, although the detection decision cannot be made on the overall packet count or the IP address distribution?

Figure 4.13 illustrates traffic images in RGB format (compare to Figure 4.11) for different resolutions $(N^{\Sigma}, N^{\Delta}) \in \{(8,8), (16,16), (32,32), (64,64), (128,128)\}$ from left to right. Each image is created over the whole attack period and shows the aggregated attack traffic in the red channel, while the legitimate traffic is shown in the blue channel. In contrast to Figure 4.8 and Figure 4.11, the pixel channel values are linearly scaled to the range $[0, 255]$ according to the overall packet count in the traffic image. This illustration reveals the packet distribution over IP address spaces for attack and legitimate traffic. A purple pixel now indicates the same amount of attack and legitimate packets, while a pronounced red or blue pixel indicates a higher amount of attack or legitimate packets, respectively. Although the IP address distributions of the attack and the legitimate traffic are the same (only purple pixels exist in Figure 4.11), the packet distributions differ. The traffic image-based DDoS detection can therefore not only learn differences in the overall packet count and the IP address distribution, but also differences in the packet distribution over the IP address space, which makes a good detection performance possible in the BelWue scenario.

To evaluate the detection performance on a dataset that neither allows a decision based on the overall packet count, nor on the IP address distribution, nor on the packet distribution, the following experiments are conducted with synthesized datasets.

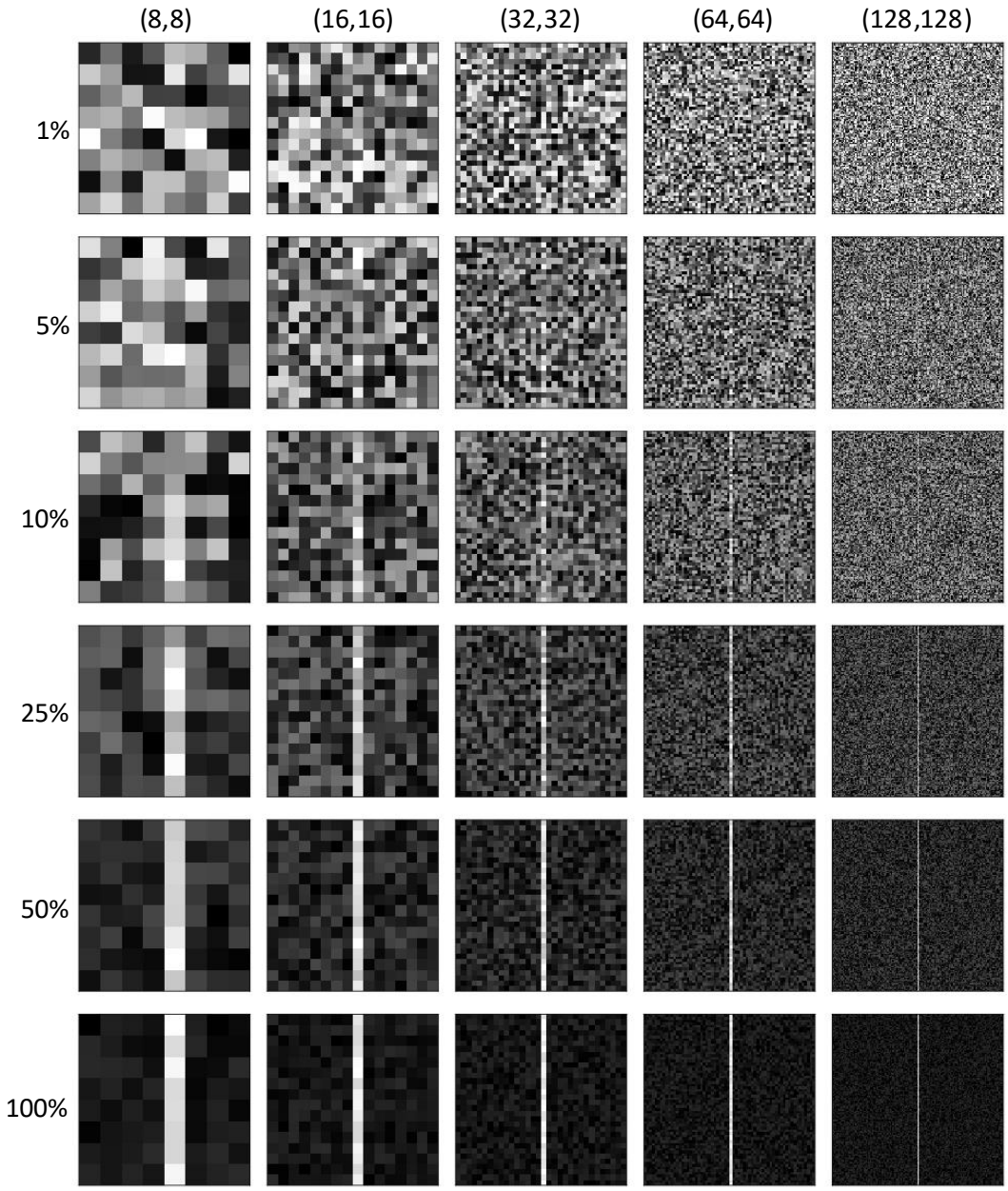


Figure 4.14: *Synthesized traffic images (uniform distribution) in greyscale representation for different attack intensities (vertical) and traffic image resolutions (horizontal).*

4.2.3 Evaluation with Synthesized Datasets

To conduct experiments, where the source IP address distribution and the packet distribution of attack and legitimate traffic are identical, synthesized datasets are created. These datasets contain traffic images with time window durations of 0.1s and a legitimate packet count of 100.000 per time window and traffic image. The legitimate packets are distributed uniformly at random over the source and destination IP address space. The attack packet count is varied for the attack intensities $\alpha \in \{0.1\%, \ldots, 1.0\%\}$ and the attack packets are also distributed uniformly at random over the source IP address space targeting a single destination IP address. A broader range of attack intensities has been evaluated, but the range $\alpha \in \{0.1\%, \ldots, 1.0\%\}$ constitutes the relevant range to show the limitations of the traffic image-based DDoS detection approach. Figure 4.14 illustrates traffic images, in the greyscale representation, for different attack intensities and resolutions. Attack intensities below one percent are not shown as the attack pattern is already invisible for the human eye at the attack intensity of 1.0%.

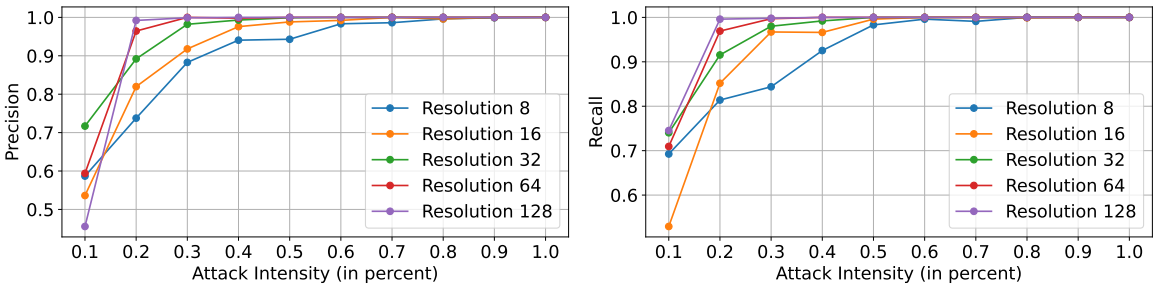


Figure 4.15: *Precision and recall for binary traffic image classification across different attack intensities and traffic image resolutions with synthesized datasets.*

Figure 4.15 shows the average precision (left) and recall (right) for all combinations of attack intensities and traffic image resolutions over 20 runs each with random seeds. First, it is observable that the precision increases monotonically with increasing attack intensities for all resolutions. The recall increases monotonically with increasing attack intensities except for one outlier at the resolution $\chi = (8, 8)$ and the attack intensity $\alpha = 0.7\%$. The correlation between an increasing attack intensity and an increasing detection performance is expected, as an increasing attack intensity leads to more attack packets aggregated in one traffic image and therefore to a more pronounced attack pattern. Second, it is observable that higher resolutions lead to a better precision and recall for the attack intensities $0.2 \leq \alpha \leq 0.8$. This result is also expected, as higher resolutions preserve more information about the packet distribution, especially over the destination IP address space (source IP distribution of attack and legitimate traffic are the same), which is crucial to differentiate between DDoS attack and legitimate traffic. For the attack intensity $\alpha = 0.1\%$, the precision and recall significantly degrade for all

resolutions, indicating a limitation of the traffic image-based DDoS detection approach. At $\alpha = 0.1\%$, the resolutions $N^{\Sigma}, N^{\Delta} \leq 32$ achieve a better precision and recall than the resolutions $N^{\Sigma}, N^{\Delta} \geq 64$. However, transferring the results to the BelWue scenario with a peak data rate of 37.26 Gbit/s (see Figure 2.5), an attack intensity of $\alpha = 0.1\%$ would correspond to an attack data rate of 37.26 Mbit/s, which cannot be considered a volumetric DDoS attack anymore. The third observation is that all resolutions achieve a precision and recall of 1.0 for the attack intensities $\alpha \geq 0.8\%$. The attack traffic is so prominent for these attack intensities that the CNN can perfectly distinguish between attack and legitimate traffic images.

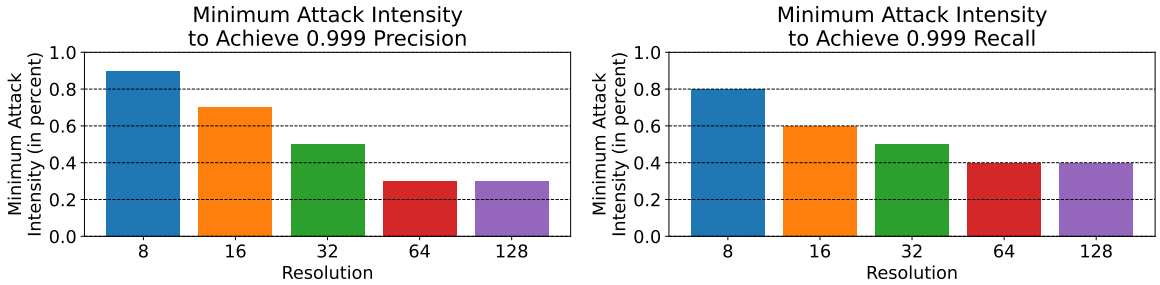


Figure 4.16: *Minimum detectable attack intensity with at least a precision (left) and recall (right) of 0.999 for different traffic image resolutions with synthesized datasets.*

Figure 4.16 presents the previous results from a different perspective. It shows the minimum attack intensity that can be detected with at least a precision (left) and recall (right) of 0.999 for different traffic image resolutions. The threshold of 0.999 is chosen for both metrics and referred to as reasonable detection performance for attack detection, if both metrics reach at least this value or exceed it.

The above illustration clarifies the trade-off between the required TCAM rules (and the memory utilization at the classification server), which scale quadratically with the traffic image resolution, and the detection performance. Higher resolutions lead to a lower minimum detectable attack intensity for both metrics, e.g., a resolution of $(8, 8)$ enables a reasonable detection of attacks with an intensity of at least 0.9% while a resolution of $(128, 128)$ enables the detection of attacks with an intensity of at least 0.4%. Furthermore, it is observable that the minimum attack intensity to achieve a precision of 0.999 is lower (0.3%) for resolutions $N^{\Sigma}, N^{\Delta} \geq 64$ than the minimum attack intensity to achieve a recall of 0.999 (0.4%). This observation covers the previous results with the BelWue dataset (see Figure 4.12), where the precision is 1.0 and only the recall degrades for lower resolutions and a time window duration of $\delta = 0.1$s.

4.2.4 Generalization Capabilities

This experiment is designed to evaluate the generalization capabilities of the traffic image-based DDoS detection approach over time. The MAWI+CAIDA dataset is utilized, where the CAIDA attack traffic is injected into eight consecutive days of MAWI traffic recorded in July 2025 (July 1st to July 8th). The CNN model is trained on the traffic images of the first day (July 1st) and evaluated on the traffic images of the seven remaining days (July 2nd to July 8th) to see if the model, trained after the first attack occurrence, is able to detect the attack in the future with varying legitimate traffic. The time window duration is fixed to 0.1s and the traffic image resolution is in the range $N^{\Sigma}, N^{\Delta} \in \{8, 16, 32, 64, 128\}$.

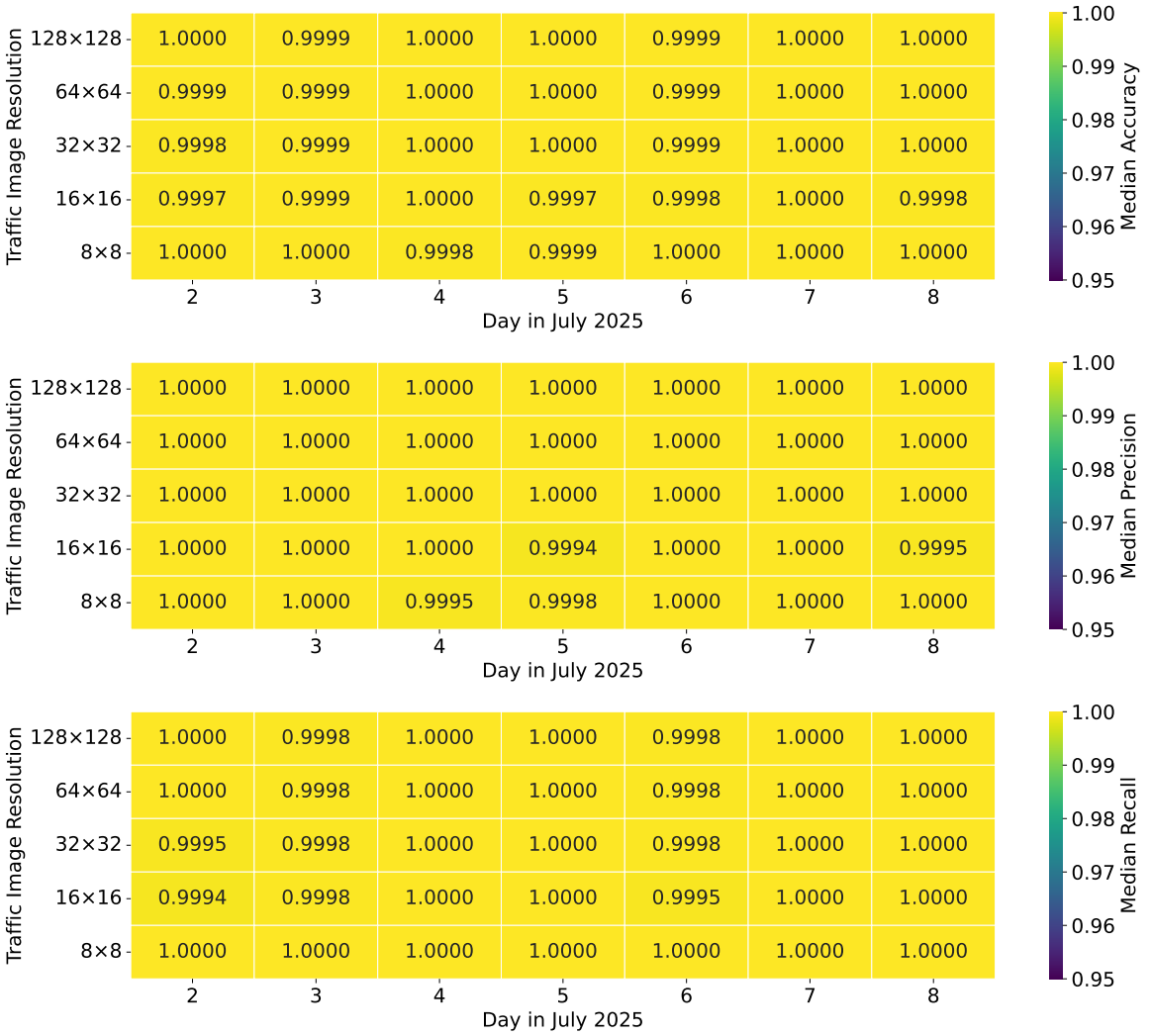


Figure 4.17: *Evaluation results for binary traffic image classification across different days with the MAWI+CAIDA dataset.*

Figure 4.17 presents the evaluation results (accuracy, precision, and recall, from top to bottom) for all combinations of testing days (July 2nd to July 8th) and traffic image resolutions ($N^{\Sigma}, N^{\Delta} \in \{8, 16, 32, 64, 128\}$). The results show that the attack can also be detected on all testing days with a precision larger than 0.9994 and a recall larger than 0.9994. The detection on July 7th achieves perfect precision and recall for all resolutions. The generalization capability is a crucial property for a DDoS detector that is deployed in a real-world scenario to avoid frequent retraining of the model.

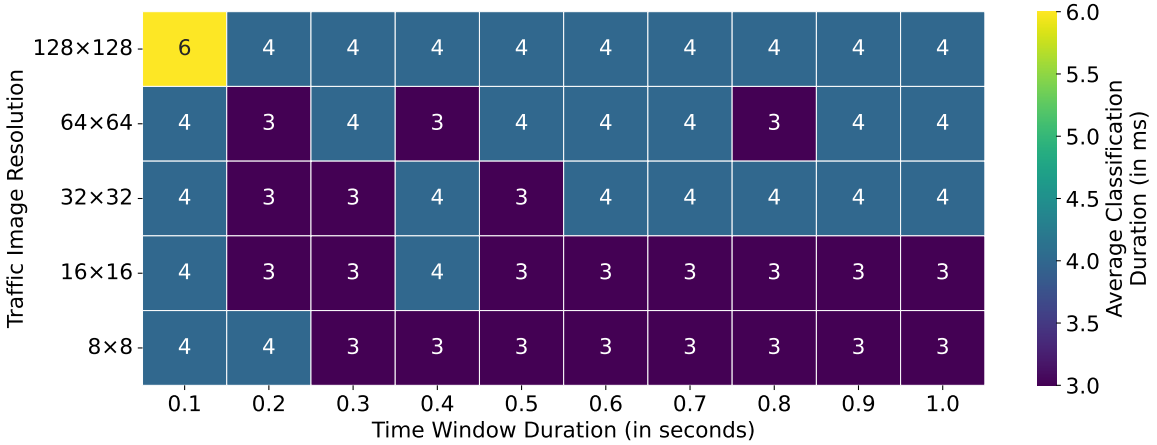


Figure 4.18: *Average classification duration per traffic image for different time window durations and traffic image resolutions.*

4.2.5 Classification Duration

The following experiment evaluates the duration the CNN model requires to classify a single traffic image. The goal of this experiment is to show that the classification is not a bottleneck in the detection pipeline, i.e., the classification duration is shorter than the time window duration to avoid a backlog of outdated traffic images and maintain a reaction time of at most two time window durations.

Figure 4.18 shows the average classification duration per traffic image of 100 runs for different time window durations and traffic image resolutions. The results indicate that neither the time window duration nor the traffic image resolution have an impact on the classification duration. These results are expected, as for all parameter combinations the same CNN architecture is trained and used for classification. The classification duration is always in the range of 3ms to 6ms, which is shorter than the smallest time window duration of 0.1s. Therefore, the classification is not a bottleneck in the detection pipeline.

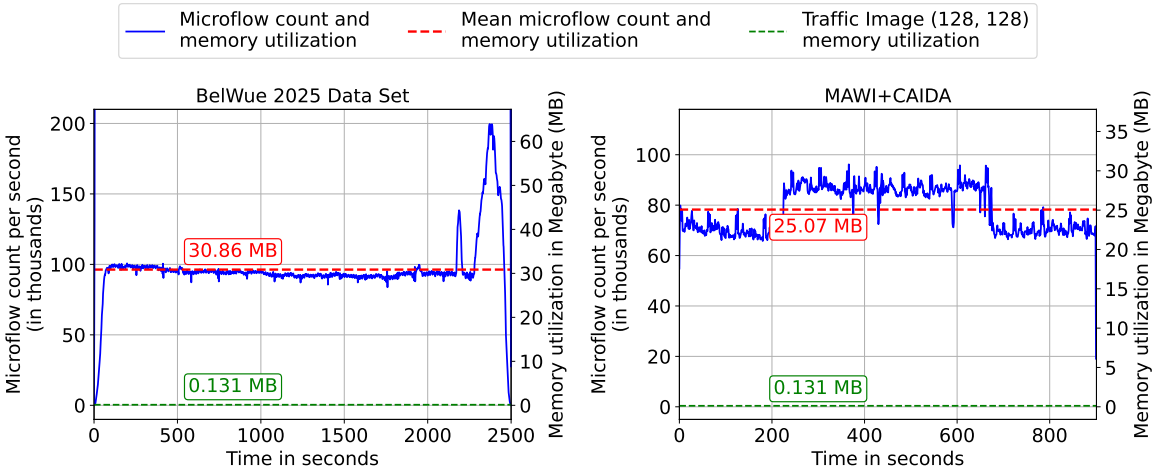


Figure 4.19: *Memory utilization comparison between the traffic image-based DDoS detection and a microflow-based approach for the MAWI+CAIDA and BelWue datasets.*

4.2.6 Comparison to Microflow-based Approaches

This section compares the memory utilization and the classification count of the traffic image-based DDoS detection approach to any approach relying on microflow-based classification. Traffic image-based detection utilizes memory corresponding to the chosen traffic image resolution. For every pixel, a memory utilization of 4 bytes is assumed. This results in a total memory utilization of $4 \cdot N^{\Sigma} \cdot N^{\Delta}$ bytes per traffic image at the classification server, i.e., 65.536 kB for a traffic image resolution of $(128, 128)$. However, to create the latest traffic image at the classification server after rule fetching, the previous traffic image is required to be stored (see Section 4.1.1). Therefore, for the highest tested resolution of $(128, 128)$, a total memory utilization of $2 \cdot 65.536 = 131.072$ kB is required at the classification server. At the ingress router, $N^{\Sigma} \cdot N^{\Delta}$ TCAM rules are required, e.g., 64 rules for the resolution $(8, 8)$ and 16.384 rules for resolution $(128, 128)$. In contrast, microflow-based approaches require memory per microflow at the classification server and the ingress router. This comparison assumes that each microflow record stores 84 metrics (e.g., CICFLowMeter), of which each metric requires 4 bytes. This results in a memory utilization of 336 bytes per microflow record at the classification server.
Figure 4.19 presents the memory utilization comparison between the traffic image-based and microflow-based DDoS detection for the BelWue (left) and MAWI+CAIDA (right) datasets. The required memory to store the microflows is shown as blue line, the average memory utilization using microflows is indicated as horizontal dashed red line, and the memory utilization for traffic images with the resolution $(128, 128)$ is shown as dashed horizontal green line.

For the BelWue scenario, the average memory utilization is 30.86 MB for microflows and 0.131 MB for traffic images with the resolution $(128, 128)$. The required memory for the traffic images is fixed during the whole deployment and does not depend on the traffic volume or the number of active microflows. For example, the required memory to store microflows more than doubles compared to the average during the second wave of the DDoS attack. The required amount of ML classifications reduces from almost 100.000 classifications per second (corresponds to microflows per second) on average for microflows to **one classification per selected time window duration** for traffic images.

For the MAWI+CAIDA scenario, the memory utilization and classification count improvement is in the same order of magnitude. The average memory utilization is 25.07 MB for microflows and 0.131 MB for traffic images with the resolution $(128, 128)$. Also the amount of required classifications reduces from almost 80.000 classifications per second on average for microflows to one classification per selected time window duration for traffic images.

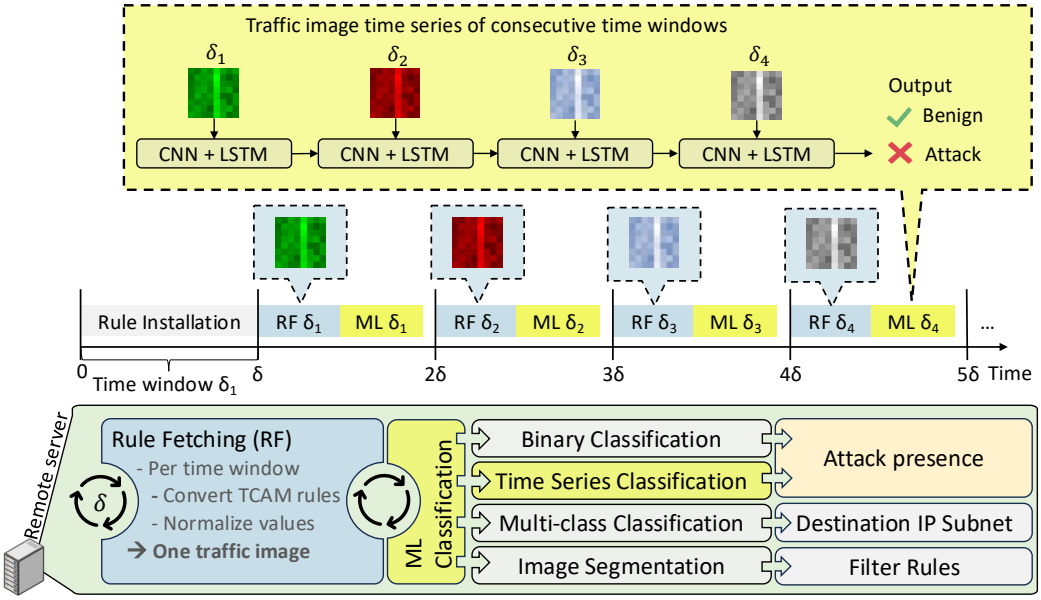


Figure 4.20: *Illustration of the time series traffic image classification approach.*

4.3 Traffic Image Series-based DDoS Attack Detection

This section explains and evaluates the **time series classification** of traffic images for DDoS attack detection. The approach is outlined together with the labeling strategy for traffic image series and the employed ML model.

Approach

A traffic image series consists of multiple traffic images created in consecutive time windows (see Figure 4.20). The series length L defines the number of traffic images in the time series. After each time window δ_t, and after the latest traffic image is created, the last L traffic images are classified by a pre-trained neural network. The neural network is a stacked Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) architecture that enables capturing both spatial (IP address distributions, packet distributions, and the overall packet count) and temporal characteristics (traffic development over time) of the traffic image series. The output is a binary classification of the traffic image series as either legitimate or attack depending on the presence of a DDoS attack in the latest traffic image of the series.

The labeling of traffic image series is defined based on the ground truth information of the utilized datasets. If at least one packet of the attack traffic arrived during the latest time window of the series, the corresponding traffic image series is labeled as **attack**, otherwise as **legitimate**. Formally, the label y_t for a traffic image series ending at time window δ_t is defined as follows:

Definition 4.5: Binary Traffic Image Series Labeling

Let δ_{t-L+1} to δ_t be the time windows of the traffic image series of length L, P_t be the set of packets arriving during time window δ_t, and $\mathscr{P}_{\text{attack}}$ be the set of packets belonging to a DDoS attack according to the ground truth. The label y_t for the traffic image series ending at time window δ_t is defined as

$$y_t = \begin{cases} 1, & \text{if } \exists p \in \mathscr{P}_{\text{attack}} : t_{\text{arrival}}(p) \in \delta_t \\ 0, & \text{otherwise} \end{cases}$$

where $t_{\text{arrival}}(p)$ denotes the arrival time of packet p.

The traffic images are stored in a first-in-first-out queue of length L. After each time window δ_t and if the queue is full, the latest traffic image is added to the queue, and the oldest traffic image is removed. L time windows after the start of the monitoring, the queue contains L traffic images. During every subsequent time window, a classification is performed. Therefore, the reaction time is not impacted by the series length L, e.g., by waiting for L new time windows.

Table 4.3: *CNN-LSTM Architecture for Binary Traffic Image Series Classification*

Architecture	
Input Layer	Image input of shape $(L, N^{\Sigma}, N^{\Delta}, 1)$
Conv2D (32 filters, 3×3)	+ ReLU activation
Conv2D (32 filters, 3×3)	+ ReLU activation
MaxPooling2D (2×2)	+ Dropout(0.1)
Conv2D (32 filters, 3×3)	+ ReLU activation
Conv2D (32 filters, 3×3)	+ ReLU activation
MaxPooling2D (2×2)	+ Dropout(0.1)
Flatten	Fully connected layers
LSTM (64 units)	
Dense (64)	Fully connected layer
Dense (32)	Fully connected layer
Output Dense (1)	Activation: `sigmoid`

The CNN-LSTM architecture for binary traffic image series classification is outlined in Table 4.3. The architecture follows the same structure as the binary traffic image classification (see Table 4.2) (as this architecture has already been optimized for traffic image classification), with the addition of an **LSTM layer** with 64 units after the flattening layer. The number of units in the LSTM layer has been determined through systematic detection performance evaluation with different unit counts (8, 16, 32, 64, 128), where 64 units achieved the best performance.

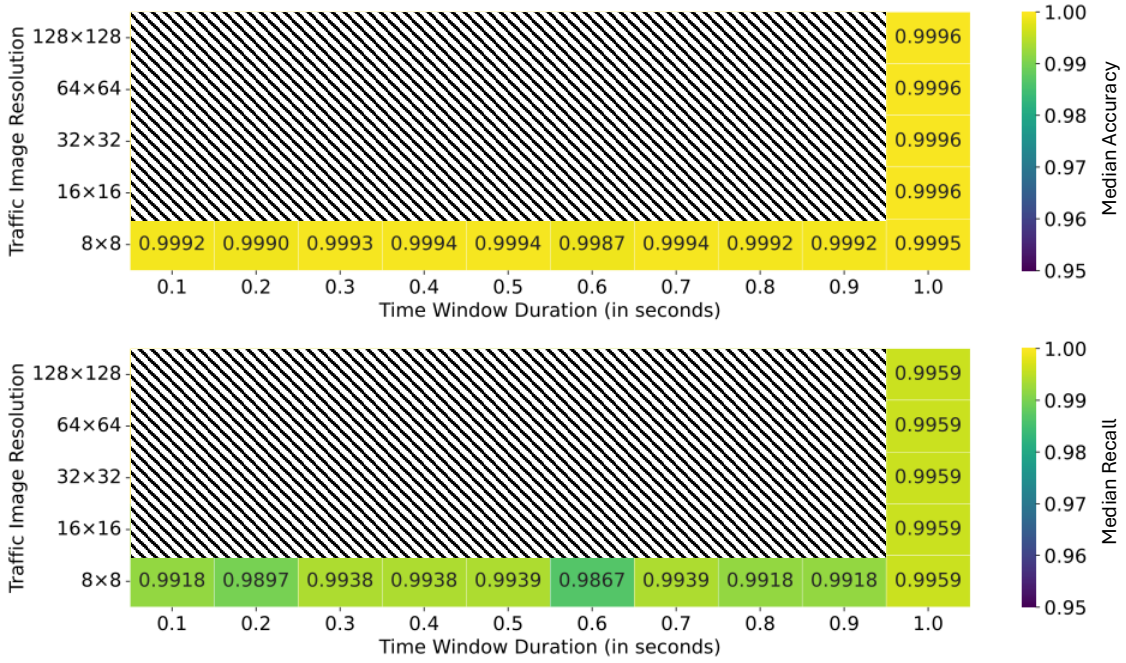


Figure 4.21: *Accuracy and recall of the binary classification (BelWue dataset) for different time window durations and traffic image resolutions. (repetition of Figure 4.12 for comparison)*

For the following experiments, the BelWue dataset is used (for selected traffic image resolutions and time window durations), as it enables the presentation of potential improvements through time series-based classification, in contrast to the MAWI+CAIDA dataset, where the binary traffic image classification already achieves perfect detection performance.

4.3.1 Improvement over Binary Traffic Image Classification

This experiment is designed to evaluate whether the time series classification of traffic images provides an improvement in the detection performance compared to the binary classification of single traffic images. Figure 4.21 repeats the accuracy and the recall

results on the BelWue dataset (see Figure 4.12) for comparison. As the precision was already perfect for all configurations, it is not shown again. The resolution $(8, 8)$ is selected for this experiment as it achieved the worst detection performance using the binary traffic image classification.

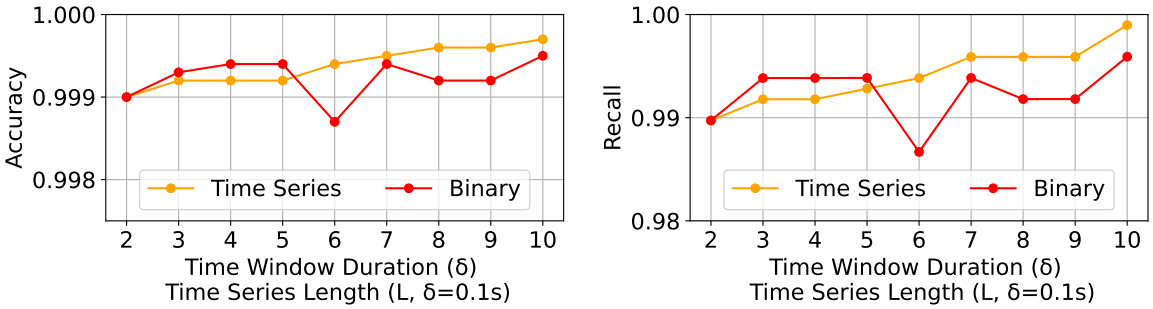


Figure 4.22: *Performance improvement of the time series classification (BelWue dataset) compared to the binary classification of single traffic images.*

Figure 4.22 presents the median (20 runs) of the accuracy (left) and the recall (right) for different time window durations (binary classification) and time series lengths (time series classification), at a fixed traffic image resolution of $(8, 8)$. Every time window duration of the binary classification is compared to a time series of traffic images spanning the same total duration. Each of the traffic images of the time series has a time window duration $\delta = 0.1$s. For example, the accuracy of the binary classification (red lines) for the time window duration $\delta = 0.5$s is compared to the accuracy of the time series-based classification (orange lines) with a time series length of $L = 5$ and a time window duration of $\delta = 0.1$s. This ensures a fair comparison, as both approaches utilize the same amount of traffic data for classification but only the temporal resolution is different. Compare the following results with the results displayed (cells in the bottom row of the heatmaps) in Figure 4.21.

The results show that the time series classification improves the accuracy and the recall for time series lengths $L \geq 6$ and the BelWue dataset. Note that the y-axes of both subfigures are different due to the varying value ranges (for better visibility). For shorter time series lengths, the recall and the accuracy slightly degrade in comparison to the binary traffic image classification. However, choosing a larger time series length L only comes with one drawback, namely that $L + 1$ traffic images need to be stored at the classification server instead of only one traffic image (L images for the time series and one to store the latest TCAM counters). On the other hand, the reaction time is not impacted by the time series length, as one classification is performed per time window (0.1s in this experiment). This results in a fixed reaction time of at most 0.1s for all time series lengths while the reaction time for the binary classification scales linearly with the

time window duration, e.g., the reaction time for a time series length of $L = 10$ is 0.1s compared to 1.0s for the binary classification with a time window duration of $\delta = 1.0$s. The previous experiment showed the potential improvement of the classification duration by utilizing longer time series of traffic images at the fixed resolution of $(8, 8)$. In contrast, the following experiment evaluates the potential detection performance improvement for **all traffic image resolutions** at a fixed time series length of $L = 10$ ($\delta = 0.1$s) compared to the binary classification with $\delta = 1.0$s. Compare the following results with the results displayed (cells in the right column of the heatmaps) in Figure 4.21.

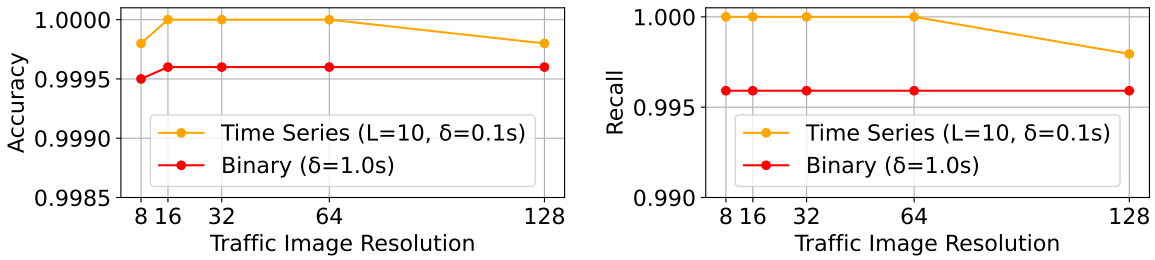


Figure 4.23: *Performance improvement for all traffic image resolutions (BelWue dataset) using time series classification with $L = 10$ compared to binary classification with $\delta = 1.0$s.*

Figure 4.23 presents the median accuracy (left) and recall (right) over 20 runs for the traffic image resolutions $N^{\Sigma}, N^{\Delta} \in \{8, 16, 32, 64, 128\}$. The results show that the median recall and accuracy improve for all traffic image resolutions when utilizing a time series of traffic images with length $L = 10$ (and $\delta = 0.1$s) compared to the binary classification with a time window duration of $\delta = 1.0$s. This implies that longer time series of traffic images is beneficial for the detection performance for all traffic image resolutions.

4.3.2 Memory Utilization and Classification Duration

The required memory at the classification server to execute the time series-based traffic image classification scales linearly with the time series length L. However, the memory utilization is fixed for a given time series length and traffic image resolution. As mentioned above, $L + 1$ traffic images need to be stored at the classification server, one image to maintain the latest fetched TCAM counters and L images for the time series. Therefore, the required memory at the classification server ranges from $8 \cdot 8 \cdot 4\text{bytes} \cdot 2 = 768$ bytes for $\chi = (8, 8)$ and $L = 2$ to $128 \cdot 128 \cdot 4\text{bytes} \cdot 11 \approx 720$ kB for $\chi = (128, 128)$ and $L = 10$, which is still negligible compared to the average memory utilization for microflow-based detection approaches (30.86 MB in average for BelWue and 25.07 MB for MAWI+CAIDA). The number of required TCAM rules remains unchanged compared to the binary traffic image classification, as it only depends on the traffic image resolution.

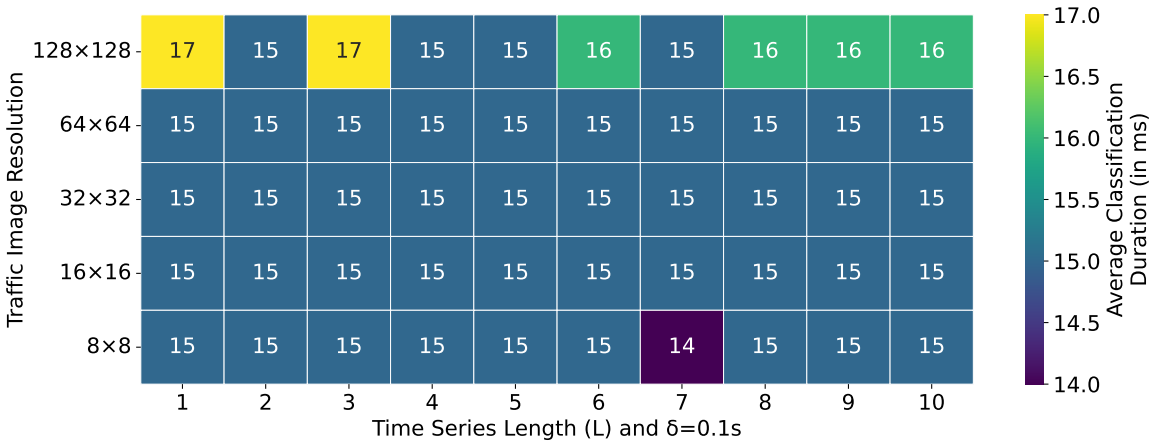


Figure 4.24: *Classification duration (ms) for different time series lengths and traffic image resolutions (BelWue dataset).*

To gain the benefits of time series-based traffic image classification in practice, the classification duration must not exceed the time window duration. Figure 4.24 presents the average classification duration (in milliseconds) for different combinations of time series lengths and traffic image resolutions (BelWue dataset). The results show that for all parameter combinations, the classification duration is between 14 ms and 17 ms. There is a deviation for the traffic image resolution of (128, 128), where the classification duration is slightly increased compared to the other resolutions. However, the classification duration is still shorter than the shortest time window duration of 0.1s and therefore not a bottleneck in the detection pipeline.

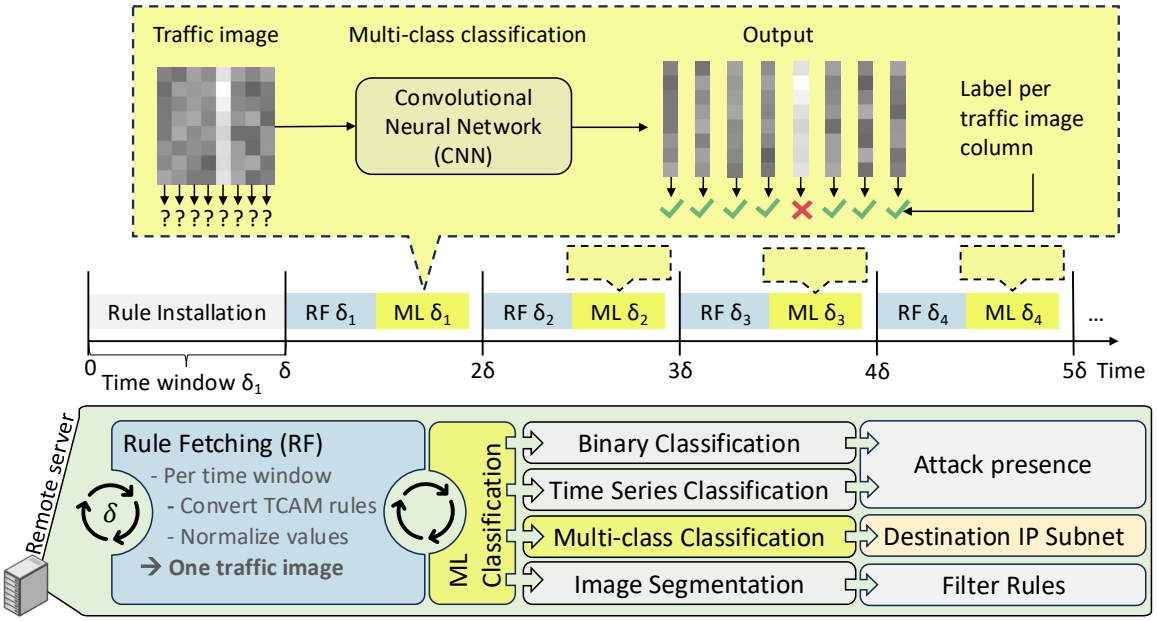


Figure 4.25: *Illustration of the multi-class traffic image classification approach.*

4.4 Destination IP Subnet Identification

This section explains and evaluates the **multi-class classification** of traffic images for DDoS **attack destination IP subnet identification**. In contrast to the binary classification approaches presented in Sections 4.2 and 4.3, and therefore beyond the detection of attack presence, multi-class classification (see Figure 4.25) aims to identify the traffic image column (see Definition 4.2 in Section 4.1) corresponding to the IP address subnet of the DDoS attack's target. One potential use case of finding the target subnet is to redirect the attack traffic (detected traffic image column) to a network function that filters attack traffic, without affecting all the legitimate traffic (remaining traffic image columns). The approach is outlined together with the labeling strategy for traffic images and the employed ML model.

Approach

The approach of multi-class traffic image classification (see Figure 4.25) is similar to the binary classification outlined in Section 4.2. In contrast, multi-class classification does not classify the input traffic image into two classes (legitimate or attack), but into $N^{\Delta}+1$ classes, where N^{Δ} is the number of traffic image columns. Each class $j \in \{1, 2, \ldots, N^{\Delta}\}$ corresponds to one traffic image column, i.e., the destination IP subnet. The class $N^{\Delta}+1$ corresponds to the legitimate class, which is assigned to traffic images without a DDoS attack. The actual label for training is one-hot encoded, i.e., a vector representation of length $N^{\Delta}+1$ with a single bit set at the index of the correct class.

Definition 4.6: Multi-Class Traffic Image Labeling

Let P_t be the set of packets arriving during time window δ_t, and $\mathscr{P}_{\text{attack}}$ be the set of packets belonging to a DDoS attack according to the ground truth. Furthermore, let $\text{IP}^{\Delta}_{\text{attack}}$ be the destination IP address subnet targeted by the DDoS attack. The label y_t for the traffic image created during time window δ_t is defined as

$$y_t = \begin{cases} j, & \text{if } \exists p \in \mathscr{P}_{\text{attack}} : t_{\text{arrival}}(p) \in \delta_t \\ & \quad \wedge \text{IP}^{\Delta}(p) = \text{IP}^{\Delta}_j \\ N^{\Delta} + 1, & \text{otherwise} \end{cases}$$

where $t_{\text{arrival}}(p)$ denotes the arrival time of packet p, and $\text{IP}^{\Delta}(p)$ denotes the destination IP address subnet of packet p.

Table 4.4: *CNN Architecture for Multi-Class Traffic Image Classification*

Architecture	
Input Layer	Image input of shape $(N^{\Sigma}, N^{\Delta}, 1)$
Conv2D (32 filters, 3×3)	+ ReLU activation
Conv2D (32 filters, 3×3)	+ ReLU activation
MaxPooling2D (2×2)	+ Dropout(0.1)
Conv2D (32 filters, 3×3)	+ ReLU activation
Conv2D (32 filters, 3×3)	+ ReLU activation
MaxPooling2D (2×2)	+ Dropout(0.1)
Flatten	Fully connected layers
Dense (64)	Fully connected layer
Dense (32)	Fully connected layer
Output Dense ($N^{\Delta} + 1$)	Activation: `sigmoid`

Table 4.4 shows the employed Convolutional Neural Network (CNN) architecture for multi-class traffic image classification. It is the same as used for the binary classification architecture (as outlined in Table 4.2), but the output layer contains $N^{\Delta} + 1$ neurons. The architecture is not further modified for multi-class classification.

4.4.1 Evaluation with Synthesized Datasets

As multi-class traffic image classification aims at detecting the traffic image column containing the attack traffic, the available real-world datasets (MAWI+CAIDA and BelWue) are not suitable for evaluation, as they only contain one DDoS attack target per dataset. This means, that for the entire dataset, only one traffic image column is assigned the attack label, which would reduce the multi-class classification to a binary classification problem. To mitigate this issue, the following experiment utilizes a synthesized dataset (same IP distributions as the synthesized dataset in Section 4.2.3), which contains a DDoS attack for every traffic image column.

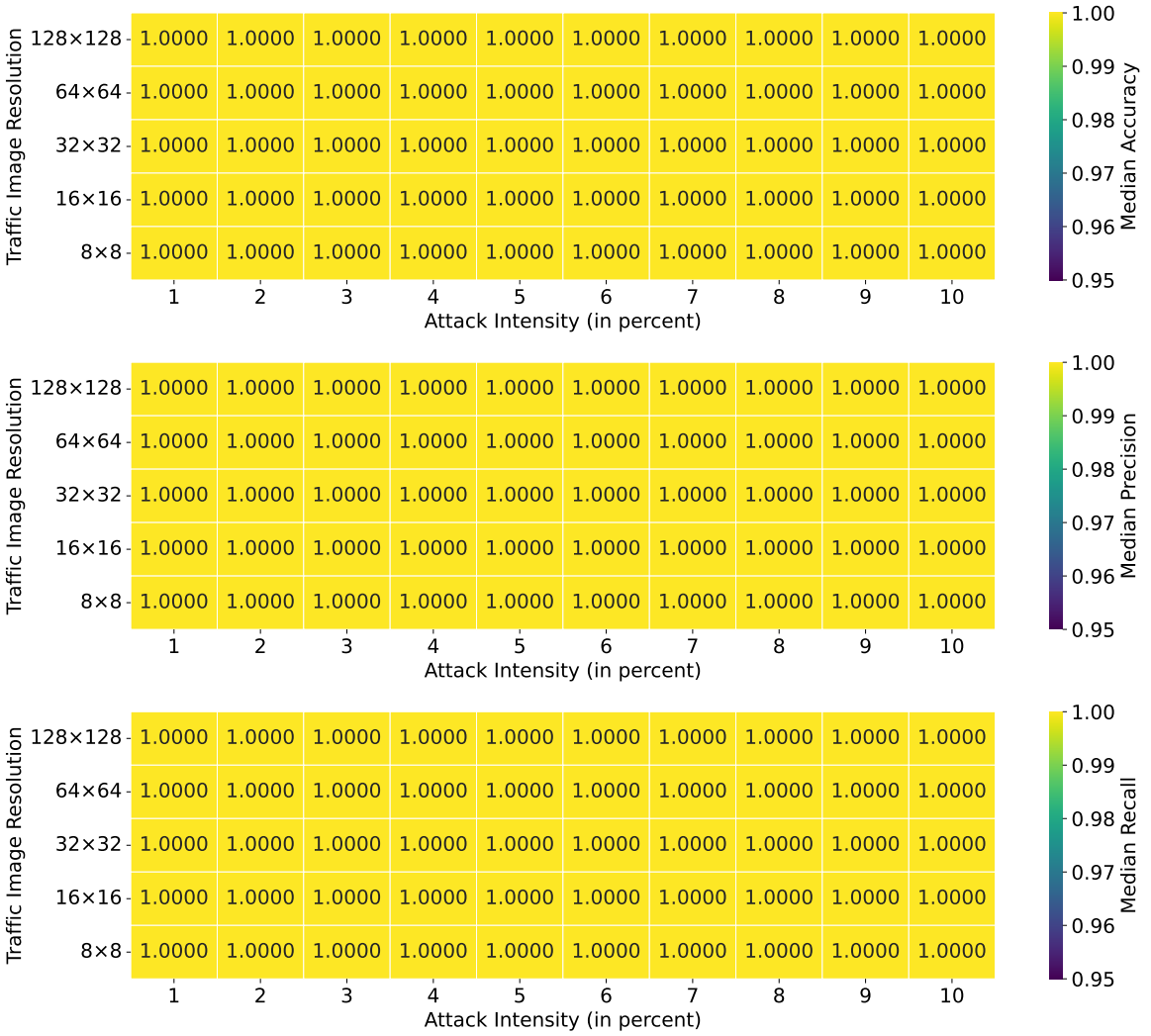


Figure 4.26: *Evaluation results of multi-class traffic image classification on synthesized datasets for different traffic image resolutions and attack intensities.*

The datasets are synthesized for the traffic image resolutions $\chi \in \{(8,8),\ldots,(128,128)\}$ and attack intensities $\alpha \in \{1\%, 2\%, \ldots, 10\%\}$. Each dataset (configuration of resolution and attack intensity) contains 100 traffic images per attack class (attack in one of the columns), and 100 images without an attack.
Figure 4.26 shows the evaluation results of the multi-class traffic image classification on the synthesized datasets for different traffic image resolutions and attack intensities. The results contain the median accuracy, precision, and recall over 20 training runs for each configuration. Although the learning task is more complex than binary classification, the results indicate that multi-class traffic image classification is feasible with perfect performance for the given attack intensities and resolutions. Similar to the binary classification results (see Figure 4.15), the multiclass classification does not show limitations above attack intensities of 1% for all evaluated traffic image resolutions.

4.4.2 Classification Duration

To ensure that multi-class traffic image classification is feasible for DDoS destination subnet identification in practice, the classification duration must not exceed the time window duration.

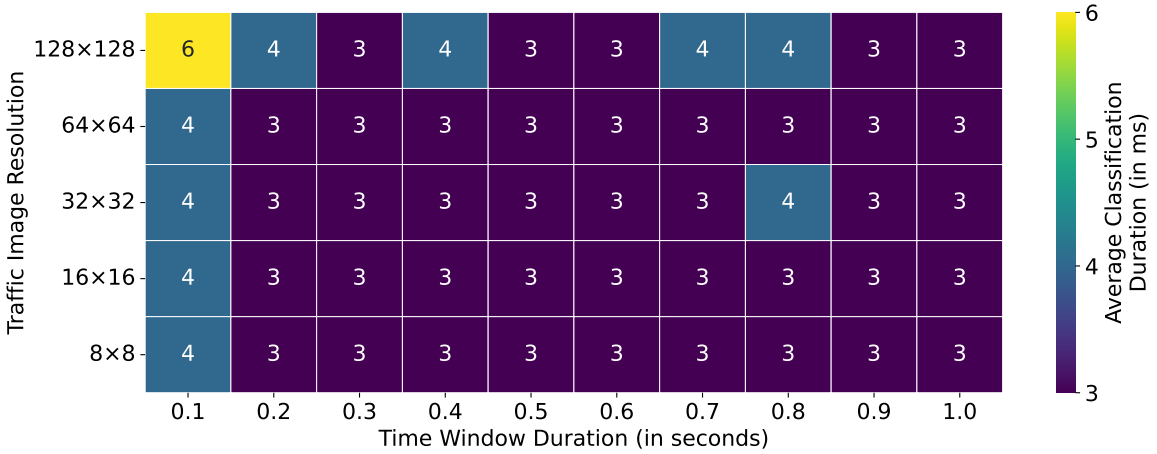


Figure 4.27: *The classification duration of multi-class traffic image-based attack detection for different time window durations and traffic image resolutions.*

Figure 4.27 shows the mean classification duration over 100 runs for different time window durations and traffic image resolutions. All values are in the range between 3 ms and 6 ms, which is below the shortest time window duration of 0.1s = 100 ms. Therefore, the classification itself is not a bottleneck for practical deployment, since only one classification is performed per time window.

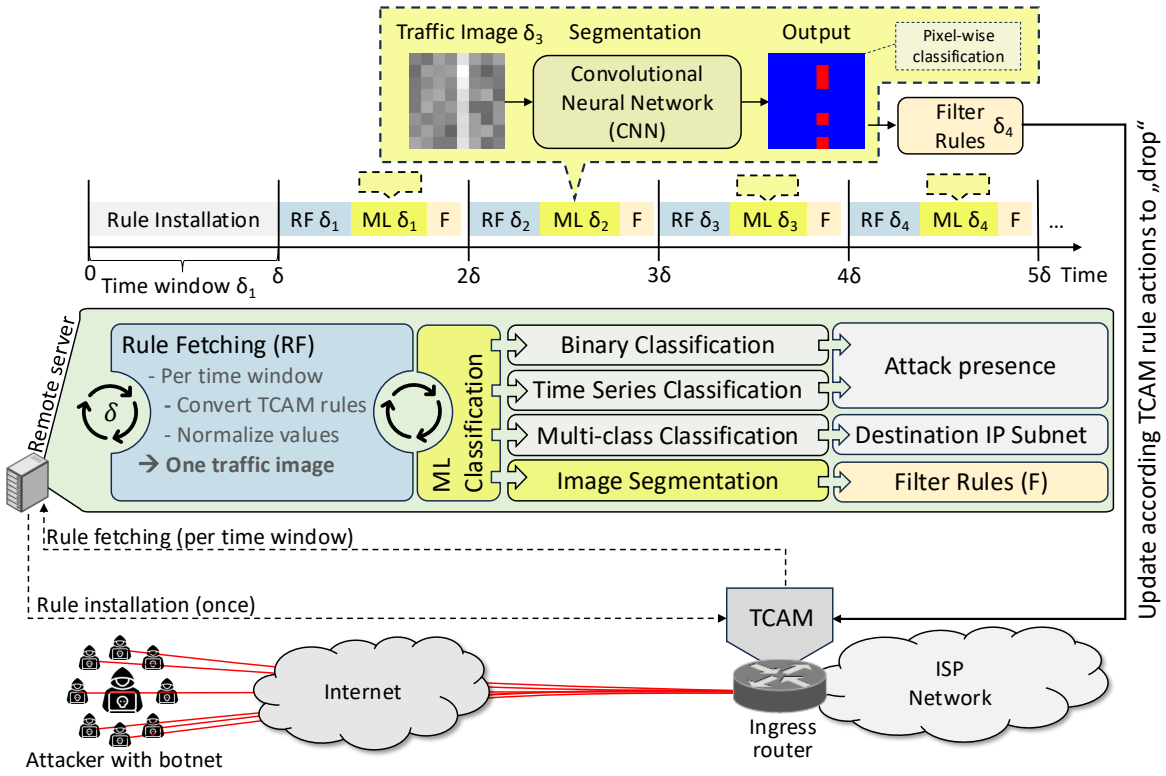


Figure 4.28: *Illustration of the traffic image segmentation approach.*

4.5 Traffic Image-based Network Filter Rules

This section explains and evaluates the traffic image segmentation approach to derive attack pixels, i.e., source-to-destination subnet pairs containing DDoS attack traffic. Identified attack pixels are then used to update the action of the already installed monitoring rules (match-action rule per pixel) in the TCAM of the ingress router to drop the matching traffic. The approach is evaluated with the MAWI+CAIDA and BelWue datasets.

Approach

The traffic image segmentation (see Figure 4.28) constitutes a pixel-wise binary classification of the traffic image, where every pixel is mapped to one of two classes, namely attack and legitimate. An attack pixel is defined as a traffic image pixel (source-to-destination subnet pair) that contains at least one packet of the DDoS attack traffic. A legitimate pixel is defined as a traffic image pixel that contains only legitimate traffic. The traffic image segmentation performs the mapping of all traffic image pixels to attack and legitimate pixels in one classification with a Convolutional Neural Network (CNN).

Definition 4.7: Traffic Image Pixel Labeling

Let P_t be the set of packets arriving during time window δ_t, and $\mathscr{P}_{\text{attack}}$ be the set of packets belonging to a DDoS attack according to the ground truth. Furthermore, let IP_i^{Σ} and IP_j^{Δ} be the source and destination IP address subnets represented by pixel (i, j) in the traffic image. The label $y_{i,j}^{\delta_t}$ for traffic image pixel (i, j) created during time window δ_t is defined as

$$y_{i,j}^{\delta_t} = \begin{cases} 1, & \text{if } \exists p \in \mathscr{P}_{\text{attack}} : t_{\text{arrival}}(p) \in \delta_t \\ & \quad \wedge \text{IP}^{\Sigma}(p) \in \text{IP}_i^{\Sigma} \wedge \text{IP}^{\Delta}(p) \in \text{IP}_j^{\Delta} \\ 0, & \text{otherwise} \end{cases}$$

where $t_{\text{arrival}}(p)$ denotes the arrival time of packet p, $\text{IP}^{\Sigma}(p)$ denotes the source IP address of packet p, and $\text{IP}^{\Delta}(p)$ denotes the destination IP address of packet p. A pixel with label $y_{i,j} = 1$ is called an **attack pixel**, while a pixel with label $y_{i,j} = 0$ is called a **legitimate pixel**.

The CNN performs one traffic image segmentation per time window. The **resulting attack pixels** are then used to update the corresponding TCAM rules' actions in the ingress router to **drop the matching traffic** (the matching rules are already installed per pixel). However, the drop actions only affect the traffic arriving after the rule update, which is assumed to be the following time window. For example, the traffic image segmented in time window δ_3 contains aggregated packets of time window δ_2. The derived filter rules from the segmentation output impact the traffic of time window δ_4. Therefore, there is a worst case delay of two time windows between attack traffic observation and filtering.

The traffic image **segmentation performance** constitutes the pixel-wise classification performance of the CNN and is therefore **evaluated with the output pixels' labels during the same time window**. The segmentation performance is not an indicator for the **filter rule performance**, as every attack pixel can contain legitimate traffic as well. In contrast, the filter rule performance is evaluated for every filtered attack pixel according to the **contained attack and legitimate packet count in the subsequent time window**.

Definition 4.8: Segmentation Performance

Let $y_{i,j}^{\delta_t}$ be the ground truth label of traffic image pixel (i,j) in time window δ_t and $\hat{y}_{i,j}^{\delta_t}$ be the predicted label of traffic image pixel (i,j) through segmentation in time window δ_t. The pixel is evaluated as true positive (TP), false positive (FP), true negative (TN), or false negative (FN) as follows:

$$E(i,j) = \begin{cases} \text{TP}, & \text{if } y_{i,j}^{\delta_t} = 1 \wedge \hat{y}_{i,j}^{\delta_t} = 1 \\ \text{FP}, & \text{if } y_{i,j}^{\delta_t} = 0 \wedge \hat{y}_{i,j}^{\delta_t} = 1 \\ \text{TN}, & \text{if } y_{i,j}^{\delta_t} = 0 \wedge \hat{y}_{i,j}^{\delta_t} = 0 \\ \text{FN}, & \text{if } y_{i,j}^{\delta_t} = 1 \wedge \hat{y}_{i,j}^{\delta_t} = 0 \end{cases}$$

Definition 4.9: Filter Rule Performance

Let p be a packet filtered in time window δ_{t+1} by the filter rules derived from the traffic image segmentation in time window δ_t. Furthermore, let $\mathscr{P}_{\text{attack}}$ be the set of packets belonging to a DDoS attack according to the ground truth. The packet p is evaluated as true positive (TP) and false positive (FP) as follows:

$$E(p) = \begin{cases} \text{TP}, & \text{if } p \in \mathscr{P}_{\text{attack}} \\ \text{FP}, & \text{if } p \notin \mathscr{P}_{\text{attack}} \end{cases}$$

Let p_n be a packet not filtered in time window δ_{t+1} by the filter rules derived from the traffic image segmentation in time window δ_t. The packet p_n is evaluated as true negative (TN) and false negative (FN) as follows:

$$E(p_n) = \begin{cases} \text{TN}, & \text{if } p_n \notin \mathscr{P}_{\text{attack}} \\ \text{FN}, & \text{if } p_n \in \mathscr{P}_{\text{attack}} \end{cases}$$

A perfect segmentation performance does not imply perfect filter rules, as traffic images cover coarse IP address subnets, e.g., /25 subnets for resolution $\chi = (128, 128)$.

Table 4.5: *UNet-inspired CNN architecture for traffic image segmentation.*

Block	**Layer Type**	**Filters / Units**
Input	Input Layer	–
	Conv2D (stride 2) + BN + ReLU	16
1st Encoder	ReLU + Conv2D + BN	32
	ReLU + Conv2D + BN	32
	MaxPooling2D (stride 2)	–
	Conv2D + BN	32
2nd Encoder	ReLU + Conv2D + BN	48
	ReLU + Conv2D + BN	48
	MaxPooling2D (stride 2)	–
	Conv2D + BN	48
1st Decoder	ReLU + UpSampling2D	–
	Conv2D + BN	48
	Conv2D + BN	48
	Conv2D + BN	48
2nd Decoder	ReLU + UpSampling2D	–
	Conv2D + BN	32
	Conv2D + BN	32
	Conv2D + BN	32
3rd Decoder	ReLU + UpSampling2D	–
	Conv2D + BN	16
	Conv2D + BN	16
	Conv2D + BN	16
Output	Conv2D (1x1) + Sigmoid	Shape $= (N^{\Sigma}, N^{\Delta}, 1)$

Table 4.5 shows the CNN architecture for traffic image segmentation. The CNN architecture is a lightweight version of the U-Net architecture [RFB15], which was originally designed for biomedical image segmentation. It has been systematically downscaled, i.e., reduced the number of en- and decoder blocks and the number of filters per layer, to fit (achieve good performance on) the traffic image segmentation task with the available datasets. Downscaling ended when further reduction of the architecture led to a segmentation performance degradation.

4.5.1 Segmentation and Filter Rule Performance – A Thought Experiment

As mentioned above, the segmentation performance is not an indicator for the filter rule performance. Figure 4.29 illustrates this difference with a thought experiment. It shows the procedure (top left) of the filter rule creation through image segmentation: First, after the traffic image of time window δ_t is received from rule fetching, it is segmented (pixel-wise binary classification) into attack and legitimate pixels. From the result, the filter rules are derived that impact the packets arriving during time window δ_{t+1}.

Three traffic images with a resolution of $\chi = (4,4)$ are shown in the figure. Traffic image δ_t is the segmented image (that contains the packet counters of time window δ_{t-1}). Traffic image δ_{t+1} is the traffic image created from the packet counters of time window δ_t (color-coded blue), and traffic image δ_{t+2} is the traffic image created from the packet counters of time window δ_{t+1} (color-coded green). Therefore, the packets aggregated in traffic image δ_{t+2} are subject to filtering. For illustration, the ground truth about the legitimate and attack pixels are shown (top right) for the traffic image δ_{t+2}. The attack spans the complete first column, while the majority of the attack traffic is concentrated in one pixel with value 50. The legitimate traffic spans the same column for the simplicity of the example.

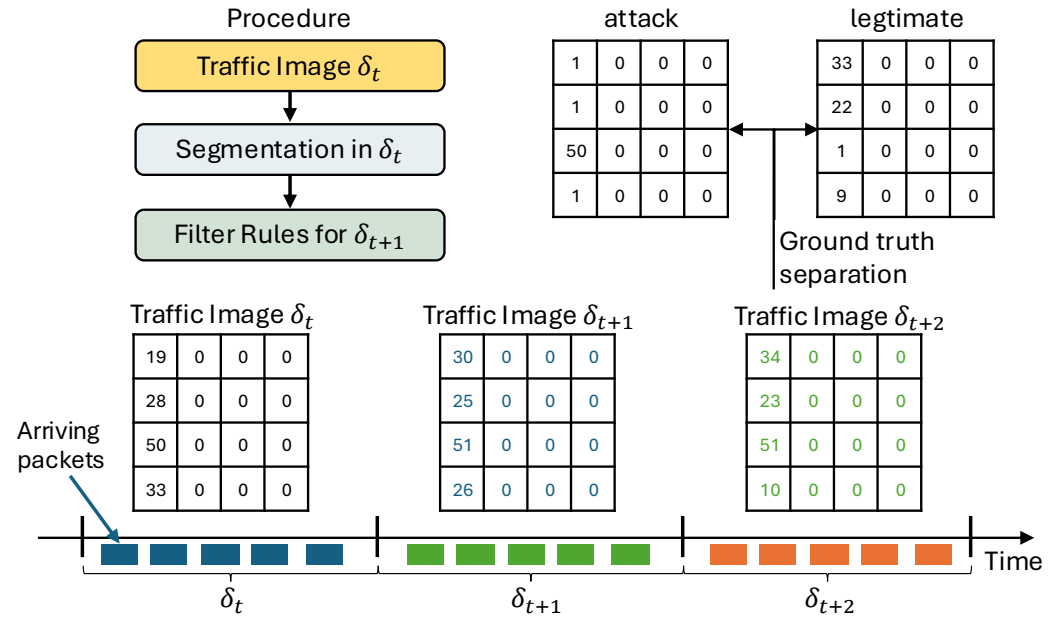
In the figure, two segmentation results are illustrated in the center, namely a perfect segmentation (left) and an imperfect segmentation (right). The perfect segmentation identifies all attack pixels (first column) while the imperfect segmentation only identifies the attack pixel with the concentrated attack traffic. As a result, the perfect segmentation achieves a recall of 100%, while the imperfect segmentation achieves a recall of only 25%.

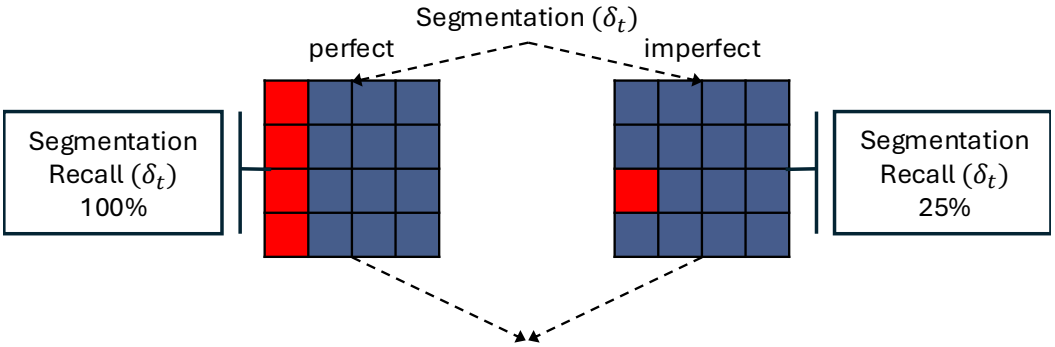
However, when evaluating the filter rule performance (bottom) in the subsequent time window δ_{t+1}, the perfect segmentation makes 65 false positive decisions. In contrast, the imperfect segmentation only makes 1 false positive and three false negative decisions, while still filtering 50 (out of 53) attack packets.

Two conclusions can be drawn from this thought experiment:

- A good segmentation performance does not imply a good filter rule performance.
- A bad segmentation performance does not imply a bad filter rule performance.

These conclusions are especially important considering the labeling strategy of attack pixels, where a pixel is assigned the attack label if at least one attack packet is aggregated into the pixel. If the segmentation misses such pixels, which is likely because they are hard to identify, the filtering is preserved from having a high collateral damage on the legitimate traffic.





Filter Rule Performance (δ_{t+1})

Metric	Perfect Segmentation	Imperfect Segmentation
TP	53	50
TN	0	64
FP	**65**	**1**
FN	**0**	**3**

Figure 4.29: *Illustration of the difference between the segmentation and the filter rule performance.*

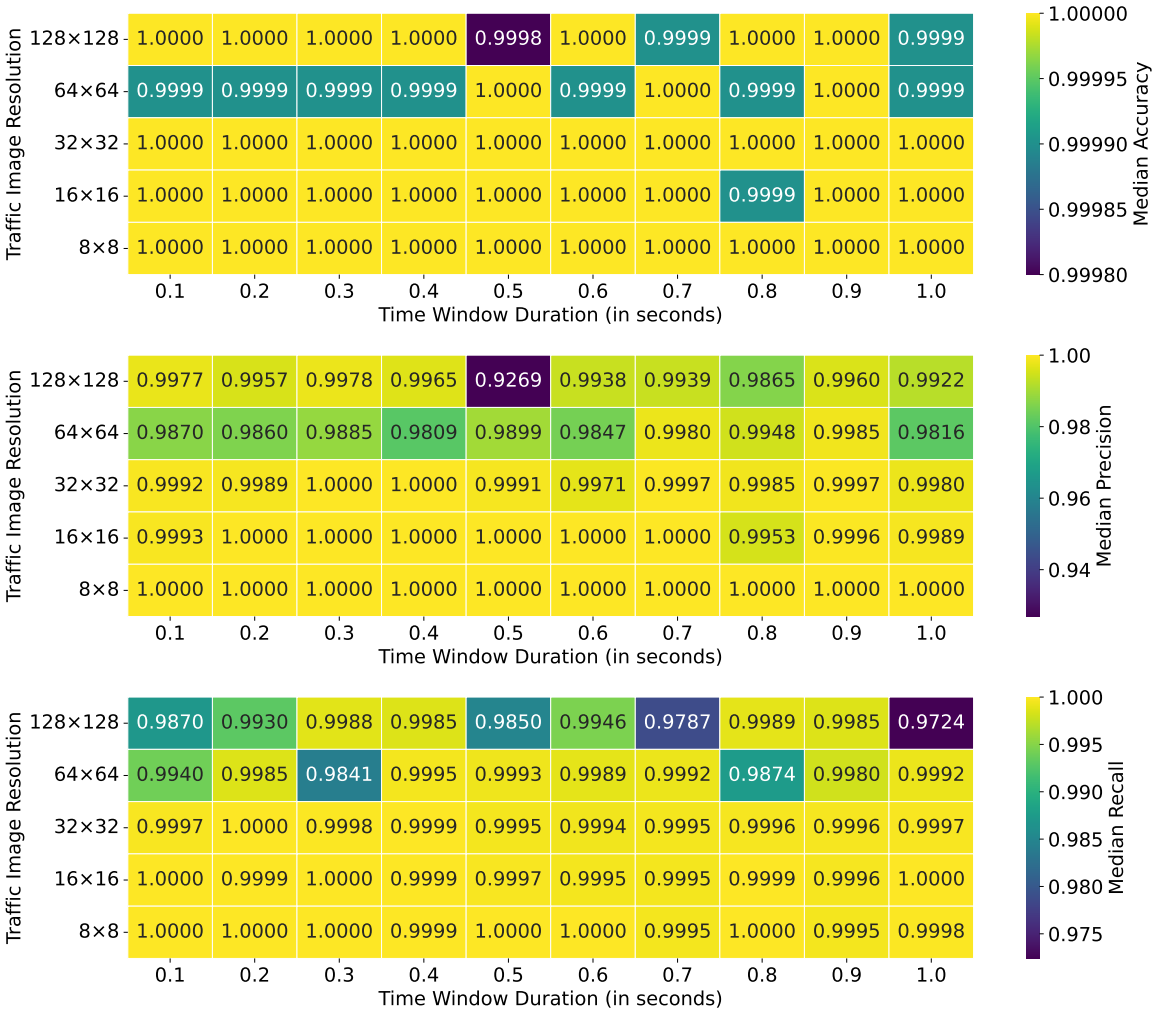


Figure 4.30: *Segmentation performance with the MAWI+CAIDA dataset trained on July 1st and evaluated on July 2nd, 2025.*

4.5.2 Evaluation with the MAWI+CAIDA Dataset

To evaluate the performance of the segmentation and the resulting filter rules, two requirements must be fulfilled: First, the training dataset must contain enough attack samples to train the traffic image segmentation, which is the most complex learning task presented in this chapter (e.g., pixel-wise binary classification in contrast to traffic image-wise binary classification). Second, the test dataset must maintain the temporal order of created traffic images, as the filter rules impact the subsequent time window. To satisfy these requirements, the following experiment conducts a combined evaluation of the segmentation and filter rule performance, as well as the generalization performance. The CNN model is trained on the MAWI+CAIDA dataset of July 1st, 2025, and evaluated on the

MAWI+CAIDA dataset of July 2nd, 2025. This ensures that enough training data is available (full dataset of July 1st) and that still enough test data is available while maintaining the temporal traffic image order (full dataset of July 2nd). The experiment is conducted for each combination of the traffic image resolutions $\chi \in \{(8,8),(16,16),\ldots,(128,128)\}$ and time window durations $\delta \in \{0.1, 0.2, \ldots, 1.0\}$s. Every configuration (resolution and time window duration) is evaluated over 20 runs and the median results are shown.

Figure 4.30 shows the segmentation performance, i.e., the pixel-wise binary classification performance of the CNN model. The results indicate that the segmentation performance decreases with an increasing traffic image resolution, which is expected due to two effects: First, **higher resolution images** contain **more pixels** (scaling quadratically) to classify, which increases the learning task's complexity. Second, higher resolution images contain **fewer packets per pixel**, which makes it harder to identify attack pixels. These **two effects in combination** impact the segmentation performance negatively.

However, the overall segmentation performance never drops below a precision of 92.69% (resolution $\chi = (128,128)$, time window duration $\delta = 0.5$s) and a recall of 97.24% (resolution $\chi = (128,128)$, time window duration $\delta = 1.0$s) for all segmented pixels over the entire evaluation period of the test dataset on July 2nd, 2025. For example, for the resolution $\chi = (128,128)$ and time window duration $\delta = 0.1$s, there are $\frac{1}{0.1s} \cdot 900s \cdot 128^2 \text{pixels} = 147.456.000$ segmented pixels, as there are ten traffic images per second containing $128 \cdot 128$ pixels each over the 15 minutes (900s) evaluation period. Nevertheless, as the segmentation performance does not imply the filter rules' performance, the following evaluation is conducted.

For each time window during the evaluation period (dataset of July 2nd, 2025), the derived filter rules from the segmentation output (subnet pairs corresponding to pixels classified as attack) are applied to the packets arriving in the subsequent time window, e.g., the traffic image of time window δ_t (containing packets monitored during time window δ_{t-1}) is segmented to derive filter rules that are applied to the packets arriving in time window δ_{t+1}.

Figure 4.31 shows the filter rules' accuracy, precision, and recall for different traffic image resolutions and time window durations. Two observations can be made from the results: First, the filter rules' recall decreases with an increasing traffic image resolution. This covers with the segmentation's recall decrease at higher resolutions, where fewer attack pixels are identified and therefore fewer attack packets are filtered. Second, the filter rules' precision increases with an increasing traffic image resolution, which means that fewer legitimate packets are filtered for higher resolutions.

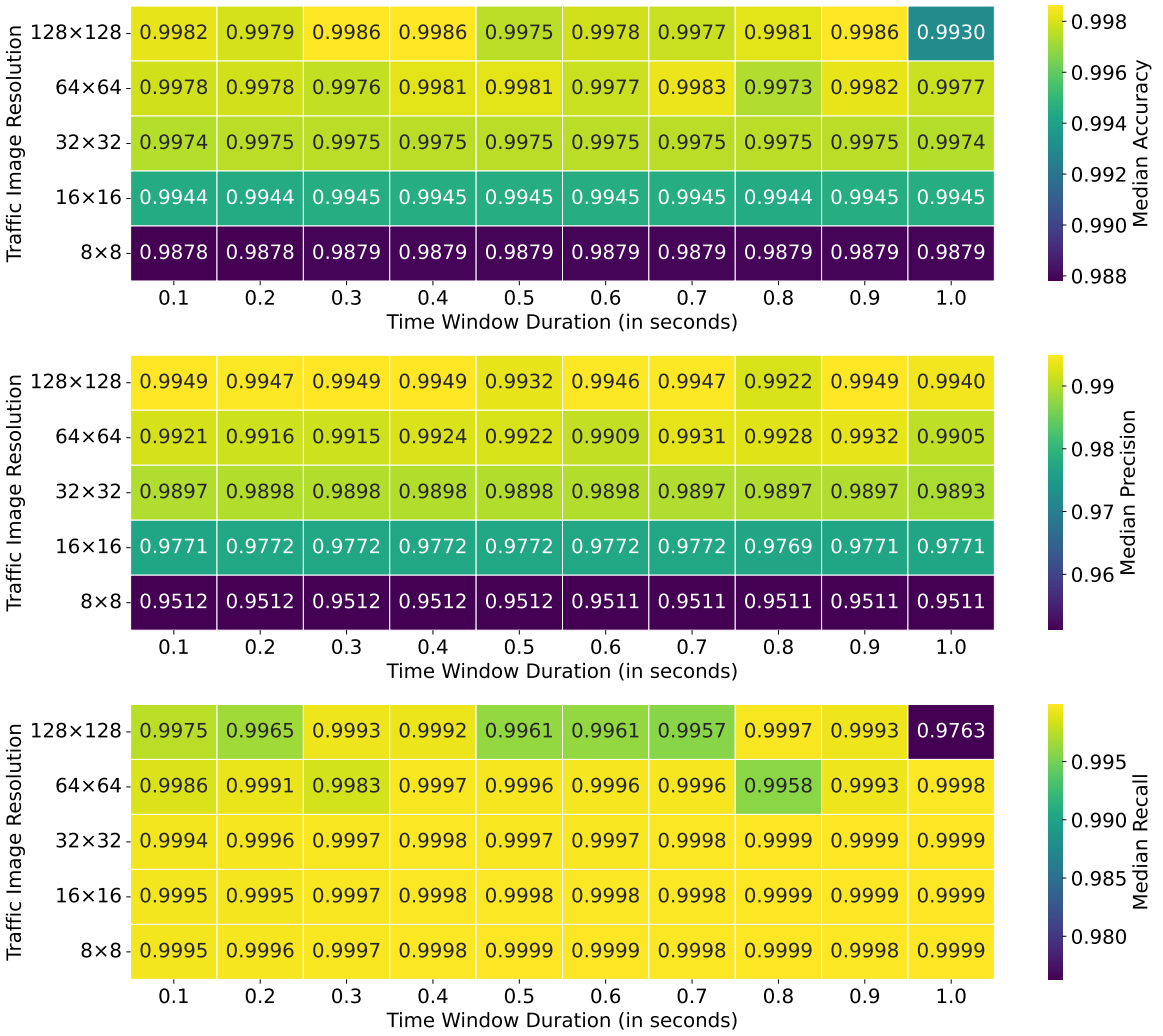


Figure 4.31: *Filter rule performance evaluation with the MAWI+CAIDA dataset trained on July 1st and evaluated on July 2nd, 2025.*

This is a counterintuitive observation, as the segmentation performance decreases for higher resolutions, and implies that attack pixels that are hard to identify (contain much legitimate traffic in contrast to attack traffic) are missed by the segmentation, which reduces the collateral damage on legitimate traffic. The result also covers the slight filter rules' recall decrease for higher resolutions as those hard-to-identify attack pixels contain only a small amount of attack traffic. The inability of the CNN model to identify attack pixels with a very small fraction of attack traffic compared to the legitimate traffic is a beneficial property of the traffic image segmentation-based filter rule creation, as it reduces the collateral damage on legitimate traffic.

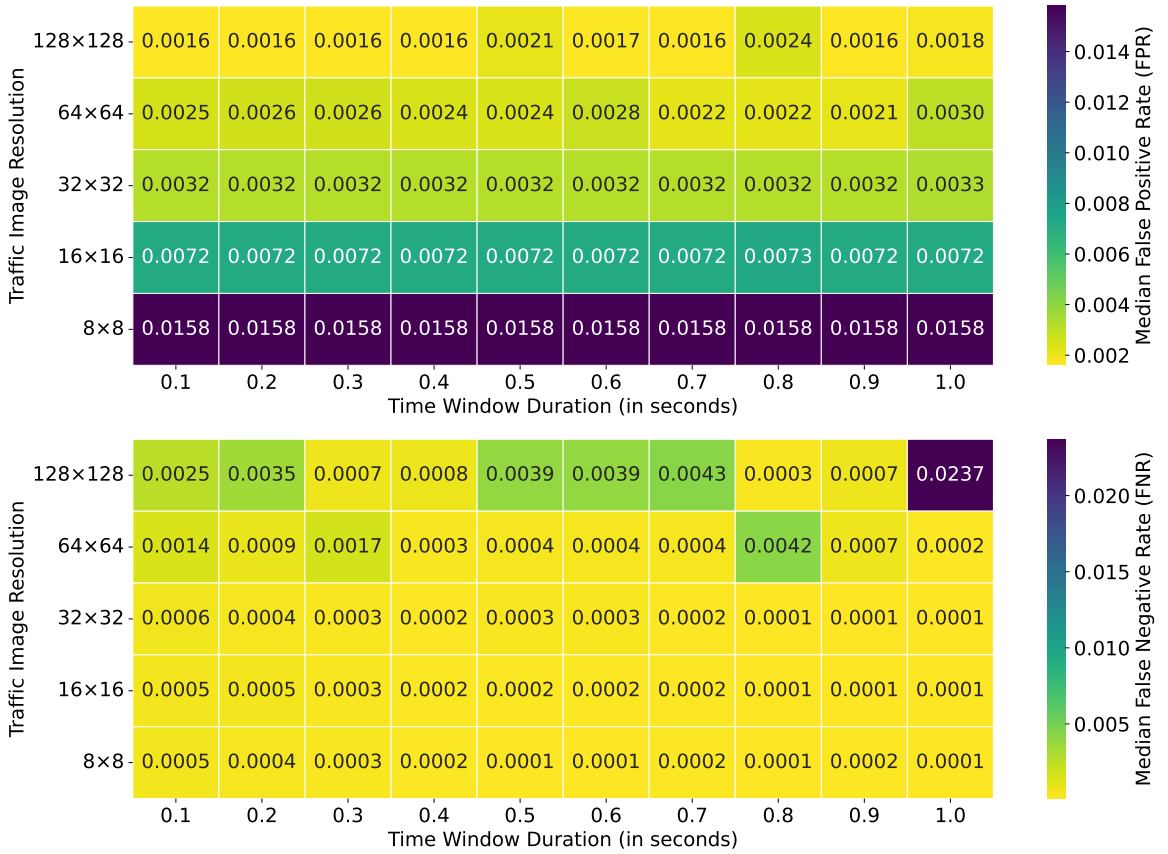


Figure 4.32: *Traffic image segmentation-based filter rules' false positive and false negative rate evaluation with the MAWI+CAIDA dataset trained on July 1st and evaluated on July 2nd, 2025.*

Figure 4.32 shows the filter rules' performance from the perspective of falsely filtered legitimate or missed attack packets, i.e., the false positive rate (FPR) and the false negative rate (FNR). The FPR $\frac{\text{FP}}{\text{FP+TN}}$ represents the fraction of all legitimate packets that are falsely filtered, while the FNR $\frac{\text{FN}}{\text{FN+TP}}$ represents the fraction of all attack packets that are missed by the filter rules. For both metrics holds that lower values are better. The results indicate that the FNR increases with an increasing traffic image resolution, which is expected due to the decreasing segmentation recall for higher traffic image resolutions (see Figure 4.30). Fewer attack pixels are identified for traffic image resolution and therefore not all attack packets are filtered. In contrast, the FPR decreases with an increasing traffic image resolution, which is also expected due to the increasing filter rule precision for higher traffic image resolutions as fewer legitimate packets are filtered. Overall, the FPR is at most 1.58% (resolution $\chi = (8, 8)$) due to the very coarse subnet granularity and goes down to 0.16% (resolution $\chi = (128, 128)$ and time window duration 0.1s). In contrast to the expectations, the time window duration has no significant impact on the

FPR and FNR. As filter rules are applied for the whole subsequent time window, more packets are impacted by the filter rules for longer time window durations. The derived filter rules would therefore have a higher impact on the FPR and FNR if the time window duration is longer. As this effect is not observable in the results, it implies that the traffic distribution does not change significantly between time windows in the selected range of time window durations.

Discussion of Traffic Image Segmentation-based Filter Rules

Although the segmentation and filter rule performance show feasible results for deployment in practice (MAWI+CAIDA backbone scenario), even with the presented circumstances of generalization (training and testing on separate days), the traffic image segmentation filter rule derivation suffers from the following limitations: The subnet granularity of the traffic image pixels is very coarse, even for the traffic image resolution $\chi = (128, 128)$, which represents /7 subnets for both source and destination IP address spaces. If the attack traffic and the legitimate traffic would share the same IP address distributions, the collateral damage on legitimate traffic would be much larger than in the presented evaluation as the traffic images in the MAWI+CAIDA dataset are sparsely populated with traffic and the intersection between attack and legitimate traffic is small in the IP address space. Furthermore, the derived filter rules only impact the traffic in the subsequent time window after the segmentation, which introduces a delay of up to two time windows between attack observation and filtering. If the attack traffic's source IP address distribution changes frequently, e.g., by activating or deactivating parts of a botnet, and in smaller periods than the time window duration, the derived filter rules miss a significant amount of attack traffic while impacting legitimate traffic.

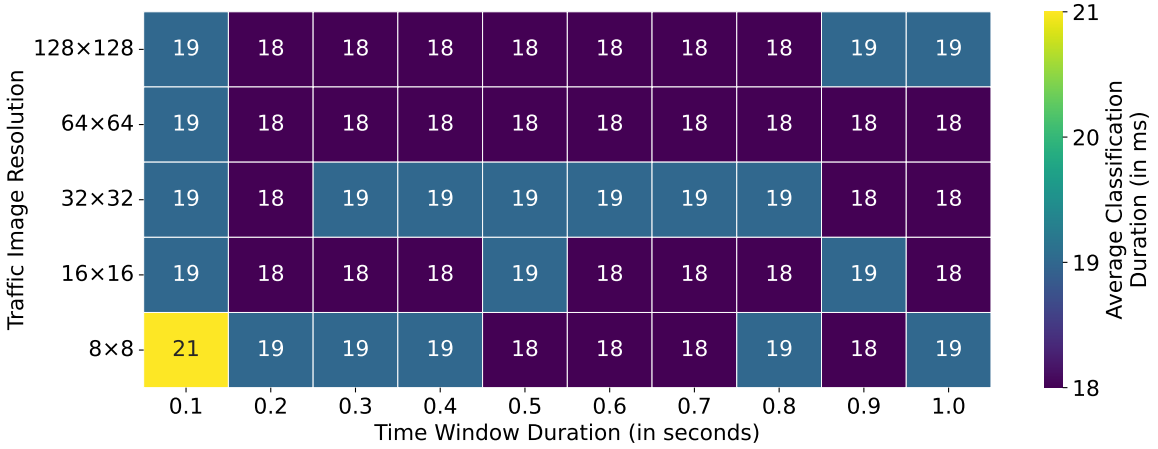


Figure 4.33: *Average segmentation duration per traffic image for different time window durations and traffic image resolutions.*

4.5.3 Segmentation Duration

The following experiment evaluates the duration the CNN model requires to segment a single traffic image. The goal of this experiment is to show that the segmentation is not a bottleneck in the classification pipeline, i.e., the segmentation duration is shorter than the time window duration.
Figure 4.33 shows the average segmentation duration per traffic image of 100 runs for different time window durations and traffic image resolutions. The results indicate that neither the time window duration nor the traffic image resolution have an impact on the segmentation duration. The segmentation duration is always in the range of 18ms to 21ms, which is shorter than the smallest time window duration of 0.1s. Therefore, the segmentation is not a bottleneck in the detection pipeline.

4.6 Traffic Image Extensions and Adaptations

This section outlines potential extensions and adaptations of the traffic image-based DDoS detection approach, namely its **adaptability to port scan detection**, its extensibility to **multi-channel traffic images** (similar to RGB images), and its **compatibility with software-based monitoring**.

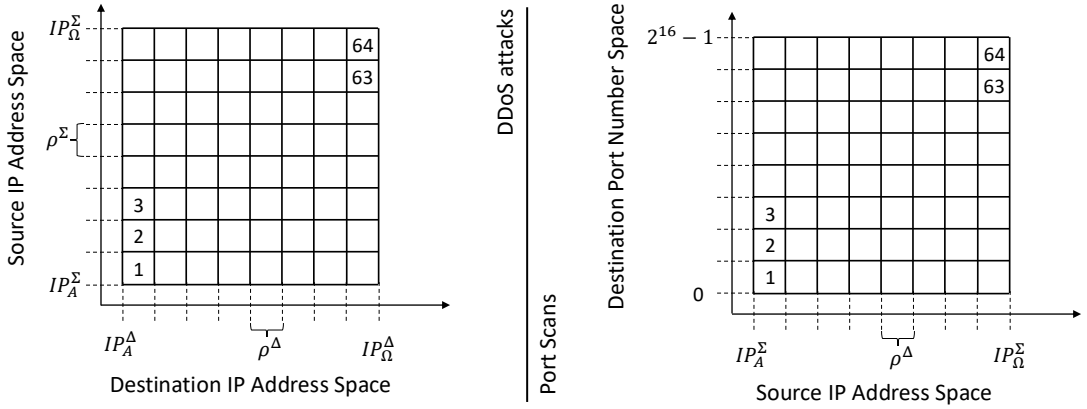


Figure 4.34: *Comparison of the traffic image for DDoS attack detection (left) and port scan detection (right).*

4.6.1 Adaptability to Port Scan Detection

While Distributed Denial-of-Service (DDoS) attacks originate from thousands of distributed sources (IP addresses) targeting a single destination (IP address), port scans typically comprise packets from a single source (IP address) to thousands of destinations (port numbers). Therefore, port scans share the same characteristics of DDoS attacks when observing different packet attributes. The traffic image-based monitoring can be adjusted for port scan detection by changing the dimensions of the traffic image (see Figure 4.34) from source and destination IP address spaces (left) to the destination port number space and the source IP address space (right). This change of dimensions makes port scans manifest in traffic image columns, such as DDoS attacks in the original traffic image. The Ternary Content Addressable Memory (TCAM) rule creation follows the same prefix-based approach for port numbers, which divides the complete port number space (2^{16} ports) into subspaces according to the traffic image resolution. For example, a traffic image resolution of $\chi = (8, 8)$ divides the source IP address space into 8 subnets (IP prefix length 3) and the port number space into 8 subspaces (port prefix length 3), where each pixel represents a pair of source IP subnet (containing 2^{29} IP addresses) and destination port number subspace (containing 2^{13} ports).

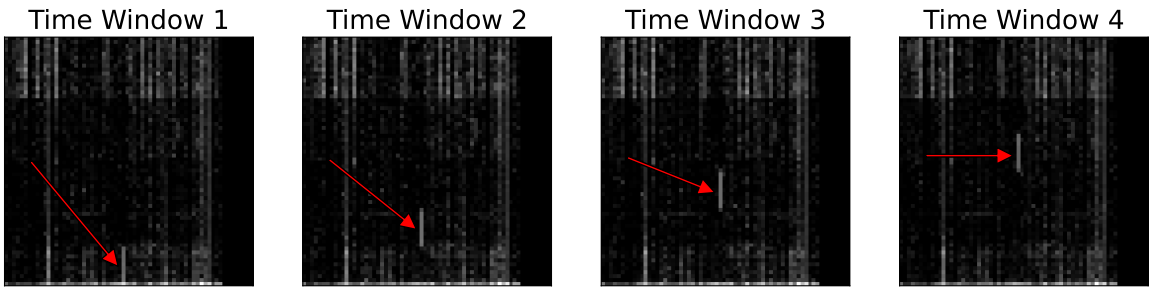


Figure 4.35: *Traffic image series example capturing an injected ordered port scan (ascending destination port numbers) into the MAWI+CAIDA dataset.*

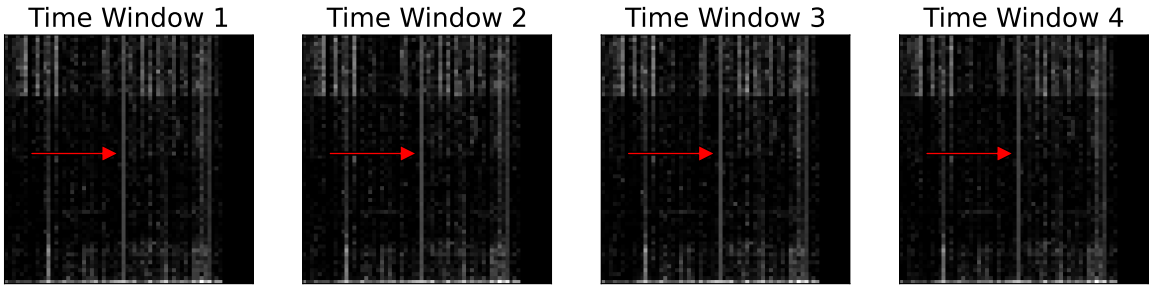


Figure 4.36: *Traffic image series example capturing an injected random port scan (random destination port numbers) into the MAWI+CAIDA dataset.*

To give a viability example of the port scan detection with traffic images, two datasets are created by injecting synthesized port scans into the MAWI dataset, i.e., the legitimate traffic belongs to the MAWI dataset and the attack traffic to the injected port scans. Two scenarios are considered for the port scan injection: First, an ordered port scan (see Figure 4.35) where the destination ports are scanned sequentially from 1 to 65535, and second, a random port scan (see Figure 4.36) where the destination ports are scanned in a random order. Both port scans are conducted with a scan rate of 10000 packets per second from a single source IP address to all destination ports (1-65535) of a target IP address (scans manifest in one traffic image column).

The red arrows in the figures point to the port scan traffic. In the first scenario (ordered port scan), the port scan manifests in a short vertical line moving from the bottom of the traffic image (destination port 1) to the top of the traffic image (destination port 65535) over time, where each traffic image represents a time window of one second from left to right. In contrast to the ordered port scan, the random port scan (second scenario) spreads over the entire traffic image column (also indicated by red arrows). The intensity (brightness in the traffic image) is lower because the 10000 packets per

second are randomly distributed over all destination ports instead of being concentrated on a few pixels.
To evaluate the traffic image-based detection performance, the binary traffic image classification approach is applied with the time window $\delta = 1$s and the traffic image resolution $\chi = (128, 128)$.

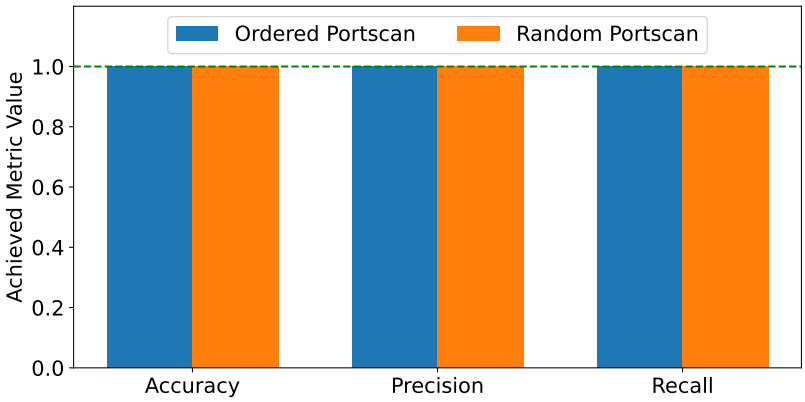


Figure 4.37: *Port scan detection performance with ordered and random port scan scenarios.*

Figure 4.37 shows the port scan detection performance for both scenarios, the ordered (blue) and the random (orange) port scan. The results indicate perfect detection performance (precision and recall equal to 1.0) for both port scan scenarios, which confirms the adaptability of the traffic image-based monitoring approach for port scan detection. However, the presented scenario comprises very aggressive port scans, i.e., having a high scan rate of 10000 packets per second, which makes them easily detectable even with the human eye.

In real-world scenarios, port scans may be conducted with lower scan rates (stealth scans) to evade detection. Sophisticated attackers, scanning only a few ports per hour, will remain undetected by the traffic image-based port scan detection. Trade-offs between the minimum detectable attack intensity (scan rate in the port scan application), the reaction time (time window duration), and the memory utilization (traffic image resolution) need to be considered for deployment.

On the other hand, the traffic image-based port scan detection approach can detect aggressive port scans effectively, while performing monitoring with routers and switches equipped with TCAM without the modification of end systems, which enables port scan detection that can be integrated seamlessly into existing network infrastructures.

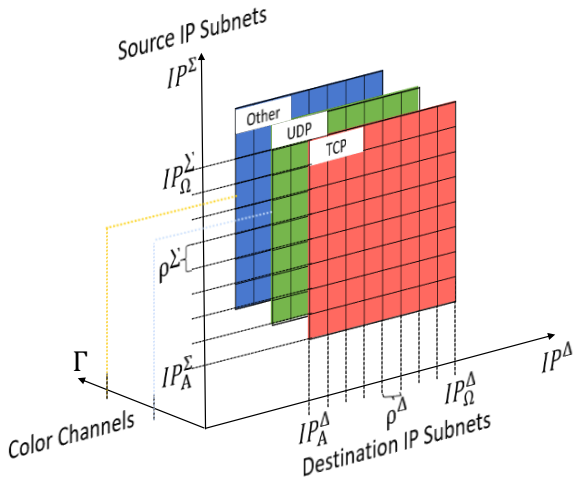


Figure 4.38: *Illustration of a multi-channel traffic image (RGB) with three color channels.*

4.6.2 Multi-channel Traffic Images

The original traffic image monitoring approach counts packets per source-to-destination IP subnet pairs, resulting in a single-channel traffic image. However, inspired by multi-channel images in computer vision (e.g., RGB images with the three color channels red, green, and blue), the traffic image concept can be extended through multiple color channels Γ (see Figure 4.38) by incorporating additional packet attributes for TCAM matching rules, such as (but not restricted to) the transport layer protocol (e.g., TCP and UDP).

Table 4.6: *Multi-channel traffic image TCAM rule structure (source and destination IP, protocol) for one pixel.*

Source IP address	Destination IP address	Protocol	Counter
100.000.000.000/7	100.000.000.000/7	00000110 (TCP)	10
100.000.000.000/7	100.000.000.000/7	00010001 (UDP)	2
100.000.000.000/7	100.000.000.000/7	XXXXXXXX (other)	3

The required information to construct TCAM rules according to the protocol channels can be obtained from the IP header's protocol field, which is eight bits in size and indicates

the transport layer protocol of the packet, e.g., TCP (6) and UDP (17). Table 4.6 shows an example of the TCAM rule structure (IP addresses in subnet notation without showing Don't Care Bits) for the monitoring of a multi-channel traffic image with three protocol channels: TCP, UDP, and other, where other has the wildcard value XXXXXXXX (Don't Care Bits) to match all remaining protocols. The wildcard value represents a catch-all channel that aggregates all packets not matching the specified protocol channels (TCP and UDP in this case). This ensures that the overall packet count is still maintained and further captures attacks that do not use the specified protocols, e.g., an ICMP-flood DDoS attack (such as the CAIDA 2007 DDoS attack). Adding color channels to the traffic image increases the contained information in the traffic image, which may improve the classification performance of the previously presented detection approaches. However, this information gain comes at the cost of an increased number of required TCAM rules, i.e., $|\Gamma|$ times more rules for $|\Gamma|$ color channels, as for every pixel and color channel a separate TCAM rule is required.

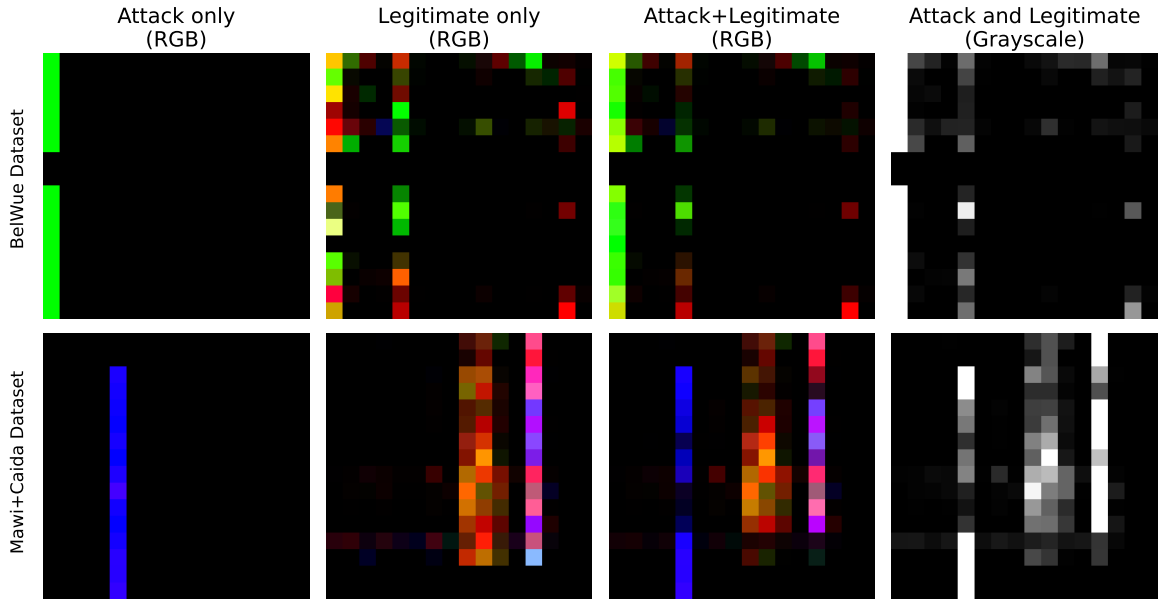


Figure 4.39: *Illustration of RGB traffic images created from the MAWI+CAIDA and the BelWue datasets.*

Figure 4.39 shows example RGB traffic images (traffic image resolution (16,16,3)), with the transport protocol channels from above, created from the MAWI+CAIDA (bottom row) and the BelWue (top row) datasets. Each of the traffic images per row (per dataset) stem from the same time window but show different traffic compositions, i.e., only attack traffic, only legitimate traffic, attack and legitimate traffic, as well as attack and legitimate traffic in the original single-channel traffic monitoring approach (grayscale). In the traffic images of the MAWI+CAIDA dataset, the ICMP-flood DDoS attack is clearly

visible in the blue color channel and does not dominate any other color channel due to the limited intersections in the IP address distributions. However, in the traffic images of the BelWue dataset, the UDP-flood DDoS attack is visible in the green color channel but also dominates the other color channels due to the significant attack volume and the high intersection in the IP address distributions.

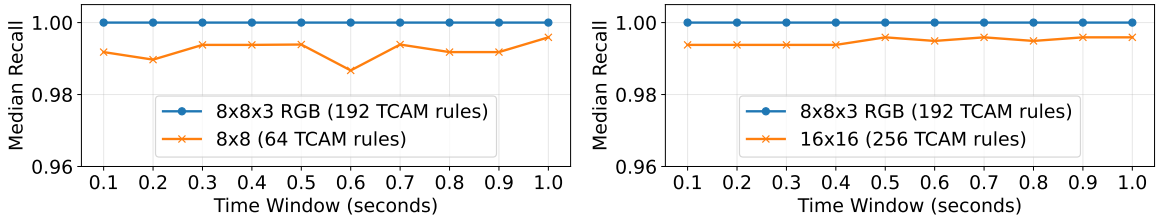


Figure 4.40: *Comparison of RGB and original traffic image binary classification results on the BelWue dataset.*

To evaluate whether the additional color channels improve the detection performance, two experiments are conducted comparing the binary classification performance of the RGB traffic image approach with the original single-channel traffic image approach. Both experiments are conducted on the BelWue dataset to enable the illustration of potential detection performance improvements.

The first experiment evaluates whether the additional color channels contribute an information gain for the classification at all and therefore facilitate a detection performance improvement. The RGB binary classification performance (the recall, as precision is already 1.0 for the original traffic images) with the resolution $\chi_{RGB} = (8, 8, 3)$ is compared to the original binary traffic image classification with resolution $\chi = (8, 8)$. Figure 4.40 (left) shows the results of this experiment for different time window durations δ. The results indicate that the RGB traffic image classification outperforms the original traffic image classification, as it enables perfect detection (recall = 1.0) for all time windows. However, the RGB traffic image approach utilizes $3 \cdot 64 = 192$ TCAM rules compared to 64 TCAM rules for the original traffic image approach, which relativizes the performance improvement.

To enable a fair comparison regarding the required TCAM rules, a second experiment is conducted where the RGB traffic image resolution remains $\chi_{RGB} = (8, 8, 3)$, but the original traffic image resolution is increased to $\chi = (16, 16)$, resulting in more utilized TCAM rules of the original approach (256 rules) than the RGB approach. The results of this second experiment are shown in Figure 4.40 (right) and indicate that the original traffic image classification approach still performs worse than the RGB approach that requires fewer TCAM rules. The RGB approach has also been successfully tested in a

major tier-1 ISP network scenario (see Section 5.2.1). Therefore, it is beneficial for the DDoS attack detection to use the available TCAM rule space for extending the monitored traffic attributes per subnet pair (multiple color channels) instead of increasing the traffic image resolution alone.

4.6.3 Compatibility with Software-based Monitoring

Although the original traffic image monitoring approach is designed for hardware-based monitoring with TCAM, it can also be adapted to software-based monitoring approaches, such as the eMinD classification pipeline. If the classification server receives a mirrored packet stream from an ingress router, it needs to extract the source and destination IP addresses from each packet header and map them to the corresponding pixel (subnet pair) in the traffic image. This mapping can be efficiently realized by prefix cutting, i.e., reading only the most significant bits of the IP addresses according to the traffic image resolution (for example the first 7 bits for the traffic image resolution $\chi = (128, 128)$). The prefixes can then be used as indices for the traffic image (in RAM) to increment the corresponding pixel counter.

In contrast to hardware-based monitoring with TCAM, software-based monitoring does not go along with strict limitations regarding the traffic image resolution (and therefore the monitoring granularity) because the traffic image monitoring uses RAM instead of TCAM. For example, a traffic image resolution of $\chi = (256, 256)$ (65536 pixels) would require $256^2 \cdot 4 \approx 262$ kB of RAM (assuming four bytes per pixel counter). Furthermore, the TCAM rule installation and rule fetching is obsolete with software-based monitoring, which enables shorter reaction times and adaptations to the monitored IP address ranges.

DDoS Attack Destination IP Address Identification
Being able to adapt the monitored IP address ranges on the classification server in RAM (without reinstalling TCAM rules) enables an iterative multi-class traffic image classification approach, where each iteration (per time window) constitutes a zoom in the destination IP address space into the identifies attack column. In the first iteration, the traffic image captures the complete source and destination IP address space, i.e., $\left[IP_A^\Sigma, IP_\Omega^\Sigma\right] = \left[0, 2^{32}-1\right]$ and $\left[IP_A^\Delta, IP_\Omega^\Delta\right] = \left[0, 2^{32}-1\right]$. If the multi-class classification identifies an attack column IP_j^Δ (destination IP subnet j) the monitored destination IP address range for the next iteration (next time window) is restricted to only the identified attack column. This new destination subnet range is again partitioned into N^Δ subnets, which leads to a new destination subnet size of $\rho_{i+1}^\Delta = \frac{\rho_i^\Delta}{N^\Delta}$, where i is the iteration index. This iterative approach is repeated until the destination subnet size reaches 1, i.e., the traffic image columns represent distinct IP addresses, and has been demon-

strated [KKZ24a] and evaluated [KKZ24b] thoroughly with synthesized and real-world datasets (MAWI+CAIDA). It has been shown that the iterative multi-class traffic image classification can identify the DDoS attack destination IP addresses with high precision and recall while achieving a subsecond reaction time. The approach cannot only be used for DDoS attack destination IP address identification but also for port scan source IP address identification by adjusting the monitored traffic image dimensions as described in Section 4.6.1.

Algorithm 5: Iterative Multi-Class Traffic Image Classification ($\chi = (256, 256)$)

Input: Global parameters $IP_A^\Delta = 0$, $IP_\Omega^\Delta = 2^{32} - 1$, $N^\Delta = 2^8$, $\rho^\Delta = 2^{24}$

```
1  while True do
       // Per-packet monitoring for time window duration
2      monitoring(δ);
3      traffic_image_current ← GetImage();
4      result_vector ← CNN.Classify(traffic_image_current);
       // Check which destination subnet is classified as attack
5      j ← arg max_k result_vector[k];
       // Class ρ^Δ represents the legitimate class
6      if j < ρ^Δ then
           // If attack detected, check current subnet granularity
7          if ρ^Δ > 2^0 then
               // Subnet granularity not /32, adjust monitored IP range
8              IP_A^Δ ← IP_A^Δ + j · ρ^Δ;
9              ρ^Δ ← ρ^Δ / N^Δ;
10             IP_Ω^Δ ← IP_A^Δ + N^Δ · ρ^Δ − 1;
11         else
               // Subnet granularity /32 reached, raise alarm
12             IPDestinationIdentifiedAlarm(IP_j^Δ);
13             IP_A^Δ ← 0, IP_Ω^Δ ← 2^32 − 1, ρ^Δ ← 2^24;
14     else
           // No attack detected, reset monitored IP range
15         IP_A^Δ ← 0, IP_Ω^Δ ← 2^32 − 1, ρ^Δ ← 2^24;
```

Algorithm 5 shows the iterative multi-class traffic image classification approach with a traffic image resolution of $\chi = (256, 256)$ exemplarily. However, the resolution only affects the global parameters and not the algorithm itself. Using a higher resolution reduces the number of required iterations to reach the /32 subnet granularity (complete IPv4 address), e.g., four iterations for $\chi = (256, 256)$ compared to eight iterations for $\chi = (16, 16)$.

4.7 Conclusion

This chapter introduced the **traffic image**, a fixed-state and IP prefix-preserving traffic aggregate. The traffic image maintains the time window-based monitoring paradigm of eMinD while shifting the network traffic monitoring to the Ternary Content Addressable Memory (TCAM) of ingress routers. This design eliminates the need for dedicated monitoring servers per ingress router. The traffic image represents the arriving network traffic as two-dimensional grid of IP subnet pairs (source to destination), counting the packets per subnet pair. This representation preserves the source and destination IP address distributions, as well as the overall packet count, which are essential for volumetric DDoS attack detection (many attack sources, one destination, and a high packet rate).

The traffic image, periodically fetched from the TCAM of an ingress router, serves as the input for different Machine Learning (ML)-based DDoS detection approaches, which are inspired by the research area of computer vision, namely binary classification, time series classification, multi-class classification, and image segmentation. Each approach follows a specific goal, i.e., binary attack detection, destination IP subnet identification, and filter rule derivation. The evaluation has been performed with real-world datasets, from the Internet backbone (MAWI+CAIDA) and from the Internet Service Provider (ISP) BelWue, as well as specifically tailored synthetic DDoS attack scenarios to assess the detection capabilities and potential trade-offs. The experiments showed the following key findings:

- The traffic image, although highly aggregating ingress traffic, provides enough information to precisely detect volumetric DDoS attacks through binary classification, which also works for low-volume distributed attacks, i.e., one percent of the legitimate traffic volume.
- Higher resolution images provide more information about the traffic's IP address distributions, improving the detection performance at the cost of increased memory utilization at the classification server and TCAM rule space.
- Time series classification improves the detection performance by capturing traffic development over time, but introduces a trade-off between memory utilization at the classification server and the detection performance, as more traffic images need to be stored.
- The traffic image segmentation enables the derivation of source-to-destination IP subnet pairs that contain attack traffic. These identified subnet pairs can be used to filter network traffic (with potentially high collateral damage) directly at the ingress router, or to reduce the load on mitigation and filtering services by redirecting only traffic from identified subnet pairs.

Chapter 5

Feedback-driven Autonomous Denial-of-Service Traffic Data Labeling

The previous chapters introduced time window-based supervised ML approaches for Distributed Denial-of-Service (DDoS) attack detection. These presented approaches rely on labeled datasets for ML model training to indicate which traffic is legitimate and which traffic is part of a DDoS attack. For all used datasets in this dissertation (CIC-IDS 2017, MAWI+CAIDA, and BelWue), the ground truth labels are available, e.g., the CIC-IDS 2017 dataset was created in a testbed environment and the attack sources, destinations, and attack periods are known. However, in real-world scenarios, such as the monitoring of network traffic in an ISP network, the ground truth labels are not available. Only the overall monitored traffic, such as microflows or traffic images, are given without any distinction. As manual labeling of the monitored traffic is infeasible at the scale of an ISP, e.g., 90 thousand microflows per second in the BelWue dataset, an automated approach for traffic data labeling is required that keeps the network expert out of the labeling loop.

This chapter explains and evaluates FeADable [KLZ25], a **Fe**edback-driven and **A**utonomous Denial-of-Service (DoS) traffic **Da**ta la**bel**ing approach. It is designed to create labeled datasets for (D)DoS attack detection in ISP networks without the need for manual data labeling. FeADable employs anomaly detection with autoencoders, facilitating temporal feedback about occurred attacks (attack periods, i.e., start time, end time, and destination IP address of an attack), to omit attack traffic from autoencoder training. The trained autoencoders are used to differentiate between legitimate and attack traffic during reported attack periods, creating labeled datasets for subsequent training of ML DDoS detection models. FeADable enables fine-granular traffic data labeling (e.g., per traffic image column or microflow) based on coarse-granular attack periods and its feasibility

has been shown in prior work with the traffic image representation of a major tier-1 ISP's border router traffic [KLZ25].

This chapter contains the following contributions:

- A detailed design of FeADable with its individual components and their respective functionality.
- An evaluation of FeADable's labeling performance regarding traffic images with tier-1 ISP border router traffic, the MAWI+CAIDA dataset and the BelWue dataset.
- An evaluation of FeADable's applicability to microflow-based traffic data labeling for DoS attack traffic with the CIC-IDS 2017 dataset.
- An evaluation of FeADable's major drawback, namely its weakness against false negative attack feedback

The chapter's structure. Section 5.1 outlines FeADable, while explaining the labeling pipeline, its individual components and their functionality in detail. Section 5.2 presents a comprehensive evaluation of FeADable with real-world datasets. First, the results of traffic image column labeling is shown with tier-1 ISP data, evaluating the labeling performance, the resulting classification performance, as well as the generalization capabilities. Second, FeADable's applicability to Denial-of-Service attack traffic in the microflow representation is shown. And third, the application to traffic image column labeling for the BelWue and the MAWI+CAIDA datasets is evaluated. Section 5.3 concludes the chapter and summarizes the key findings.

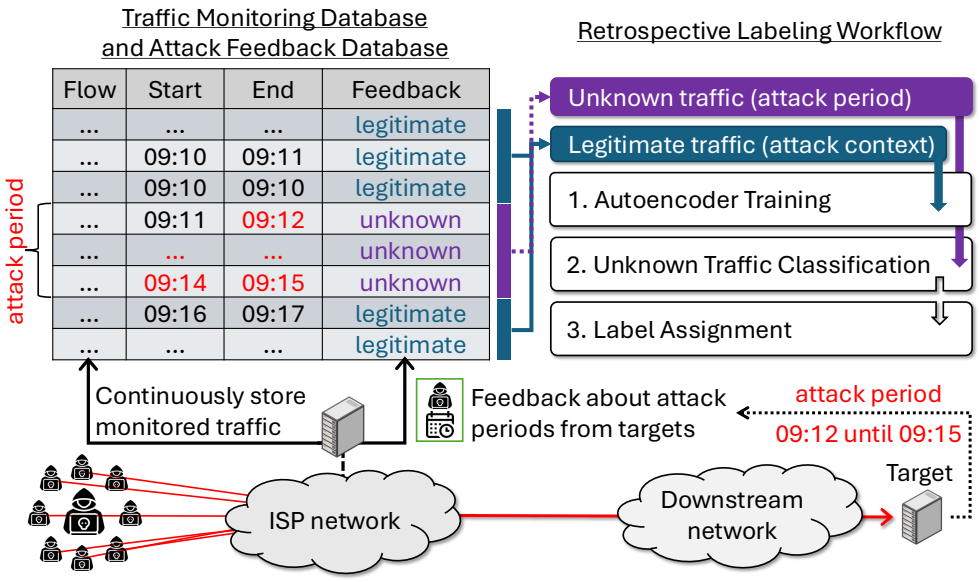


Figure 5.1: *The FeADable monitoring, feedback obtainment, and labeling approach.*

5.1 Functionality of FeADable

FeADable (see Figure 5.1) runs on a server in control of the ISP and requires access to two data sources: First, the traffic monitoring database and second, the attack feedback database. The traffic monitoring database stores the complete monitored border router traffic (e.g., traffic images or microflows). FeADable assumes that entries in the traffic monitoring database contain at least the start time and end time of the monitored traffic data, e.g., the time window of a traffic image or the start and end time of a microflow. FeADable assumes that continuous network traffic monitoring of the ISP's border router traffic is implemented.

Definition 5.1: Traffic Monitoring Database

The traffic monitoring database is defined as a set of monitored traffic data entries $D_{\text{traffic}} = \{d_1, d_2, \ldots, d_n\}$, where each entry d_i contains at least the start time t^i_{start} and end time t^i_{end} of the data entry recording together with the monitored traffic data.

The attack feedback database stores attack feedback reported from customers' end systems or obtained through polling. FeADable assumes that a service is deployed at the ISP that enables customers to report attack feedback to the ISP (details in Section 5.1.1). Attack feedback holds the following information about reported (D)DoS attacks: The

attack's start time, the attack's end time, and the attack's target IP address (the victim end system).

Definition 5.2: Attack Feedback Database

The attack feedback database is defined as a set of attack feedback entries $D_{\text{feedback}} = \{f_1, f_2, \ldots, f_m\}$, where each entry f_i is a three-tuple $(t^i_{\text{start}}, t^i_{\text{end}}, \text{IP}^i_{\text{dst}})$ containing the attack's start time t^i_{start}, the attack's end time t^i_{end}, and the attack's destination IP address IP^i_{dst} of the reported attack.

If the traffic monitoring database and the attack feedback database hold data with intersecting time periods of the monitored traffic and the attack feedback entries, FeADable's labeling pipeline starts, which consists of three main steps:

(1) **Training data selection** (Section 5.1.2): For each received attack feedback, FeADable selects legitimate traffic data before and after the attack period from the traffic monitoring database.

(2) **Autoencoder training** (Section 5.1.3): FeADable trains an autoencoder with the selected legitimate traffic data to learn the legitimate traffic characteristics.

(3) **Label assignment** (Section 5.1.4): FeADable uses the trained autoencoders to classify the monitored traffic during the attack period (unknown traffic as it can either be legitimate or attack traffic) and assigns labels according to the classification results.

5.1.1 Attack Feedback Obtainment

Before the labeling pipeline can be initiated, a DDoS attack must have occurred and the corresponding attack feedback must be obtained and stored in the attack feedback database. This section proposes two approaches for feedback obtainment from ISP customers. As a first option for feedback obtainment, the ISP can provide a **polling** service running on the associated polling server that can be subscripted by customers for desired end systems, such as web servers. The customer enters the IP address of the end system and the corresponding service type (provided by this end system, e.g., an HTTP web application serving port 80 and 443). After the subscription by the customer, the corresponding end system is polled (by the polling server) periodically through sending requests, e.g., HTTP GET requests (depends on the service type), and checking for received responses. If the polling server does not receive a response from the subscripted end system, it is assumed to be unavailable and an according attack feedback entry is stored in the attack feedback database. However, the polling approach is prone to false

positive attack feedback because end systems can be unavailable for different reasons other than (D)DoS attacks, e.g., maintenance or network issues. This chapter's evaluation provides experiments that show FeADable's resilience to false positive feedback, maintaining polling as a viable option for attack feedback obtainment. On the other hand, polling does not suffer from false negative attack feedback, i.e., an attack period is not recognized, as it is assumed that (D)DoS attacks deny the availability of the target end system. Furthermore, as polling receives responses periodically, the temporal precision of the attack feedback (the precise start and end time of the attack) depends on the polling interval.

As a second option for feedback obtainment, the ISP can implement a **reporting** service that allows customers to actively report (e.g., through Distributed Denial-of-Service Open Threat Signaling [Red+20]) occurring (or occurred) DoS attacks to the ISP with the corresponding attack period and target IP address. Reporting can strictly avoid false positive attack feedback as customers have access to detailed information, such as end system logs, to determine the reason for the unavailability, e.g., maintenance or Internet connectivity issues. In addition to more detailed information about the unavailability cause, the reported attack feedback can provide a higher temporal precision of the attack period, as customers can determine the precise start and end time of the attack through end system logs. In contrast to polling (passively performed from the perspective of the customer), reporting requires active adaptations on the customer side, i.e., availability monitoring and sending attack feedback.

Table 5.1: *Benefits and drawbacks of feedback polling and reporting.*

Benefits	**Polling**	**Reporting**
Precise attack periods	✗	✔
No false positives	✗	✔
No false negatives	✔	✔
Scales with end system count	✗	✔
No end system adaption	✔	✗

Table 5.1 summarizes the benefits and drawbacks of the polling and reporting feedback obtainment approaches. Reporting enables precise attack periods, avoids false positive and false negative attack feedback, and scales with the number of registered end systems, but requires adaptations on the customer side. In contrast, polling is performed passively from the customer's perspective and avoids false negatives, but cannot provide precise attack periods while being prone to false positive attack feedback.

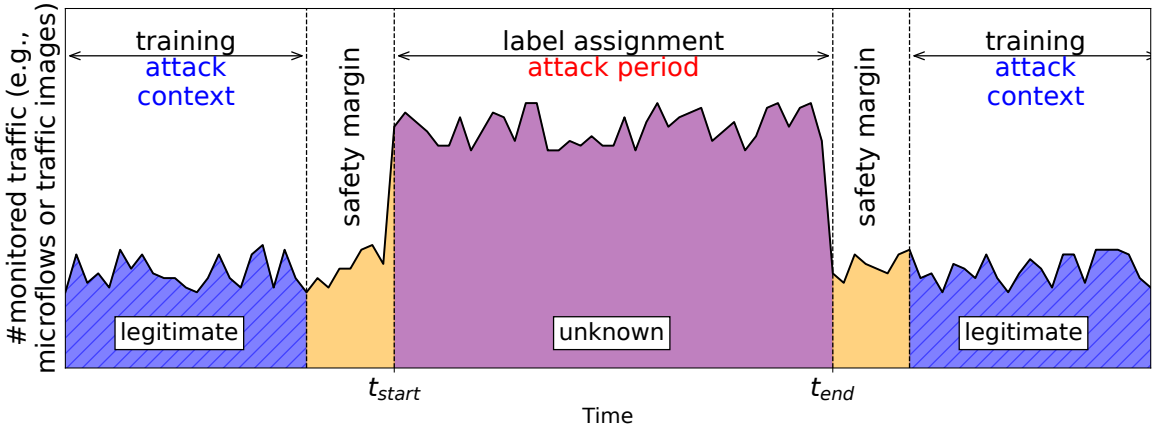


Figure 5.2: *Illustration of the attack period, safety margin, and attack context for an attack feedback entry.*

5.1.2 Training Data Selection

For every attack feedback entry in the attack feedback database, FeADable determines three time periods: the **attack period** (between the attack feedback's start time and end time), the **safety margin** (immediately before and after the attack period), and the **attack context** (the time periods before and after the safety margin) (see Figure 5.2).

The **attack period** constitutes the time interval from the reported start time to the end time of the attack feedback entry. Therefore, this period is given. All entries in the traffic monitoring database whose start and end time intersect with the attack period are considered as **unknown data** and will be labeled by FeADable later. They are unknown because they can either belong to legitimate traffic or attack traffic. For example, a microflow that starts before the attack period and ends during the attack period is considered unknown data, such as a microflow that starts during the attack period and ends after the attack period.

The **attack context** defines the time intervals before and after the attack period that contain the traffic data for training the autoencoder, which is assumed to be legitimate traffic. In the experiments of this chapter, the attack context always represents the complete data set without the attack period as the available data sets are very small. Further reducing the available training data by restricting the attack context would lead to insufficient training data for the autoencoder.

The **safety margin** defines time intervals immediately before and after the attack period that exclude traffic data from the attack context to avoid including attack traffic in the

training data due to imprecise attack feedback in real-world deployments. However, as the ground truth of the attack periods in the used data sets is known, the safety margin is set to zero in the experiments of this chapter. Instead, one experiment is conducted to evaluate the impact of imprecise attack feedback on the labeling performance of FeADable.

After the time period determination above, FeADable extracts all entries from the traffic monitoring database that belong to the attack context as **training data** for the autoencoder. This selection can be further restricted to all traffic monitoring database entries matching the destination IP address of the reported attack feedback.

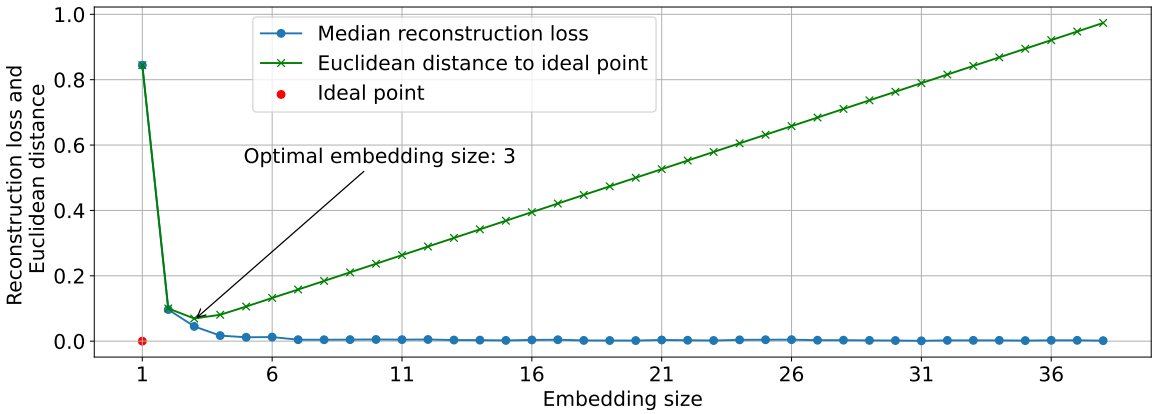


Figure 5.3: *FeADable hyperparameter optimization example results for the CIC-IDS 2017 dataset.*

5.1.3 Autoencoder Training

FeADable uses the selected training data to train an autoencoder to reconstruct the traffic data during the attack context through a lower-dimensional embedding layer (lower dimension than the input features). The selection of the **embedding size** (number of neurons in the embedding layer) is crucial as it defines the amount of the input features' information that can be retained in the embedding. If the embedding size is too small, traffic characteristics of the training data might not be preserved, which leads to high reconstruction losses even for traffic similar to the training data. On the other hand, if the embedding size is too large (e.g., equal to the input feature size), the autoencoder might learn to copy the input data directly to the output without learning traffic characteristics, which would lead to low reconstructions losses for all types of traffic, including traffic that has not been encountered during the training (making a later distinction between legitimate and attack traffic impossible). Therefore, the first step of the autoencoder training is to find hyperparameters, including the number of hidden layers, the number

of neurons per layer, and the embedding size, that achieve low reconstruction losses for the training data while keeping the embedding size as small as possible.

Hyperparameter Optimization
To find the best hyperparameters for the autoencoder, FeADable performs a grid search over a predefined hyperparameter space including the number of hidden layers, the neurons per layer, and the embedding size. The embedding size is varied from 1 (smallest possible embedding) to the input feature size (no compression). The hyperparameter optimization is performed directly on the selected training data, which is split into training and validation sets (80%/20%). For each hyperparameter combination, an autoencoder is trained on the training set, and the reconstruction loss (Mean Squared Error) is evaluated on the validation set after training. All results can be assigned to two conflicting objectives:

- Minimize the median reconstruction loss on the validation set.
- Minimize the embedding size.

These two objectives are conflicting, as smaller embedding sizes typically lead to higher reconstruction losses. Therefore, FeADable calculates the euclidean distance of each result's median reconstruction loss and embedding size to an **ideal point** defined as $(1, 0)$, i.e., an embedding size of 1 and a median reconstruction loss of 0. The hyperparameter combination with the smallest distance to the ideal point is selected for the final autoencoder training (see Figure 5.3). As the embedding size has a higher impact on the distance to the ideal point than the reconstruction loss, the embedding sizes are normalized to the range $[1, 2]$ and the reconstruction losses to $[0, 1]$ before calculating the distances, which assigns both objectives equal importance.

The autoencoder that achieves the best trade-off between reconstruction loss and embedding size is selected for the label assignment step. This hyperparameter optimization is performed for each attack feedback entry individually, as the traffic in the attack context can differ significantly between different attack feedback entries.

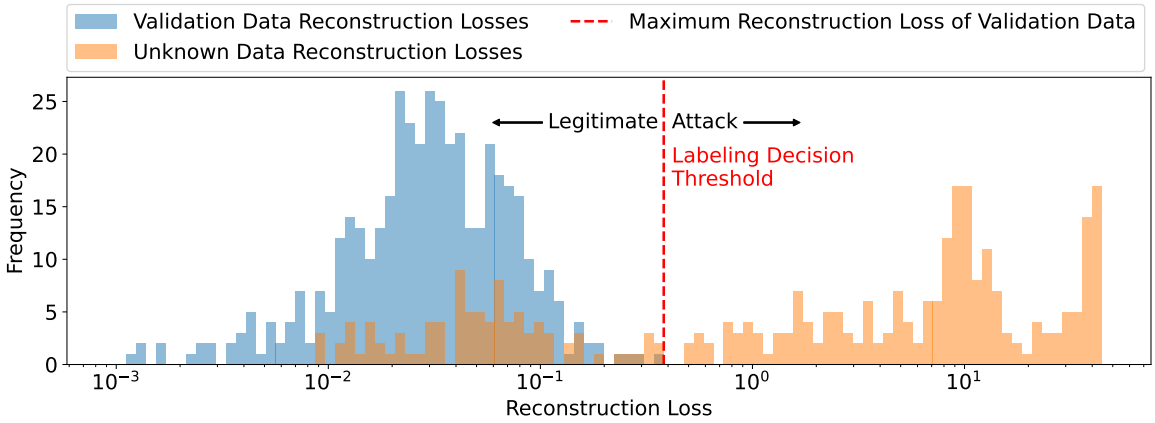


Figure 5.4: *Illustration of FeADable's labeling decision threshold based on reconstruction losses of the validation set and unknown data.*

5.1.4 Label Assignment

After training data selection, hyperparameter optimization, and autoencoder training, FeADable uses the trained autoencoder to classify the unknown data during the attack period. Based on the achieved reconstruction losses of the unknown data, FeADable decides whether the data belongs to the legitimate or the attack traffic. For this decision, FeADable defines the **labeling decision threshold**, which is defined as the maximum reconstruction loss encountered in the validation set during autoencoder training. If unknown data achieves a reconstruction loss smaller than or equal to this threshold, it is assumed that the autoencoder has encountered similar traffic data during training and the traffic data is labeled as legitimate traffic. Otherwise, if the reconstruction loss is larger than the threshold, it is assumed that the autoencoder has not encountered similar traffic data during training and the traffic data is labeled as attack traffic. Figure 5.4 illustrates this concept with a histogram (data from the MAWI+CAIDA dataset) of reconstruction losses for the legitimate training data (validation set in blue) and the unknown data (orange). The labeling decision threshold (dashed vertical line) separates the histogram in two areas: The area left of the threshold containing traffic data similar to the training data (labeled as legitimate) and the area right of the threshold containing traffic data dissimilar to the training data (labeled as attack traffic).

The labeling decision threshold is explicitly chosen to be the maximum reconstruction loss of the validation set to avoid labeling legitimate traffic as attack traffic (false positives). The threshold can also be chosen smaller to capture more attack traffic (true positives), but this increases the risk of false positives. It can also be chosen larger to further avoid false positives, but this increases the risk of missing attack traffic (false negatives). This trade-off is later evaluated in this chapter.

5.2 Evaluation

This section covers the evaluation of FeADable with four different real-world traffic datasets: (Section 5.2.1) a major Tier-1 ISP dataset, (Section 5.2.2) the CIC-IDS2017 dataset, (Section 5.2.3) the BelWue dataset, and (Section 5.2.4) the MAWI+CAIDA dataset. The first three datasets contain real-world DDoS attack traffic in the traffic image representation. The CIC-IDS2017 dataset contains a Denial-of-Service (Slowloris) attack in the microflow representation and is used to demonstrate the applicability of FeADable to different traffic monitoring approaches and attack vectors. The labeling performance of FeADable is examined as well as the detection performance of a subsequently trained DDoS detector. Furthermore, FeADable's resilience to false positive and its vulnerability to false negative attack feedback is shown (Section 5.2.5). In addition, the impact of the labeling decision threshold selection is analyzed.

5.2.1 Evaluation with Major Tier-1 ISP Traffic Data

The first experiment is conducted on 10 consecutive days of a major Tier-1 ISP's border router traffic in February 2025. The dataset comprises one hour of complete border router traffic per day, where 15 minutes of the CAIDA DDoS attack is injected in the middle of each hour. Perfect attack feedback is assumed, i.e., the start and end time of the attack are known exactly and no safety margin is applied. The experiment is conducted with traffic images, time window duration $\delta = 1s$ and resolution $\chi = (256, 256, 4)$, where each image has four color channels representing the protocols TCP, UDP, ICMP, and Others (see Section 4.6.2). This high resolution is possible, as the traffic images are not monitored in TCAM but created from border router flow exports (IPFix and NetFlow), which enables monitoring in RAM. The labeling is performed based on traffic image columns. This means that all traffic images are separated into their individual columns, and each column is labeled individually as attack or legitimate. Only the columns with the destination subnet under attack are used, which is obtained from the attack feedback.

The evaluation goals of this experiment are twofold: First, the labeling performance of FeADable is examined regarding the precision and the recall on all ten days. Second, the detection performance of an ML-based DDoS detector (fully connected neural network classifying traffic image columns) is evaluated. This ML-based DDoS detector is trained with the resulting labels of FeADable on the first day (February 1, 2025) and evaluated on the remaining nine days without retraining. The goal is to show that a viable DDoS detector can be trained with FeADable's labels, that it generalizes well to unseen data, and that it is therefore suitable for real-world deployment.

Due to confidentiality reasons, only the results can be presented and no detailed information about the dataset can be disclosed.

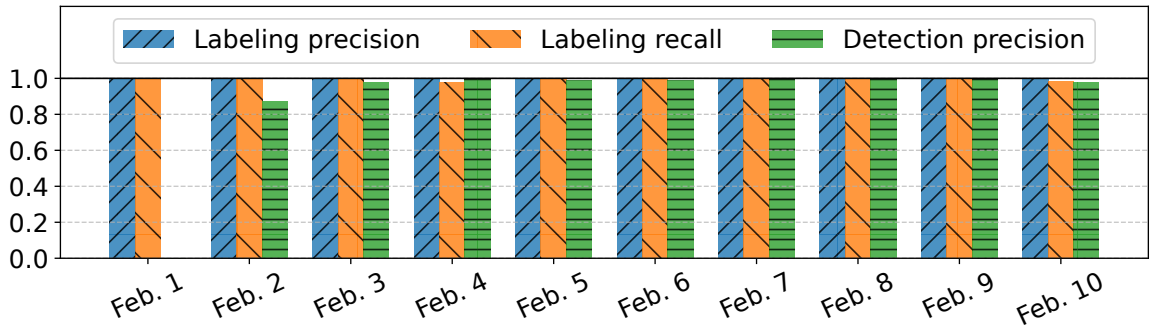


Figure 5.5: *Labeling performance of FeADable (with traffic images) on a major Tier-1 ISP's border router traffic in February 2025 (10 consecutive days).*

Figure 5.5 shows FeADable's labeling performance (precision and recall) on the major Tier-1 ISP dataset on all ten days, as well as the detection performance (precision) of the subsequently trained ML-based DDoS detector (remaining nine days). The labeling precision is indicated by the blue bars, the labeling recall by the orange bars, and the detection precision of the ML-based DDoS detector by the green bars.

The labeling precision is 1.0 on all ten days, which means that no false positive labels are created by FeADable. This is crucial for real-world deployment, as false positive labels would lead to the training of an ML-based DDoS detector that classifies legitimate traffic as attack traffic and therefore raises false alarms. The recall is almost 1.0 on all ten days, except for the fourth and the tenth day, where it is slightly lower. However, almost all attack columns are detected by FeADable and no attack column is falsely labeled.

The detection precision of the ML-based DDoS detector is 1.0 on the fourth, the seventh, the eighth, and the ninth day. The precision on the third, the fifth, the sixth, and the tenth day shows a small degradation. The precision on the second day is below 0.9. This degradation effect arises as the overall dataset has not been sanitized and contains other attack traffic besides the injected CAIDA DDoS attack. Consequently, it leads to falsely detected attack columns (that could in fact be true positives) according to the ground truth labels, which only consider the injected CAIDA DDoS attack.

The results were achieved without retraining the ML model on the remaining nine days and show that **FeADable enables precise labeling and subsequent training of a supervised ML-based DDoS detector in a real-world and large-scale ISP environment**.

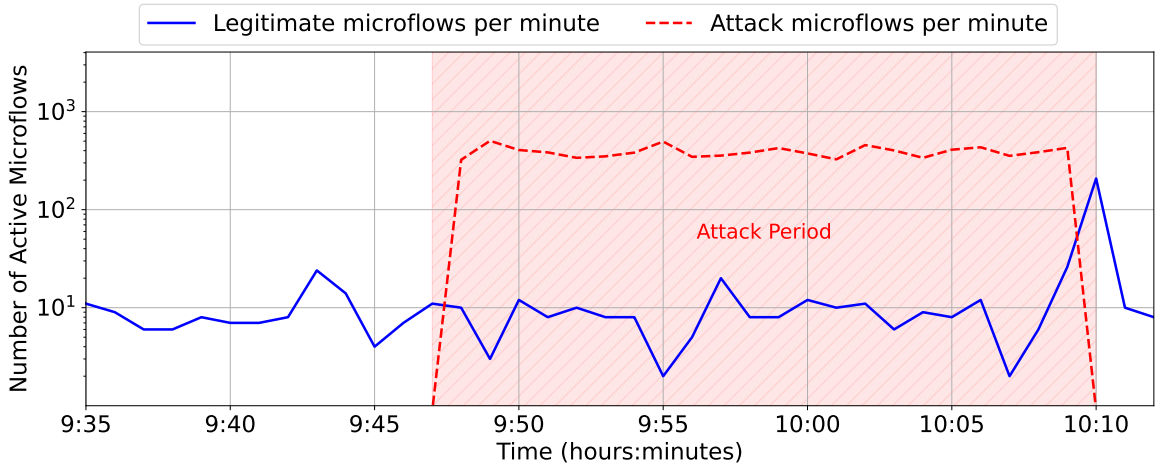


Figure 5.6: *Microflow count per minute for the CIC-IDS2017 Slowloris attack.*

5.2.2 Application to Denial-of-Service Attacks and Microflows

This section demonstrates the applicability of FeADable to the Slowloris Denial-of-Service (DoS) attack and microflow-based traffic monitoring. The CIC-IDS2017 dataset contains a Slowloris attack (in the Wednesday traffic recording) that is conducted between 9:47am and 10:10am. Figure 5.6 shows the number of active microflows over time for the legitimate traffic (blue line) and the Slowloris attack traffic (dashed red line). Note that the y-axis is scaled logarithmically and there is significantly less legitimate traffic than attack traffic. The overall traffic is already filtered to only include microflows towards the target IP address of the Slowloris attack, as FeADable restricts the traffic scope according to the obtained attack feedback containing the attack's target IP address. The attack period is indicated by the red shaded area.

All microflow features provided through the original dataset are used for the autoencoder training, except the source and destination IP addresses to avoid overfitting. The source and destination ports are transformed through binary encoding (each port number is represented by 16 binary features representing the bits of the port number) and the transport protocol is one-hot encoded (one binary feature per protocol).

The attack context is chosen from 9:35am to 10:14am excluding the attack period (9:47am to 10:10am). The end of the attack context is selected 4 minutes after the attack ends, as a new DoS attack starts at 10:14am in the dataset. The attack feedback is assumed to be perfect, i.e., the start and end time of the attack are known exactly as well as the target IP address. Therefore, no safety margin is applied. With this setup,

the hyperparameter optimization is conducted and the labeling decision threshold is determined with training data exclusively from the attack context.

Table 5.2: *FeADable's labeling performance on CIC-IDS2017 Slowloris attack*

Metric	Value
Precision	0.9997
Recall	0.5798
True Positives	3244
False Positives	1
True Negatives	192
False Negatives	2351

Labeling Performance
Table 5.2 shows FeADable's labeling performance on the CIC-IDS2017 Slowloris attack for different metrics (relative and absolute). Overall, FeADable achieves a very high precision of 0.9997 as only one false positive label is created. As already mentioned, this property is crucial for real-world deployment to avoid the training of an ML-based DDoS detector that raises false alarms. The recall is 0.5798, meaning that approximately 58% of all attack microflows are identified by FeADable. While a higher recall is desirable, FeADable still outputs 3244 true positive microflows that can be used for training an ML-based DDoS detector. Microflows labeled as legitimate (true negatives and false negatives) are omitted for potential subsequent training for two reasons: First, FeADable is designed to identify attack traffic with high precision to avoid false positives while not guaranteeing a high recall. Second, the traffic monitoring database contains enough legitimate traffic for dataset creation without further reliance on the legitimate traffic during attack periods. Training a simple fully-connected neural network for binary microflow classification with FeADable's resulting attack labels and the legitimate traffic of the attack context shows why the low recall is acceptable.

Detection Performance
The simple neural network consists of three hidden layers with 128, 64, and 32 neurons. ReLU activation functions are used for all hidden layers and a sigmoid activation function for the output layer. The model training is conducted over 500 epochs with early stopping regarding the validation loss (patience of 10 epochs). The trained model is evaluated on the false negatives and true negatives of FeADable's labeling results to show that the model can identify unseen attack microflows. This simple neural network achieves a recall of 94.6% and a false positive rate of 0.52%, which means that it is able to identify

most of the attack microflows that were not labeled by FeADable during training, while maintaining a low false positive rate.

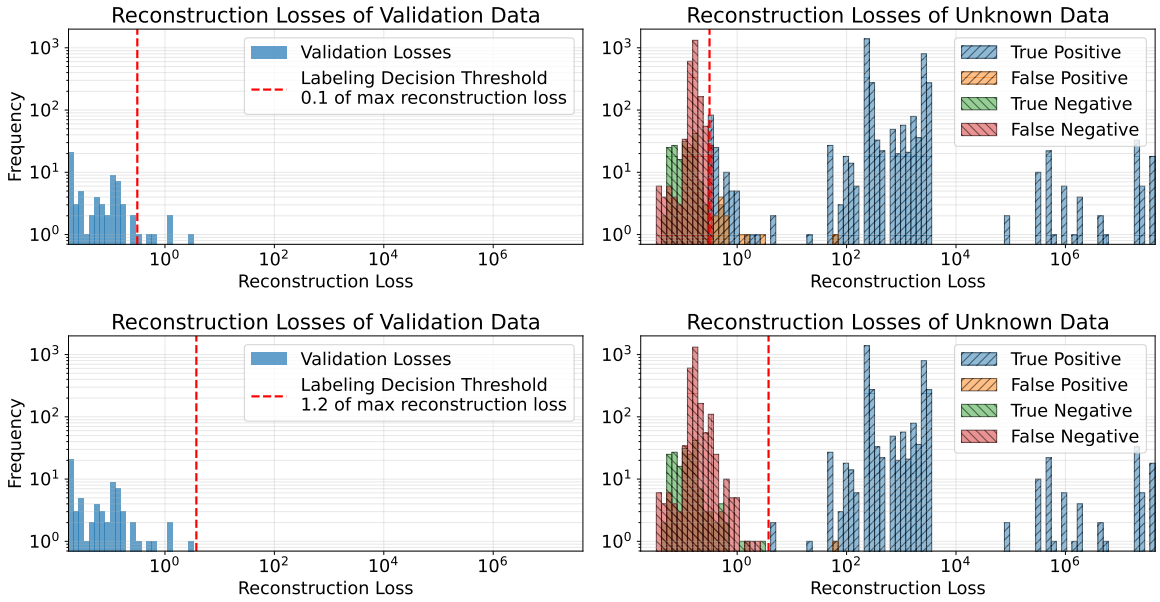


Figure 5.7: *Reconstruction loss distributions of the validation data and the unknown data for the CIC-IDS2017 Slowloris attack.*

Labeling Decision Threshold

To maintain high labeling precision, FeADable is designed to have its labeling decision threshold set to the maximum reconstruction loss observed on the validation data during the training. The training data is assumed to contain only legitimate traffic and every unknown microflow, having a reconstruction loss within the loss range of the training data, should be labeled as legitimate. To show, why this design decision is important to maintain high precision, the following experiment is conducted: FeADable is trained as before but the labeling decision threshold is varied between 0% and 120% of the maximum reconstruction loss observed on the validation data to show the impact on the resulting labeling precision and recall.

To give an understanding of the labeling decision threshold's impact, Figure 5.7 shows the results for two selected labeling decision thresholds (10% and 120%) in each row. On the left side, the reconstruction loss distribution of the validation data is shown with the corresponding labeling decision threshold (dashed, vertical red line). The right side shows the reconstruction loss distribution of the unknown data colored by the labeling result. The respective labeling decision threshold is again indicated. All samples to the left of the labeling decision threshold are labeled as legitimate and all samples to the right as attack. Therefore, left of the threshold, only true negatives and false negatives

are present, while to the right, only true positives and false positives exist. Increasing the labeling decision threshold from 10% to 120% shifts the dashed red line to the right, which leads to more samples being labeled as legitimate and less samples being labeled as attack. This inherently increases the precision (less false positives – less yellow) but decreases the recall (more false negatives – more red). Imagining the extreme case of setting the labeling decision threshold to 0%, all unknown samples would be labeled as attack, leading to a recall of 1.0 but a very low precision. Conversely, setting the labeling decision threshold to a very high value would label all unknown samples as legitimate, resulting in a recall of 0.0 but no false positives at all.

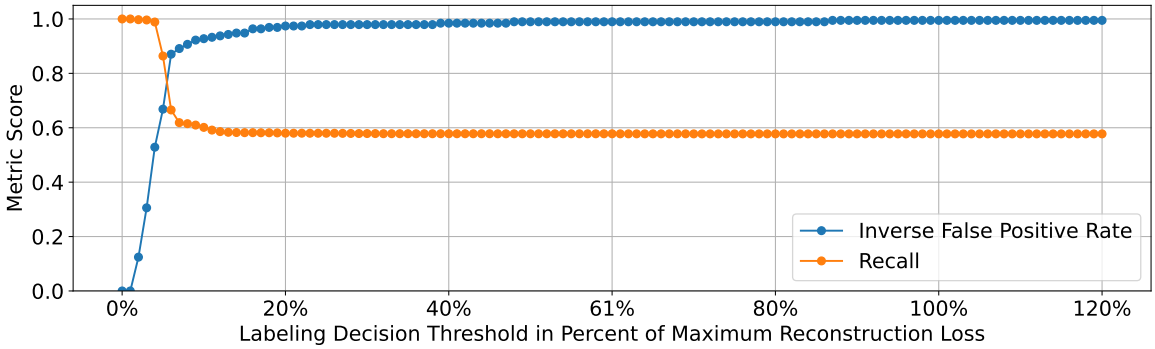


Figure 5.8: *Trade-off between false positive rate and recall for different labeling decision thresholds (relative to the maximum reconstruction loss for the validation data) on the CIC-IDS2017 Slowloris attack.*

Figure 5.8 shows the results (recall and inverse FPR) for all labeling decision thresholds between 0% and 120% (in steps of 1%) of the maximum reconstruction loss observed on the validation data. The recall is indicated by the orange line and the inverse FPR by the blue line. As expected, the recall decreases with increasing labeling decision threshold, while the inverse FPR increases. This shows the inherent trade-off between the labeling recall and the precision and justifies the design decision of FeADable to set the labeling decision threshold to the maximum reconstruction loss observed on the validation data to maintain high precision.

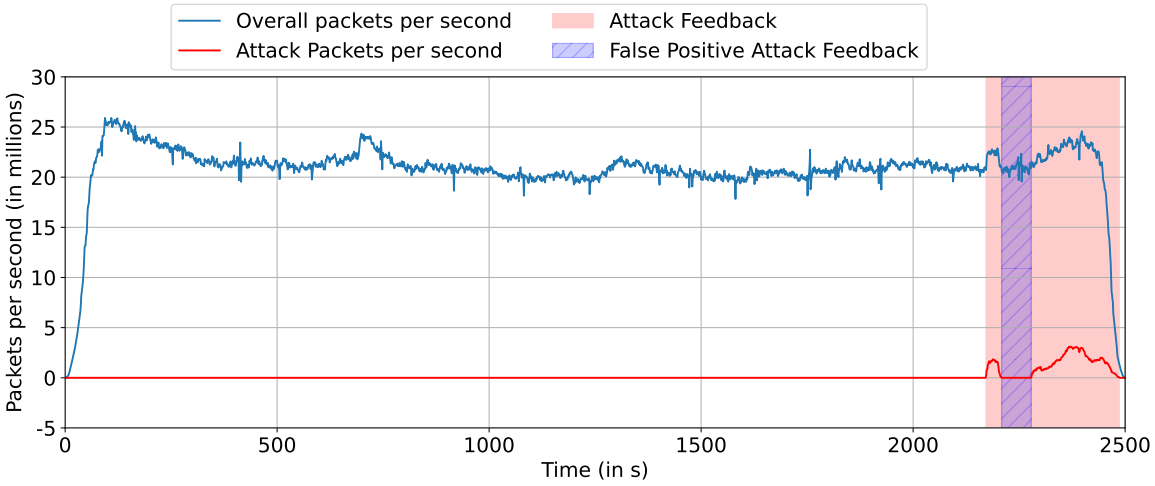


Figure 5.9: *Packet count over time for the BelWue dataset, showing the overall packet count in blue, the attack packet count in red, and the attack feedback.*

5.2.3 Evaluation with the BelWue Dataset

This section evaluates FeADable applied to traffic images generated from the BelWue dataset. Traffic image-based labeling with FeADable operates on traffic image columns, as the attack feedback provides the destination IP address under attack (besides the attack start and end time). This enables FeADable to restrict the traffic scope to only the traffic image columns that contain traffic towards the attacked destination IP address.

Figure 5.9 shows the packet count over time for the BelWue dataset (overall packet count in blue and attack packet count in red). The attack traffic contains two attack waves separated by a gap period of only legitimate traffic. However, the used attack feedback for FeADable (indicated by the red area) starts with the beginning of the first attack wave and ends with the end of the second attack wave, i.e., the attack feedback spans both attack waves and the gap period. This results in false positive attack feedback during the gap period (indicated by the blue area), as an attack is reported while no attack traffic is present.

The goal of this experiment is twofold: First, the labeling performance of FeADable is evaluated in a real-world ISP scenario. Second, the resilience of FeADable to false positive attack feedback is demonstrated, i.e., FeADable should not label legitimate traffic as attack traffic during the gap period.

The attack context is chosen as the complete dataset before the attack period. The data during the attack period (including the gap period) is used as unknown data for

labeling. With the attack context, the hyperparameter optimization is conducted, which contains a full grid search for all traffic image resolutions and the following hyperparameters: number of hidden layers $(1, 2, 3, \ldots, 10)$, number of neurons per hidden layer $(16, 32, 64, \ldots, 512)$, and the embedding size $(1, 2, 3, \ldots, N^{\Sigma}$, where N^{Σ} is the number of source IP subnets in the traffic image columns depending on the resolution). The aggregated results of the grid search for all resolutions are presented in Section A.2.1.

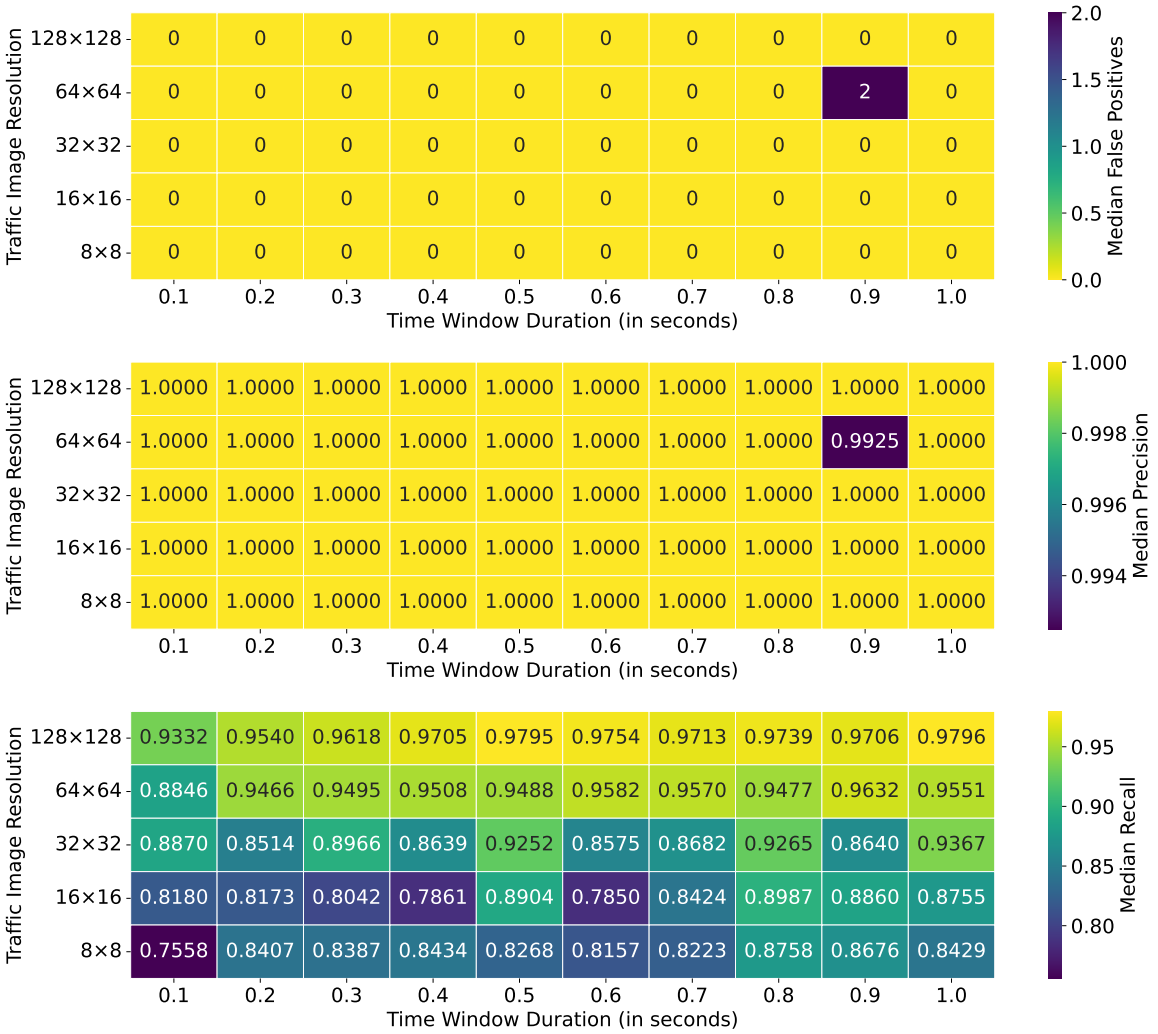


Figure 5.10: *FeADable labeling performance of the BelWue dataset for different time window durations and traffic image resolutions.*

Figure 5.10 shows the labeling performance results for different time window durations and traffic image resolutions. The median absolute false positives, the median precision, and the median recall across 20 runs are shown from top to bottom. The first observation is that FeADable achieves a very high precision of 1.0 for all time window durations and traffic image resolutions, except one outlier for the time window duration of 0.9s and the resolution of $(64, 64)$, where two false positives occur. The second observation is that the resolution of $(128, 128)$ outperforms all other resolutions regarding the recall. This constitutes a trade-off between memory consumption (higher resolution requires more memory) and labeling performance.

The worst achieved recall is approximately 75% for the time window duration of 0.1s and the resolution of $(8, 8)$. This means that three out of four attack columns are identified by FeADable, while no false positive labels are created. As shown for the CIC-IDS2017 dataset, an effective ML-based DDoS detector can be trained with the resulting attack labels despite the low recall ($\approx$ 58% in the CIC-IDS2017 DoS scenario). The important property is that no false positive labels are created to avoid false alarms of the subsequently trained DDoS detector. The best achieved recall is approximately 98% for the time window duration of 1.0s and the resolution of $(128, 128)$, meaning that almost all attack columns are identified by FeADable while maintaining a perfect precision.

The results of this experiment show that FeADable is applicable to real-world ISP scenarios and is able to maintain a high labeling precision even in the presence of false positive attack feedback.

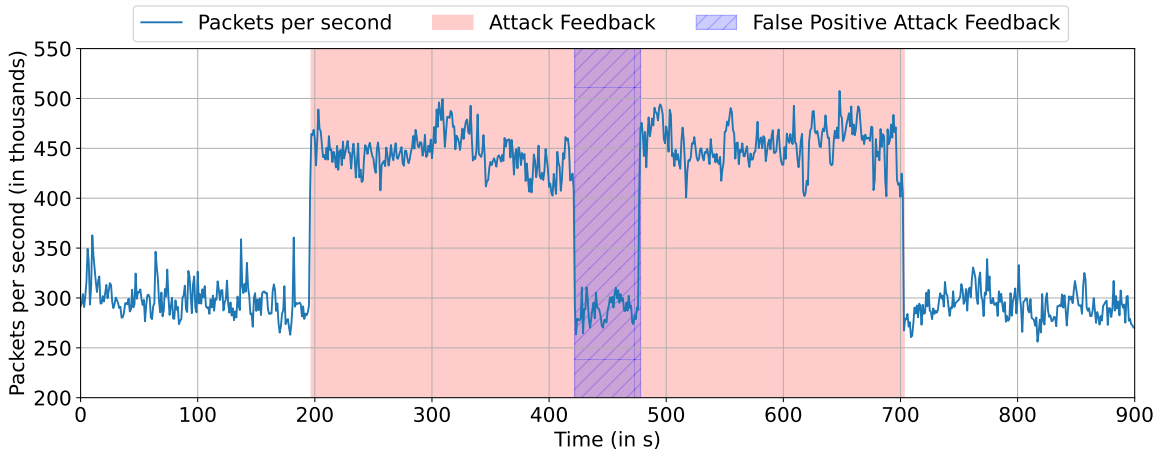


Figure 5.11: *Packet count over time for the MAWI+CAIDA dataset for the labeling evaluation with indicated attack feedback.*

5.2.4 Evaluation with the MAWI+CAIDA Dataset

This section evaluates FeADable applied to traffic images generated from the MAWI+CAIDA dataset. Previously, the MAWI+CAIDA dataset contained attack traffic (CAIDA) that was coherently injected into the legitimate traffic (MAWI) in the middle of the 15-minute traffic trace. In this experiment, the CAIDA attack traffic is split in two parts that are injected in the middle of the dataset with a gap period that contains only legitimate traffic (see Figure 5.11). In total, 450 seconds of attack traffic and 450 seconds of legitimate traffic are maintained (as in the traffic image-based detection experiments). The dataset can consequently be split into five parts:

(1) $450s \cdot \frac{7}{16} = 196.875s$ of legitimate MAWI traffic

(2) $225s$ with injected CAIDA attack traffic (first part of the attack)

(3) $450s \cdot \frac{2}{16} = 56.25s$ of legitimate MAWI traffic (gap period)

(4) $225s$ with injected CAIDA attack traffic (second part of the attack)

(5) $450s \cdot \frac{7}{16} = 196.875s$ of legitimate MAWI traffic

The first and the fifth part constitute the attack context, while the second to fourth part constitute the attack period (unknown data), which includes the gap period. The gap period is introduced to evaluate FeADable's resilience to false positive attack feedback, similar to the BelWue dataset experiment that contained two attack waves separated by a gap period per se. Consequently, the attack feedback for this experiment is assumed to span both attack parts and the gap period, i.e., the attack feedback indicates the attack duration $t_{start} = 196.875s$ to $t_{end} = 196.875s + 225s + 56.25s + 225s = 703.125s$.

The autoencoder hyperparameter optimization is again performed per traffic image resolution with the data of the attack context. A full grid search is conducted for all traffic image resolutions and the following hyperparameters: number of hidden layers $(1, 2, 3, \ldots, 10)$, number of neurons per hidden layer $(16, 32, 64, \ldots, 512)$, and the embedding size $(1, 2, 3, \ldots, N^{\Sigma}$, where N^{Σ} is the number of source IP subnets in the traffic image columns depending on the resolution). The aggregated results of the grid search for all traffic image resolutions are presented in Section A.2.2.

The labeling performance results for different time window durations and traffic image resolutions are presented in Figure 5.12. The absolute median false positives, the median precision, and the median recall across 20 runs are shown from top to bottom.
From the results, the following observations can be made: First, FeADable achieves 6 false positives in the worst case (time window duration of $1.0s$ and traffic image

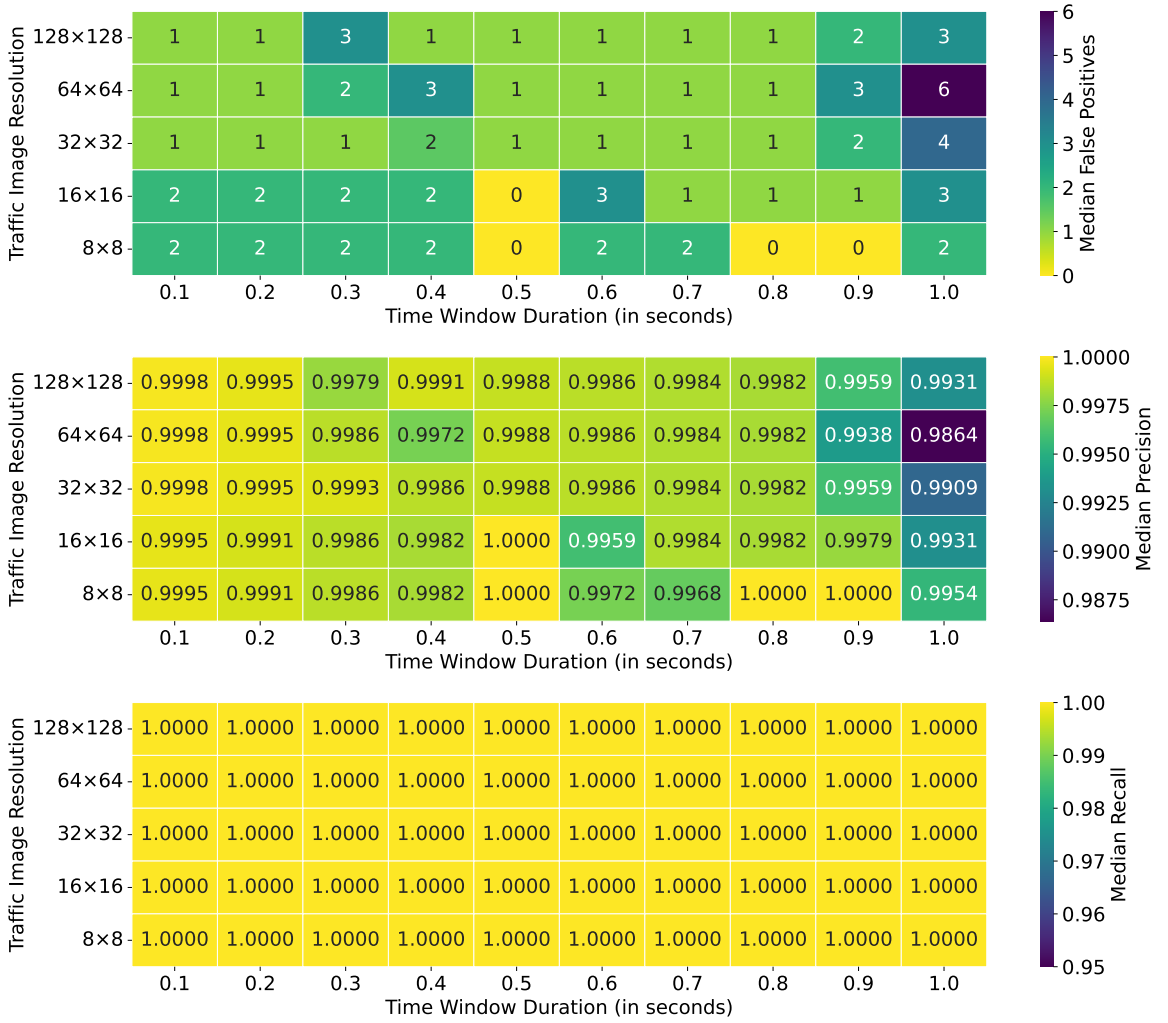


Figure 5.12: *Labeling performance of FeADable on the MAWI+CAIDA dataset for different time window durations and traffic image resolutions.*

resolution $(64, 64)$) considering all combinations of time window durations and traffic image resolutions. Second, FeADable achieves perfect recall across all time window durations and traffic image resolutions. This means that all attack columns are identified by FeADable. The overall worse labeling precision compared to the BelWue dataset experiment can be explained by the lack of training data in the attack context. The MAWI+CAIDA dataset contains only approximately $394s$ of legitimate traffic in the attack context, which results in 394 traffic image columns for a time window duration of $1.0s$. In contrast, the BelWue dataset contain more than $2100s$ of legitimate traffic in the attack context, resulting in more than 2100 traffic image columns for a time window duration of $1.0s$.

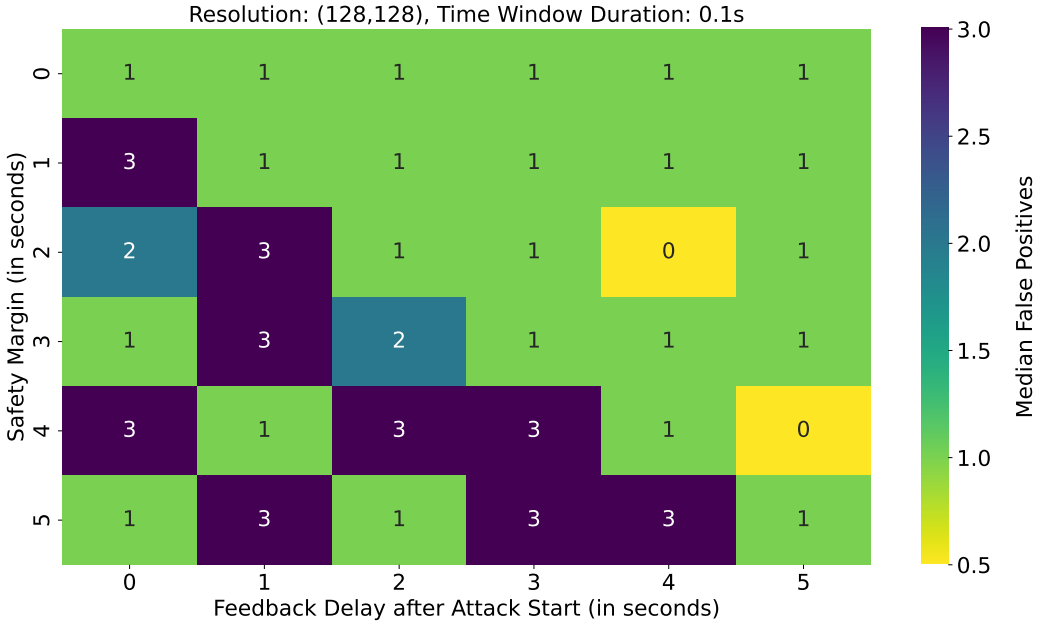


Figure 5.13: *Median false positives for different combinations of safety margins and attack feedback delays on the MAWI+CAIDA dataset.*

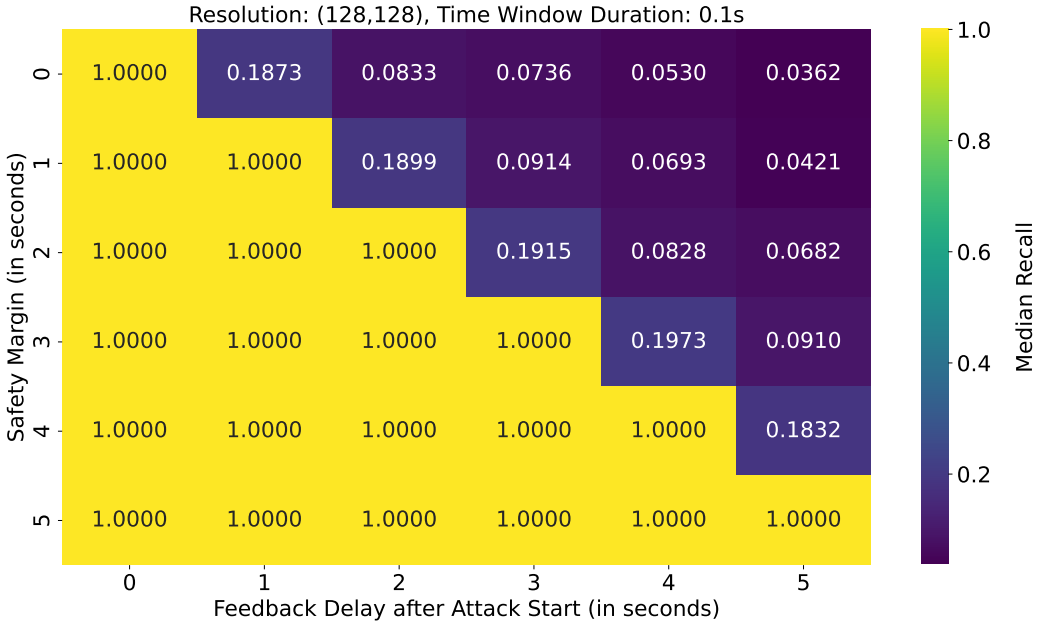


Figure 5.14: *Median recall of different combinations of safety margins and attack feedback delays on the MAWI+CAIDA dataset.*

5.2.5 Impact of Imprecise Attack Feedback

This section evaluates the impact of imprecise attack feedback on FeADable's labeling performance and shows FeADable's major drawback: its vulnerability to false negative attack feedback. The experiment is conducted with a time window duration of $0.1s$ and a traffic image resolution of $(128, 128)$ exemplarily.

Imprecise attack feedback is created by adding a **feedback delay** to the start time of the attack in the reported attack feedback. For example, a feedback delay of $5s$ means that the attack feedback indicates an attack start time $5s$ after the actual attack start time. This would lead to $5s$ of false negative attack feedback at the beginning of the attack period if no safety margin is applied. In this experiment, all combinations of feedback delays ($0s$ to $5s$ in steps of $1s$) and safety margins ($0s$ to $5s$ in steps of $1s$) are evaluated with the MAWI+CAIDA dataset used in the previous section (including the legitimate gap period). The expected behavior is that FeADable maintains its labeling performance if the safety margin is greater than or equal to the feedback delay. If the feedback delay is greater than the safety margin, false negative attack feedback occurs, and attack traffic is present in the attack context that is used for autoencoder training.

Figures 5.13 and Figure 5.14 show the median false positives and the median recall across 20 runs, respectively. The diagonal cells from the top-left to the bottom-right represent the parameter combinations where the safety margin is equal to the feedback delay. The cells below the diagonal represent the combinations where the safety margin is greater than the feedback delay, and the cells above the diagonal represent the combinations where the feedback delay is greater than the safety margin.

From the results, the following observations can be made: First, FeADable maintains perfect recall (zero false negatives) if the safety margin is greater than or equal to the feedback delay (cells on and below the diagonal), while achieving false positives between 1 and 3. This result is expected according to the previous experiment, where FeADable achieved perfect recall and a low number of false positives.
Second, if the feedback delay is greater than the safety margin (cells above the diagonal). FeADable's recall degrades significantly (many false negatives occur), while the number of false positives remains low. In these cases, the attack context contains attack traffic due to false negative attack feedback, and the autoencoder learns to reconstruct attack traffic as well. Consequently, FeADable can also reconstruct attack traffic during the classification of unknown data with a small reconstruction loss, leading to less data labeled as attack in total.

These results show that FeADable labeling recall strongly degrades considering if the

attack feedback is imprecise in such way that the attack period is not fully covered through the attack feedback. It emphasizes the importance of the safety margin to ensure that no attack traffic is present in the attack context. However, even if the attack traffic is present in the attack context used for training, FeADable still outputs only very few false positive labels.

5.3 Conclusion

This chapter explained and evaluated FeADable, an autonomous DoS attack traffic labeling approach that leverages autoencoder-based anomaly detection enriched with feedback about occurred attacks. FeADables enables fine-granular label derivation for attack traffic, e.g., microflows or traffic image columns, from coarse attack information, i.e., the start time, the end time, and the destination IP address of a DoS attack. FeADable utilizes traffic monitored immediately before and after reported attack periods for autoencoder training, which enables to capture the legitimate traffic characteristics at the time of the attack. This enables FeADable to differentiate between legitimate and attack traffic during the reported attack periods based on the reconstruction loss of the trained autoencoder.

FeADable's labeling capabilities have been shown with multiple real-world datasets, including traffic from a major tier-1 ISP's border routers, the MAWI+CAIDA dataset, the BelWue dataset, and the CIC-IDS 2017 dataset. The evaluation of FeADable demonstrated its high labeling precision, which is a crucial requirement for attack traffic labeling as the resulting labels are used for training DDoS detectors. If the labeling precision is low, many false positive labels would occur in the training data and the trained DDoS detector would incorrectly classify legitimate traffic as attack traffic.

Despite the high labeling precision, FeADable's labeling recall depends on the quality of the attack feedback. If the feedback is imprecise and attack traffic is present in the training data of the autoencoder, the labeling recall degrades significantly. However, even in these cases, FeADable still maintains a high labeling precision.

To summarize, this chapter made the following contributions:

- Explaining the design of FeADable, a feedback-driven and autonomous DoS traffic data labeling approach
- Evaluation with multiple real-world datasets, demonstrating high labeling precision including an ISP scenario
- Investigating a trade-off between the labeling precision and the labeling recall depending on the parameterization of FeADable (labeling decision threshold)

- Showing FeADable's applicability to different traffic representations, including traffic images and microflows
- Demonstrating the drawback of FeADable when facing imprecise attack feedback

Chapter 6

Conclusion

This dissertation focused on resource-predictable machine learning-based detection of volumetric Distributed Denial-of-Service (DDoS) attacks. In particular, it addressed the challenge of maintaining fixed (and low) computational and memory resource consumption for traffic monitoring and classification, even in the context of large-scale DDoS attacks with tens of thousands of attack sources and millions of attack packets per second.

This challenge was faced by applying a time window-based traffic monitoring paradigm, which performs network traffic monitoring for the whole arriving network traffic according to a fixed set of inspected packet attributes and within fixed-size time periods. In contrast to prevalent microflow-based monitoring for ML-based detection, a time window-based approach enables fixed memory consumption depending on the number of inspected attributes (reduction of $> 99\%$ compared to microflow-based monitoring) and strictly limits the number of required ML classifications to one per time window (reduction of $> 99.99\%$ compared to microflow-based monitoring), independent of the arriving traffic's volume or microflow count. Requiring only one classification per time window enables a fixed and sub-second reaction time to DDoS attacks.

Based on this time window-based monitoring paradigm, two novel DDoS detection approaches were presented in this dissertation, namely eMinD and traffic image-based DDoS detection. eMinD is an efficient and microflow-independent DDoS detection pipeline that utilizes traffic aggregates, e.g., estimates of unique IP addresses and port numbers. It is designed for software-based monitoring on servers collocated with ingress routers. Traffic image-based DDoS detection leverages a novel traffic representation, the traffic image, which maintains source-to-destination IP address distributions of network

traffic within time windows. This representation enables monitoring in the Ternary Content Addressable Memory (TCAM) of ISP ingress routers, which allows monitoring each arriving packet within one clock cycle and makes fetching and aggregating traffic images from multiple routers possible. Furthermore, traffic images enable the application of well-studied computer vision approaches to DDoS detection, such as binary image classification and image segmentation, which makes the derivation of source-to-destination subnet pairs possible that contain attack traffic and therefore enables the derivation of network filter rules or redirection rules to mitigation services.

This dissertation also addressed the challenge of creating labeled datasets for DDoS attack detection in real-world scenarios, such as ISP networks, where no ground truth labels are available and manual data labeling is infeasible due to the large amounts of monitored network traffic. For this purpose, a feedback-driven and autonomous DDoS traffic data labeling approach was designed and evaluated, which leverages autoencoder-based anomaly detection enriched with attack feedback information (start time, end time, and destination IP address of the attack) to create labeled datasets for DDoS attack detection without the need for manual labeling.

The presented approaches were evaluated with multiple real-world datasets, including traffic from a major tier-1 ISP's border routers, the MAWI+CAIDA dataset, the BelWue dataset, and the CIC-IDS 2017 dataset, demonstrating the applicability of the presented approaches to real-world scenarios. Furthermore, to show the benefits and limitations of the presented approaches, synthesized datasets with different attack intensities and IP address distributions were used.

6.1 Future Research

The most important next step to enable reasonable DDoS detection and mitigation research in general is the creation and publication of a large-scale, real-world, benchmark ISP dataset containing real-world legitimate traffic and real-world DDoS attacks comprising different attack vectors, simultaneous attacks, different attack intensities, different attack durations, and different target subnets. This would enable the research community to evaluate and compare different DDoS detection and mitigation approaches in a realistic setting. It would further enable showcasing promising approaches to ISP operators and therefore foster the adoption of research prototypes in production networks. Regarding the presented approaches in this dissertation, future research could address the following topics:

- Traffic image-based DDoS detection should be co-deployed with FeADable to enable a self-improving detection system that autonomously labels attack traffic from

undetected attacks and continuously retrains the DDoS detector using the newly labeled data.

- Traffic image-based multi-class classification should be extended to a multi-label classification setting to account for simultaneous DDoS attacks targeting multiple subnets within a single time window, or alternatively, the classification paradigm should be shifted from image-level classification to image-column classification, which inherently requires multi-channel traffic images.
- Once comprehensive and large datasets of real-world DDoS attacks become available, novel machine learning model architectures for traffic image-based DDoS detection, such as transformer models [Kha+22], should be systematically evaluated.

Appendix A

Appendix

A.1 Hyperparameter Optimization of eMinD

The hyperparameters of the ML models, e.g., the number of trees in a Random Forest (RF) or the learning rate of a Multi-Layer Perceptron (MLP), are tuned using a full grid search approach, i.e., all possible combinations of predefined hyperparameter values are evaluated by measuring the performance of the ML model on the validation set. The hyperparameter combination that achieves the best performance on the validation set is selected as the final model. The scikit-learn [Noa; Var+15] implementation is used for all models to ensure reproducibility.

Table A.1 shows the hyperparameter search space for the MLP model, resulting in 8,640 combinations, while Table A.2 lists the best performing hyperparameter combination.

Table A.3 shows the hyperparameter search space for the Decision Tree model, resulting in 2,304 combinations, while Table A.4 lists the best performing hyperparameter combination.

Table A.5 shows the hyperparameter search space for the Random Forest model, resulting in 3,456 combinations, while Table A.6 lists the best performing hyperparameter combination.

Table A.7 shows the hyperparameter search space for the SVM model, resulting in 192 combinations, while Table A.8 lists the best performing hyperparameter combination.

Table A.1: *MLP hyperparameter grid search configuration*

Hyperparameter	**Values**	**Count**
Hidden Layers	1, 2, 3, 4	4
Units per Layer	64, 128, 256, 512	4
Activation Function	ReLU	1
Output Activation Function	Sigmoid	1
Loss Function	Binary Crossentropy	1
Dropout Rate	0.0, 0.1, 0.2	3
Optimizer	Adam	1
Learning Rate	10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}	4
Batch Size	4, 8, 16, 32, 64	5
Epochs	100, 500, 1000	3
Early Stopping Patience	3, 5, 10	3
Total Combinations		**8,640**

Table A.2: *MLP best hyperparameter configuration*

Hyperparameter	**Best Value**
Hidden Layers	2
Units per Layer	128
Activation Function	ReLU
Output Activation Function	Sigmoid
Loss Function	Binary Crossentropy
Dropout Rate	0.0
Optimizer	Adam
Learning Rate	0.01
Batch Size	32
Epochs	1000
Early Stopping Patience	10

Table A.3: *Decision Tree hyperparameter grid search configuration*

Hyperparameter	**Values**	**Count**
Split Criterion	Gini, Entropy	2
Split Strategy	Best, Random	2
Maximum Depth	None, 5, 10, 15, 20, 30	6
Minimum Samples to Split	2, 5, 10, 20	4
Minimum Samples per Leaf	1, 2, 4, 8	4
Maximum Features	None, Sqrt, Log2	3
Class Weighting	None, Balanced	2
Total Combinations		**2,304**

Table A.4: *Decision Tree best hyperparameter configuration*

Hyperparameter	**Selected Value**
Split Criterion	Gini
Split Strategy	Best
Maximum Depth	None
Minimum Samples to Split	2
Minimum Samples per Leaf	1
Maximum Features	Sqrt
Class Weighting	None

Table A.5: *Random Forest hyperparameter grid search configuration*

Hyperparameter	**Values**	**Count**
Number of Trees	50, 100, 300, 500	4
Split Criterion	Gini, Entropy	2
Maximum Depth	None, 10, 20, 30	4
Minimum Samples to Split	2, 5, 10	3
Minimum Samples per Leaf	1, 2, 4	3
Maximum Features	None, Sqrt, Log2	3
Bootstrap Sampling	True, False	2
Class Weighting	None, Balanced	2
Total Combinations		**3,456**

Table A.6: *Random Forest best hyperparameter configuration*

Hyperparameter	**Selected Value**
Number of Trees	50
Split Criterion	Gini
Maximum Depth	None
Minimum Samples to Split	2
Minimum Samples per Leaf	1
Maximum Features	Sqrt
Bootstrap Sampling	True
Class Weighting	None

Table A.7: *SVM hyperparameter grid search configuration*

Hyperparameter	**Values**	**Count**
Regularization Parameter (C)	0.1, 1, 10, 100	4
Kernel	Linear, Polynomial, RBF, Sigmoid	4
Polynomial Degree	2, 3, 4 (Polynomial only)	3
Kernel Coefficient (γ)	$\frac{1}{n_{\text{features}}}$, $\frac{1}{n_{\text{features}} \cdot \text{Var}(X)}$	2
Class Weighting	None, Balanced	2
Total Combinations		**192**

Table A.8: *SVM best hyperparameter configuration*

Hyperparameter	**Best Value**
Regularization Parameter (C)	1
Kernel	Linear
Polynomial Degree	3
Kernel Coefficient (γ)	$\frac{1}{n_{\text{features}} \cdot \text{Var}(X)}$
Class Weighting	None

A.2 Hyperparameter Optimization for FeADable

This section contains the complete results of the hyperparameter optimization for FeADable's autoencoder training for the BelWue dataset (Section A.2.1) and the MAWI+CAIDA dataset (Section A.2.2).

For both datasets, the hyperparameter optimization contains 20 runs per traffic image resolution and each combination of the following hyperparameters:

- Number of hidden layers: $\{1, 2, 3, \ldots, 10\}$
- Neurons per hidden layer (hidden layer size): $\{16, 32, 64, 128, 256, 512\}$
- Embedding size (bottleneck layer size): $\{1, 2, 3, \ldots, N^{\Sigma}\}$, where N^{Σ} is the number of source IP subnets in the traffic image representation

For each combination of dataset and resolution, two heatmaps are shown:

- The optimal embedding size according to the minimum euclidean distance to the ideal point $(1, 0)$
- The corresponding euclidean distance to the ideal point $(1, 0)$ for the optimal embedding size

Results for embedding sizes different to the optimal one are omitted for clarity. In all figures, the parameterization that achieves the best euclidean distance to the ideal point is highlighted with a red border.

Note that for both datasets and all resolutions the optimal embedding size is 1. This shows that underlying traffic distribution can be effectively captured in a one-dimensional space.

A.2.1 BelWue Dataset

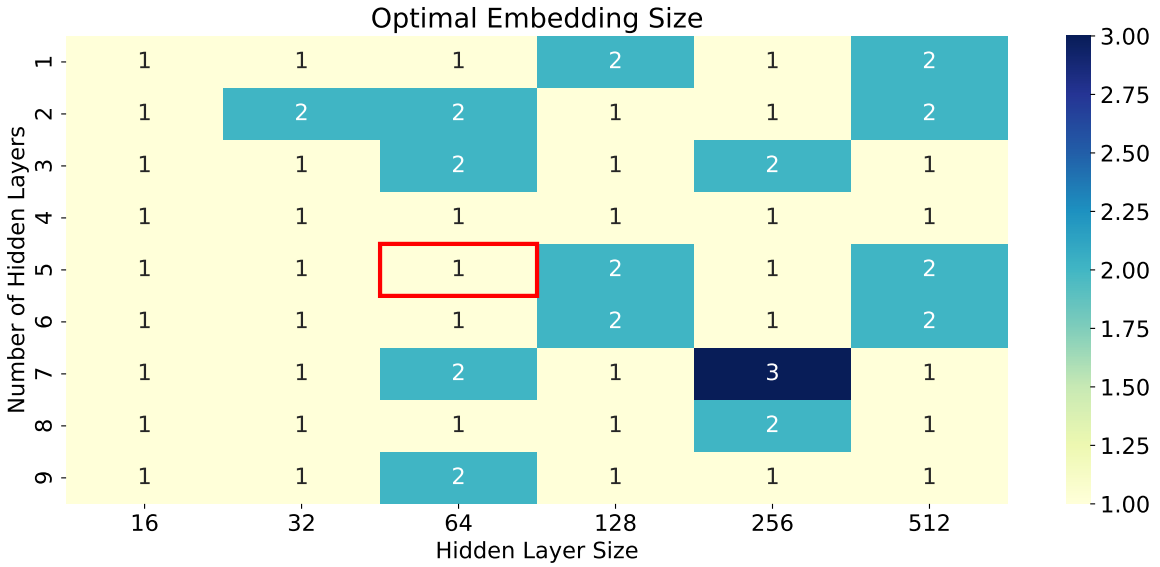


Figure A.1: *Optimal embedding size for the BelWue dataset with traffic image resolution 8*

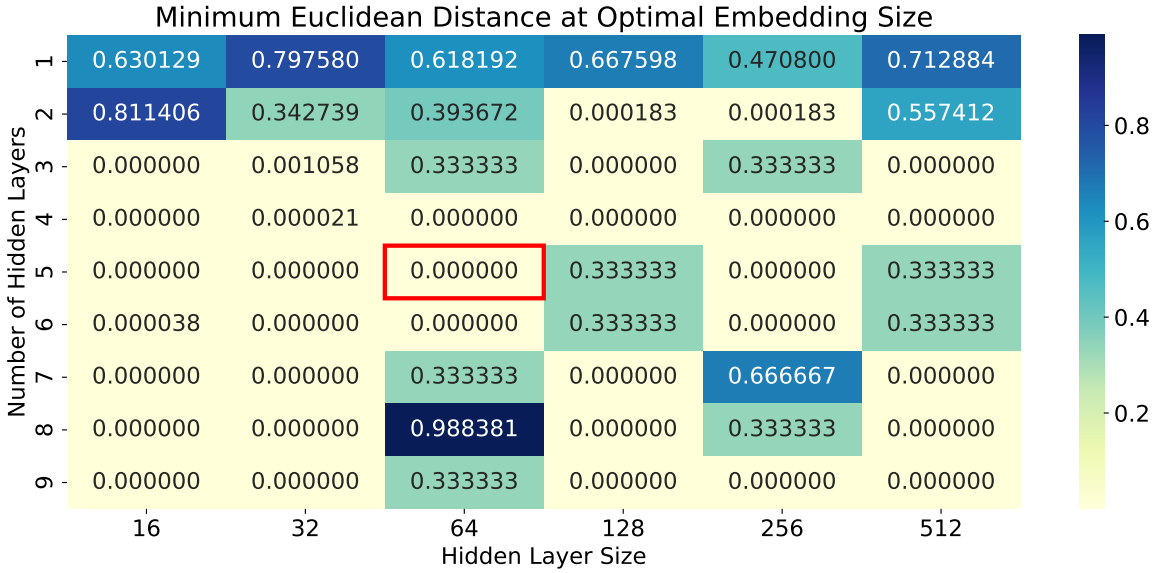


Figure A.2: *Euclidean distance to the ideal point for BelWue dataset with traffic image resolution 8*

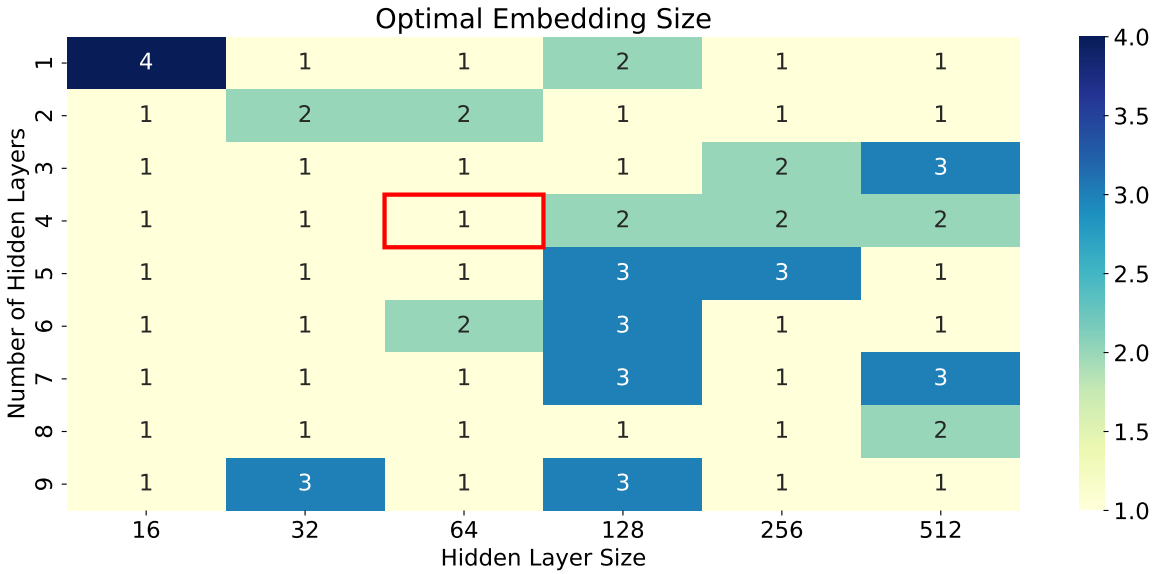


Figure A.3: *Optimal embedding size for the BelWue dataset with traffic image resolution 16*

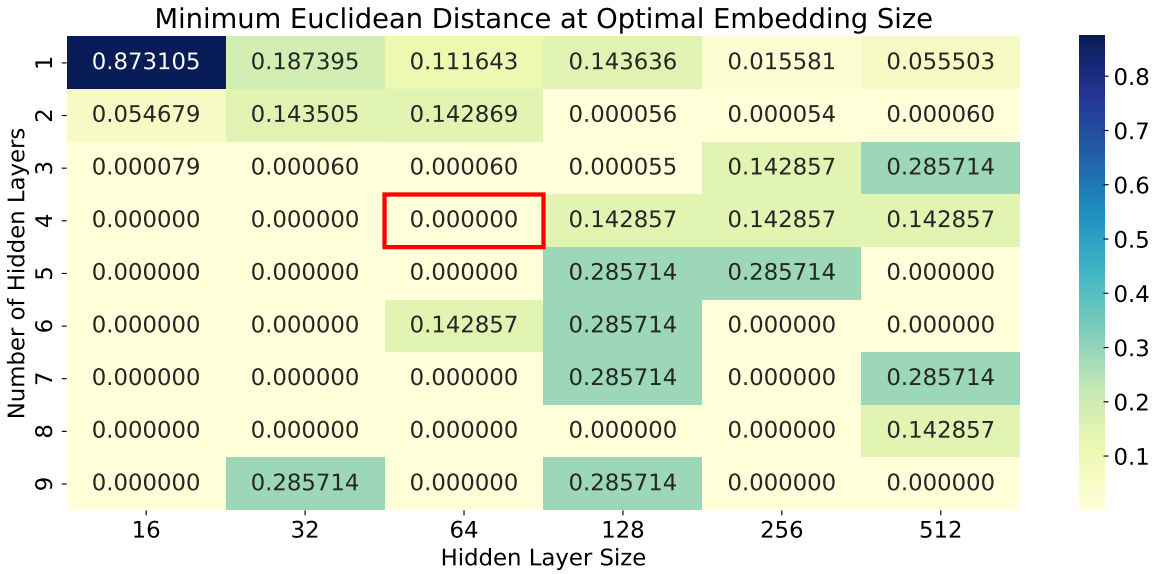


Figure A.4: *Euclidean distance to the ideal point for BelWue dataset with traffic image resolution 16*

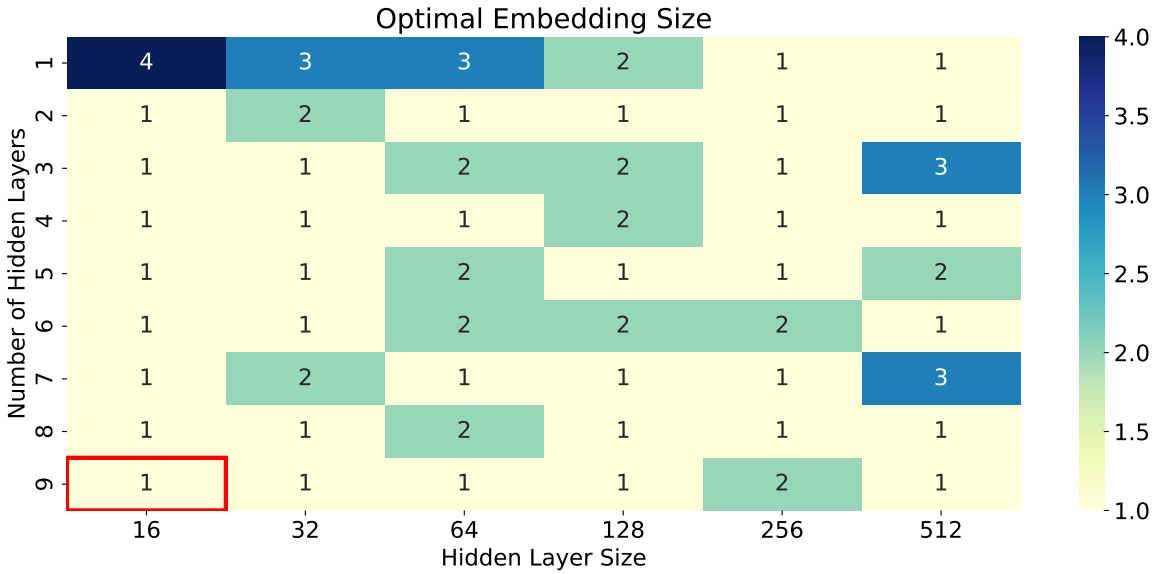


Figure A.5: *Optimal embedding size for BelWue dataset with traffic image resolution 32*

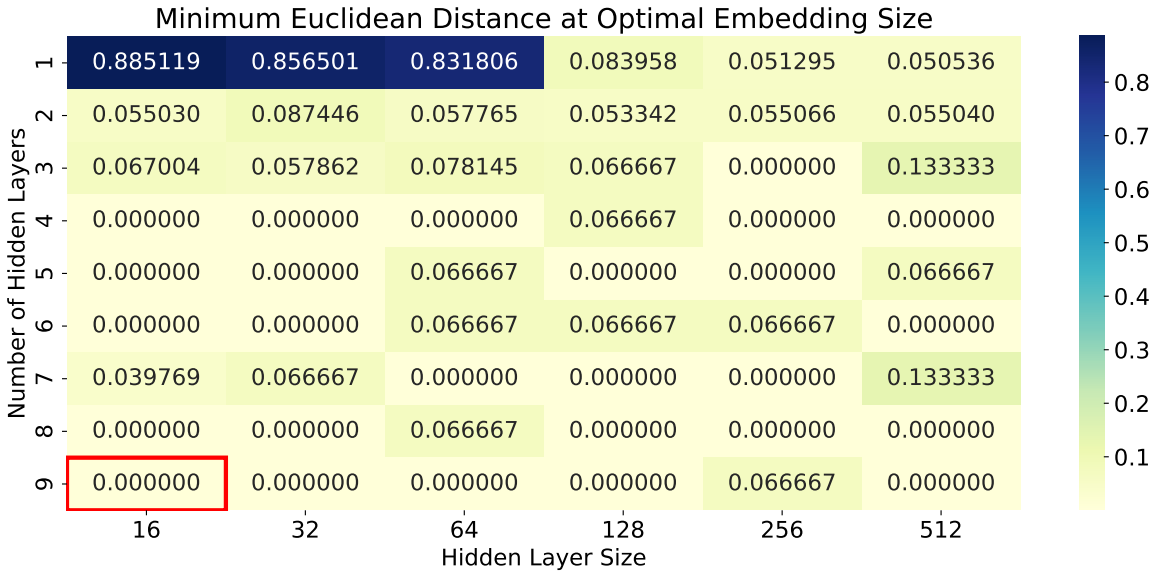


Figure A.6: *Euclidean distance to the ideal point for BelWue dataset with traffic image resolution 32*

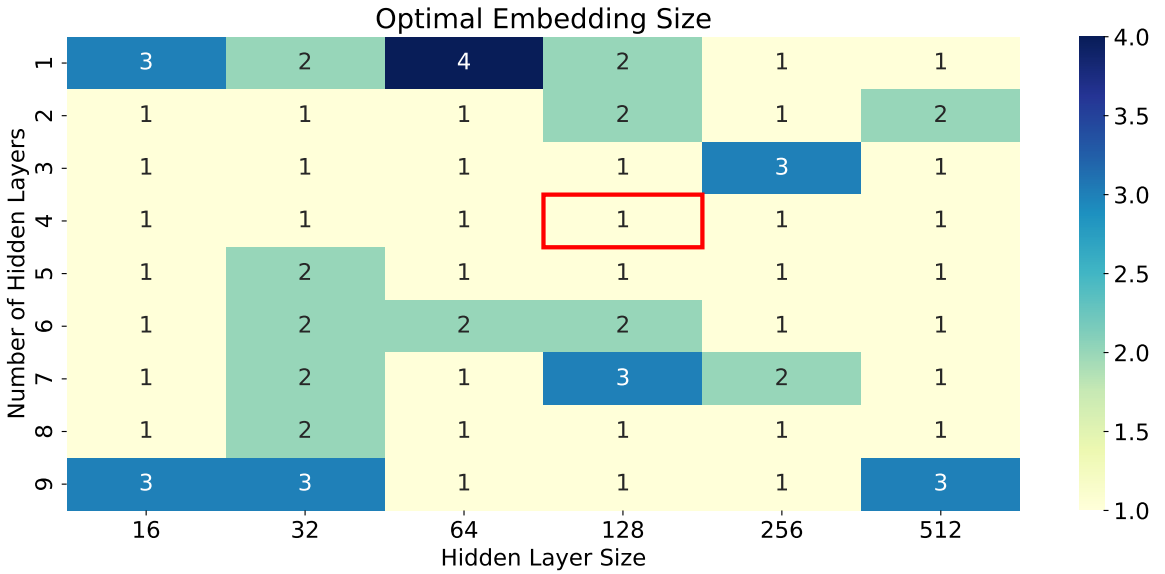


Figure A.7: *Optimal embedding size for the BelWue dataset with traffic image resolution 64*

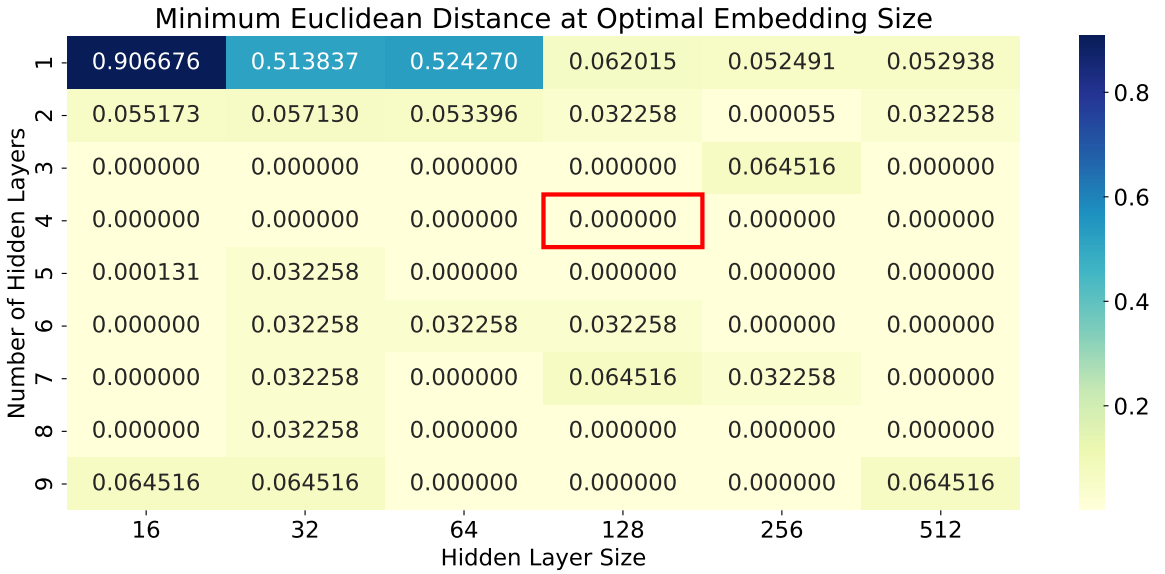


Figure A.8: *Euclidean distance to the ideal point for the BelWue dataset with traffic image resolution 64*

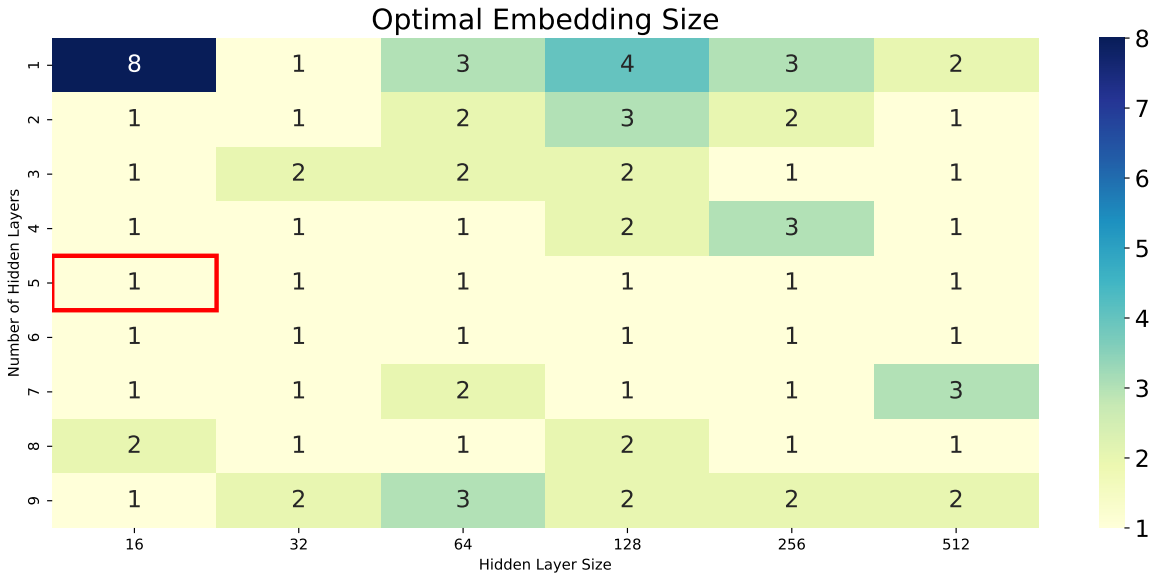


Figure A.9: *Optimal embedding size for the BelWue dataset with traffic image resolution 128*

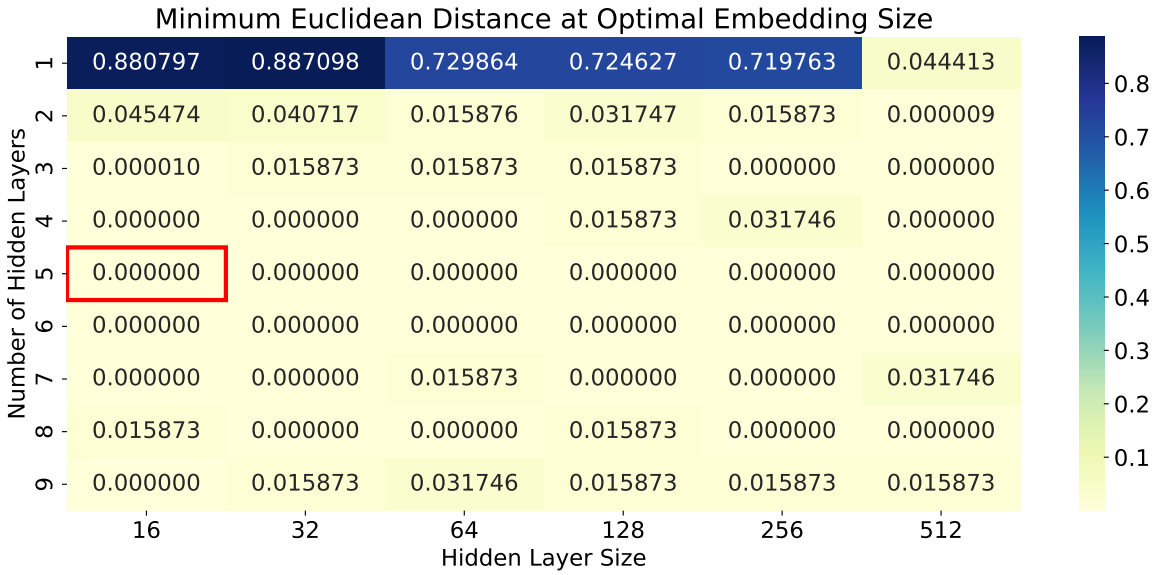


Figure A.10: *Euclidean distance to the ideal point for the BelWue dataset with traffic image resolution 128*

A.2.2 MAWI+CAIDA Dataset

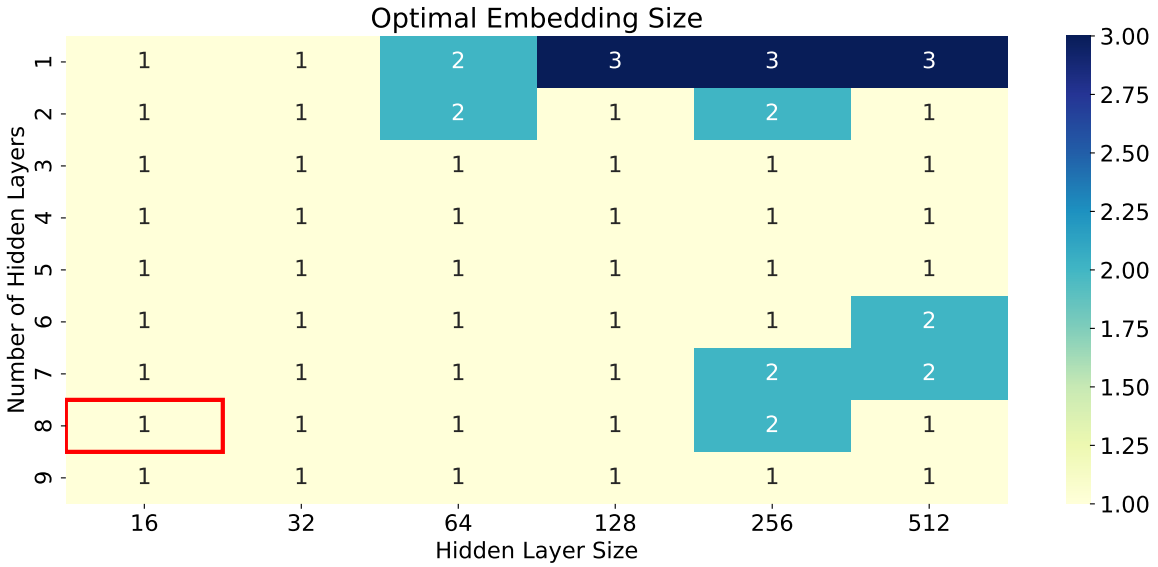


Figure A.11: *Optimal embedding size for the MAWI+CAIDA dataset with traffic image resolution 8*

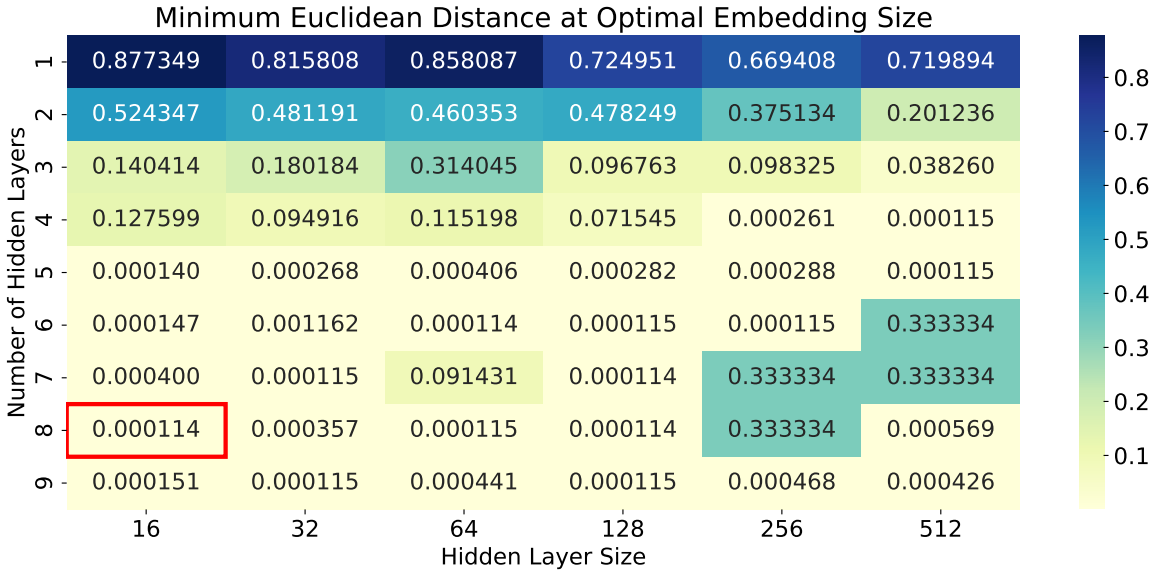


Figure A.12: *Euclidean distance to the ideal point for the MAWI+CAIDA dataset with traffic image resolution 8*

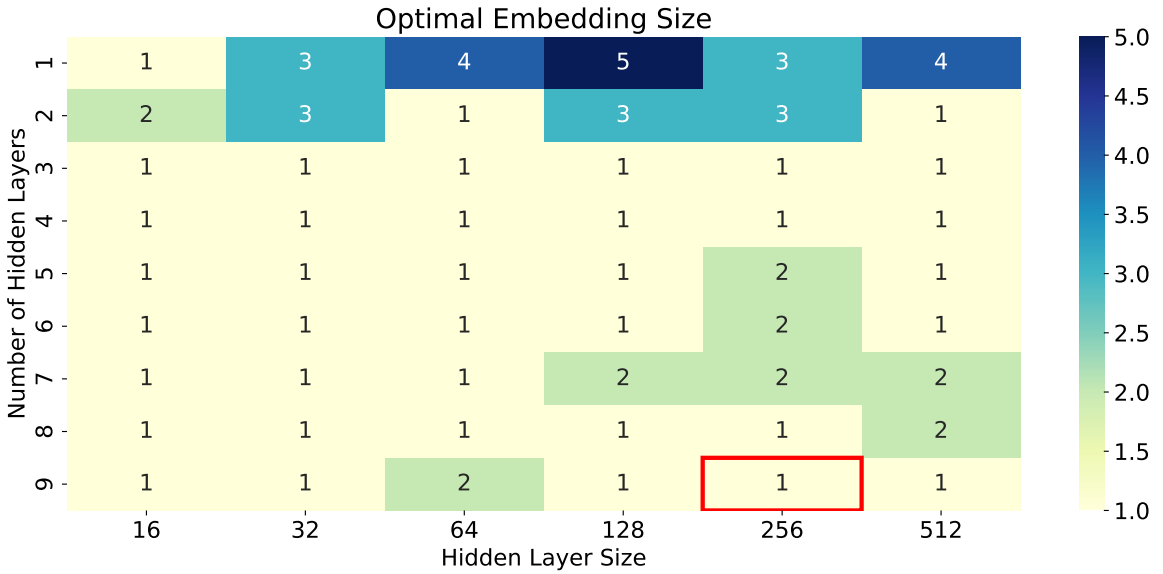


Figure A.13: *Optimal embedding size for the MAWI+CAIDA dataset with traffic image resolution 16*

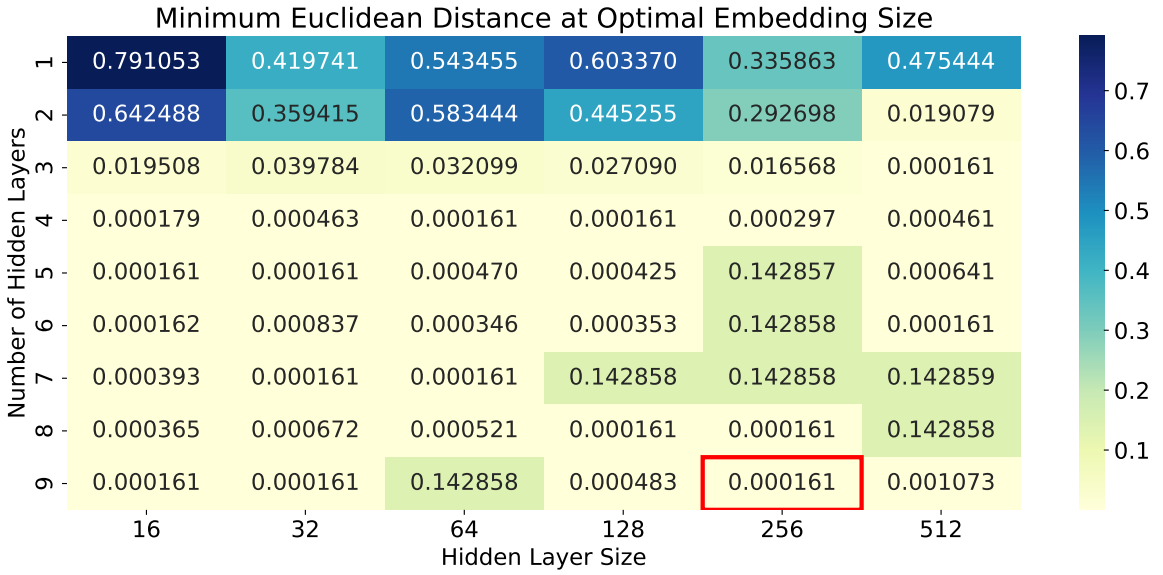


Figure A.14: *Euclidean distance to the ideal point for the MAWI+CAIDA dataset with traffic image resolution 16*

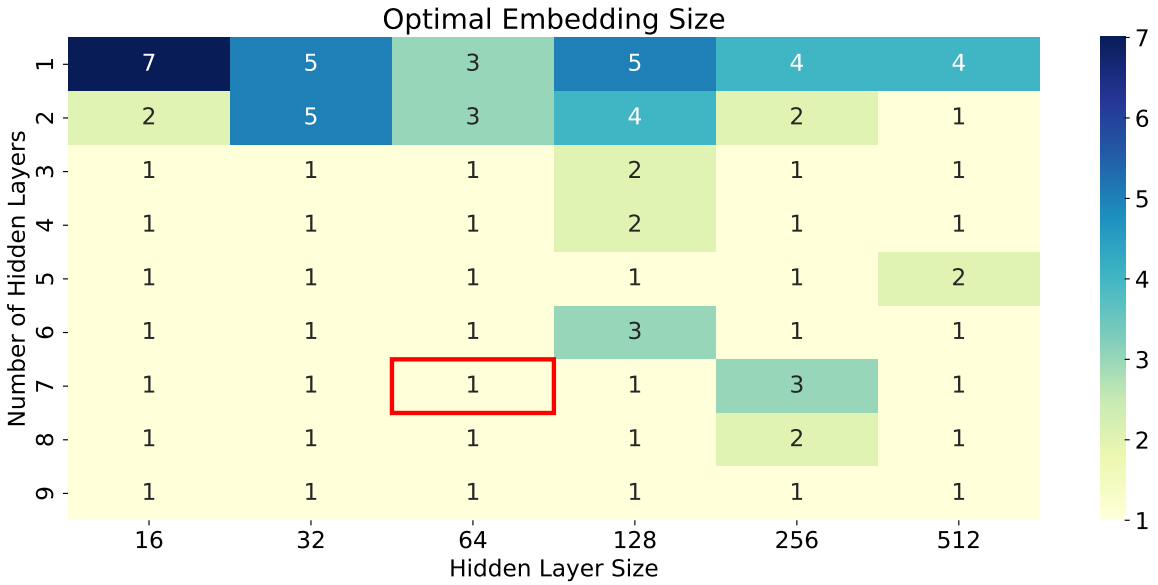


Figure A.15: *Optimal embedding size for the MAWI+CAIDA dataset with traffic image resolution 32*

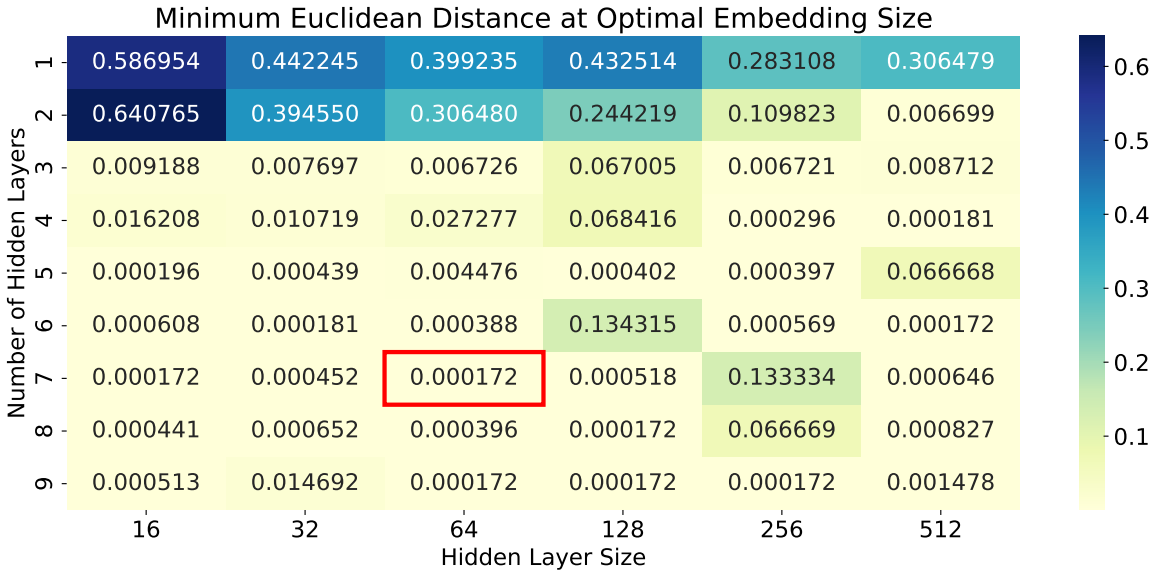


Figure A.16: *Euclidean distance to the ideal point for MAWI+CAIDA dataset with traffic image resolution 32*

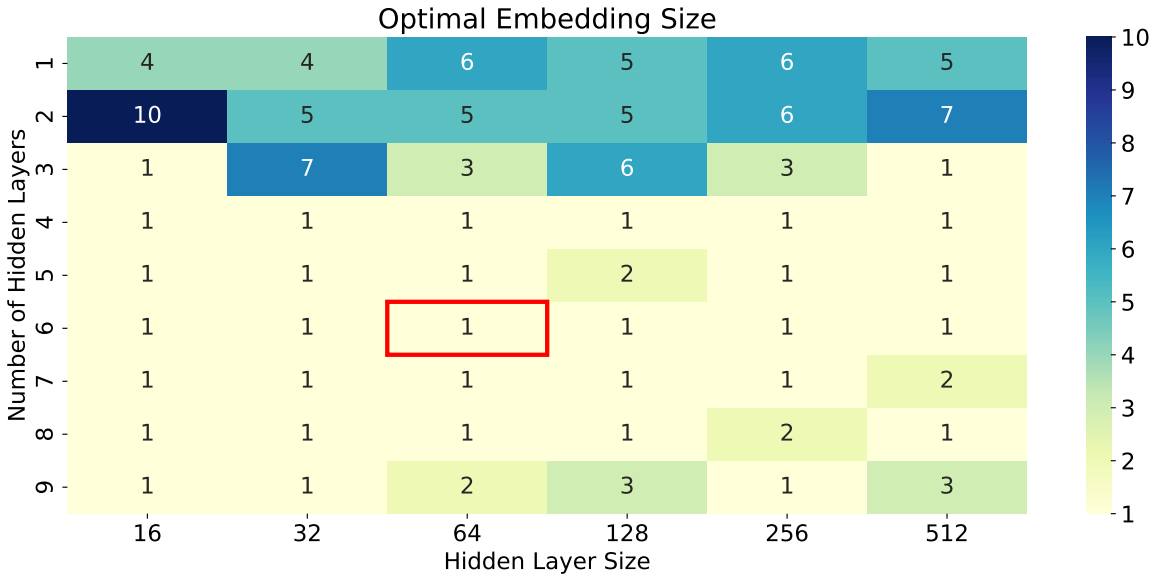


Figure A.17: *Optimal embedding size for the MAWI+CAIDA dataset with traffic image resolution 64*

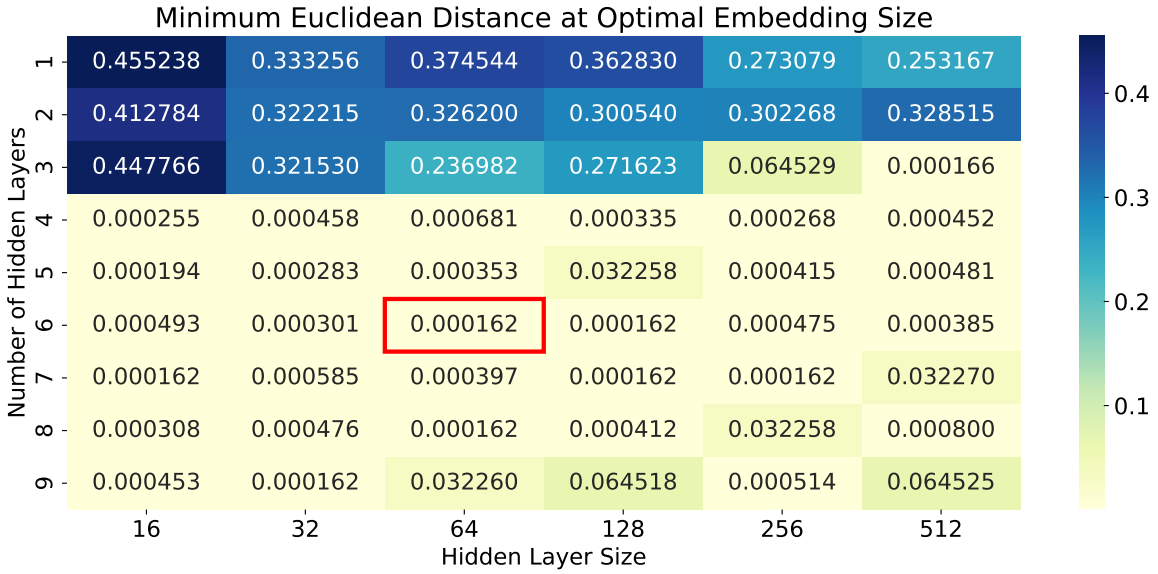


Figure A.18: *Euclidean distance to the ideal point for the MAWI+CAIDA dataset with traffic image resolution 64*

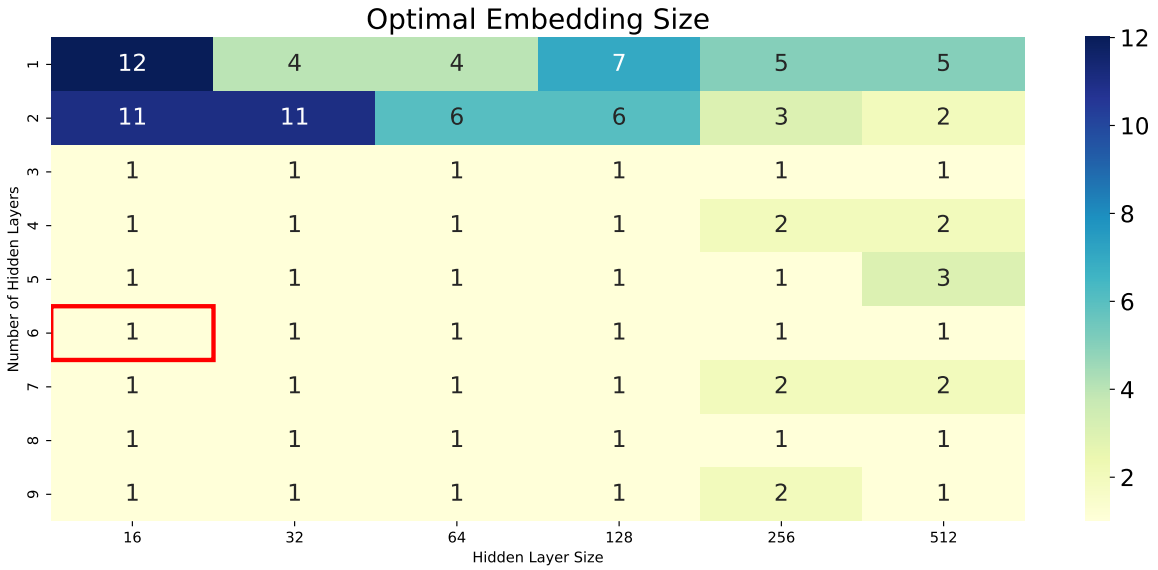


Figure A.19: *Optimal embedding size for the MAWI+CAIDA dataset with traffic image resolution 128*

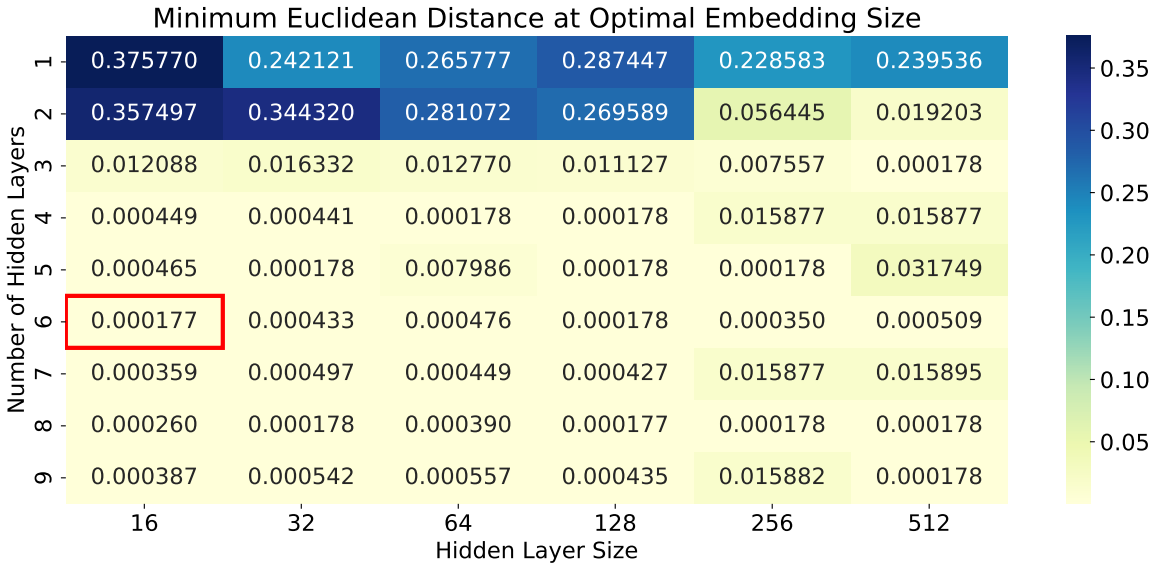


Figure A.20: *Euclidean distance to the ideal point for the MAWI+CAIDA dataset with traffic image resolution 128*

Bibliography

[AZ21] M. Aamir and S. M. A. Zaidi. “Clustering based semi-supervised machine learning for DDoS attack classification.” In: *Journal of King Saud University-Computer and Information Sciences* 33.4 (2021), pp. 436–446.

[Agr+22] G. Agrafiotis et al. “Image-based Neural Network Models for Malware Traffic Classification using PCAP to Picture Conversion.” In: *Proceedings of the 17th International Conference on Availability, Reliability and Security*. ARES ’22. Vienna, Austria: Association for Computing Machinery, 2022. ISBN: 9781450396707. DOI: 10.1145/3538969.3544473. URL: https://doi.org/10.1145/3538969.3544473.

[Ala+23] A. A. Alahmadi et al. “DDoS Attack Detection in IoT-Based Networks Using Machine Learning Models: A Survey and Research Directions.” In: *Electronics* 12.14 (2023). ISSN: 2079-9292. DOI: 10.3390/electronics12143103.

[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, 693–706. ISBN: 9781450394208. DOI: 10.1145/3544216.3544263. URL: https://doi.org/10.1145/3544216.3544263.

[ACM23] T. E. Ali, Y.-W. Chong, and S. Manickam. “Machine Learning Techniques to Detect a DDoS Attack in SDN: A Systematic Review.” en. In: *Applied Sciences* 13.5 (Jan. 2023). Publisher: Multidisciplinary Digital Publishing Institute, p. 3183. ISSN: 2076-3417. DOI: 10.3390/app13053183.

[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. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[Bae+17] S. Baek et al. “Unsupervised labeling for supervised anomaly detection in enterprise and cloud networks.” In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE. 2017, pp. 205–210.

[Bah+23] A. A. Bahashwan et al. “A Systematic Literature Review on Machine Learning and Deep Learning Approaches for Detecting DDoS Attacks in Software-Defined Networking.” en. In: *Sensors* 23.9 (Jan. 2023). Publisher: Multidisciplinary Digital Publishing Institute, p. 4441. ISSN: 1424-8220. DOI: 10.3390/s23094441. URL: https://www.mdpi.com/1424-8220/23/9/4441.

[Bel] *BelWü - Landeshochschulnetz Baden-Württemberg*. URL: https://www.belwue.de/.

[Blo70] B. H. Bloom. “Space/time trade-offs in hash coding with allowable errors.” In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[CBG12] C. A. Catania, F. Bromberg, and C. G. Garino. “An autonomous labeling approach to support vector machines algorithms for network traffic anomaly detection.” In: *Expert Systems with Applications* 39.2 (2012), pp. 1822–1829.

[Cen07] Center for Applied Internet Data Analysis (CAIDA). *CAIDA UCSD "DDoS Attack 2007" Dataset*. https://www.caida.org/catalog/datasets/ddos-20070804_dataset. University of California San Diego, San Diego Supercomputer Center, Aug. 2007.

[Cen25] Center for Applied Internet Data Analysis (CAIDA). *About CAIDA*. 2025. URL: https://www.caida.org/about/.

[Che+17] Z. Chen et al. “Seq2Img: A sequence-to-image based approach towards IP traffic classification using convolutional neural networks.” In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 1271–1276. DOI: 10.1109/BigData.2017.8258054.

[CNN22] P. Choobdar, M. Naderan, and M. Naderan. “Detection and multi-class classification of intrusion in software defined networks using stacked auto-encoders and CICIDS2017 dataset.” In: *Wireless Personal Communications* 123.1 (2022), pp. 437–471.

[CTA13] B. Claise, B. Trammell, and P. Aitken. *RFC 7011: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. USA, 2013.

[Cla04] B. Claise. *Cisco systems netflow services export version 9*. Tech. rep. 2004.

[Clo] Cloudflare. *Famous DDoS attacks*. URL: https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/.

[Clo24] Cloudflare. *DDoS threat report for 2024 Q4*. 2024. URL: https://radar.cloudflare.com/reports/ddos-2024-q4.

[Clo25a] Cloudflare. *DDoS threat report for 2025 Q2*. 2025. URL: https://radar.cloudflare.com/reports/ddos-2025-q2.

[Clo25b] Cloudflare. *DDoS threat report for 2025 Q3*. 2025. URL: https://blog.cloudflare.com/de-de/ddos-threat-report-2025-q3/.

[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, 435–448. ISBN: 9781450332132. DOI: 10.1145/2663716.2663717. URL: https://doi.org/10.1145/2663716.2663717.

[DKT03] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. "Longest prefix matching using bloom filters." In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '03. Karlsruhe, Germany: Association for Computing Machinery, 2003, 201–212. ISBN: 1581137354. DOI: 10.1145/863955.863979. URL: https://doi.org/10.1145/863955.863979.

[ERJ21] G. Engelen, V. Rimmer, and W. Joosen. "Troubleshooting an intrusion detection dataset: the CICIDS2017 case study." In: *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2021, pp. 7–12.

[Gu+19] Y. Gu et al. "Semi-Supervised K-Means DDoS Detection Method Using Hybrid Feature Selection Algorithm." In: *IEEE Access* 7 (2019), pp. 64351–64365. DOI: 10.1109/ACCESS.2019.2917532.

[Hal+22] A. Halbouni et al. "CNN-LSTM: Hybrid Deep Neural Network for Network Intrusion Detection System." In: *IEEE Access* 10 (2022), pp. 99837–99849. DOI: 10.1109/ACCESS.2022.3206425.

[He+15] K. He et al. *Deep Residual Learning for Image Recognition*. arXiv:1512.03385 [cs]. Dec. 2015. DOI: 10.48550/arXiv.1512.03385. URL: http://arxiv.org/abs/1512.03385.

[Hou+22] Y. Hou et al. "Handling Labeled Data Insufficiency: Semi-supervised Learning with Self-Training Mixup Decision Tree for Classification of Network Attacking Traffic." In: *IEEE Transactions on Dependable and Secure Computing* (2022), pp. 1–14. DOI: 10.1109/TDSC.2022.3195534.

[IJJ24] K. Ibrahimi, M. Jouhari, and Z. Jakout. "Enhancing Intrusion Detection Systems Using Machine Learning Classifiers on the CSE-CIC-IDS2018 Dataset." In: *2024 11th International Conference on Wireless Networks and Mobile Communications (WINCOM)*. 2024, pp. 1–6. DOI: 10.1109/WINCOM62286.2024.10655131.

[KJ22] M. I. Kareem and M. N. Jasim. “DDOS Attack Detection Using Lightweight Partial Decision Tree algorithm.” In: *2022 International Conference on Computer Science and Software Engineering (CSASE)*. 2022, pp. 362–367. DOI: 10.1109/CSASE51777.2022.9759824.

[Kha+22] S. Khan et al. “Transformers in Vision: A Survey.” In: *ACM Comput. Surv.* 54.10s (Sept. 2022). ISSN: 0360-0300. DOI: 10.1145/3505244. URL: https://doi.org/10.1145/3505244.

[KHZ22a] S. Kopmann, H. Heseding, and M. Zitterbart. “HollywooDDoS: Detecting Volumetric Attacks in Moving Images of Network Traffic.” In: *2022 IEEE 47th Conference on Local Computer Networks (LCN)*. 2022, pp. 90–97. DOI: 10.1109/LCN53696.2022.9843465.

[KHZ22b] S. Kopmann, H. Heseding, and M. Zitterbart. “MIDA: Micro-flow Independent Detection of DDoS Attacks with CNNs.” In: *Advances in Service-Oriented and Cloud Computing*. Ed. by C. Zirpins et al. Cham: Springer Nature Switzerland, 2022, pp. 32–43. ISBN: 978-3-031-23298-5.

[KHZ23] S. Kopmann, H. Heseding, and M. Zitterbart. “Toward Joining DDoS Mitigation and Image Segmentation.” In: *Datenschutz und Datensicherheit - DuD* 47.8 (Aug. 2023), pp. 475–477. ISSN: 1862-2607. DOI: 10.1007/s11623-023-1801-1. URL: https://doi.org/10.1007/s11623-023-1801-1.

[KKZ24a] S. Kopmann, T. Krack, and M. Zitterbart. “Demonstrator 7D: Demonstrating Drill-Down DDoS Destination Detection.” In: *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. 2024, pp. 1–2. DOI: 10.1109/ICMLCN59089.2024.10624795.

[KLZ25] S. Kopmann, D. Lachos, and M. Zitterbart. “Feedback-driven Autonomous Data Set Labeling for Denial-of-Service Attack Traffic.” In: *2025 IEEE 50th Conference on Local Computer Networks (LCN)*. 2025, pp. 1–9. DOI: 10.1109/LCN65610.2025.11146338.

[KZ23] S. Kopmann and M. Zitterbart. “eMinD: Efficient and Micro-Flow Independent Detection of Distributed Network Attacks.” In: *2023 14th International Conference on Network of the Future (NoF)*. 2023, pp. 159–167. DOI: 10.1109/NoF58724.2023.10302763.

[KZ24] S. Kopmann and M. Zitterbart. “Importance Analysis of Micro-Flow Independent Features for Detecting Distributed Network Attacks.” In: *IEEE Transactions on Network and Service Management* 21.6 (2024), pp. 5947–5957. DOI: 10.1109/TNSM.2024.3460082.

[KKZ24b] T. Krack, S. Kopmann, and M. Zitterbart. “Evaluating Drill-Down DDoS Destination Detection.” In: *2024 IEEE 49th Conference on Local Computer Networks (LCN)*. 2024, pp. 1–7. DOI: 10.1109/LCN60385.2024.10639629.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012.

[Liu+21] Z. Liu et al. “Jaqen: A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches.” In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 3829–3846.

[Mar+17] J. Margolis et al. “An In-Depth Analysis of the Mirai Botnet.” In: *2017 International Conference on Software Security and Assurance (ICSSA)*. 2017, pp. 6–12. DOI: 10.1109/ICSSA.2017.12.

[Mas+21] Z. K. Maseer et al. “Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset.” In: *IEEE Access* 9 (2021), pp. 22351–22370. DOI: 10.1109/ACCESS.2021.3056614.

[Mea25] Measurement and Analysis on the WIDE Internet. *About MAWI*. 2025. URL: https://mawi.wide.ad.jp/.

[NZM23] M. Najafimehr, S. Zarifzadeh, and S. Mostafavi. “DDoS attacks and machine-learning-based detection methods: A survey and taxonomy.” en. In: *Engineering Reports* 5.12 (2023), e12697. ISSN: 2577-8196. DOI: 10.1002/eng2.12697.

[RS24] S. R and V. S. “An Improving Intrusion Detection Model Based on Novel CNN Technique Using Recent CIC-IDS Datasets.” In: *2024 International Conference on Distributed Computing and Optimization Techniques (ICDCOT)*. 2024, pp. 1–6. DOI: 10.1109/ICDCOT61034.2024.10515433.

[Red+20] T. Reddy.K et al. *Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification*. RFC 8782. May 2020. DOI: 10.17487/RFC8782. URL: https://www.rfc-editor.org/info/rfc8782.

[Roe+99] M. Roesch et al. “Snort: Lightweight intrusion detection for networks.” In: *Lisa*. Vol. 99. 1. 1999, pp. 229–238.

[RFB15] O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation.” In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[Ros+22] A. Rosay et al. “Network intrusion detection: A comprehensive analysis of CIC-IDS2017.” In: *8th international conference on information systems security and privacy*. SCITEPRESS-Science and Technology Publications. 2022, pp. 25–36.

[Ros14] C. Rossow. “Amplification Hell: Revisiting Network Protocols for DDoS Abuse.” In: *NDSS*. 2014, pp. 1–15.

[Rus+15] O. Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV]. URL: https://arxiv.org/abs/1409.0575.

[Saj+24] M. Sajid et al. “Enhancing intrusion detection: a hybrid machine and deep learning approach.” In: *Journal of Cloud Computing* 13.1 (2024), p. 123.

[Noa] *scikit-learn: machine learning in Python — scikit-learn 1.8.0 documentation*. URL: https://scikit-learn.org/stable/.

[SS19] T. Shapira and Y. Shavitt. “FlowPic: Encrypted Internet Traffic Classification is as Easy as Image Recognition.” In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2019, pp. 680–687. DOI: 10.1109/INFCOMW.2019.8845315.

[SHG18] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani. “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.” In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP*. INSTICC. SciTePress, 2018, pp. 108–116. ISBN: 978-989-758-282-0. DOI: 10.5220/0006639801080116.

[Sha+19] I. Sharafaldin et al. “Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy.” In: *2019 International Carnahan Conference on Security Technology (ICCST)*. 2019, pp. 1–8. DOI: 10.1109/CCST.2019.8888419.

[SZ15] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556 [cs]. Apr. 2015. DOI: 10.48550/arXiv.1409.1556. URL: http://arxiv.org/abs/1409.1556.

[Sze+14] C. Szegedy et al. *Going Deeper with Convolutions*. arXiv:1409.4842 [cs]. Sept. 2014. DOI: 10.48550/arXiv.1409.4842. URL: http://arxiv.org/abs/1409.4842.

[TL20] A. Thakkar and R. Lohiya. “A review of the advancement in intrusion detection datasets.” In: *Procedia Computer Science* 167 (2020), pp. 636–645.

[Var+15] G. Varoquaux et al. “Scikit-learn: Machine Learning Without Learning the Machinery.” In: *GetMobile: Mobile Comp. and Comm.* 19.1 (June 2015), pp. 29–33. ISSN: 2375-0529. DOI: 10.1145/2786984.2786995. URL: https://dl.acm.org/doi/10.1145/2786984.2786995.

[WK18] S. Wankhede and D. Kshirsagar. “DoS Attack Detection Using Machine Learning and Neural Network.” In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. 2018, pp. 1–5. DOI: 10.1109/ICCUBEA.2018.8697702.

[WVZT90] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. “A linear-time probabilistic counting algorithm for database applications.” In: *ACM Trans. Database Syst.* 15.2 (June 1990), 208–229. ISSN: 0362-5915. DOI: 10.1145/78922.78925. URL: https://doi.org/10.1145/78922.78925.

[Wic+22] M. Wichtlhuber et al. “IXP scrubber: learning from blackholing traffic for ML-driven DDoS detection at scale.” In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM ’22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, 707–722. ISBN: 9781450394208. DOI: 10.1145/3544216.3544268. URL: https://doi.org/10.1145/3544216.3544268.

[Xu+01] J. Xu et al. “On the design and performance of prefix-preserving IP traffic trace anonymization.” In: *Proceedings of the 1st ACM SIGCOMM Workshop on Internet measurement*. IMW ’01. New York, NY, USA: Association for Computing Machinery, Nov. 2001, pp. 263–266. ISBN: 978-1-58113-435-3. DOI: 10.1145/505202.505234. URL: https://doi.org/10.1145/505202.505234.

[Xu+02] J. Xu et al. “Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme.” In: *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* ISSN: 1092-1648. Nov. 2002, pp. 280–289. DOI: 10.1109/ICNP.2002.1181415. URL: https://ieeexplore.ieee.org/document/1181415.

[Zha+20] M. Zhang et al. “Poseidon: Mitigating volumetric ddos attacks with programmable switches.” In: *the 27th Network and Distributed System Security Symposium (NDSS 2020)*. 2020.

[ZKA23] R. Zhour, C. Khalid, and K. Abdellatif. “Hybrid intrusion detection system based on Random forest, decision tree and Multilayer Perceptron (MLP) algorithms.” In: *2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM)*. 2023, pp. 1–5. DOI: 10.1109/WINCOM59760.2023.10322983.