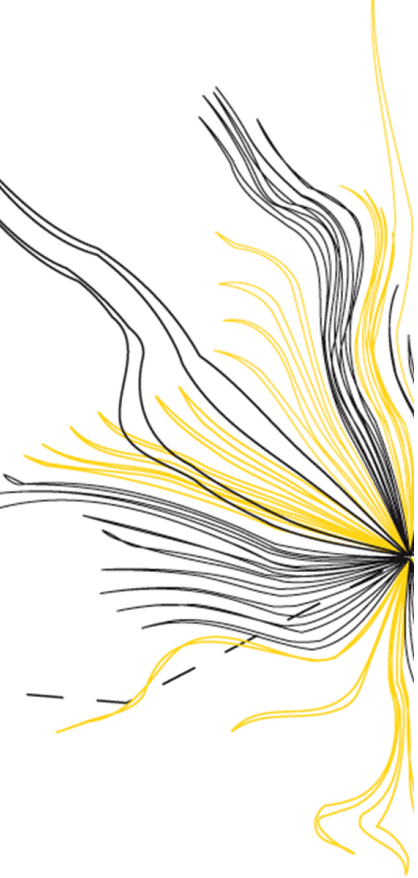


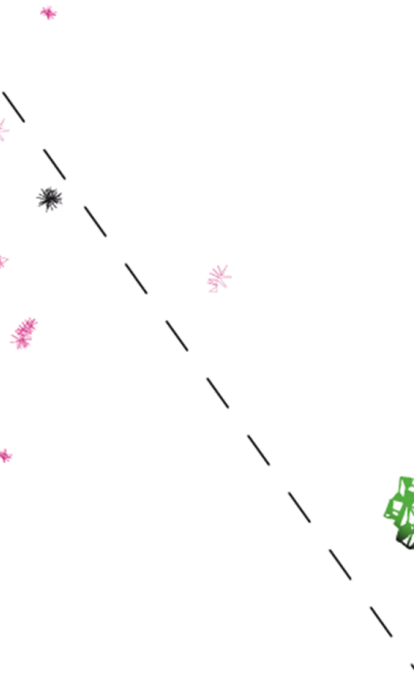
UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

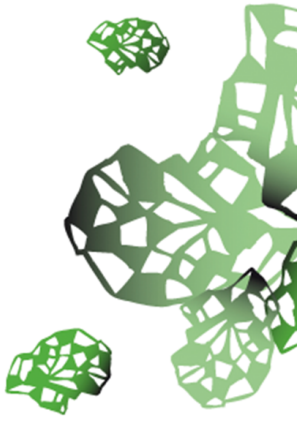
# A Scalable Recommendation System for Selective Retention of DDoS Traffic: Entropy-Based Network Traffic Analysis

**Bas Bleijerveld**

**M.Sc. Thesis**
**October 2024**







**Committee:**
prof. dr. ir. R.M. van Rijswijk-Deij
dr. ir. M. Jonker
dr. D. Bucur

Design and Analysis of Communication Systems
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

# Abstract

This study aims to enhance data retention strategies in the context of DDoS attacks by designing, implementing, and evaluating a novel scalable system. The system is developed to provide near real-time recommendations for retaining only relevant data related to DDoS attacks, thereby reducing storage requirements. Our work is structured into three stages: design, implementation and deployment, and evaluation. The system is designed to utilize IPFIX flow data, which aggregates network traffic into concise records based on unencrypted header fields and metadata, to reduce processing overhead while maintaining an appropriate representation of ongoing network behavior. The system's architecture leverages Apache Kafka and Apache Spark for scalable distributed processing of flow data in near real-time. By employing entropy-based features in a sliding window approach, the system aims to detect anomalies indicative of DDoS attacks. Using a synthetic dataset with self-generated DDoS attacks, the system was evaluated under different cost scenarios related to falsely retained traffic and undetected attacks. The results confirm the feasibility of achieving significant storage reductions while retaining traffic from most DDoS attack types. However, difficulties were encountered in detecting stealthy attacks, as demonstrated by the Slowloris attack. By leveraging the widely supported IPFIX protocol, the system facilitates integration with various existing network infrastructures, making it a viable solution for real-world applications. This research contributes to the field of network traffic data retention by providing a scalable solution to manage the retention of DDoS traffic more efficiently in large high-speed networks.

# Contents

# List of acronyms

**ACK** Acknowledgment

**API** Application Programming Interface

**ATU** Adaptive Threshold Updater

**AUPRC** Area Under the PR Curve

**CPU** Central Processing Unit

**Crypto-PAn** Cryptography-based Prefix-preserving Anonymization

**CSV** Comma-Separated Values

**CUSUM** Cumulative Sum

**CV** Coefficient of Variation

**DDoS** Distributed Denial-of-Service

**DES** Double Exponential Smoothing

**DNS** Domain Name System

**DoS** Denial-of-Service

**DPI** Deep Packet Inspection

**DSF** Data Serialization Format

**EDNS** Extension Mechanisms for DNS

**EMA** Exponential Moving Average

**EWMA** Exponentially Weighted Moving Average

**EWMMD** Exponentially Weighted Moving Mean Difference

**FIN** Finish

**FN** False Negative

**FP** False Positive

**FTP** File Transfer Protocol

**GB** Gigabyte

**Gbps** Gigabits per second

**GUI** Graphical User Interface

**HDFS** Hadoop Distributed File System

**HIDS** Host Intrusion Detection System

**HTTP** Hypertext Transfer Protocol

**HTTPS** Hypertext Transfer Protocol Secure

**ICMP** Internet Control Message Protocol

**IDS** Intrusion Detection System

**IETF** Internet Engineering Task Force

**IMAP** Internet Message Access Protocol

**IoT** Internet of Things

**IPFIX** Internet Protocol Flow Information Export

**IP** Internet Protocol

**IPS** Intrusion Prevention System

**IPv4** IP version 4

**IRC** Internet Relay Chat

**ISCX** Information Security Centre of Excellence

**ISP** Internet Service Provider

**JAR** Java ARchive

**JSON** JavaScript Object Notation

**LAN** Local Area Network

**LOIC** Low Orbit Ion Cannon

**MAC** Media Access Control

**MAD** Median Absolute Deviation

**MA** Moving Average

**Mbps** Megabits per second

**MTU** Maximum Transmission Unit

**NaN** Not a Number

**NAT** Network Address Translation

**NaWas** National Scrubbing Center against DDoS attacks

**NBIP** Dutch National Internet Providers Management Organization

**NetBIOS** Network Basic Input Output System

**NIC** Network Interface Controller

**NIDS** Network Intrusion Detection System

**NLNOG** Netherlands Network Operators' Group

**NTP** Network Time Protocol

**OVS** Open vSwitch

**PCAP** Packet Capture

**PMF** Probability Mass Function

**POP3** Post Office Protocol 3

**PR** Precision-Recall

**RAM** Random-Access Memory

**RDD** Resilient Distributed Dataset

**RFC** Request for Comments

**RIPE** Réseaux IP Européens

**RQ** Research Question

**RST** Reset

**RTT** Round-Trip Time

**RUDY** R-U-Dead-Yet

**S3** Simple Storage Service

**SBT** Scala Build Tool

| | |
|---|---|
| **SES** | Single Exponential Smoothing |
| **SLA** | Service-Level Agreement |
| **SMTP** | Simple Mail Transfer Protocol |
| **SQL** | Structured Query Language |
| **SSH** | Secure Shell |
| **SYN** | Synchronize |
| **TCP** | Transmission Control Protocol |
| **TES** | Triple Exponential Smoothing |
| **TN** | True Negative |
| **TP** | True Positive |
| **UDAF** | User-Defined Aggregate Function |
| **UDP** | User Datagram Protocol |
| **VoIP** | Voice over IP |
| **XML** | Extensible Markup Language |
| **YAML** | Yet Another Markup Language |

# Chapter 1

# Introduction

This introduction chapter is divided into four sections. The first section explains the research problem and provides relevant context. Following this, the second section details the research goal, along with specific objectives and research questions to guide this study. The third section presents the contributions of the study. Finally, the fourth section provides a concise overview of this report, outlining its structure and the content covered in each chapter.

## 1.1 Research Problem

Distributed Denial-of-Service (DDoS) attacks can disrupt critical infrastructure and essential services, posing a serious threat to businesses, organizations, and even society as a whole. Cloudflare's DDoS Threat Report for 2022 Q4 [1] shows a sustained increase in the prevalence and sophistication of DDoS attacks.

This emphasizes the importance of network forensics, a subfield of digital forensics that investigates security incidents on computer networks. It involves collecting and retaining network traffic to identify and analyze malicious events, such as DDoS attacks and unauthorized access. The findings are reported to relevant stakeholders and serve two main applications: improving network security and providing legal evidence for law enforcement [2]. The analysis of attack traces can provide valuable insights into their nature, methods and impact. This is important information for improving defensive strategies and increasing the overall resilience of networks. In cases where security incidents lead to legal action, the analysis of network data collected through network forensics can provide crucial evidence that is admissible in court, helping to support investigations and prosecutions of cybercriminals.

Network traffic is volatile data, which means it exists only temporarily in memory or in transit [2]. This makes it crucial to take proactive measures to ensure the acquisition and availability of relevant network traffic data for further analysis. There are two main approaches to this: "catch it as you can" and "stop, look, and listen" [3]. The first approach captures everything moving over the network. However, due to

storage constraints and privacy concerns, it is impractical to simply store all network traffic data for extended periods [4]. The latter approach examines all network traffic, but only stores data that is considered valuable for further analysis. To capture this data, the best method is to use a moving window, limited by the time in which a security incident could potentially be detected [2]. Additionally, rudimentary data analysis is performed during this window to detect any suspicious traffic and provide recommendations regarding extended data retention.

However, to collect and analyze network traffic in real-time, large-scale distributed and parallel processing engines are required to provide the necessary processing capacity [2]. Therefore, it is important to ensure that the method used to capture and analyze relevant network traffic data is both lightweight and effective, while also scalable to support networks of various sizes and to accommodate potential increases in overall network traffic volume.

Another important aspect to consider is that a significant portion of network traffic is encrypted, as indicated by the Google Transparency Report on HTTPS encryption on the web [5]. Consequently, it is essential to take this into account and make sure that the information used for the analysis is accessible.

This study explores the utilization of flow data, a concise representation of network traffic based on unencrypted information, such as header fields and metadata. This approach not only reduces the volume of data that needs to be processed for the analysis, but also ensures the availability of network traffic information by avoiding potential encryption.

Although such an analysis method would be useful for various cyber threats, this study specifically focuses on identifying network traffic data related to DDoS attacks.

## 1.2 Goal, Objectives and Research Questions

The overarching goal of this study is to enhance data retention solutions, particularly in the context of DDoS attacks. The findings should provide a solid foundation for the development of practical solutions that enable organizations to improve their data retention strategies. This will be accomplished through the design, implementation, deployment and evaluation of a novel scalable system. It should be able to provide low-latency recommendations for retaining only relevant data related to DDoS attacks.

In this study, we will research whether this envisioned system proves feasible in reducing storage requirements for retaining DDoS attack traffic data, by addressing the following Main Research Question (RQ).

**Main Research Question***: Is it feasible to reduce storage requirements for retaining DDoS attack traffic data through the deployment of a scalable analysis system that offers near real-time retention recommendations?*

Our study is structured around the following three stages:

1. **Design** - The first stage involves reviewing literature to gain understanding of the research area and operational context, as well as to identify useful methods and techniques for the envisioned system. Based on these gained insights, a system design is proposed which includes the components of the processing architecture and methods for the applied analysis.

   **Objective 1***: Develop a suitable design for a scalable system that provides low-latency data retention recommendations related to DDoS attacks based on a lightweight analysis.*

2. **Implementation and Deployment** - The second stage is dedicated to the implementation and deployment of the designed system, i.e. developing a prototype.

   **Objective 2***: Develop an implementation and deployment of the proposed retention recommendation system, while documenting the process, as well as any encountered considerations and trade-offs.*

3. **Evaluation** - The third stage involves evaluating the system's performance for different cost scenarios related to falsely retained traffic and undetected attacks. This evaluation is conducted using a synthetic dataset with self-generated attacks based on DDoS characteristics from the latest extensive report on DDoS attacks by the Dutch National Internet Providers Management Organization (NBIP) [6]. Further details about the dataset are provided in Section 3.4.

   First, the performance of each implemented feature is evaluated individually to identify features that perform poorly in distinguishing DDoS attack traffic from benign traffic. These underperforming features are excluded from the utilized feature selection and are not futher considered for the system's evaluation.

   Subsequently, we determine the optimal sensitivity configuration of the remaining features for different cost scenarios related to falsely retained traffic and undetected attacks.

   Finally, we assess the retention performance and the reduction in storage size that can be achieved with the recommendation system in place.

   The following RQs guide this process and give a specific and research-oriented direction to the evaluation stage of this study.

   **Research Question 1***: How effective is each implemented feature in detecting various types of DDoS attacks, and which of these features perform poorly on this task?*

   **Research Question 2***: What are the optimal sensitivity configurations for various cost scenarios related to falsely retained traffic and undetected attacks?*

**Research Question 3***: What is the retention performance and potential storage reduction that can be achieved by utilizing the recommendation system with optimal sensitivity configurations for detecting DDoS attacks in various cost scenarios?*

The primary focus of this stage is to answer these three RQs, thereby evaluating the system.

After progressing through all three stages, we discuss the results and conclude by answering the Main RQ.

## 1.3 Contributions of the Study

This study contributes to the field of network traffic data retention by developing a novel scalable system that provides near real-time recommendations for retaining only relevant data related to DDoS attacks. This scalable solution addresses the challenge of effectively managing large volumes of network traffic data by aiming to improve the retention strategies of large high-speed networks.

The main contribution of this research is the creation of this scalable system that integrates well with existing network infrastructure through the use of IPFIX flows, a protocol that is widely supported by enterprise network devices. This ease of integration lowers the barrier for organizations to adopt such a system, thereby increasing its potential for real-world application.

Additionally, our study includes the creation of a synthetic dataset and the simulation environment used to generate the DDoS attack traffic. These resources can serve as tools for future research and the evaluation of other DDoS-related solutions.

The code of our system and our scripts used to simulate DDoS attack traffic to create the synthetic dataset are publicly available in the following GitLab repository, hosted by the University of Twente:
`https://gitlab.utwente.nl/BasBleijerveld/scalable-ddos-traffic-retention`

## 1.4 Overview of the Thesis

This report is structured into five chapters, of which this introduction chapter is the first. Chapter 2 provides a thorough literature review on DDoS attacks, network monitoring, anomaly detection techniques, scalable stream processing technologies, evaluation metrics, and related work. Chapter 3 details the methodology, covering system design, implementation, and deployment, followed by the evaluation processes and dataset acquisition. Chapter 4 presents our study's results and analysis, including data visualization, evaluating the detection performance of each feature, optimizing the sensitivity configurations for various cost scenarios, and assessing the retention performance and potential storage reduction. Chapter 5 concludes

this report by addressing the main research question, interpreting the results, comparing them with related work, discussing the implications and limitations, proposing future research directions, and providing a final conclusion.

# Chapter 2

# Background and Related Work

This chapter serves as a comprehensive literature review and is the first step to accomplish Objective 1. The design of the envisioned system is based on the information presented in this chapter, and we discuss relevant evaluation metrics. The chapter is organized into five sections, each addressing a different aspect.

The first section provides insights into DDoS attacks in general. The second section explores the process of network monitoring and its application for intrusion detection. The third section delves into anomaly detection techniques specifically tailored to DDoS attacks. The fourth section investigates scalable stream processing technologies that enable near real-time flow analysis. The fifth section examines evaluation metrics used to assess the effectiveness of the recommendation system. The final section reviews related work that can be used later in the report to compare our approach with existing methods.

## 2.1 DDoS Attacks: Fundamentals, Techniques and Types

This section provides insight into DDoS attacks. It explains what they are, why they exist and how they work. The section begins with fundamental information about DDoS attacks, then proceeds to explore attack techniques, and finally outlines some common attack types. These insights are important as they establish the necessary understanding of the adversary at hand.

### 2.1.1 Fundamentals

A Denial-of-Service (DoS) attack is a type of cyber-attack in which the attacker attempts to disrupt machines and/or network resources, with the aim of preventing legitimate use of a service.

DoS attacks can be broadly classified into two categories [7]:

- **Semantic** - These exploit a specific feature or implementation bug of a protocol or application to consume excessive resources of the victim.
- **Volumetric** - Also known as flooding attacks, these flood the victim with an excessive volume of network traffic and/or request packets, to overwhelm its resources.

Countermeasures against semantic attacks typically involve modifying the deployed protocol or application, which can shift the attack mechanism into the volumetric category due to the inevitable limitation of resources [7]

In addition to resource limitations, the current design of the Internet also contributes significantly to the prevalence of DoS attacks [7]. The end-to-end paradigm, a fundamental design principle of the Internet, places the complexity on end hosts while the network is only responsible for packet forwarding. Although this design provides a solid foundation for implementing various features on top of the Internet, it lacks the capability to effectively address malicious actors. As a result, the responsibility for addressing DoS attacks is fragmented, falling on end hosts and network operators, which makes it extremely challenging to enforce robust security measures against these attacks.

A DDoS attack is a type of DoS attack that deploys multiple attacking entities. By distributing the attack across multiple sources, DDoS attacks can generate higher volumes of traffic and requests, making them more powerful than single-source DoS attacks. The distributed nature of DDoS attacks also increases their resilience against mitigation efforts, as identifying and blocking multiple attacking sources is a more challenging task.

Various types of attackers have different incentives for carrying out DoS attacks. Understanding their incentives can be used for targeted strategies to mitigate attackers' interests. The incentives can be categorized into five groups [8]:

- **Financial** - Targets for financial gain, such as through ransom or gaining an advantage in the market by targeting a competitor.
- **Ideological** - Motivated by ideological beliefs, such as political views, also referred to as hacktivism.
- **Cyberwarfare** - Weaponized by military or terrorist organizations to target critical infrastructure or services like financial institutions.
- **Revenge** - Involves frustrated individuals attacking in response to perceived injustice.
- **Intellectual Challenge** - Hacking enthusiasts attack to experiment, learn, and showcase their capabilities, as a form of entertainment or to prove their skills.

### 2.1.2 Techniques

This subsection presents an overview of the various techniques used by attackers to carry out the attacks.

#### Tools

Attackers have a range of tools they can use to carry out these attacks, some of which are freely available online, such as Low Orbit Ion Cannon (LOIC) [9] and R-U-Dead-Yet (RUDY) [10]. LOIC is a network stress-testing tool that can generate a high volume of traffic to overwhelm the target, while RUDY uses a semantic approach by sending specially crafted Hypertext Transfer Protocol (HTTP) requests that consume excessive server resources. Since these tools can be used for legitimate purposes such as testing the resilience of a network, they are usually distributed as legal stress-testing tools [11]. However, attackers use them against targeted systems for malicious purposes. These attack tools typically feature an intuitive user interface that minimizes the technical skills required to operate them.

As mentioned previously, it is advantageous to perform the attacks in a distributed manner, whereby multiple devices are used to launch an attack simultaneously. For that reason, some tools are equipped with a built-in feature to remotely control multiple clients at once, which can be used to launch a distributed attack. An example of this functionality is the HiveMind feature present in LOIC [12].

#### Botnets

Acquiring a large amount of devices to perform the attack can be quite challenging. Therefore a common infrastructure used for this is a so-called botnet, which is a network of devices that are infected with malware and under control of single entity, known as the master. The master can use these devices, known as the bots, to launch coordinated attacks on targeted systems and networks, without the knowledge or consent of the device owners.

Some botnets are self-propagating, the infected devices also scan for vulnerable devices and try to install the malware on these devices to increase the size of the botnet [13]. With the rise in popularity of the Internet of Things (IoT), a new landscape for botnets has emerged, as these devices are often poorly configured and secured, making them vulnerable to infection by botnets. This enables the botnets to grow to enormous sizes, such as the infamous Mirai botnet that maintained a size of around 200,000 to 300,000 devices at its steady state [14].

The control structure of botnets can be roughly divided into two categories: client/server and peer-to-peer [13]. The client/server model uses a centralized resource for its command-and-control center, the bots connect to this resource using protocols like HTTP and Internet Relay Chat (IRC). The simplicity of this structure allows the master to easily update the instructions, but it makes the botnet susceptible to a single point of failure. As a result, botnets have evolved to use the peer-to-peer

model, which distributes the command-and-control among devices in the network. This allows the devices to share instructions from the master among themselves, eliminating the single point of failure.

Booter services provide a paid service where customers can use these botnets to launch a DDoS attack against a target of their choice [15]. This makes it easy and affordable for almost anyone to carry out a large-scale DDoS attack.

#### Spoofing and Amplification

An attacker can also increase the power and resilience of their attacks by using various methodological techniques. Spoofing is a technique in which an attacker falsifies the source address of an Internet Protocol (IP) packet to impersonate another device on the network [16]. This allows attackers to hide their identity, making it more difficult to take measures against the attacks.

A threat associated with IP spoofing is the reflection attack in which the attacker impersonates the target. The attacker sends request packets to various services, such as the Domain Name System (DNS) or a Network Time Protocol (NTP) server, which then send their replies to the target.

The reflection technique can be extended by requesting large amounts of data, causing an increase in the traffic volume sent to the victim. This is known as an amplification attack because the volume of traffic sent by the attacker is magnified before reaching the victim.

### 2.1.3 Types

This subsection provides an overview of common DDoS attack types.

#### Ping (ICMP) Flood Attack

A ping flood is an attack that overwhelms a target with a large amount of Internet Control Message Protocol (ICMP) echo request packets [17]. ICMP is an Internet layer protocol used by network devices to communicate and diagnose connectivity issues. The attack causes the target device to consume an excessive amount of resources, including processing power and network bandwidth, as the target attempts to respond to the large amount of ICMP echo request packets. When the volume of requests exceeds the target's capacity, the result is a DoS that affects legitimate traffic.

#### UDP Flood Attack

A User Datagram Protocol (UDP) flood is an attack that overwhelms a target with a large amount of UDP packets [18]. UDP is a transport layer protocol that sends data packets between devices without establishing a connection beforehand, enabling faster transmission but lower reliability compared to Transmission Control

Protocol (TCP). The attack leverages the server's packet processing and response mechanism, as for each incoming UDP packet, the server utilizes resources to check if an application is running on the destination port. If there is no application receiving the packets at the destination port, the server sends an ICMP packet to notify the sender, which further increases resource utilization. This process becomes a vulnerability when a large number of UDP packets are sent to the target, overwhelming the server's resources and causing a DoS for legitimate traffic.

### TCP SYN Flood Attack

A TCP Synchronize (SYN) flood is an attack that overwhelms a target with a large amount of TCP SYN packets [19]. This attack exploits a vulnerability in the TCP protocol's three-way handshake process, which results in the server allocating an excessive amount of resources. To initiate a connection, the attacker sends a TCP SYN packet to the server, which is the first step in the handshake process. The server responds with a SYN-Acknowledgment (ACK) packet and allocates resources while awaiting an ACK response. The attacker may use a spoofed IP address or intentionally ignore responses to keep the server's resources allocated until the waiting time expires. By overwhelming the server with a large amount of these SYN packets, the server runs out of resources to maintain the connections, eventually causing a DoS for legitimate traffic.

### HTTP Flood Attack

An HTTP flood is an attack that targets a web server by overwhelming it with a high volume of HTTP requests [20]. HTTP GET requests can request specific files from the server, or even files that don't exist. In contrast, HTTP POST requests are used to send data to the server, for example to update a database. Both types of requests cause the server to waste resources on processing these requests. When the number of incoming requests exceeds the server's capacity, it results in a DoS for legitimate users of the web server.

### DNS Amplification Attack

A DNS amplification attack uses publicly accessible DNS servers to flood a target with a large volume of data [21]. The attack works by spoofing the source IP address of the DNS query packet to impersonate the target, and sending these packets to a DNS server. The server then sends a DNS response to the target, which is much larger than the original query, thereby amplifying the amount of traffic directed at the target. The amplification is possible because DNS servers use the UDP protocol, which allows for the sending of data without first completing a handshake. By sending multiple DNS queries in this manner, the target is overwhelmed with an excessive amount of packets, resulting in a DoS for legitimate traffic.

### NTP Amplification Attack

An NTP amplification attack exploits a feature in NTP called monlist, which allows an attacker to request a list of the last 600 clients that have accessed the NTP server [22]. Similar to DNS amplification, the attacker spoofs the source IP address of the request packets to impersonate the target. The NTP servers then flood the target with large responses, leading to a DoS for legitimate traffic.

### Memcached Amplification Attack

A memcached amplification attack exploits vulnerable UDP memcached servers to overwhelm a target with a large amount of traffic [23]. Memcached is a database caching system used for speeding up queries. The attacker sends spoofed requests to the server, which then floods the target with response data. The excessive amount of packets received by the target results in a DoS for legitimate traffic. Since memcached version 1.5.6 the UDP protocol is disabled by default [24], effectively mitigating this type of attack.

### Low and Slow Attack

A low and slow attack targets a thread-based web server, it aims to evade detection by gradually consuming the available threads over an extended period of time [25]. Rather than sending a large volume of traffic all at once, a low and slow attack uses a low volume of traffic at a slow rate. This makes it more difficult for mitigation mechanisms to distinguish between legitimate traffic and malicious traffic.

An example of a low and slow attack is Slowloris [26]. It involves sending incomplete HTTP requests to the target server in small chunks, leaving multiple connections open and exhausting available threads for new connections. By tying up all available threads, the attack makes it impossible for legitimate users to connect to the server.

Another example of a low and slow attack is RUDY [27]. It works by sending HTTP POST requests to the target server with a large body of data spread over multiple small packets. The attacker sends the data very slowly, one byte at a time, and keeps the connections open for as long as possible. This exhausts the server's available threads, which results in a DoS for legitimate users.

## 2.2 Network Monitoring and Intrusion Detection

This section explores the process of network monitoring. It first covers the different types of network monitoring, then discusses methods for capturing and analyzing network traffic, and concludes with its application to detect network intrusions.

### 2.2.1 Network Monitoring

Network monitoring is the process of continuously observing and analyzing computer networks to ensure optimal performance and security. This involves using tools to monitor devices connected to the network and the traffic passing through it. The aim of network monitoring is to enhance network performance, prevent downtime, and safeguard against cyber-attacks by maintaining constant vigilance over network activity. By implementing effective network monitoring strategies, organizations can improve network reliability, enhance user experience, and mitigate the risk of security incidents. Network monitoring is important for networks of all sizes, ranging from a Local Area Network (LAN) to the Internet as a whole.

### 2.2.2 Active and Passive Monitoring

Network monitoring approaches are typically grouped into two categories [28]:

- **Active Monitoring** - Active network monitoring involves actively probing a network or its services to obtain specific metrics by injecting packets. Sending requests and receiving responses allow for the measurement of various network performance metrics such as latency, packet loss, and network topology. The network diagnostic tools Ping and Traceroute are examples of active monitoring. Active approaches to network monitoring require additional resources from both the network and the target being monitored, which can potentially disrupt normal operations. Therefore, active monitoring should be carried out at an acceptable intensity level to avoid such disruption. Large-scale active monitoring typically requires multiple monitoring locations, which can be expensive and challenging to set up. A popular solution is provided by cooperative monitoring platforms, such as Réseaux IP Européens (RIPE) Atlas [29] and Netherlands Network Operators' Group (NLNOG) Ring [30]. These platforms rely on participants hosting monitoring nodes, enabling resource sharing and reducing the cost and effort required to establish a large-scale monitoring infrastructure.
- **Passive Monitoring** - Passive network monitoring involves examining the network traffic passing through one or multiple observation points, such as network switches or specialized network taps, without actively sending probes by injecting packets. This makes it non-intrusive on the network while providing insights into network behavior to proactively identify issues, such as bottlenecks and security incidents. However, despite its non-intrusive nature, it is important to note that retrieving information from passive monitoring devices can still generate network traffic. Furthermore, passive monitoring can require the processing of large amounts of packets, which can present a significant challenge, particularly for networks with high traffic volumes. Additionally, passive monitoring raises privacy concerns due to the potential interception and analysis of sensitive information in network traffic.

The objective of this study is to develop a system capable of detecting malicious activity in passing network traffic. Passive monitoring is the most relevant approach to achieve this objective and is therefore examined in more detail through its two main steps: traffic duplication and various analysis methods.

### 2.2.3 Traffic Duplication

Passive network traffic monitoring requires traffic duplication, which involves creating a copy of the traffic from a network link for analysis. Two common approaches to traffic duplication are inline tapping and port mirroring [31].

- **Inline Tapping** - Inline tapping is a traffic duplication method that involves inserting a physical device into a network link by splitting the observed line to create a copy of the traffic. This dedicated device is called a network tap and usually has three ports. Two of these ports reform the link that has been split and enable the traffic to pass through. The tap duplicates the observed traffic, and the third port is used for sending a copy of the traffic to another device for analysis.
  Alternatively, a computer with two network interfaces can be used to create a similar setup without the need for a dedicated network tap. The network interfaces are configured using software as a network bridge, and the traffic passing through it can be observed. This approach can integrate traffic duplication with traffic analysis, as both operations are possible on the same computer. A consumer-grade Network Interface Controller (NIC) is capable of working in this configuration, but it might result in a disconnected link due to software crashes or hardware issues such as power loss. Specialized bypass NICs exist to address this issue by bypassing the two network interfaces whenever a failure occurs.
- **Port Mirroring** - Port mirroring is a feature built into enterprise network switches and routers that duplicates network traffic from selected ports and sends it to another port for analysis. A significant advantage of this method is that it does not require additional hardware for traffic duplication. However, port mirroring also has its downsides. The mirror port can become congested if the total throughput of the observed ports exceeds what the mirror port can transmit. Additionally, during periods of peak traffic, both switching and mirroring functions require sufficient computational power. The hardware may not be able to handle both functions and prioritizes its primary function, potentially causing issues with its mirroring functionality.

Both inline tapping and port mirroring have their advantages and limitations, the choice of method depends on factors such as network topology, traffic volume, monitoring requirements, and budget.

### 2.2.4 Packet Capture

Packet capture is the process of duplicating network traffic from a network interface by capturing each IP packet along with its associated timestamp. An IP packet consists of a header and a payload. The header contains control information required to deliver the packet, such as the source and destination addresses, the total length of the packet, and an identifier for the protocol used in the payload. The payload contains the user data that is transmitted, which is formatted according to a different protocol that operates on the next layer. The captured information can be saved for future inspection or analyzed in real time.

#### Packet Capture and Analysis Tools

Libpcap [32] is a powerful library that provides low-level network access for packet capture and analysis, making it an essential component in most packet capture tools. The library is written in the C programming language and developed for Unix-like operating systems. Several libraries implement bindings to libpcap for other programming languages, including Pcap4J [33] for Java and Scapy [34] for Python. On Microsoft Windows, libpcap requires a driver and a library to access this driver. WinPcap [35] was a popular port for Windows, but has been unmaintained since 2013. The currently recommended port for Windows is Npcap [36], which is the packet capture library developed and maintained by the Nmap Project.

Two common open source tools for packet capture and analysis are tcpdump [32] and Wireshark [37], both utilize the libpcap library.

- **Tcpdump** - Tcpdump is a command-line tool that can capture and analyze network packets from various interfaces, and also supports loading already captured packets from files. Tcpdump can identify different protocols and parse the packets accordingly, enabling users to analyze the packets based on their content. Filters can be applied to the captured packets based on the header fields, providing insights into specific aspects of the network traffic.
- **Wireshark** - Wireshark is similar to tcpdump but has a Graphical User Interface (GUI) that displays the captured data in a more visually appealing and informative manner. Wireshark provides additional features compared to tcpdump and also offers a command-line version called TShark.

Commercial packet capture and analysis tools exist, but these are not considered in this study.

Network traffic analysis tools can operate using various packet inspection techniques. An essential distinction can be made based on the inspection level. While some analysis tools only look at the packet headers, others might also examine the payload, often referred to as Deep Packet Inspection (DPI) [31]. DPI can provide more information about the network traffic but requires more resources and raises

privacy concerns due to its inspection of packet content. Encryption can, in many cases, be used to evade DPI.

Another key distinction in packet inspection techniques can be made between stateful and stateless packet inspection. Stateful packet inspection analyzes packets within the context of the network connection, while stateless packet inspection analyzes each packet independently. The stateful packet inspection technique provides more comprehensive information about network traffic, but requires more resources for tracking the states.

The choice of inspection techniques depends on the specific needs of network analysis, as well as the available resources and potential trade-offs between obtaining the desired information and ensuring the privacy of network users.

### Packet Capture Storage

The widespread use of the libpcap library has contributed to the popularity of the Packet Capture (PCAP) format for packet capture storage. Despite the lack of a standardized format, the PCAP format has become the de facto standard due to its simplicity and the fact that it is implemented by libpcap. This format consists of a global header containing general information about the capture, followed by a packet header and raw binary data of each captured packet. This format consists of a file header containing general information about the capture, followed by a packet record for each captured packet. Each packet record contains a packet header, which includes the timestamp and packet length, followed by the packet's raw binary data.

PCAPs are a valuable resource for network analysis as they store detailed information about each captured packet. The downside is that this level of detail can lead to large file sizes that require significant storage space. For this reason, standard compression methods are commonly used for archiving purposes. However, since compression is often only desired for long-term storage, other techniques may also be used to minimize storage while retaining essential information. These traffic data reduction techniques include packet filtering, packet truncation, packet sampling, and flow aggregation [38].

- **Packet Filtering** - PCAP files can be filtered based on various features such as source addresses, destination addresses, and protocol types. Filtering can be used to remove unnecessary packets and store only the packets that carry the desired information. Depending on how many packets of the capture actually need to be kept, this technique can significantly reduce the required storage space.
- **Packet Truncation** - This technique selects only a predetermined number of bytes to reduce the amount captured data. For example, only the packet header fields can be selected, and the rest of the packet can be discarded. It is common practice to truncate packets in an adaptive fashion, which means

that the number of bytes captured varies depending on the header length of each packet. Packet truncation can also be used to address privacy concerns, as sensitive information from the captured packet's payload can be removed.

- **Packet Sampling** - Sampling is a technique that can be used to reduce the amount of captured data while preserving the aggregated properties of the entire packet stream with reasonable accuracy. Only a certain percentage of the total packet stream is captured, and randomness can be used in this sampling process to avoid synchronization with periodic patterns in the network traffic. An industry-standard packet sampling protocol is sFlow [39].
- **Flow Aggregation** - Flow aggregation involves grouping multiple packets that share common properties into a flow. For this grouping, each flow is identified by a unique flow key consisting of these common properties, such as source and destination IP addresses, ports, and protocol type. Flow statistics, including duration and total packet count, are tracked for each flow. Only packet header information is used, making flow data less privacy-sensitive than packet captures, and it also avoids dependence on potentially encrypted packet information. The flow data is exported in the form of flow records, which is usually supported by the network infrastructure itself.

The concise and aggregated representation of network activity provided by flows enables the capture and analysis of network traffic patterns and behavior in a more efficient and scalable way than analyzing individual packets [40]. Furthermore, considering that a significant portion of network traffic data is encrypted [5], flow data stands out by not relying on this potentially encrypted information.

These benefits make network analysis based on flow monitoring a relevant approach for this study and is therefore explored in more detail.

### 2.2.5 Flow Monitoring

A flow monitoring system typically consists of three main components: a flow exporter, a flow collector, and a flow analysis application [40]. Each component performs some of the operations within the system's pipeline. Figure 2.1 provides an overview of this pipeline, showing the flow of data from packet observation to flow analysis on the left and the component responsible for each operation on the right. It is important to note that flow monitoring systems can have a many-to-one or one-to-many correspondence between their components, meaning that multiple exporters, collectors, and analysis applications can be involved. This allows for customized configuration of flow monitoring systems to suit different network architectures and workloads. This section provides more information on the operations of each component within a flow monitoring system.

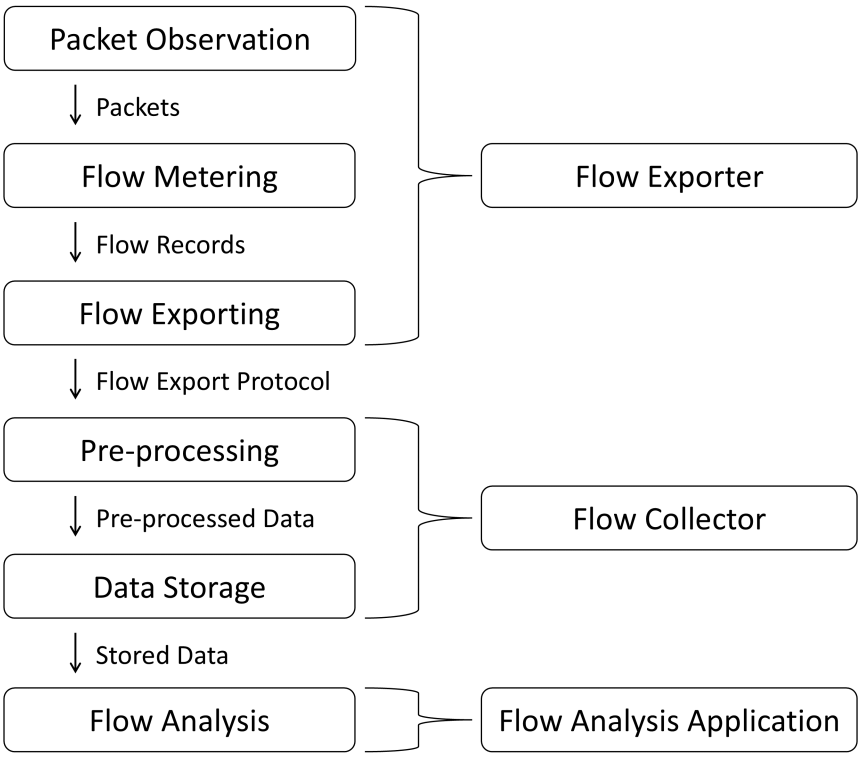



**Figure 2.1:** Overview of a Flow Monitoring System

### Flow Exporter

The flow exporter is responsible for observing packets passing through a network, grouping these packets together into flows, and exporting the flow data to the flow collector.

- **Packet Observation** - The process of packet observation entails capturing packets and is performed as described previously. Packet observation is not always an integral part of the exporter, as packet capture can be performed on a separate device, especially in research environments where analysis of already captured packets is common. In such scenarios, a PCAP file is typically used as input for the exporter instead of live network traffic on an interface.
- **Flow Metering** - Flow metering is the process of aggregating packets into flows while tracking various statistics about each flow. The packets are grouped based on common properties, including source and destination IP addresses, ports, and protocol type. The statistics consist of information such as flow duration and total number of packets. The common properties, along with the associated statistics, are recorded as attributes within a flow record. A flow record is stored in the flow cache during the metering process of an active flow and is exported upon expiration. The expiration of flows can be based on various policies [40]:
  - *Active Timeout* - An active timeout reports on the activity of long-lived

flows periodically, typical active timeout values range from 120 seconds to 30 minutes. Flow entries expired by the active timeout remain in the flow cache with their counters reset and start and end times updated.

  - *Idle Timeout* - Flows without observed packets for a specified time period are expired based on the idle timeout, with values typically ranging from 15 seconds to 5 minutes.
  - *Resource Constraints* - In case of resource constraints, timeouts can expire flows prematurely to free up space in the flow cache.

  Other policies can also be used, depending on the implementation of the flow exporter:

  - *Natural Expiration* - For example, a TCP packet with a Finish (FIN) or Reset (RST) flag indicates the flow has terminated and can trigger the expiration.
  - *Emergency Expiration* - A number of flow entries are immediately expired when the flow cache becomes full.
  - *Cache Flush* - All flow cache entries are expired to completely flush the flow cache. This can be forced by the operator or caused by an unexpected situation, such as a significant change in system time after time synchronization.

  Following expiration, the flow records of expired flows are forwarded to the flow exporting process.

- **Flow Exporting** - The flow exporting process involves sending the flow records to a flow collector, which is typically done using a flow export protocol. In this field, there are two predominant protocols for exporting flow records [40]:
  - *NetFlow* - NetFlow is a network monitoring technology patented by Cisco in 1996. The first widely adopted version of NetFlow was NetFlow v5, released in 2002. NetFlow technology is often integrated into devices within the network infrastructure. NetFlow v9 is a more flexible and widely used version that supports customizable templates for flow records and IPv6. NetFlow defines a format for flow records and a protocol for exporting the flow records to a flow collector.
  - *IPFIX* - Internet Protocol Flow Information Export (IPFIX) is a flow export protocol developed by the Internet Engineering Task Force (IETF). It is similar to and based on NetFlow v9, however IPFIX is an open standard since 2013.

### Flow Collector

The flow collector receives the exported flow records, performs some pre-processing on the flow data if required, and stores the data in an appropriate format for the

analysis.

- **Pre-processing** - The amount and type of pre-processing operations that need to be performed can vary widely depending on the requirements of the system and its applications. For example, some systems may require very minimal pre-processing, such as just removing invalid or irrelevant data, while others may require more complex operations like anonymization of the data. It is recommended to use the Cryptography-based Prefix-preserving Anonymization (Crypto-PAn) algorithm for anonymization purposes [40]. This algorithm allows anonymized IP addresses to be grouped by prefix into anonymized networks, which significantly increases the utility of the data for analysis.
- **Data Storage** - The choice of storage type for flow monitoring data is highly dependent on the requirements of system and the analysis. In-memory storage may be sufficient for some on-the-fly analysis, whereas persistent storage options such as flat files or database solutions may be more suitable for long-term analysis and advanced queries.

### Flow Analysis Application

The flow analysis application is responsible for processing the stored flow data to obtain the desired insights. The objectives of the analysis can be broadly categorized into three distinct domains [40]:

- **Flow Analysis and Reporting** - Flow Analysis and Reporting involves examining flow data to obtain certain statistics and generate reports and alerts. The reports can include information like bandwidth usage to reveal top-talkers (the hosts or services that exchanged the most traffic), while alerts can notify network managers of events such as traffic thresholds being exceeded or the usage of unwanted applications or protocols. The analysis of flow data can reveal anomalies in network behavior, which can then be further investigated to determine the actual nature of the anomalous traffic.
- **Intrusion Detection** - Intrusion detection on computer networks involves identifying malicious behavior by analyzing traffic patterns. However, only a subset of malicious network activities can be identified based on flow data, including DDoS attacks, network scans, worm spreading, and botnet communication. An advantage of using flow data is that it works in encrypted environments, as it does not rely on packet payloads. The detection of these patterns can be performed in an automated fashion by using a Network Intrusion Detection System (NIDS).
- **Performance Monitoring** - Performance monitoring observes network services to verify Service-Level Agreement (SLA) compliance and identify network events affecting end-user experience. Analysis applications process flow

data to report on metrics like Round-Trip Time (RTT), delay, jitter, response time, packet loss, and bandwidth usage.

Among these analysis objectives, intrusion detection is particularly relevant to this study. Therefore, intrusion detection is examined in more detail.

### 2.2.6 Intrusion Detection

Intrusion detection is a critical component of cybersecurity, which involves the monitoring and analysis of a system or network to detect any unauthorized or malicious activities. By using a range of techniques, intrusion detection aims to ensure the confidentiality, integrity, and availability of digital infrastructure.

#### Intrusion Detection Systems

Automated intrusion detection can be achieved through the use of Intrusion Detection Systems (IDSs). These systems can be categorized based on two distinct data sources: logs residing on host systems, and network data [41]. This categorization results in the following two types of IDSs [42]:

- **Host Intrusion Detection Systems (HIDSs)** - These monitor individual host systems and provide insights into host-level activities to identify potential intrusions.
- **Network Intrusion Detection Systems (NIDSs)** - These monitor network data to detect malicious activities within the network infrastructure.

Upon detection, these systems can generate alerts for further action, such as triggering an Intrusion Prevention System (IPS) to address the detected incidents.

#### Detection Approaches

An IDS can utilize various techniques for detection, which can be broadly categorized into two groups [41, 42]:

- **Misuse Detection** - Misuse detection aims to identify explicitly defined intrusions by relying on predefined signatures. These signatures represent specific patterns and characteristics of malicious behavior. The detection process involves comparing observed activities with a signature database to identify matches and alert about detected incidents. Generally, it is effective in detecting well-known intrusions with low false alarm rates when using well-defined signatures. However, it may fail to detect novel or sophisticated intrusions that deviate from existing signatures, potentially resulting in undetected malicious activities.

- **Anomaly Detection** - Anomaly detection, on the other hand, focuses on identifying deviations from normal behavior. Instead of relying on predefined intrusion signatures, anomaly detection aims to model the expected or normal behavior of the system or network. Any activities or events that significantly deviate from this model are considered potential intrusions and raise alerts. The advantage of anomaly detection is that it is capable of detecting novel and unknown incidents. However, anomaly detection is less specific, which can result in false alarms and a lack of explanatory information. Undetected intrusions can occur due to deficient behavioral models or measurements that do not contain distinctive information necessary for detection.

In this study, the detection of any DDoS attacks, including unknown ones, is of great importance, even if it leads to some false alarms. Therefore, the use of anomaly detection is essential for the envisioned system.

## 2.3 Anomaly Detection Techniques for DDoS Attacks

Anomaly detection can utilize various techniques categorized into statistical analysis, machine learning, and data mining to model normal behavior and detect abnormalities [43]. While advanced data-driven techniques like machine learning and data mining offer extensive capabilities for anomaly detection, implementing them within a parallel processing system can become quite complex. Additionally, acquiring a large, high-quality dataset necessary for the effective utilization of these methods poses a significant challenge. To reduce complexity and ensure the feasibility of this study, more simple yet effective statistical techniques are explored for realizing the system. Although this approach may not be as effective as advanced data-driven techniques in capturing complex or subtle attacks, it provides a solid baseline for assessing the potential capabilities of the system. The possibility of improving the system with more advanced techniques can be considered for future research.

A simple statistical approach requires feature engineering to extract meaningful information from the data, followed by deviation detection to uncover any anomalies. Techniques for both processes are discussed in this section.

### 2.3.1 Feature Engineering

Feature engineering involves transforming attributes of network flow data into informative features that capture relevant characteristics indicative of the targeted anomalies. The process includes parsing the flow records and converting their attributes into numerical values that provide quantitative data about the network traffic's behavior.

### Counters

Simple counters, such as the number of flows, packets, and unique IP addresses observed within a specific time window, can be used for this quantification. However, a limitation of this counter-based approach is its strong dependence on traffic volume as the main determining factor [44].

### Entropy

Entropy-based features provide a valuable alternative to overcome this limitation by taking into account the distribution of attributes. This approach provides a more comprehensive understanding of network behavior, enabling improved anomaly detection [44]. It is important for this to use bidirectional flows, as unidirectional flows may introduce bias in the properties of the underlying distributions [45]. A bidirectional flow captures the network communication in both directions between the source and destination.

Entropy is a measure of disorder and originates from thermodynamics, where it was first proposed by Rudolf Clausius in the early 1850s [46]. Later, Claude Shannon developed his definition of entropy in 1948 [47], which became one of the fundamental concepts in information theory.

### Shannon Entropy

Shannon entropy is a measure of uncertainty or information content in a given set of data or a probability distribution. Higher entropy values indicate more unpredictable and dispersed distributions, reflecting a higher level of uncertainty or information content. Conversely, lower entropy values correspond to more predictable and concentrated distributions, representing a higher level of predictability or information redundancy.

Mathematically, Shannon entropy $H_S(X)$ is defined for a Probability Mass Function (PMF) $p(X = x_i)$ [48], which represents the probability distribution of a discrete random variable $X$, as denoted by Equation 2.1.

$$H_s(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i) \tag{2.1}$$

In this Equation 2.1, $H_S(X)$ is calculated by summing over all possible values $x$ of $X$. Each value $x_i$ has a corresponding probability $p(x_i)$. The logarithm is computed with respect to a specified base $b$, commonly base $2$, which measures entropy in bits. However, alternative bases can be used, such as the natural logarithm with base $e$, resulting in entropy being measured in the natural unit of information, or base $10$, which measures entropy in decimal digits, known as the hartley unit.

In the context of network flows, Shannon entropy can be used to summarize the distribution of flow attributes into a single value. This makes it a powerful tool for aggregating network flow data into informative and quantitative features. For

**Table 2.1:** Flow Attributes with Corresponding Probability Mass Functions (PMFs)

| **Flow Attribute** | **Probability Mass Function** $p(X = x_i)$ |
|---|---|
| Source address | $\frac{\text{number of flows with } x_i \text{ as source address}}{\text{total number of flows}}$ |
| Destination address | $\frac{\text{number of flows with } x_i \text{ as destination address}}{\text{total number of flows}}$ |
| Source port | $\frac{\text{number of flows with } x_i \text{ as source port}}{\text{total number of flows}}$ |
| Destination port | $\frac{\text{number of flows with } x_i \text{ as destination port}}{\text{total number of flows}}$ |
| Protocol | $\frac{\text{number of flows with } x_i \text{ as protocol}}{\text{total number of flows}}$ |
| Flow duration | $\frac{\text{number of flows with } x_i \text{ as duration}}{\text{total number of flows}}$ |
| Flow size (packets) | $\frac{\text{number of flows with } x_i \text{ (packets) as flow size}}{\text{total number of flows}}$ |
| Flow size (bytes) | $\frac{\text{number of flows with } x_i \text{ (bytes) as flow size}}{\text{total number of flows}}$ |

example, an increasing entropy of the source address attribute indicates a greater dispersion of source addresses, which is a behavior expected to be observed during an attack with a distributed nature.

The PMFs of flow attributes are typically estimated based on the number of times each unique instance of the attribute occurs within a given time window [44, 49]. Table 2.1 presents possible PMFs based on flow attributes. It should be noted that the flows included in the numerator and denominator are limited to the respective time window.

Lakhina et al. [50] showed that Shannon entropy is an effective metric for capturing unusual changes induced by anomalies in traffic feature distributions. In their study, the authors used Shannon entropy to summarize the distributions of flow attributes. They subsequently applied unsupervised learning to these entropy values, allowing them to successfully cluster and detect anomalies in the analyzed network traffic.

### Normalized Entropy

Entropy takes into account both the uncertainty captured by the probabilities $p(x_i)$ themselves and the number of possible values $n$ that the discrete random variable $X$ can take [44]. Normalization can be used to obtain a measure of uncertainty that is not influenced by the size of the distribution, enabling a better comparison of probability distributions with different sizes. A common technique for entropy nor-

malization is calculating the efficiency. Efficiency measures how close the actual entropy of a distribution is to the maximum possible entropy for a distribution of the same size. Maximum entropy occurs when the probabilities $p(x_i)$ are equal for all possible values of X, resulting in $\log_b(n)$ as the maximum entropy for a distribution of size $n$. Efficiency is calculated by dividing the actual entropy by the maximum possible entropy, as shown in Equation 2.2.

$$H_N(X) = \frac{H(X)}{\log_b(n)} \tag{2.2}$$

The efficiency value ranges from $0$ to $1$, where a value of $0$ indicates that the distribution captures no uncertainty at all and a value of $1$ indicates that the distribution captures the maximum uncertainty possible.

### Relative Entropy

The relative entropy, also referred to as Kullback-Leibler divergence, is a measure that serves as a quantification of uncertainty between two probability distributions [48]. It captures the extent of differences between an observed distribution $P$ and an expected or approximated distribution $Q$, both defined on the same probability space $X$. The Kullback-Leibler divergence is mathematically defined by Equation 2.3.

$$D_{KL}(P||Q) = \sum_{i=1}^{n} p(x_i) \log_b(\frac{p(x_i)}{q(x_i)}) \tag{2.3}$$

This measure becomes particularly valuable in cases where not only the degree of uncertainty is significant, but also the magnitude of changes between the assumed distribution $Q$ and the observed distribution $P$ is relevant.

### Parameterized Entropy

Parameterized entropies are a class of entropy measures that go beyond conventional entropy measures like Shannon entropy. Shannon entropy assumes an implicit trade-off between contributions from the main mass and the tails of the probability distribution [51]. This single perspective makes Shannon entropy somewhat restrictive, and it would be useful to control this trade-off. To address this limitation, parameterized entropies, such as Rényi entropy and Tsallis entropy, offer a more flexible entropy calculation. These entropies introduce an entropic parameter, with $\alpha$ being the parameter for Rényi entropy and $q$ for Tsallis entropy, that modifies the entropy calculation. By adjusting the value of the entropic parameter, one can customize the entropy measure to emphasize different aspects of the probability distribution.

Rényi entropy was introduced by Alfred Rényi [52], and its mathematical formulation is presented in Equation 2.4.

$$H_{R\alpha}(X) = \frac{1}{1-\alpha} \log_b(\sum_{i=1}^{n} p(x_i)^{\alpha}) \tag{2.4}$$

Tsallis entropy was proposed by Constantino Tsallis [53], and its mathematical expression can be found in Equation 2.5.

$$H_{T\alpha}(X) = \frac{1}{1-q}(\sum_{i=1}^{n} p(x_i)^q - 1) \tag{2.5}$$

Smaller values of $\alpha$ and $q$ highlight the contributions of rare events, while larger values of $\alpha$ and $q$ give more weight to the most probable outcomes. Both Rényi entropy and Tsallis entropy converge to Shannon entropy when their entropic parameter approaches 1.

Various studies have explored the use of Renyi and Tsallis entropy for the detection of DDoS attacks. Xiang et al. [54] used Renyi entropy to successfully detect low-rate DDoS attacks, while Ziviani et al. [55] effectively employed Tsallis entropy for DDoS detection. David and Thomas [49] reported positive results for both entropies in their DDoS detection methods.

Bereziński et al. [44] reviewed several studies to determine appropriate entropic parameter values, but the studies they consulted reported varying results. Consequently, they concluded that the optimal parameter value depends on either the specific anomaly under consideration, the legitimate traffic used as a baseline, or both. Bereziński et al. emphasized that finding the optimal parameter value remains unachieved.

In their own research, Bereziński et al. demonstrated that Renyi and Tsallis entropy outperformed Shannon entropy. However, they did not specify a single optimal value for the parameter and instead suggested to use a range of values. It is worth noting that Shannon entropy performed only slightly less effective, while it has the advantage of being a fixed metric that does not require a specifically tuned parameter.

Tellenbach et al. [56] demonstrated that multiple entropic parameter values can be used to create a spectrum of entropy calculations. This approach provides a broader view to detect anomalies and even enables the classification of attacks based on the patterns observed within the spectrum.

Overall, entropy seems to be a promising metric to properly capture and summarize network behavior in quantitative features.

### 2.3.2 Defining Normal Behavior and Detecting Deviations

Detecting deviations from expected normal behavior in quantitative features is essential for identifying anomalies in network traffic. This section explores simple statistical techniques for establishing a definition of expected normal behavior and detecting deviations from it.

### Moving Average

The Moving Average (MA) is a statistical method used for time series analysis and forecasting [57], which can serve as an estimate of expected normal behavior. It calculates the average value of preceding data points within a sliding window, which is particularly useful when observations are recorded at regular intervals over time. By sliding the window along the dataset and recalculating the average at each point, it allows for the creation of a smoothed representation of the data. This technique reduces the impact of short-term fluctuations, such as noise or outliers, thereby providing insight into longer-term smoothed behavior of the analyzed network traffic. However, the MA only considers the level of the data and it does not account for trend or seasonality, resulting in forecasts that disregard gradual shifts or periodic variations.

The size of the window affects the trade-off between smoothing and capturing sudden shifts, so it should be determined based on the characteristics of the data being analyzed and the desired level of smoothing. A larger window provides smoother results, but may also result in a delayed or less accurate representation of rapid changes in the data. On the other hand, a smaller window provides a better response to short-term changes, but can introduce more noise into the smoothed data.

The mathematical expression for the MA is shown in Equation 2.6.

$$M_t = \frac{X_t + X_{t-1} + \cdots + X_{t-N+1}}{N} \tag{2.6}$$

The MA $M_t$ at time $t$ is calculated by summing the data points $X_t + X_{t-1} + \cdots + X_{t-N+1}$ within the considered window and dividing this sum by the number of data points $N$ in this window.

In the study conducted by No and Ra [58], the authors employed the MA technique on entropy values to detect anomalies or deviations by calculating the absolute difference between the current entropy and the MA. This difference was then compared to a threshold, which was based on the standard deviation of the entropy values within the window, but multiplied by a parameter $\beta$ to allow for sensitivity adjustments. Exceeding the threshold suggested a significant deviation from the expected behavior, indicating a potential anomaly. Conversely, if the difference fell below the threshold, the data was considered to be within the normal range. By continuously monitoring and comparing the current entropy with the MA, No and Ra were able to detect deviations exceeding the threshold, which could potentially indicate the presence of anomalies.

In a follow-up study by No and Ra [59], an adaptive threshold for the $\beta$ value was implemented to account for varying traffic conditions. The authors observed that a high $\beta$ value hindered the detection of attacks during stable channel conditions, while a low beta value resulted in an increased number of false positives during burst channel conditions. To overcome these issues, an Adaptive Threshold Updater

(ATU) was proposed, which adjusted the threshold multiplication parameter $\beta$ based on the new entropy value compared to the MA.

### Exponential Smoothing

Another commonly used method for time series analysis and forecasting is exponential smoothing [60]. The core idea of exponential smoothing is to calculate a weighted average, where the weights for each data point decrease exponentially over time. This technique places more emphasis on recent data points while gradually decreasing the influence of older data. The rate at which these weights decrease can be controlled by adjusting the smoothing constants that are part of the calculation. Exponential smoothing involves several methods, each tailored to different types of time series with distinct characteristics:

- **Single Exponential Smoothing** - Simple or Single Exponential Smoothing (SES) is the most basic form and is also known as the Exponentially Weighted Moving Average (EWMA) [61]. This method differs slightly from the MA as it does not employ a finite sliding window, instead it relies on weights that progressively approach zero without ever reaching absolute zero. The EWMA $S_t$ is calculated by taking a weighted combination based on the smoothing constant $\alpha$ of the current value $y_t$ and the previous smoothed value $S_{t-1}$, as shown in Equation 2.7.

  $$S_t = \alpha y_t + (1 - \alpha) S_{t-1} \tag{2.7}$$

  The smoothing constant $\alpha$ is a value between $0$ and $1$. Values of $\alpha$ close to $1$ give more weight to recent data points, making the average more responsive to changes. On the other hand, values of $\alpha$ closer to zero shift the emphasis to older data, resulting in a smoother average that responds more slowly. The resulting average can be used as a forecast for the subsequent data point [62]. Similar to the MA, SES only considers the average of the time series, also referred to as the level component, and it does not account for the effects of trend or seasonality. However, it can be extended to also consider these properties of the data.

- **Double Exponential Smoothing** - Double Exponential Smoothing (DES), also known as Holt's method, captures not only the level but also the trend of the time series, thereby improving forecast accuracy in the presence of a consistent upward or downward trend [63].

  There are two types of trends commonly observed in time series [64]:

  - *Additive Trend* - This pattern shows a constant increase or decrease over time, regardless of the level of the time series.
  - *Multiplicative Trend* - In contrast, this pattern shows a proportional increase or decrease over time, relative to the level of the time series.

These trends require a different implementation of DES. Since only additive trends are considered within the scope of related work, this section will exclusively focus on DES regarding additive trends.
To also capture an additive trend, the SES method has been extended with a trend component $b_{t-1}$, as shown in Equation 2.8, where $b_{t-1}$ represents the trend at the previous time step.

$$S_t = \alpha y_t + (1 - \alpha)(S_{t-1} + b_{t-1}) \tag{2.8}$$

The trend $b_t$ is defined by Equation 2.9).

$$b_t = \gamma(S_t - S_{t-1}) + (1 - \gamma)b_{t-1} \tag{2.9}$$

It is calculated by taking a weighted combination of the previous trend $b_{t-1}$ and the difference between the current and previous smoothed value, denoted as $S_t - S_{t-1}$. The weights for this combination are determined by the smoothing constant $\gamma$, which is a value between $0$ and $1$. This constant controls the influence of the previous trend on the current trend estimation. The forecasts for subsequent time steps of DES can be obtained using Equation 2.10 [65].

$$F_{t+m} = S_t + mb_t \tag{2.10}$$

The forecasted value for the time step $t+m$ is denoted by $F_{t+m}$, where variable $m$ indicates the number of time steps ahead the forecasted value corresponds to.

- **Triple Exponential Smoothing** - Triple Exponential Smoothing (TES), or Holt-Winters' method, further expands on DES by also incorporating seasonality [66]. It introduces a seasonal component that accounts for periodic patterns in the data, such as hourly or weekly variations. By considering both trend and seasonality, TES enhances forecasting capabilities for data with a gradual shift and periodic patterns.
  Similar to trend, seasonality can exhibit two main types of patterns [64]:
  - *Additive Seasonality* - In this pattern, the seasonal fluctuations have a relatively constant magnitude throughout the time series, regardless of the level of the series.
  - *Multiplicative Seasonality* - In contrast to additive seasonality, this pattern involves seasonal fluctuations that are proportional to the level of the time series.

  As with trends, additive and multiplicative seasonality require different implementations of exponential smoothing. This section will exclusively focus on TES regarding additive seasonality, as only additive seasonality is considered within the scope of the related work.

To capture additive seasonality, the DES method is further extended with a seasonal term $I_{t-L}$, as shown in Equation 2.11, where $L$ is the number of data points that make up a single seasonal cycle.

$$S_t = \alpha(y_t - I_{t-l}) + (1-\alpha)(S_{t-1} + b_{t-1}) \tag{2.11}$$

The additive seasonal term is defined by Equation 2.12.

$$I_t = \gamma(y_t - S_t) + (1-\gamma)I_{t-l} \tag{2.12}$$

It is calculated by taking a weighted combination of the previous seasonal term $I_{t-1}$ and the difference between the current observation and the smoothed value, denoted as $y_t - S_t$. The definition of the trend $b_t$ remains unchanged.
The forecast for an upcoming data point of TES, denoted as $F_{t+m}$, is given by Equation 2.13, where $m$ indicates the number of time steps ahead the forecasted value corresponds to.

$$F_{t+m} = S_t + mb_t + I_{t-l+1+(m-1) \mod l} \tag{2.13}$$

This equation combines the level component $S_t$, the trend component $mb_t$, and the seasonal component $I_{t-l+1+(m-1) \mod l}$ to obtain the forecasted values.

The values for the smoothing constants are typically optimized based on historical data. The optimization process involves finding the values of the smoothing constants that minimize the difference between the forecasted values and the actual values in the historical data, for example by minimizing the mean squared error [66].

In the study by Bojović et al. [67], the authors employed Shannon entropy to measure the diversity of communication pairs, which are the combinations of source and destination IP addresses. By utilizing two EWMA indicators with varying smoothing constants, they obtained a fast EWMA to capture short-term trends and a slow EWMA to capture longer-term trends in entropy changes. The difference between the fast and slow EWMA values served as an indication of potential DDoS attacks, and a threshold was applied to perform the actual detection.

The proposed detection method in the study by Lapolli et al. [68] relies on the entropy values of source and destination IP addresses. The authors utilize the EWMA to determine the central tendency of the entropy values, while the Exponentially Weighted Moving Mean Difference (EWMMD) is calculated as an indicator of dispersion. To ensure an accurate definition of normal expected behavior, any entropy measurements identified as malicious by the detection method are excluded from the EWMA and EWMMD calculations. If the entropy value for a source IP address exceeds the previous central tendency estimate plus a configurable parameter $k$ multiplied by the previous dispersion estimate, it triggers an alarm for a DDoS attack. Similarly, if the entropy value for a destination IP address falls below the previous central tendency estimate minus $k$ multiplied by the previous dispersion estimate,

an alarm is triggered. The parameter $k$ provides the flexibility to adjust the detection thresholds, allowing for a configurable balance between undetected attacks and false alarms.

In the study conducted by Hofstede et al. [69], the authors propose two detection algorithms based on exponential smoothing techniques. Both algorithms use the number of flow record creations as input for detection. The first algorithm employs the EWMA, which forecasts the next value based on previous values. An anomaly is detected when the measured value significantly deviates from the forecasted value. The second algorithm extends the first one by incorporating seasonality to more accurately capture daily traffic patterns, although this does require additional resources. It should be noted that this algorithm differs from TES, as the authors do not utilize trend in their implementation. Both algorithms use an upper threshold to detect anomalies. If the measured value exceeds the upper threshold, it is flagged as an anomaly. Hofstede et al. focused on detecting flood attacks, which by definition lead to an increase in the number of flow record creations. Therefore, their approach only considers an upper threshold for detection purposes. The upper threshold is constructed using the forecasted value and the standard deviation of previous forecasting errors. The allowed deviation from the forecasted value is determined by multiplying the standard deviation by a constant factor. This constant can be adjusted to control the sensitivity of the detection process. To maintain stability, a minimal margin is implemented to prevent small peaks during quiet periods from being falsely flagged as anomalies. The Cumulative Sum (CUSUM) technique is used to minimize false positives, as it filters out transient overshoots and focuses on detecting anomalies that persist over a somewhat longer period of time.

### Z-score

The Z-score is a widely used statistical measure that quantifies the deviation of a specific data point from the rest of the dataset in a normalized manner. Unlike previously discussed methods that depend on the sequential order of data points in a time series, the Z-score is applied to data points without considering their patterns based on temporal order. Instead, the Z-score relies on the assumption of normality, which means that the data is expected to follow an underlying normal distribution [70]. A normal distribution, also known as a Gaussian distribution, is a symmetrical probability distribution characterized by its bell-shaped curve. Data points that deviate significantly from the expected range of values based on the normal distribution may indicate potential anomalies. In the study conducted by Van de Meent et al. [71], the authors state that network traffic may be assumed to be fairly Gaussian in general. Therefore, the application of Z-scores to detect anomalies in network traffic data can be justified.

There are two distinct types of Z-scores that are commonly used [70]:

- **Traditional Z-score** - The traditional Z-score provides a measure of deviation, indicating the number of standard deviations the considered data point is away

from the mean. The mean and standard deviation are usually calculated based on a sample from the dataset, they serve as estimators of central tendency and variability, respectively.
The standard deviation is calculated according to Equation 2.14.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)} \tag{2.14}$$

In this equation, $\sigma$ represents the standard deviation, $N$ is the total number of data points, $x_i$ represents each individual data point, and $\mu$ denotes the mean of the data points.
To calculate the traditional Z-score, the mean is subtracted from the considered data point and this result is divided by the standard deviation. The formula for the traditional Z-score is shown in Equation 2.15.

$$Z = \frac{x - \mu}{\sigma} \tag{2.15}$$

In this equation, $Z$ represents the traditional Z-score, $x$ is the considered data point, $\mu$ is the mean, and $\sigma$ is the standard deviation. Positive Z-scores indicate that the data point is above the mean, negative Z-scores indicate that the data point is below the mean, and a Z-score of zero indicates equality to the mean. The magnitude of the Z-score indicates the distance from the mean, expressed in the number of standard deviations.
The traditional Z-score has a maximum value constraint of $\frac{n-1}{\sqrt{n}}$, where $n$ represents the number of data points in the used sample. This constraint can potentially distort the interpretation of the traditional Z-score when working with small samples.
Additionally, the traditional Z-score may provide misleading outcomes when analyzing datasets that contain extreme outliers. This is because extreme values can significantly affect both the mean and the standard deviation. The breakdown point of an estimator is defined as the largest proportion of the data that can be replaced with arbitrary values without causing the estimated value to become infinite [72]. Both the mean and the standard deviation have a breakdown point of zero, as moving a single data point to infinity would also make them infinite.
Since using sufficiently large samples may not always be possible and simply deleting outlying observations from the analysis is typically unwanted, Iglewicz and Hoaglin [72] recommend using the modified Z-score instead for such situations.

- **Modified Z-score** - The modified Z-score is an alternative statistical measure that addresses the discussed limitations of the traditional Z-score. It utilizes the median and the Median Absolute Deviation (MAD) as robust estimators of central tendency and variability, respectively.

The breakdown point of the median is approximately 50%, the exact percentage depends on whether the sample size is even or odd [72]. Due to this high breakdown point, the median is used as a replacement for the mean. Similarly, the MAD, which represents the median of the absolute deviations from the median of the data points, serves as a suitable replacement for the standard deviation. The MAD also has a breakdown point of approximately 50% and is relatively easy to compute. These adjustments make the modified Z-score more robust against extreme values.
Additionally, the modified Z-score is not constrained by sample size, enabling a more accurate assessment of data even with limited observations. However, it is important to note that larger sample sizes generally provide more accurate results.
The MAD is calculated according to Equation 2.16.

$$MAD = \text{median}(|x_i - \mu_{1/2}|) \tag{2.16}$$

In this equation, MAD represents the Median Absolute Deviation, $x_i$ represents each individual data point, and $\mu_{1/2}$ denotes the median of the data points.
To calculate the modified Z-score, subtract the median from the considered data point and divide the result by the MAD. The formula for the modified Z-score is shown in Equation 2.17.

$$M = \frac{0.6745(x - \mu_{1/2})}{MAD} \tag{2.17}$$

In this equation, $M$ represents the modified Z-score, $x$ is the considered data point, $\mu_{1/2}$ is the median, and $MAD$ is the Median Absolute Deviation (MAD). The constant value of $0.6745$ serves as a scaling factor that makes the modified Z-score comparable to the traditional Z-score when the data follows a normal distribution. This specific value is used because the expected value of the MAD is equal to $0.6745$ times the standard deviation for large sample sizes.

In the context of detection, the absolute value of the Z-score is commonly used, since the direction of the deviation is not that relevant. The resulting Z-score value is evaluated on whether it exceeds a predefined threshold, which is generally considered to be indicative of a potential outlier. The threshold value can be adjusted to meet the specific requirements of the detection application.

In the study conducted by Majed et al. [73], the authors used the traditional Z-score in their proposed detection method for identifying DDoS attacks. NetFlow statistics are collected and Z-scores are calculated for various aggregated features, including the number of flows, packets, bytes, and source IP addresses per destination address. The obtained Z-scores are compared to their corresponding threshold values to detect a DDoS attack. Majed et al. used a dynamic threshold based on the Coefficient of Variation (CV), which represents the ratio of standard deviation to the mean. The CV provides a statistical measure of the dispersion of the data points. If

the Z-score value for any of the features exceeded their respective threshold, it indicated the presence of a DDoS attack. The authors were able to differentiate between various attack types based on the specific features that exceeded their thresholds. The method was applied to aggregated data in a time window and the researchers examined the optimal time window required for accurate detection. Their findings indicated that the proposed method required a time window of 30 seconds to confidently identify the occurrence of a DDoS attack.

In the previously discussed studies conducted by No and Ra [58, 59], the authors utilized the MA technique and standard deviation of entropy values within a time window to detect anomalies. This method bears resemblance to the traditional Z-score method, although this was not explicitly mentioned by the authors. The calculation of the difference between the current entropy and the MA is similar to the numerator of the traditional Z-score in Equation 2.15. To determine the presence of an anomaly, this disparity is compared to a threshold defined by multiplying a parameter $\beta$ with the standard deviation. By using the standard deviation as the denominator, this parameter $\beta$ effectively represents the threshold value in the context of the traditional Z-score. Therefore, their method can be understood as an adaptation of the Z-score technique for detecting anomalies based on entropy values.

In the study conducted by Blaise et al. [74], the authors propose a detection method that aims to identify anomalies in time-series data by comparing feature values to their expected distributions. The method analyzes these values within time windows and assumes that each feature follows a normal distribution. The authors initially considered using the traditional Z-score. However, the traditional Z-score can be influenced by extreme values and was therefore replaced by the modified Z-score. An anomaly is identified if the absolute value of the modified Z-score exceeds a predefined threshold. The authors suggest a threshold of 3.5, as recommended by Iglewicz and Hoaglin [72].

## 2.4 Scalable Stream Processing Technologies for Near Real-Time Flow Analysis

This section provides insight into scalable stream processing technologies that enable near real-time analysis of flow data. First, Stream4Flow, a system that leverages Apache Kafka and Apache Spark for analyzing streaming flow data, is introduced. A more detailed explanation of Apache Kafka and Apache Spark follows, highlighting their essential functionalities in the context of streaming data analysis. Then Docker is explained, as its container technology is a popular solution for simplified application deployment, and containers play an important role in scalable systems. Finally, Kubernetes is introduced as a tool for deploying Spark applications on a cluster, providing a robust and scalable infrastructure.

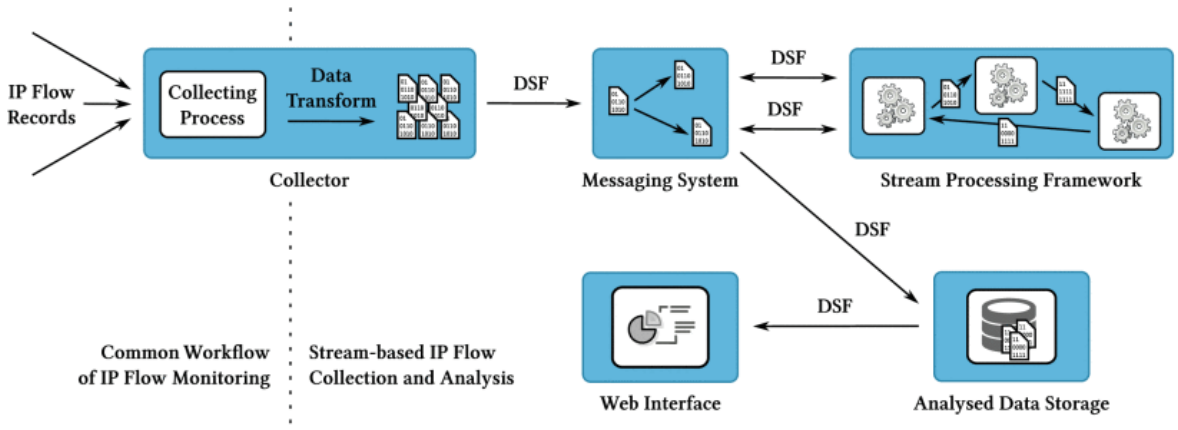



**Figure 2.2:** Overview of Stream4Flow's Architecture [78]

### 2.4.1 Stream4Flow

In response to the increasing volume of network traffic and the growing demand for near real-time traffic analysis, Jirsik et al. [75] proposed the Stream4Flow framework [76] as a novel approach to flow monitoring and analysis. Stream4Flow is a scalable and stream-based flow analysis framework that employs distributed processing, enabling it to analyze large volumes of flow records immediately upon observation. This scalable and near real-time analysis of network traffic can facilitate rapid threat detection and reduce the need for extensive data storage, even as network demands continue to rise.

#### Distributed Processing

Stream4Flow leverages distributed processing, which involves breaking down large tasks into smaller units that can be executed concurrently across a cluster of interconnected processing nodes. Collectively, these nodes can manage significantly large workloads, which is especially useful in the context of big data. Distributed processing supports horizontal scaling, as additional nodes can be added to the cluster to meet increasing demands.

#### Framework

The primary contribution of Stream4Flow lies in its framework design and the development of a prototype for such a system. Stream4Flow's implementation details are based on insights gathered from assessments of various systems [77]. The overall architecture of Stream4Flow is shown in Figure 2.2 and it consists of the following five components [75, 78]:

- **Collector** - Similar to the common flow monitoring workflow, flow records are collected from flow exporters within the network. These records are then converted into a suitable Data Serialization Format (DSF) to ensure compatibility with subsequent processing in the framework.

In Stream4Flow, this process is facilitated by the IPFIXcol [79] collector. IPFIXcol is responsible for collecting flow records from the exporters, converting these records into the JavaScript Object Notation (JSON) format, and transmitting this JSON-formatted data to the messaging system.

- **Messaging System** - The converted flow records are received by the messaging system, which acts as the input interface for the stream processing framework.
  Stream4Flow uses the Apache Kafka [80] messaging system to serve as entry point for data into the stream processing framework. Its efficient data distribution is essential to enable distributed processing of the streaming flow data.
- **Stream Processing Framework** - A stream processing framework is employed to process the data streams. The flow data undergoes immediate analysis through various operations, such as filtering, aggregation, and sorting.
  Stream4Flow uses the Apache Spark [81] processing framework to achieve near-real-time analysis capabilities for substantial volumes of network traffic, as it is a framework that supports distributed processing of streaming data. Apache Spark can be configured with custom applications, enabling it to conduct various types of near real-time flow analysis.
- **Analysed Data Storage** - The analysis results are stored in a query-supporting database, which acts as an interface for visualization tools to present the flow analysis results to the user.
  Stream4Flow stores its results in the Elastic Stack [82] platform, which comprises Logstash [83] for data collection and transformation, Elasticsearch [84] as the database, and Kibana [85] as data visualization tool.
- **Web Interface** - Stream4Flow's framework is complemented by a web interface that provides access to interactive dashboards with the visualized results created by Kibana, as well as providing access to various administrative configurations.

#### Implemented Applications

Two applications were developed to showcase the capabilities of Stream4Flow:

- **Host Monitoring** [75] - The host monitoring application employs distributed stream-based processing to transform connection-based IP flow data into host-based information. This involves filtering connections of interest, creating analysis windows, and computing statistics for each IP address. The dynamic transformation provides a real-time and detailed profile of every host on the network, with a flexible level of detail when needed.
- **PatternFinder** [86] - The PatternFinder application serves as a solution for real-time detection of network attacks based on flow data. PatternFinder's objective is to detect network attacks as they occur by continuously analyzing

incoming flow records and comparing them against predefined attack patterns. Distance functions, weights, and thresholds are employed to effectively assess the similarity between incoming flows and the predefined patterns. A flow with a high degree of similarity is identified as an attack. PatternFinder's real-time detection capabilities have been proven effective in a scenario involving Secure Shell (SSH) authentication attacks.

Stream4Flow's scalable and near real-time analysis capabilities make it an interesting framework for this study. Apache Kafka and Apache Spark are fundamental to Stream4Flow's robust flow analysis framework. The next two sections explore some of their key concepts to better understand how they work.

### 2.4.2 Apache Kafka

Apache Kafka [80] is an open-source, distributed streaming platform that facilitates the ingestion, storage, and distribution of data within distributed systems. Its distributed and scalable architecture is designed for high-throughput, fault-tolerant, and real-time data streaming. Kafka's design accommodates horizontal scaling through the addition of new nodes to a Kafka cluster, ensuring its capability to meet growing demands.

#### Kafka Concepts

In order to gain a better understanding of Apache Kafka, it is helpful to explore some of its key concepts [87]:

- **Broker** - A Kafka broker is a single server or node within a Kafka cluster. It acts as a message broker, receiving, storing, and delivering messages between producers and consumers. Multiple brokers can work together to form a Kafka cluster.
- **Producer** - A Kafka producer is a component or application responsible for ingesting data into the Kafka ecosystem. Producers send messages to Kafka brokers, which are then stored and made available to consumers by publishing the data to Kafka topics.
- **Consumer** - A Kafka consumer is a component or application that consumes and processes the published messages from Kafka brokers. Consumers subscribe to Kafka topics to retrieve and process the messages from them. They enable real-time processing of streaming data by making it available to various downstream applications or systems.
- **Event** - In Kafka, an event refers to a single message or piece of data. Events are produced by producers and consumed by consumers. They represent the unit of data that flows through Kafka's streaming platform.

- **Topic** - A Kafka topic is a feed name to which messages are published by producers. Topics logically represent specific streams of events within a defined category. Producers write messages to specific topics, and consumers subscribe to these topics to read and process the messages.
- **Partition** - A Kafka topic can be divided into multiple partitions, allowing for parallelism and scalability within a topic. Messages within a partition are ordered and immutable, and each partition can be stored on a separate broker within a Kafka cluster. Partitioning enables efficient distribution and processing of data across multiple nodes.
- **Offset** - Each message within a Kafka topic is assigned a unique identifier called an offset. Offsets represent the position of a message within a particular partition of a topic. Consumers use offsets to track their progress in consuming messages from a topic, ensuring they do not miss any data.
- **Replication** - Kafka provides fault tolerance and high availability through data replication. Each partition in Kafka can have multiple replicas, allowing for data redundancy. Replicas ensure that if a broker fails, the data can still be accessed and served from other replicas, ensuring data durability and availability.

These concepts form the foundation of Kafka's distributed streaming platform. They enable reliable and scalable data ingestion, storage, and processing. This makes Kafka a popular choice for building real-time streaming applications and data pipelines.

### 2.4.3 Apache Spark

Apache Spark [81] is an open-source, distributed framework designed to handle large-scale data processing and analysis. Its distributed and fault-tolerant platform leverages in-memory data processing to efficiently execute various tasks on extensive datasets, supporting both batch-based processing and stream-based processing. Spark can be scaled horizontally by adding new nodes to its cluster, ensuring its ability to keep up with the ever-growing demands of data processing.

#### Spark Concepts

To better understand Apache Spark, just like with Kafka, it is helpful to explore some of its key concepts [88]:

- **Cluster** - In the context of Apache Spark, a cluster is a group of computers that work together to process large datasets. A Spark cluster consists of a driver program, a cluster manager, and multiple worker nodes.
- **Driver Program** - The driver program in Apache Spark is the entry point for a Spark application. It runs the main function and coordinates the execution of

tasks across the cluster. The driver program maintains information about the application and interacts with the cluster manager to allocate resources and manage the lifecycle of the application.

- **Cluster Manager** - Apache Spark can independently create a cluster in standalone mode, but can also be integrated with cluster managers, such as Kubernetes [89], to manage the allocation and scaling of resources for the Spark applications. Kubernetes provides a container orchestration platform that simplifies the deployment and scaling of Spark applications in a cluster, making resource management more efficient and flexible.
- **Worker Nodes** - Worker nodes are the compute nodes within a Spark cluster. They are responsible for executing tasks on the data distributed across the cluster. Worker nodes receive instructions from the driver program and execute them in parallel, efficiently using the available resources.
- **Executor** - An executor is a process that runs on a worker node to perform computations for its Spark application. Each application has its own set of executors. Executors manage data storage, execute tasks, and communicate with the driver program. Multiple executors can run on a single worker node, enabling concurrent processing of tasks.
- **Task** - In Apache Spark, a task represents a unit of work that is executed on a partitioned subset of the data. Tasks are distributed to executors across worker nodes and perform data transformations, calculations, or other operations as directed by the Spark application. The parallel execution of tasks is a key feature of Spark's performance optimization.
- **Partition** - Data in Spark is divided into smaller, manageable pieces called partitions. Partitions are the fundamental unit of parallelism in Spark, as each task typically operates on a single partition of the data. Partitioning allows Spark to distribute data across the cluster efficiently, enabling parallel processing and scalability.
- **Resilient Distributed Dataset (RDD)** - An RDD is a low-level abstraction for managing distributed data collections in Spark. It plays a fundamental role in Spark's distributed nature, facilitating fault-tolerant parallel processing through operations like map, filter, and reduce.
- **DataFrame and Dataset** - DataFrames and Datasets are higher-level abstractions built upon RDDs for enhanced structured data processing in Spark. They provide a schema and allow for Structured Query Language (SQL)-like operations, making it easier to work with structured data sources. DataFrames resemble tables in relational databases, offering a structured, column-based data representation that makes them particularly well-suited for general structured data handling. A Dataset can be considered a strongly typed extension of the DataFrame concept, ensuring that each column or field adheres to a

specific data type, thereby enhancing data integrity and reducing runtime errors in Spark applications.

- **Batch Processing** - In Spark, batch processing is the conventional method of data processing where data is organized into fixed-sized, discrete batches. This approach allows for ad-hoc data analysis as well as periodically scheduled processing. However, it is not well-suited for real-time applications involving continuous data streams, which is why Spark offers a dedicated streaming solution.
- **Spark Streaming (DStream)** - Spark Streaming, often referred to as DStream (short for Discretized Stream), is a component of Spark designed for real-time processing of continuously incoming data streams. It operates by splitting the data into micro-batches at short intervals, making it suitable for low-latency processing tasks, such as real-time analytics and event-driven applications. Currently, DStream has been deprecated in favor of a more modern Application Programming Interface (API) called Structured Streaming.
- **Spark Structured Streaming** - Spark Structured Streaming builds on the concepts of Dataframes and Datasets to offer a higher-level, continuous stream processing API. It abstracts streaming data as an ever-expanding table of structured data, providing a unified API that simplifies the development of both batch and real-time processing applications.

These concepts broadly explain how Spark works and the tools it provides for implementing custom processing applications. It is noteworthy that Spark's partitioning model aligns with the partitions created in Apache Kafka, resulting in a seamless integration of both solutions. The partitions from Kafka can be consumed by separate Spark tasks, facilitating the efficient distribution and processing of streaming data within a Spark cluster.

Due to its popularity, Kubernetes [89] is the primary choice as a cluster manager. Since it revolves around containers, which have become the de facto standard for deploying applications, especially in cloud environments, it is useful to gain a better understanding of this container technology. Docker [90] emerges as the foremost platform for container applications. The following two sections explore this area of application deployment, covering both Docker and Kubernetes.

### 2.4.4 Docker

Docker [90] is a widely-used open-source platform that simplifies the development, deployment, and management of applications by packaging them in lightweight, portable containers. These containers encapsulate an application and its dependencies, ensuring consistent behavior across various environments.

### Docker Concepts

This segment explores some of Docker's key concepts [91]:

- **Container** - A container is an isolated environment that encapsulates an application along with all the necessary components for its execution, including libraries and configurations. Containers offer consistency and portability, simplifying the process of migrating and running applications across various systems without the need to be concerned about dependencies or conflicts.
- **Image** - An image is a read-only template that specifies the content and configuration of a container. It serves as a blueprint for creating containers. Docker images are reusable and shareable, allowing developers to package applications once and and easily deploy them to any environment.
- **Dockerfile** - A Dockerfile is a text-based script that contains a set of instructions for building a Docker image. It specifies the base image, adds application code, installs dependencies, and sets up configuration settings. Dockerfiles enable developers to define the desired state of containerized applications in a code-like format.
- **Docker Compose** - Docker Compose is a tool for defining and managing containerized applications. Through a Yet Another Markup Language (YAML) configuration file, developers can specify the services (containers), networks (communication channels), and volumes (data storage) required for an application. Docker Compose is especially useful for the orchestration of complex applications involving multiple containers.
- **Volume** - A volume in Docker is a mechanism for persisting and sharing data between containers or between a container and the host system. Volumes ensure that data is retained even if a container is stopped, removed, or replaced. They are typically used for storing database files, configuration data, and other forms of persistent data.
- **Container Orchestration** - Container orchestration is the automated management of containerized applications at scale. This encompasses tasks such as container deployment, scaling, load balancing, and health monitoring. Kubernetes [89] is a popular container orchestration platform that streamlines the management of containerized applications in production environments.

Docker has had a significant impact on the landscape of application development and deployment, providing a standardized and efficient method for packaging, distributing, and running applications within containers.

In the context of this study, containers offer a convenient solution for deploying various components of the envisioned system.

Regarding Spark, it can effectively harness this technology by utilizing multiple containers to meet its processing requirements. Custom Spark images can be used

to instantiate containers that can operate as nodes within a Spark cluster. A container orchestration platform like Kubernetes can serve as the cluster manager, responsible for both managing and scaling the resources needed by operational Spark applications.

### 2.4.5 Kubernetes

Kubernetes [89], often abbreviated as K8s, is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides a flexible and efficient way to manage clusters that execute multiple containerized applications, ensuring high availability, resource optimization, and seamless scalability. These features make it an essential part of most modern cloud-native infrastructure.

#### Kubernetes Concepts

To gain a better understanding of Kubernetes, it is helpful to delve into some of its key concepts [92]:

- **Cluster** - A Kubernetes cluster is a collection of interconnected machines that work together to manage and orchestrate containerized applications, ensuring efficient deployment and scalability. Clusters consist of two main components, namely a control plane in combination with one or multiple nodes.
- **Node** - A node, also known as a worker node, is an individual machine within a Kubernetes cluster. Nodes can be physical servers or virtual machines and provide the computational resources required to run the containerized applications.
- **Control Plane** - The control plane serves as the management component of a Kubernetes cluster. It abstracts the cluster's complex infrastructure and acts as the primary interface for interacting with it. Kubernetes adopts a declarative approach to define the cluster's desired state, and the control plane is responsible for maintaining this state. It efficiently distributes the workload across the nodes in the cluster, aiming to balance resource utilization and prevent overloading or underutilizing nodes. Additionally, the control plane regulates the desired state of various resources running in the cluster, such as ensuring the specified number of deployed replicas is maintained.
- **Pod** - A pod is the smallest deployable unit in Kubernetes. It encapsulates one or more coupled containerized applications and their shared resources, such as storage, within the same network namespace. Containers within a pod share the same IP address and can communicate with each other via localhost. This design simplifies network configuration and enables efficient inter-container communication within the pod. Pods serve as self-contained

units, bundling related components within a single entity, simplifying application deployment and enabling straightforward replication for scalability.

- **Volume** - A volume is a directory that can be mounted into one or more containers within a pod, providing a way for containers to store and share data. Volumes ensure data persistence even if containers are terminated or relocated to different nodes.
- **Deployment** - A deployment is a fundamental aspect of Kubernetes' declarative approach for managing applications. It specifies the desired state of an application, such as the number of replicas, and the control plane orchestrates the creation and scaling of pods to match that desired state. This enables automatic self-healing and also simplifies management tasks, for instance, updating containers through adjustments to the deployment specification.
- **Service** - In Kubernetes, a service acts as an abstraction for a group of pods that provide the same functionality. It establishes a stable network endpoint to access these pods, regardless of their number or location in the cluster. Services enable load balancing, automatic failover, and DNS-based discovery, to ensure the reliable availability of applications to others, both internal and external to the cluster.
- **Minikube** - Minikube is a tool for running a single-node Kubernetes cluster locally. It simplifies both the development of Kubernetes applications and the process of experimenting with them, by providing a convenient, isolated environment, without the need for a full-scale cluster.

The integration of Kubernetes with Apache Spark creates a powerful combination for managing large-scale data processing and analysis. Kubernetes offers a dynamic, container-based orchestration platform that simplifies both the deployment and scaling of Spark clusters. By leveraging Kubernetes, Spark clusters can benefit from automatic scaling, resource optimization, and efficient management, ensuring that available processing power can seamlessly adapt to changing demands.

## 2.5 System Evaluation Metrics

In this section, we explore appropriate metrics to assess the effectiveness of the envisioned recommendation system and examine an approach to determine the system's optimal configuration.

### 2.5.1 Binary Classifier

Understanding the operational mechanism of the envisioned system is crucial for selecting the right metrics to conduct an effective evaluation. The system will provide recommendations for data retention based on the deviation of the assessed network

traffic compared to other traffic in the considered reference data. Elevated deviations signify anomalies in the analyzed network traffic, potentially indicating the presence of a DDoS attack. Therefore, a recommendation for data retention can be triggered when the deviation surpasses a certain threshold. The recommendation system acts as a binary classifier, since estimating whether the network traffic contains a DDoS attack or not is a two-class classification problem.

### 2.5.2 Confusion Matrix

Evaluation of a classifier is typically conducted using a confusion matrix, which contains the information needed to calculate various evaluation metrics. It is a table that summarizes the performance of a classifier, where the rows represent the actual classes, and the columns represent the predicted classes. Since the actual classes must be known to construct a confusion matrix, it is necessary to let the system make predictions on a dataset for which the ground truth is available. The recommendation system works with only two classes, resulting in a simple confusion matrix with four cells. The presence of attack traffic is predicted, so the positive class represents network traffic containing attacks, and the negative class represents network traffic without attacks. A binary classifier can provide either a correct or incorrect prediction for each actual class, resulting in four potential outcomes. The corresponding confusion matrix is shown in Figure 2.3.

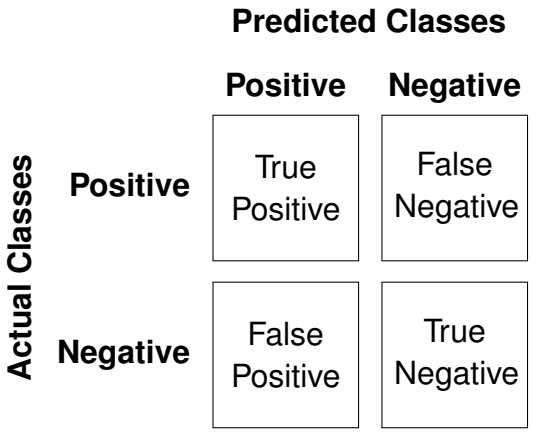



**Figure 2.3:** Confusion Matrix

The four cells in this confusion matrix represent the different possible outcomes:

- **True Positive (TP)** - This is correctly classified network traffic that contains an attack.
- **True Negative (TN)** - This is correctly classified network traffic that does not contain an attack.
- **False Positive (FP)** - This is benign network traffic that has been incorrectly classified as containing an attack, i.e. falsely retained traffic.

- **False Negative (FN)** - This is network traffic that contains an attack that is incorrectly classified as benign traffic, i.e. an undetected attack.

For the assessment of the recommendation system, predictions are made based on a dataset with available ground truth, and the results are arranged in a confusion matrix. Subsequently, the outcomes in this matrix are used to derive appropriate metrics for the evaluation of the system.

### 2.5.3 Accuracy

A common and intuitive metric for the evaluation of classifiers is accuracy, which is the ratio of correct predictions to all predictions. Accuracy is calculated with Equation 2.18.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.18}$$

However, the problem with the accuracy metric is that it is vulnerable to the accuracy paradox. This occurs when there is an imbalance in the dataset, and the presence of one class is more dominant than the other. In an example situation where 99% of the dataset consists of one class, the classifier can achieve an accuracy of 99% by simply always predicting this one class. This preference for the dominant class can lead to the neglect of the other class, even if it is the correct classification of the minority class that matters. A similar situation applies to the recommendation system, if we are to assume that attacks are the exception rather than the norm. Therefore, accuracy is not a suitable metric for evaluating this system. Fortunately, two other metrics exist that can be used to avoid this bias, namely precision and recall.

### 2.5.4 Precision and Recall

Precision and recall are performance metrics commonly employed to assess the accuracy and completeness of positive predictions, particularly in scenarios where the identification of positive instances is important.

#### Precision

Precision measures the accuracy of positive predictions by indicating the ratio of true positive predictions to all positive predictions. This is calculated with Equation 2.19.

$$Precision = \frac{TP}{TP + FP} \tag{2.19}$$

A high precision value indicates a low rate of false positives. This means that when the system predicts a positive outcome, it is more likely to be accurate, thereby minimizing falsely retained traffic. However, the pitfall of solely considering precision is that the system tends to not classify attacks at all. Therefore, the use of the recall metric is important as well.

### Recall

Recall measures the system's ability to identify positive instances by indicating the ratio of true positive predictions to all actual positives. This is calculated with Equation 2.20.

$$Recall = \frac{TP}{TP + FN} \tag{2.20}$$

A high recall value indicates a low rate of false negatives. This means that the system can identify a signification portion of the actual positive instances, thereby minimizing undetected attacks. However, only considering recall has a similar effect as solely relying on precision, but in reverse. In this case, the system would tend to classify everything as an attack. Hence, both metrics are typically used together for a proper evaluation.

### Precision-Recall Curve

In short, precision ensures that the system does not raise false alarms, while recall ensures that the system does not overlook attacks. A higher threshold can reduce false alarms, but it increases the likelihood of undetected attacks. Conversely, a lower threshold can decrease the number of undetected attacks, but it raises the occurrence of false alarms. This illustrates a trade-off between both metrics, and it can be graphed with a Precision-Recall (PR) curve, which is a plot of multiple PR pairs. These pairs are obtained by creating confusion matrices based on different threshold values and calculating the corresponding precision and recall values. Essentially, they form triplets, with each PR pair having an associated threshold value. By iterating over all thresholds that result in different PR pairs, it becomes possible to comprehensively analyze the trade-off. The pairs are plotted to graph the PR curve and visualize this trade-off. This plot displays precision on the Y-axis and recall on the X-axis. Given that both metrics represent a part-to-whole ratio, their values range from 0 to 1. Each point of the curve corresponds to a threshold value, and its position on the plot indicates the precision and recall when that threshold is applied. An effective classifier can maintain a high precision as its recall increases.

A baseline can be added to the plot, indicating the PR curve of a dummy classifier with no skill in predicting the correct classes. A dummy classifier adds no predictive value, either by predicting a random class or a constant class, so its predicted classes are just samples of the dataset. Precision represents the proportion of true positives in the positive predicted class. As a consequence, the precision of a dummy classifier is determined by the proportion of true positives in the taken sample. If the sample is large enough, it approximates the same class proportions as observed in the entire dataset. Therefore, the precision of a dummy classifier is approximately equal to the proportion of true positives in the dataset. The precision of a dummy classifier remains constant for any threshold value, as the threshold only affects the sample size while the proportions remain consistent. Recall, on the other hand, is affected by the threshold value, as it causes the number of correctly pre-

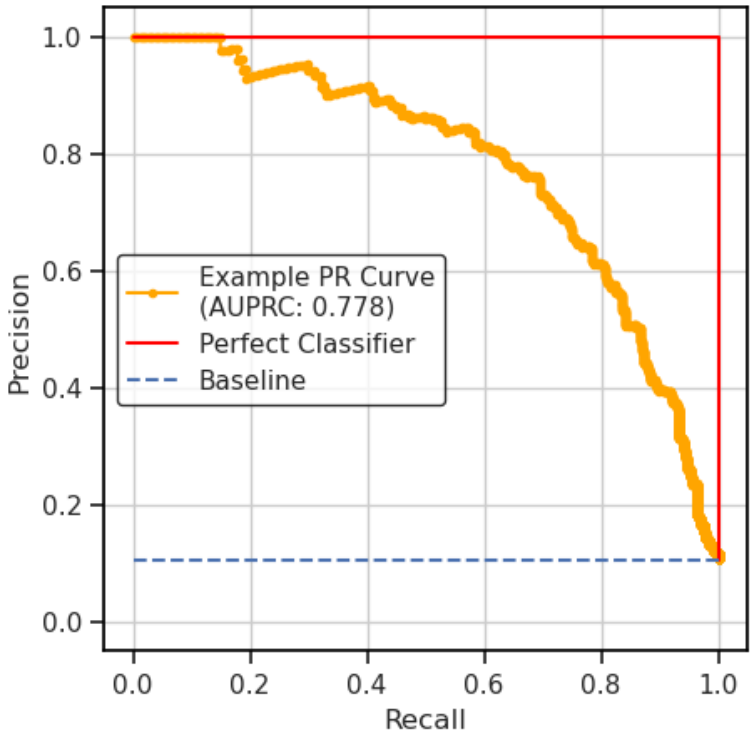



**Figure 2.4:** Example Precision-Recall (PR) Curve

dicted positives to vary while the number of actual positives in the dataset remains the same. This results in a baseline where the precision for each value of recall is equal to the proportion of true positives in the dataset. An effective classifier resides above the baseline, whereas an ineffective classifier falls below it. However, reversing the predictions of a classifier below the baseline results in a curve above the baseline. Some classifiers can have both, with parts of the curve above and other parts below the baseline, depending on the threshold value.

A perfect classifier can completely distinguish both classes within a specific range of thresholds, where both precision and recall are 1. Moving the threshold further beyond either side of this range affects only one metric at a time. As a result, the precision remains at 1 for each recall value, and when the recall is 1, the precision ranges from 1 to the baseline. The more effective the classifier, the closer its curve gets to the curve of the perfect classifier.

In Figure 2.4, an example PR curve is depicted, also featuring the baseline and the curve of a perfect classifier. This example PR curve represents the performance of a logistic regression classification model on a dataset with a distribution of 10% positive class and 90% negative class, placing the corresponding baseline at 0.1 precision.

The PR curve provides a comprehensive overview of the system's performance across the entire spectrum of varying thresholds, encompassing all different operating points. This offers valuable insights into the performance that can be expected during an actual deployment. However, for it to be representative, it is crucial to

use a dataset that accurately reflects the proportions of attacks and benign traffic observed in real-world scenarios. This is particularly important because the recall metric depends on the proportions of actual classes within the dataset, which can significantly affect the evaluation.

The Area Under the PR Curve (AUPRC) serves as an indicator of the classifier's overall performance. This metric facilitates comparing multiple classifiers based on their performance across the entire threshold spectrum.

### 2.5.5 Optimal Operating Point Based on F-scores

In this research, the evaluation focuses on the performance of the recommendation system at various threshold values, with a particular interest in its performance at the optimal operating point. The optimal operating point of the recommendation system depends heavily on the costs of undetected attacks and falsely retained traffic. Different costs create different cost scenarios, each with its own optimal operating point and corresponding threshold value.

The F-score can be used to find the optimal operating point for various scenarios. This metric combines precision and recall into a single value by calculating their harmonic mean. The harmonic mean, as shown in Equation 2.21, is used to calculate the average of rates and ratios.

$$Harmonic\ mean = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} \tag{2.21}$$

The traditional F-score considers precision and recall equally, but a more generalized F-score allows precision and recall to be valued differently. This traditional F-score, also called $F_1$-score, is the harmonic mean of precision and recall. It is calculated with Equation 2.22.

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2\frac{precision \cdot recall}{precision + recall} \tag{2.22}$$

The generalized form is called the $F_\beta$-score, as it uses a $\beta$ factor to influence the weight of recall. This $F_\beta$-score is calculated with Equation 2.23.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{2.23}$$

Recall is considered $\beta$ times more important than precision with a $\beta$ value greater than 1. If precision should be considered more important, the inverse can be used for $\beta$. Note that using a $\beta$ of 1 results in the traditional F-score, which is why it is also referred to as $F_1$-score.

The costs associated with undetected attacks and falsely retained traffic determine the most appropriate $\beta$ for a given cost scenario. Using this $\beta$, the F-scores for all PR pairs on the curve can be calculated. The pair with the highest F-score indicates the optimal operating point, and the corresponding threshold produces the

most favorable results by accounting for the costs related to undetected attacks and falsely retained traffic. The precision and recall values provide insight into the performance that can be expected using this threshold. However, for these results to be representative, it is necessary that the dataset closely resembles the network traffic observed in real-world scenarios.

## 2.6 Related Work

In this section we discuss related work, including three studies on entropy-based DDoS detection and a study on storage reduction for network traffic retention. These studies use metrics similar to those in our study, enabling a meaningful comparison during the discussion.

### 2.6.1 4EMA: A Hybrid Detection Method for DDoS Attacks

Bojović et al. [67] developed a DDoS detection method that combines feature-based and volume-based detection, by utilizing both packet diversity and packet rates. They measured the packet diversity by calculating the Shannon entropy of communication pairs, which are the combinations of source and destination IP addresses. Their approach computes short-term and long-term trends for both metrics using the EWMA. Bojović et al. use the term Exponential Moving Average (EMA) instead of EWMA, but these refer to the same concept. Since their method uses it four times, both short-term and long-term for each of the two metrics, they called it 4EMA. 4EMA triggers a DDoS detection when the differences between these short-term and long-term trends exceed predefined thresholds.

The method was tested in an academic network with about 300 workstations and 20 servers. The tests, which included ICMP flood and TCP SYN attacks, were executed using a simulated botnet. Attack scripts were distributed via Microsoft Windows Active Directory and selectively deployed to logged-in student computers in classrooms. This caused intensity fluctuations that simulate the variability in real-world DDoS scenarios, as computers were powered on and off and students logged in and out.

The results showed that the method successfully detected high-rate ICMP flood attacks, with minimal false positives. It was also reliable at detecting low-rate TCP SYN flood attacks, but it occasionally generated multiple alerts for a single low-rate attack. Although all attacks were detected, the 4EMA method could not always accurately determine the attack periods due to delays introduced by the EWMA. However, using the EWMA did effectively reduce the number of false positives during short bursts of regular traffic. The 4EMA method achieved a precision of 90% and a recall of 76% when it was optimized for the F1-score, i.e. when equal costs were associated with false alarms and undetected attacks.

### 2.6.2 D-FACE: An Anomaly-Based Distributed Approach for Early Detection of DDoS Attacks and Flash Events

Behal et al. [93] introduced D-FACE, a distributed system designed for the early detection of DDoS attacks and flash events within Internet Service Provider (ISP) networks. D-FACE deploys its traffic analysis at multiple points of presence across the network infrastructure. Each point independently calculates the traffic rate of its observed network traffic and the generalized entropy of the source IP addresses from unique network flows within that traffic. These metrics are then sent to a central point, where they are aggregated to determine the overall traffic rate and generalized entropy across the network. By distributing the analysis workload, this approach reduces the required computation and memory resources at any single point.

The detection process of D-FACE involves analyzing network traffic in time windows of 1 second. The aggregated traffic rates and generalized entropy values are compared with their baselines to identify anomalies. These baselines and corresponding deviation thresholds are predetermined by analyzing the behavior of the network traffic without the presence of attacks. The absolute difference between the observed entropy and the baseline entropy is used as an indicator, which is called the information distance. D-FACE uses the traffic rate and information distance to classify the traffic in each window into one of the following four categories:

- **High-Rate DDoS Attacks** - High packet rate and high information distance.
- **Low-Rate DDoS Attacks** - Low packet rate and high information distance.
- **Flash Events** - High packet rate and low information distance.
- **Legitimate Traffic** - Low packet rate and low information distance.

The D-FACE system was validated against various DDoS attacks and flash events using a hybrid testbed environment that combines both physical and virtual infrastructure. This testbed consists of 75 physical nodes organized into three clusters, along with a web server that is designated as the victim of the attacks. These clusters and the victim web server are interconnected by 4 routers, which also serve as the points of presence of the D-FACE system within this network infrastructure. Additionally, the testbed utilizes emulation to scale up the number of nodes, as each physical node emulates 16 virtual clients and four software routers.

The authors used legitimate traffic traces from the MIT Lincoln [94], CAIDA [95], and FIFA World Cup [96] datasets to establish the baseline traffic behavior. Furthermore, traffic traces from the CAIDA and FIFA World Cup datasets were used to generate the anomalies in the testbed. Both low-rate and high-rate DDoS attack scenarios were sourced from the CAIDA dataset, and the flash event traces were sourced from the FIFA World Cup dataset. In addition to this, the authors simulated a botnet by generating synthetic botnet traffic in the testbed using multiple instances of the BoNeSi [97] botnet simulation tool.

In these validation tests, the D-FACE system achieved an overall precision of 97% and an F-score of 95% when its configuration was optimized. These results imply a recall of 93%. The authors also emphasize the system's capabilities for early detection and accurate classification of the network anomalies included in this validation.

### 2.6.3 An Online Entropy-Based DDoS Flooding Attack Detection System With Dynamic Threshold

Tsobdjou et al. [98] developed an online entropy-based system for detecting DDoS flooding attacks in client-server environments. Their method detects attack connections by first calculating the normalized Shannon entropy of source IP addresses observed during a given time window, and counting the number of packets for each connection within that window. Detection is then performed based on these features using dynamic thresholds that adapt to variations in legitimate traffic.

Their proposed system consists of five modules:

1. **Features Extraction and Connections Construction** - This module produces bidirectional network connections of the observed network traffic within a given time window. Each connection is defined by the source and destination IP addresses, and this module tracks their packet counts in both the forward and backward directions, as well as the timestamp of the first packet of the connection.

2. **Suspicious Activity Detection** - Using the connections built in the first module, this module calculates the normalized entropy of source IP addresses within each time window. A window is flagged as suspicious activity if the entropy falls below a dynamically updated entropy threshold.

3. **Attack Connections Detection** - This module is triggered when the second module detects suspicious activity within a time window. It identifies attack connections by comparing the number of packets in each connection against a dynamic packet count threshold. Connections that exceed this threshold are marked as potential attack connections.

4. **Alert Generation** - For each detected attack connection, this module generates an alert containing the connection identifier, timestamp, and detection time. These alerts are intended for network administrators to take appropriate actions.

5. **Threshold Update** - If no suspicious activity is detected, this module updates both the entropy and packet count thresholds based on the observed legitimate traffic. The thresholds are created based on the interval given by Chebyshev's theorem, using the lower bound ($\mu - k \cdot \sigma$) as the threshold for the normalized entropy, and the upper bound ($\mu + k \cdot \sigma$) as the threshold for the number of

packets in a connection. Here, $\mu$ is the mean, $\sigma$ is the standard deviation, which are both calculated based on recent previous values. The $k$ factor is varied during the evaluation to find an optimal value for improving detection performance. This approach with upper and lower bounds is used because the authors claim that the entropy decreases during an attack, while the number of packets in the connections increases.

The proposed detection system was evaluated through simulation of synthetic attacks and with a publicly available dataset. The simulations were conducted using the GNS3 network simulator [99], in which the simulation environment consisted of four LANs. Three of these LANs each contained two legitimate clients and one malicious client, while the fourth LAN contained two Apache web servers [100] and the DDoS detection system. Legitimate traffic was generated using the httperf tool [101], and attack traffic for ICMP flood and SYN flood attacks was simulated using the hping3 tool [102]. Each simulation ran for six minutes, with the attacks occurring during the fourth minute. For the evaluation based on this simulated traffic, the system achieved a 100% detection rate, also referred to as recall, and a 100% precision.

Additionally, the publicly available CICIDS2017 dataset [103], created by Sharafaldin et al. [104], was used to further evaluate the system. Only a specific section of this dataset was used, that contains both legitimate traffic and DDoS attack traffic generated by the LOIC tool [9]. The system's evaluation based on this subset of the dataset resulted in a 100% detection rate (recall) and a 100% precision.

Thus, the system showed an overall recall of 100% and a precision of 100% in both evaluations, which involved the simulation with synthetic attacks and the publicly available CICIDS2017 dataset.

### 2.6.4 Smart Collection and Storage Method for Network Traffic Data

A related technical report in the field of network traffic data management, by Horneman and Dell [105], describes a smart collection and storage method aimed at optimizing storage efficiency for network security monitoring. Their approach shows organizations how to selectively store network traffic based on its criticality and effectiveness, thereby balancing the trade-off between the availability of information for analysis and storage-related constraints.

Before describing their methodology, the authors outline two important categories of considerations for evaluating different capture and storage solutions:

- **Filtering** - This involves methods for choosing which network traffic to store, including predefined or customizable filters, which can be based on network or application-level characteristics. Filtering can occur in-line during the capture

or post-process after the data has been written to disk, both impacting performance and storage differently. Other filtering methods are also discussed, like sampling with pseudorandom selection or only saving specific parts of packets.

- **Storage Methods** - This involves the techniques for packaging and storing the captured data, including local or cloud-based solutions, centralized or distributed systems, and file-based or database storage. Each option has its own costs, security implications, and technical feasibilities. Additionally, file formats and compression are discussed, as they can significantly impact storage efficiency and costs.

To determine the optimal storage policy, Horneman and Dell suggest that organizations should quantify the risks and effectiveness of storing different types of network traffic at various capture tiers and then evaluate how these values change over time. These values can be combined with related costs into a cost-benefit analysis.

The authors first emphasize the importance of understanding the benefits, limitations, and risks associated with the following three capture tiers, which store the traffic with distinct levels of detail:

- **Full Packet Capture** - This provides the most comprehensive information by storing all the packet data, but it requires significant storage and processing resources. It also poses risks such as storing sensitive data and legal implications for data protection.
- **Network Flow Capture** - This is the most storage-efficient option as it does not store payload data and combines information from multiple packets of the same flow into one record. It is effective for long-term storage but it offers limited analytical capabilities and it is therefore not useful for detailed investigations.
- **Augmented Flow Capture** - This provides a middle ground by storing flow records enriched with additional data. Depending on the augmentation, storage requirements for augmented flows can approach or exceed those for full packet captures. Organizations should treat these files with the same caution as full packet captures due to the potential presence of sensitive data.

The authors also highlight the need to consider the specific features of different traffic protocols. For example, the information in DNS traffic can provide valuable insights, while the content of encrypted protocols is generally inaccessible, and capturing Voice over IP (VoIP) data may even present legal issues as it is a form of telephony. It is also beneficial to check if other existing applications already store the data to prevent duplication. For instance, email is often already stored long term on an exchange server. By tailoring the storage strategies to these specific characteristics of protocols, organizations can optimize their data management practices.

Determining the optimal storage duration for different types of network traffic is another important aspect. Organizations should evaluate the duration of data

usability, by considering how long the data remains useful or how long it takes to become useful. In addition, they must ensure that the traffic storage meets legal obligations and service level agreements.

In order to formulate a filtering and storage policy that yields the highest value while staying within the organization's constraints, Horneman and Dell introduced a method that involves ranking the criticality and effectiveness of different traffic types. This method consists of the following six steps:

1. Identify applicable reasons for using captured data.
2. Determine how long it is desirable to store the captured data.
3. Identify the attack categories that are most relevant to the organization and the data necessary to investigate them.
4. Find the storage time frame that is most effective for storing data related to each specific attack category.
5. Determine the essentiality of each traffic type per capture tier, by evaluating and balancing the criticality of attacks, the relevance and effectiveness of each traffic type for investigation, and the associated storage risks. Additionally, determine the maximum number of useful days for each traffic type based on its effectiveness duration for investigating various attacks.
6. Determine the total storage requirements and appropriate storage tiers for each traffic type by calculating the expected traffic volume and accounting for future growth. If the storage requirements exceed the available capacity, use the criticality and effectiveness values to prioritize important traffic types and their storage days.

The outcome is a prioritized storage policy that aligns with the organization's needs and resource constraints.

The authors emphasize that organizations must plan for future network growth when they purchase or implement a capture solution. This involves considering the expected lifespan of the solution, making future bandwidth projections, assessing potential growth within the current infrastructure, anticipating changes in network usage policies that could affect traffic types and volumes, and analyzing trends in network traffic and protocol-specific data.

After outlining their method, Horneman and Dell provide three examples to show how the method is applied to actual networks of different sizes, classified as small, medium, and large. These examples illustrate that their method can offer storage policies tailored to the specific needs of different organizations. The results reveal that a tiered storage approach can significantly reduce storage requirements while still retaining the necessary information for effective analysis.

The method achieved the highest percentages of storage reduction when it was applied to the large network. Table 2.2 shows the achieved savings in storage for the large network with different filtering options at each capture level.

**Table 2.2:** Storage Savings Achieved in the Large Network Example

| **Traffic Kept** | **Savings for Network Flow** | **Savings for Augmented Flow** | **Savings for Full Packet Capture** |
|---|---|---|---|
| All | - | - | - |
| All but Encrypted | 30.49% | 30.49% | 29.43% |
| All but Encrypted and Email | 31.41% | 31.41% | 29.83% |
| All but Encrypted, Email, DNS, NTP, and ICMP | 49.84% | 49.84% | 29.96% |
| All but Encrypted, Email, DNS, NTP, ICMP, and VoIP | 50.43% | 50.43% | 29.97% |

# Chapter 3

# Methodology

In this chapter, we explain our methodology, detailing how we addressed both research objectives and our approach to answer the research questions. We begin with the design of the system, followed by the implementation and deployment. Subsequently, the evaluation process is explained, encompassing the methodologies used to assess the effectiveness of the system and to answer the related research questions. The last section provides insights into the dataset utilized for evaluation, by outlining the acquisition process of the dataset.

## 3.1 System Design

This section is dedicated to achieving Objective 1 of this study. In Chapter 2, an extensive literature review was conducted to gain a better understanding of the research area and its operational context, as well as to identify useful methods and techniques for the envisioned system. This section proceeds to outline the design of our system based on the gained insights from the literature review. It begins with the design of the analysis technique and subsequently addresses the design of the processing architecture to apply the analysis.

### 3.1.1 Analysis Design

The envisioned system should perform a rudimentary analysis of network traffic to provide recommendations regarding the potential presence of traffic related to DDoS attacks. These recommendations need to be provided with low-latency to prevent the accumulation of temporarily stored traffic, aiming to ensure that only DDoS-related data is retained for more extensive analysis.

The design of the analysis technique involves three distinct steps:

1. Determining the input data source for the analysis.
2. Establishing how this information can serve as an indicator for DDoS attacks.
3. Defining the mechanism to detect anomalies based on this indicator.

### Data Source

The data source for the analysis should reflect the patterns and behavior of the network traffic, enabling it to effectively detect ongoing DDoS attacks. Simply processing all individual network packets incurs a significant computational overhead, especially in large networks with high data throughput, where packet capture requires expensive hardware and substantial infrastructure for storage and analysis [40]. Moreover, the large-scale use of encryption makes a significant part of the packet data inaccessible [5]. The literature has shown that the use of flow data proves to be a useful solution to address these issues [40]. This approach involves aggregating network traffic information into a more concise format, focusing solely on unencrypted header fields and accessible metadata. This reduces the overall volume of data that requires analysis, while preserving the ability to represent traffic patterns and behavior through its flow attributes. For an improved representation and to reduce bias in the underlying attribute distributions, it is recommended to use bidirectional flows [45], which captures the requests as well as the responses within a single flow record. This approach enhances the comprehensiveness of information regarding communication between network entities.

Such a flow-based approach aligns with the recommendation by Sperotto et al. [106], who advocate the utilization of flow data in a two-stage analysis process, which bears resemblance to our envisioned system. In the initial stage, flow-based methodologies can be effectively harnessed to detect network traffic attacks. Subsequently, in the second stage, a more detailed packet inspection can be employed to conduct advanced analyses, focusing on the activities detected during the initial stage.

However, Hofstede et al. [69] highlighted a notable drawback related to flow monitoring. The aggregation process involves collecting data throughout the entire duration of a flow, which means the data becomes available only after the flow ends. To overcome this, they proposed an approach that integrates intrusion detection into the flow metering process. They implemented this as a plugin for INVEA-TECH's FlowMon platform [107], which provides a highly customizable plugin-based architecture. They achieved promising results, but their implementation seems too specific and therefore limited, as it would not be compatible with commonly deployed flow monitoring infrastructure that does not support such a plugin-based approach. In these other scenarios, integrating intrusion detection into the flow metering process would require more complex modifications to the existing infrastructure.

Typically, the delays are affected by the configured IP flow expiration timeouts [69, 78], so a more straightforward alternative is to set these timeouts relatively low. However, it is important to strike a balance and avoid excessively short timeouts to ensure that the distinction between short- and long-duration flows remains clear. Moreover, the use of very short timeouts would diminish the benefits of using flows significantly, as less data gets aggregated, resulting in more processing overhead and larger storage requirements. For this study, we assumed that an effective com-

promise in this regard is a 30-second active timeout, adeptly managing the trade-off between near real-time updates and the preservation of flow duration information. This assumption was made to limit the scope of our research. The result is a more practical approach that minimizes delays, avoiding the need to integrate intrusion detection into the flow metering process, which would require more complex modifications within the existing flow monitoring infrastructure.

The flow data can consist of various attributes depending on the configuration of the flow monitoring setup. For this study, we focused on the following flow attributes: source address, destination address, source port, destination port, protocol, flow duration, and flow size in packets and bytes. These attributes are widely recognized as the core attributes of flow data in network flow analysis and are typically available across all flow monitoring systems. So, this selection aligns with common attribute standards in the field, ensuring compatibility with most established flow monitoring practices and existing flow monitoring infrastructures. While this study focuses on these core attributes, future research may consider exploring additional attributes to improve the system's performance.

### Feature Engineering

The flow data can be used for the detection of DDoS attacks since these attacks affect metrics that can be derived from the flow records [40]. Therefore, converting the flow data into quantitative features is important to make them serve as actual indicators of such attacks.

As discussed in Chapter 2, entropy is a metric that can aggregate attributes from multiple flows into a quantitative feature. This feature represents the measure of concentration or dispersion in the distribution of the considered attribute, which can change significantly during a DDoS attack and thus serve as a proper indicator. As mentioned by Nychis et al. [45], bidirectional flows should be used to mitigate bias in the attribute distributions.

There are several types of entropy that can be applied, including the well-known Shannon entropy, as well as the Renyi and Tsallis entropies. The latter two can offer more extensive perspectives due to their entropic parameters.

Bereziński et al. [44] reviewed multiple studies in an attempt to identify the ideal entropic parameter for detecting network traffic anomalies. However, they found that this issue remains unresolved due to inconsistent results across various studies.

In their own research, Bereziński et al. compared the Shannon, Renyi and Tsallis entropies for detection of anomalies in network traffic. They observed that the Renyi and Tsallis entropies performed slightly better than Shannon entropy, but the authors refrained from specifying a single entropic parameter value and suggested a range of values instead.

Given the uncertainty surrounding the selection of entropic parameters, we opted for Shannon entropy in our study. The fixed nature of Shannon entropy eliminates the need for entropic parameter tuning, while only slightly degrading performance.

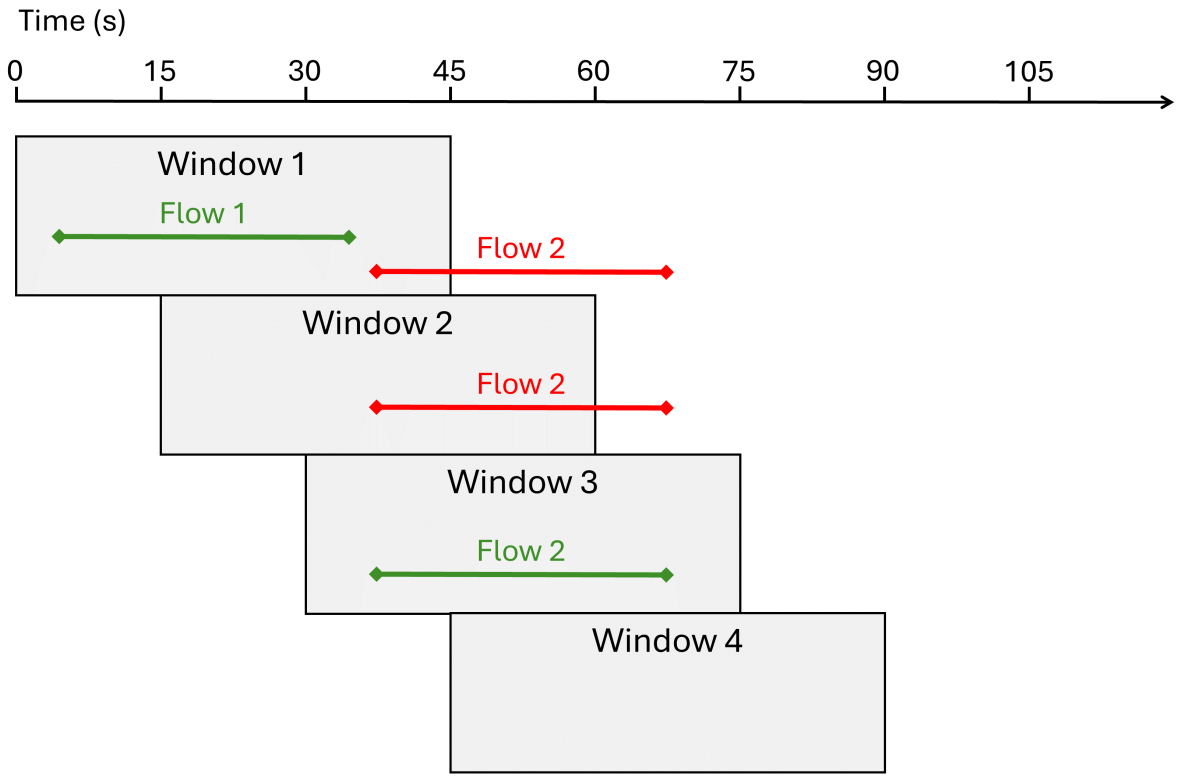



**Figure 3.1:** Sliding windows of 45 seconds with 15-second intervals, along with two example flows that have the maximum duration of 30 seconds, to show how flows have their starting points in multiple overlapping windows while fitting entirely within at least one of these windows.

Using Shannon entropy makes it possible to establish a clear performance baseline within the time frame of this study, leaving parameter tuning with the other entropies to future research.

Because the calculation of entropy involves aggregating grouped flows, it is required to establish a method for grouping multiple flows together. In this study, time-based windowing is employed for this purpose, with flows grouped based on their starting timestamps. A window size of 45 seconds is used within a sliding window approach, where a new window is created every 15 seconds. This ensures that every flow entirely fits within at least one window, given that the maximum duration of a flow is limited to 30 seconds. Figure 3.1 visualizes this by showing two example flows, both with a maximum duration of 30 seconds. Flow 1 starts and fits entirely in Window 1. On the other hand, Flow 2 has a starting point that falls in Windows 1, 2, and 3. However, it exceeds the ends of the first two windows, but it does fit entirely in Window 3.

Using shorter sliding intervals does increase the number of windows, which results in a higher granularity and reduced delays, but it comes at the cost of a higher computational workload since more windows must be processed. Conversely, employing longer intervals results in fewer windows, which decreases the granularity

**Table 3.1:** Flow Attributes with Corresponding Entropy-based Features

| **Flow Attribute** | **Feature** |
|---|---|
| Source Address | Shannon Entropy of the Source Address Distribution |
| Destination Address | Shannon Entropy of the Destination Address Distribution |
| Source Port | Shannon Entropy of the Source Port Distribution |
| Destination Port | Shannon Entropy of the Destination Port Distribution |
| Protocol | Shannon Entropy of the Protocol Distribution |
| Flow Duration | Shannon Entropy of the Flow Duration Distribution |
| Flow Size (Packets) | Shannon Entropy of the Flow Size (Packets) Distribution |
| Flow Size (Bytes) | Shannon Entropy of the Flow Size (Bytes) Distribution |

and increases the delays as the windows are processed less often. This aligns with the study by Cermak and Celeda [78], where they pointed out that longer time windows can indeed lead to increased delays in identifying malicious activities, which is not compatible with our low-latency goal from Objective 1. For this study, the use of a 45 second sliding window every 15 seconds is considered a practical balance, with the aim of providing sufficient granularity and responsiveness within a manageable workload.

So, we compute the Shannon entropy for each of the selected flow attributes, aggregating them within 45-second sliding windows. The entropy calculation requires the PMF of each flow attribute as input. We use the PMFs of the flow attributes within each sliding window as defined in Table 2.1. The resulting entropy values are then used as features for the detection in the next step. An overview of the flow attributes and their corresponding features is provided in Table 3.1.

### Detection Mechanism

We implemented a detection mechanism using these entropy-based features derived from flow attributes, as shown in Table 3.1. These features represent the state of the network within specific time windows. This mechanism is used to distinguish anomalous windows, as these may contain DDoS-related traffic. In Chapter 2, various methods for this purpose have been explored. The modified Z-score stands out for its robustness and simplicity. It requires other data points to calculate the median and the Median Absolute Deviation (MAD), which are used as estimators of central tendency and variability, respectively. In other words, these represent the expected normal behavior of the network traffic.

The use of the median in its estimators ensures it is more robust against extreme outliers, which is important since these outliers are expected during attacks. As mentioned, it has a breakdown point of approximately 50%, meaning that a substantial portion of the data points would have to exhibit deviant behavior before it

starts to influence the estimators.

The simplicity of the modified Z-score is another advantage. It requires less complex parameter tuning than some of the other methods, making it a good fit within the time frame and scope of this study. To tune the modified Z-score to be as accurate as possible, we have to determine which other data points are used for its calculation and what threshold is used for the detection.

In this study, the data points used to represent this normal behavior are the entropy values observed within multiple windows preceding the currently assessed window. This choice is made with the intent of accommodating natural fluctuations in network activity over time. We opted for a two-hour time frame based on our assumption that it enables the expected normal behavior to adapt to changing conditions during the day, while still maintaining a relatively stable reference point.

It is important to note that with the two-hour time frame, any DDoS attack exceeding the duration of one hour would be considered part of the normal behavior, potentially leading to an oversight in recognizing attacks. However, according to the latest extensive report on DDoS attacks by the NBIP [6], the vast majority (87.60%) of attacks observed on the National Scrubbing Center against DDoS attacks (NaWas) platform in 2020 had a duration of less than an hour.

The modified Z-score quantifies the extent to which the assessed window deviates from the expected normal behavior. This value is subsequently compared against a threshold to determine if it is considered an anomaly. The threshold is an important parameter that influences the sensitivity of the system, and its impact is thoroughly examined during the evaluation stage of this study.

In this study, we focus on the general design of the system and the impact of the threshold value, rather than aiming for complete optimization of all parameters. Consequently, we have chosen to keep the other parameters, such as the chosen timeout, window size, sliding interval, and the size of the normal behavior window, constant throughout our research and we made deliberate choices for their values based on lessons learned from existing work. We do acknowledge that investigating the effects of these parameters with different values would be interesting for future research.

### 3.1.2 Architecture Design

This section delves into the design of the processing architecture for conducting the flow-based analysis, briefly outlining the creation process and providing an overview of the architecture with explanations of its main components. Further details on the specific implementation and deployment are provided later in Section 3.2.

Considering the flow-based nature of the designed analysis approach, the foundation of the architecture design is based on the general architecture of a flow monitoring system. As shown in Section 2.2.5, a flow monitoring system typically consists of three main components: a flow exporter, a flow collector, and a flow analysis appli-

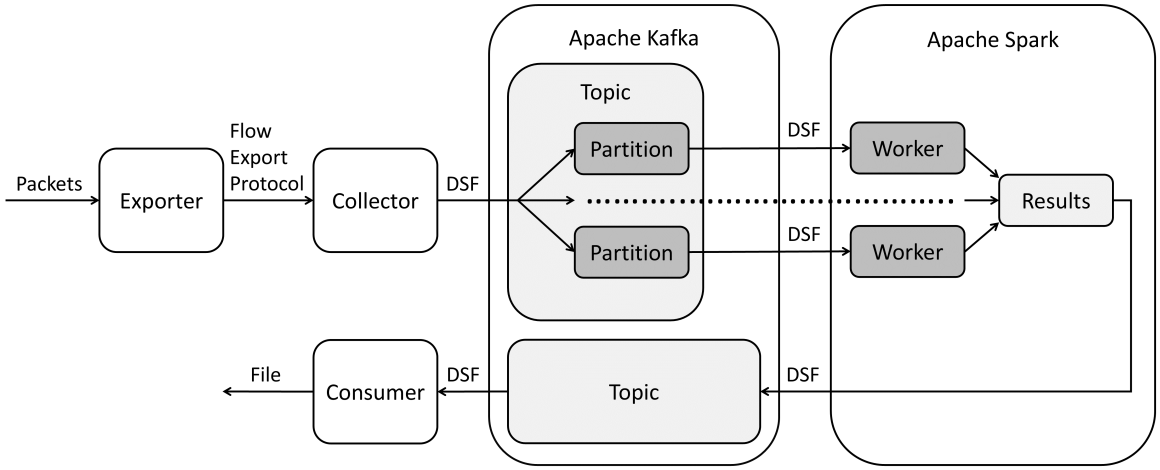



**Figure 3.2:** Overview of the Designed Processing Architecture

cation. The combination of these components creates the backbone of a flow-based analysis infrastructure.

For the analysis application, the architecture is primarily based on the design of Stream4Flow, which was discussed in Section 2.4.1, as it has introduced an innovative approach that enables scalable and near real-time analysis of flow data. Their approach involves integrating Apache Kafka and Apache Spark to establish a streamlined processing framework for implementing the flow analysis application. This integration requires data transformation by the collector into a DSF that is compatible with Kafka and Spark. Subsequently, Kafka and Spark process the flow data in a stream-based and distributed manner, ensuring scalability to accommodate the demands of large-scale networks while enabling near real-time analysis.

The analysis results are sent back to Kafka, where they can be accessed for further applications. Stream4Flow includes a database and a web interface for queryable storage and visualization of the results. However, these additions exceed the scope of this study, in which we primarily focus on the processing aspect of the system. For that reason, a simpler approach is adopted, utilizing a consumer that retrieves the streaming analysis results from Kafka and stores it in a regular file.

An overview of the designed processing architecture is shown in Figure 3.2, consisting of the following five components:

1. **Exporter** - The exporter is responsible for generating flow records based on the packets observed on its network interface. It can be integrated into network devices such as routers, switches, and firewalls or installed as software on servers and other network endpoints. The exporter uses a flow export protocol for transmitting the flow data to the collector. Flow data is typically sourced directly from network devices, and to enhance data collection, multiple exporters can be strategically deployed at various vantage points within the network. However, we use a single exporter in this study, as we consider it sufficient to both demonstrate the feasibility of the system design and achieve

our stated objectives.

2. **Collector** - The collector receives the flow data from the exporter and converts it into a DSF compatible with the subsequent components of the system. After conversion, the collector forwards the data to the messaging system. Multiple collectors can be used in parallel to manage higher workloads. However, we employ only one collector instance in this study, as we consider it sufficient to both demonstrate the feasibility of the system design and achieve our stated objectives.

3. **Apache Kafka** - The Apache Kafka messaging system facilitates the ingestion, storage, and distribution of data to enable distributed processing by the Apache Spark stream processing framework. Kafka is known for its ability to handle high-throughput, fault-tolerant, and real-time data streaming with low latency. Kafka categorizes data streams as topics, and the topics are divided into partitions to facilitate parallel processing, enabling horizontal scaling of workers in big data analysis frameworks like Apache Spark. Kafka itself can also be scaled horizontally, but to keep it more simple and within the scope of this study, only one instance is used, while it does provide multiple partitions to support parallel processing within Spark.

4. **Apache Spark** - The Apache Spark stream processing framework conducts the flow analysis. Spark can leverage multiple workers nodes to process data from various Kafka partitions in parallel. In this study, Spark is deployed with multiple worker nodes to demonstrate its distributed processing capabilities. The analysis results are sent back to Kafka using a compatible DSF. Other applications can retrieve these reported results from Kafka for further use.

5. **Consumer** - A basic consumer is used to retrieve streaming data from the Kafka topic containing the results and save it to a file. This file should use a versatile and clearly structured data format to facilitate evaluating the system of this study.

## 3.2 System Implementation and Deployment

This section is dedicated to Objective 2 and outlines the implementation and deployment of the system. The deployment process can be divided into two main parts, namely the Spark application, which is deployed on a Spark cluster, and the remaining system components, which are deployed within Docker containers. This separation enables the Spark application to demonstrate its utilization of a Spark cluster for distributed computing, while the Docker containers facilitate seamless deployment of the other components. The implementation and deployment of the Dockerized system components are discussed first, followed by that of the Spark application. The code for the implementation and deployment of the system is

publicly available in the following GitLab repository, hosted by the University of Twente:
`https://gitlab.utwente.nl/BasBleijerveld/scalable-ddos-traffic-retention/-/tree/main/system`

### 3.2.1 Dockerized System Components

Docker containers encapsulate dependencies and configurations, establishing a self-contained environment for each component. We adopted this technology to ensure a consistent and easily maintainable deployment across diverse computing environments. To manage the interactions among multiple containers, Docker Compose is utilized. As mentioned in Section 2.4.4, this tool streamlines the configuration of a multi-container setup, ensuring that the components work cohesively as an integrated system. Each component of the system is deployed within its own container, this includes the exporter, collector, messaging system and the consumer. The subsequent part of this section delves into the implementation and deployment of each of these components, followed by their integration using Docker Compose.

#### Exporter

As mentioned during the architecture design in Section 3.1.2, a single instance of a flow exporter is used as the source of flow records in this study. For the implementation, we decided to use softflowd [108], because it is an open-source and accessible software-based flow exporter that supports both NetFlow and IPFIX as flow export protocol. In this study, IPFIX is used since it is the open standard developed by the IETF.

Softflowd is installed within the Docker container and configured to export unidirectional flows, although our initial plan, as outlined during the analysis design in Section 3.1.1, was to use bidirectional flows in accordance with the recommendation by Nychis et al. [45]. However, we encountered an unexpected implementation of bidirectional flows by the developers of Softflowd. Specifically, Softflowd assigns the source of the flow based on the lowest value of the two communicating IP addresses, which does not accurately reflect the actual initiator of the flow. Due to this limitation and the lack of a suitable alternative, we opted to use unidirectional flows, as this results in a representative configuration that produces flow records in line with other flow exporters. Future work could involve an implementation with proper bidirectional flows to evaluate their potential improvement of the results.

Additionally, there are several other configurations that require specification. The time format used for the exported flow records is denoted in seconds, as this provides a sufficient level of granularity for the designed analysis approach.

A maximum flow lifetime of 30 seconds is enforced, which triggers the expiration and exportation of all flows exceeding this duration. This ensures timely updates of long living flows, as discussed in Section 3.1.1. We disabled the interval for expiry

checks, allowing the expiration and exportation of flows to occur as fast as possible in rapid succession. TCP is utilized as the transport protocol and the collector is configured as the destination for the exported flow records.

The softflowd exporter can use either a network interface or a PCAP file as an input source for the observed packets. In this study, the source of network packets is a dataset provided in the form of PCAP files. Although softflowd supports PCAP files as input, it processes and exports them as fast as possible, resulting in the loss of correct arrival times between the exported flows, which can cause potential problems in the Spark processing framework. Efficiently working with streaming data in Spark requires the specification of a threshold for late or out-of-order data arrivals. In Spark, this threshold is referred to as the watermark, and it is essential for managing the tracked states during the analysis. For this threshold to hold any significance, it is required that the timestamps associated with the input stream of flow records reflect their actual arrival rates. Unfortunately, softflowd cannot be configured to adhere to the relative timings of packets in a PCAP file. To overcome this issue, we decided to pair softflowd with tcpreplay [109], which is a tool designed for replaying PCAP files on a network interface while preserving the relative timing of packets based on their timestamps. This ensures an accurate reflection of the arrival timings of packets and the duration of the capture file. Tcpreplay is also installed within the Docker container, and a dummy network interface is created. This virtual network interface serves as a dedicated connection point for both programs, allowing softflowd to only observe the packets replayed by tcpreplay without any additional traffic. Tcpreplay is configured to replay the PCAP file on this dummy interface, while softflowd is configured to listen for packets on the same interface and create flow records based on the observed packets. A specifically crafted sleep function is employed to initiate tcpreplay precisely on the whole minute mark. This approach allows for a more straightforward comparison of the results with the labels of the dataset.

After tcpreplay finishes processing the PCAP file, the container waits for 60 seconds to make sure all flows have sufficient time to expire and be exported. Then, softflowd is shut down, and the exporter container is closed.

### Collector

In line with the architectural design outlined in Section 3.1.2, the collector of the system consists of a single instance. Its main functionality encompasses the conversion of flow records into a DSF compatible with the subsequent system components, as well as the ingestion of the converted data into the Apache Kafka messaging system.

Stream4Flow utilized IPFIXcol [79] for this purpose, which is an open-source flow collector. Based on its proven effectiveness within the Stream4Flow framework, this study chooses to utilize IPFIXcol as well. However, a successor to IPFIXcol, namely IPFIXcol2 [110], is currently available. Therefore, we decided to make use of IPFIXcol2 to ensure our prototype benefits from the improvements introduced in

version 2 of IPFIXcol.

Stream4Flow uses JSON as its DSF, and since JSON is a popular and easy-to-use format, it is also employed in this study. IPFIXcol2 is installed in a Docker container and it is started with a configuration provided in an Extensible Markup Language (XML) file. The configuration specifies that the collector should listen for TCP packets with flow data, convert the received flow data into JSON, and submit the converted data to a specific topic in the Kafka broker.

### Apache Kafka

In accordance with the architecture design outlined in Section 3.1.2, a single instance of a Kafka broker is employed and this broker provides multiple partitions to facilitate parallelism in the Spark analysis. The Kafka broker is deployed in a Docker container, utilizing the readily available image 'bitnami/kafka' from Docker Hub [111]. Configuration settings for the Kafka broker, including the number of partitions for each topic, are provided through environment variables. In this study, we opted to set the number of partitions to 2 to demonstrate parallelism within the Spark analysis application. Another noteworthy configuration is that Kafka is set to use the self-managed KRaft mode [87], eliminating the now-deprecated need for Apache ZooKeeper and thereby reducing system complexity. The creation of the topics must be handled by an external admin client, which is taken care of by the consumer component of the system.

### Consumer

The main function of the consumer is to retrieve results from the Kafka topic and store them in a file. Additionally, it serves as an admin client for creating the required topics within the Kafka broker. While Kafka does have a configuration setting that allows topics to be created if they do not exist, this feature did not work seamlessly with Spark, potentially resulting in system crashes. To ensure smooth operation, we chose to implement a solution where topics are first created using a Python application before components subscribe to these topics. While a separate Docker container could have been deployed for this purpose, integrating this functionality into the consumer container provided a more streamlined and manageable setup within the Docker environment.

Both functionalities are executed through Python applications, running consecutively within a Docker container. The first Python application is responsible for creating the topics in Kafka, while the second application subscribes to the topic with the results, retrieves the entries, and stores them in a Comma-Separated Values (CSV) file used for evaluating the system. The CSV format is chosen because it is a clearly structured and versatile data format. A Kafka consumer like this demonstrates the potential for real-world deployment scenarios, where it can trigger further actions based on the reported results.

#### Docker Compose

Docker Compose is employed to deploy these containers and manage their integration into a coherent system. It initiates all the containers and establishes the connections among them within Docker using a bridge network. Health checks are utilized to ensure the availability of the services and to appropriately manage the order of connecting the containers. It is important to note that the deployment of the Spark application is not included in this Docker Compose configuration and deployment because the Spark application must be submitted to a Spark cluster, which requires a different deployment method.

### 3.2.2 Spark

This section delves into the implementation and deployment of the Spark application. The initial part outlines the implementation of the Spark application, followed by its deployment on a Spark cluster.

#### Implementation of the Spark Application

The source code for a Spark application specifies the series of operations that can be executed on a Spark cluster to manipulate the data, such as filtering, mapping, aggregating, and joining. These operations are specified in a declarative manner, and the Spark deployment autonomously manages and optimizes their execution across the distributed computing cluster.

Before delving into an overview of the implemented operations, it is imperative to indicate the specific version of Spark employed in this study. The utilization of Spark version 3.5.0 is important because this release introduces watermark propagation among operators [112], enabling the use of multiple stateful operators in Spark Structured Streaming [113]. This feature is leveraged in the implementation to reduce the time required for obtaining the analysis results. Equally significant is the decision to utilize the Scala version of Spark, driven by its support for User-Defined Aggregate Functions (UDAFs), which is also used in the implementation. Notably, this feature is unavailable in the Python version of Spark. Further details on both features will be provided as they come up during this implementation overview:

Operation 1: The initial operation of the Spark application is to retrieve the streaming input data by subscribing to the Kafka topic containing the IPFIX records.

Operation 2: Following this, it parses the JSON-formatted data into a DataFrame with each IPFIX field residing in a separate column.

Operation 3: Watermarking is applied to the column containing the start time of flows to establish a threshold for late and out-of-order data, creating a point in time at which the results are considered final. It is important to

note that flows do not have a specific event time but rather a duration, bounded by a start and end time. This means that short-lived flows can terminate and arrive before long-lived flows, even if they started later on. To account for this, the watermark should allow for late arrivals of at least the maximum duration of flows, which is 30 seconds in this case, as configured in the exporter. An extra 15 seconds is added as a safety margin. To limit the scope of this study, we assumed this to be a generous margin for the arrival times of flow records. Future research can provide better insights into these arrival times to configure this margin more tightly.

To visualize this effect of watermarking in Spark on flow processing, Figure 3.3 and Figure 3.4 show two example flows with different durations and how different watermark lengths affect whether the flows are accepted for processing or dropped. In both examples, Flow 1 has a 30-second duration, while Flow 2 lasts for 10 seconds, with Flow 2 starting after Flow 1 but finishing earlier.

In Figure 3.3, with a 10-second watermark, Flow 1 is marked as "Too Late" and dropped, as the watermark has advanced beyond the starting time of Flow 1 after Flow 2 was already processed. In Figure 3.4, the watermark is extended to 20 seconds, which allows Flow 1, despite starting earlier than Flow 2, to be still considered "On-Time" and processed.

These figures demonstrate how adjusting the watermark affects whether late-arriving flows are accepted or dropped, particularly in these cases where flows can have different durations. This is why a 30-second watermark is used in our system, allowing flows with a duration of up to 30 seconds to still be processed as on-time. Additionally, we have added an extra safety margin to account for potential delays in arrivals and processing.

Operation 4: After adding the watermark, a new column is introduced, representing the duration of each flow. This duration is calculated based on their start and end times.

Operation 5: The flows are then grouped into 45-second windows with a sliding duration of 15 seconds, as part of the designed analysis approach outlined in Section 3.1.1.

Operation 6: Subsequently, the Shannon entropy values are calculated based on the distributions of the selected flow attributes within each window.

To perform this entropy calculation, the occurrence ratio of each attribute variation within these windows is required, as this serves as the attribute distribution. This necessitates both the total number of flows

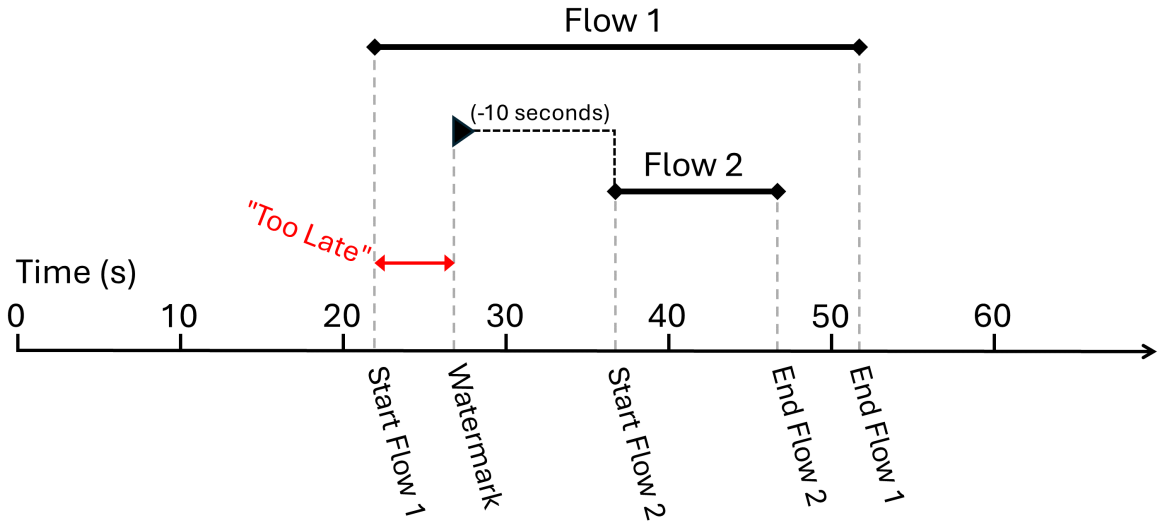



**Figure 3.3:** Visualization of a late arrival using two example flows and a 10-second watermark. Flow 1, with a 30-second duration, is marked as "Too Late" and dropped, as the watermark has advanced beyond the starting time of Flow 1 after the shorter Flow 2, with a 10-second duration, was already processed.

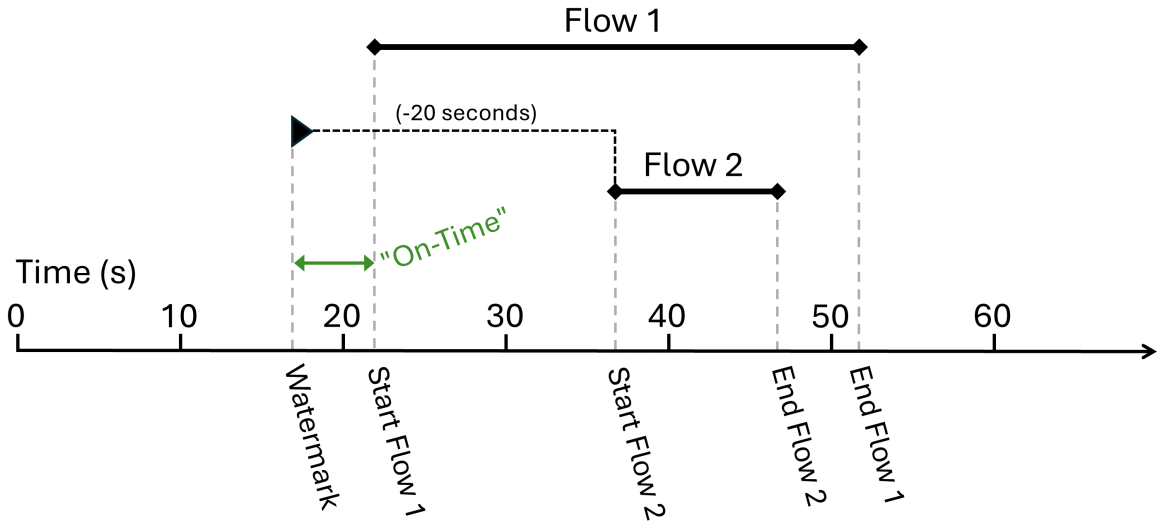



**Figure 3.4:** Visualization of an on-time arrival using two example flows and a 20-second watermark. Flow 1, with a 30-second duration, is marked as 'On-Time' and processed, despite starting before the shorter Flow 2, with a 10-second duration, that was already processed, as the extended watermark allows for its late arrival.

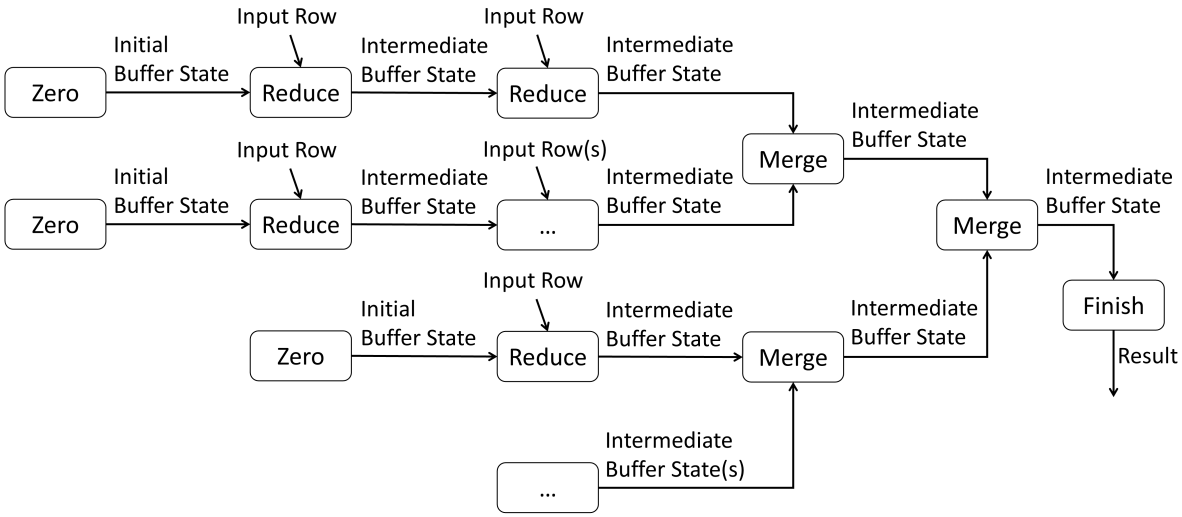



**Figure 3.5:** Overview of UDAF Function Interactions

and the count for each attribute variation within each window. Unfortunately, Spark lacks a built-in function for either directly calculating the entropy based on the windowed flow attributes or for calculating their occurrence ratios.

Initially, a potential approach involved combining multiple streams. One stream groups solely based on the start time to count the total number of flows within each window, while other streams group based on both the start time and a specific attribute column. This additional grouping allows for counting the occurrence of each variant of that attribute within the windows as well. Subsequently, these streams could be merged using a stream-stream join on matching windows to get data representing the attribute distributions. However, this approach required joining the streams, and due to a bug [114], the watermark propagation does not function correctly during stream-stream joins, making it impossible without introducing significant delays into the analysis. Consequently, another solution was devised, involving the implementation of a UDAF for the entropy calculation.

Implementing a UDAF requires four processing functions and two encoders. An overview of how these functions are combined into a UDAF can be seen in Figure 3.5. We implemented the following four functions to define our novel aggregation process for calculating entropy:

- **Zero** - In the aggregation process, each partition maintains an individual buffer state to facilitate distributed processing. This function initializes the buffer state used for processing a single partition. In the entropy calculation, the buffer stores the occurrence counters for each observed attribute variation. We implemented

this functionality using a HashMap for its ts efficiency in quickly retrieving and updating values.

- **Reduce** - The reduce function is responsible for processing data rows within each partition to update the partition's buffer state. For the entropy calculation, we implemented this function to track the occurrences of attribute variations by updating the counters in the HashMap. If an attribute variation is encountered for the first time, a new entry is created with a counter set to 1. For variations with an existing entry, the counter is incremented by 1.
- **Merge** - The merge function plays an important role in the aggregation process by combining results from different partitions, which is a crucial step in distributed computing for handling the aggregation of partial results. In the entropy calculation, we implemented this function to merge the HashMaps of two buffer states. If an attribute variation is present in only one buffer, the counter is taken as it is. If the variation is present in both buffers, the counters are summed. This merging continues until a single buffer remains, resulting in the combined occurrence counters from all partitions. Our implementation utilizes the efficient mergeByKey-With function from scala-collection-contrib [115], a module providing user-contributed additions to the Scala collections library.
- **Finish** - The finish function represents the finalization step in the aggregation process. This function takes the fully aggregated buffer state and produces the final result of the aggregation. This is where the actual entropy calculation occurs in our implementation. We programmed the function to calculate the occurrence ratios of the attribute variations and then compute the entropy value. The result is the Shannon entropy for the analyzed attribute within the considered window.

In addition to these functions, two encoders for the UDAF must also be defined. For distributed computing, Spark utilizes encoders to optimize communication between its nodes. These encoders allow Spark to serialize the intermediate states and final results of the UDAF into a compact binary format, which improves efficiency in data transfer. Spark has some built-in encoders tailored towards common data types and structures. These specific encoders offer the best performance. However, in cases where these specific encoders are not available, such as for the intermediate state HashMaps, Kryo serialization serves as a more general-purpose alternative. Unlike Java's generic serialization, Kryo serialization offers greater speed [116], making it a preferred choice for transmitting objects efficiently. Thus, we employed Kryo seri-

alization to encode the intermediate buffer, while we encode the output value with the built-in Scala Double encoder.

A notable observation regarding this implementation is that the buffer can become exceptionally large when dealing with a substantial number of attribute variations. For most of the considered flow attributes, the number of variations is already limited within a safe range by definition. However, this issue may still arise, particularly concerning the number of addresses and flow size variants. In such scenarios, solutions like address truncation or size bucketing can be employed. However, each of these introduces a trade-off between buffer size and the ability to distinguish certain variants. Although these solutions are not implemented in this study, exploring the effects of this trade-off can be an interesting topic for future research.

The aggregation to calculate entropy is performed on each of the analyzed attribute columns, resulting in the entropy-based features of the designed analysis approach. Based on these features, the modified Z-score is calculated to conduct the detection.

This is where the support for multiple stateful operators comes into play. Since grouping the entropy values over a broader time window and aggregating them into a modified Z-score constitutes a second stateful operation.

In versions prior to Spark 3.5.0, a sequence of operations like this was not supported. Consequently, intermediate results had to be sent back to Kafka and re-consumed for processing as a second stream. This approach introduced additional delays, as the second stream required an extra layer of watermarking based on the results of the first stream, rather than the input for the first stream.

Fortunately, Spark 3.5.0 introduced support for multiple stateful operators with watermark propagation [112, 113]. This allows for linking these operations into a single stream, enabling the watermarking of the second stateful operation to expire based on the input of the first stream, eliminating these additional delays.

Operation 7: Following the analysis design from Section 3.1.1, the entropy values are grouped in windows of two hours, serving as data points representing expected normal behavior for the calculation of the modified Z-score.

Operation 8: The grouped data is then aggregated to obtain the information required for calculating the modified Z-score. Specifically, it retrieves the latest 45-second window present within the 2-hour window. And for each feature, it acquires the entropy value associated with this latest window,

the median of entropy values within the 2-hour window, and a list containing all entropy values in the 2-hour window. The size of this list is limited by the number of 45-second windows that fit within the 2-hour window. This limitation is fixed and ensures the retrieval of a modest amount of data, which is an important aspect to guarantee safe processing on a single node, as the entire list gets consolidated into a single row. The list is obtained to calculate the MAD later on with a user-defined function, as Spark does not have a built-in function for this purpose.

Operation 9: A filter is applied to retain only the 2-hours windows where the end of the latest 45-second window aligns with the end of the 2-hour window. This ensures that the modified Z-score calculation is only applied to windows where the considered 45-second window is actually present.

Operation 10: Subsequently, user-defined functions are employed to calculate the MAD and, using the MAD, calculate the modified Z-score for each feature.

An issue arises when the MAD equals 0, causing a division by 0 in the modified Z-score calculation. However, in this study, this is not dealt with in the analysis, and the system reports Not a Number (NaN) when encountered. Strategies to address this issue can be considered in future iterations of the system.

Operation 11: In final versions of the system, the modified Z-scores can be mapped to booleans based on the selected thresholds, resulting in a clear yes or no detection. However, as part of this study, the system is evaluated with various thresholds. Therefore, this version of the system reports the modified Z-score values, facilitating the comparison of different threshold values during the evaluation process.

The column containing the 45-second window information and the columns holding the modified Z-score values are selected and parsed into the JSON format.

Operation 12: The resulting JSON is then written back to Kafka, to a topic specifically designated for the results.

### Deployment on a Spark Cluster

As mentioned, the deployment of the Spark application involves a different approach that sets it apart from the Dockerized components. Unlike the latter, the Spark application is deployed on a Spark cluster. This intentional separation is driven by the unique requirements of Spark, demanding a specific deployment to demonstrate its distributed computing capabilities.

Initially, the idea was to deploy the application on a Spark cluster managed by Kubernetes, using a local instance of Minikube. However, due to the extensive configurations involved, it was decided to opt for a simpler approach. Therefore, Spark's standalone mode [117] was chosen, which utilizes Spark's native cluster manager that requires minimal configuration, making it a better fit within the workload of this study.

Setting up a Spark cluster with this mode requires creating a standalone master server and attaching one or more worker instances to it. In this study, a master server with two worker instances is used, and these are all running on the same machine. Both worker instances are configured with 5GB of Random-Access Memory (RAM) and 2 Central Processing Unit (CPU) cores. The employment of two worker instances facilitates the demonstration of distributed processing, as these workers could also be located on different physical machines.

Once the cluster is in place, it is ready to receive and execute the Spark application. However, before the Scala application can be submitted to the cluster, it needs to be compiled. To ensure a more self-contained and easily distributable package, it is compiled into a fat Java ARchive (JAR) file containing all dependencies. In this study, the Scala Build Tool (SBT) [118] is used to compile the application, with the sbt-assembly plugin [119] used for creating the fat JAR. This JAR can then be submitted to the cluster master, which manages its execution across the cluster.

Various configuration settings are specified during submission, including the deploy mode, checkpoint location, number of partitions, and the amount of RAM used by the executors. The deploy mode of the Spark application determines whether the driver program runs locally on the client used for submission or remotely on one of the nodes within the cluster. In this study, the deploy mode is set to cluster, ensuring that the entire application operates within the cluster.

The checkpoint location is crucial for Spark, as it needs to track streaming states to recover from failures and to enable stateful processing by maintaining states across batches. This is achieved through checkpointing, and therefore, a checkpoint location must be specified. To ensure accessibility by all nodes, the checkpoint location is typically on a distributed file system such as the Hadoop Distributed File System (HDFS) [120] or Amazon's Simple Storage Service (S3) [121]. However, within the scope of this study, a simpler and more accessible solution can be used that still provides sufficient functionality. Since the whole cluster operates on one physical machine, a local directory is accessible to the entire cluster. This directory can be easily configured as the checkpoint location, which is why this location is used in this study.

Additionally, the Spark application is configured to use 2 partitions, aligning with both the number of partitions in Kafka and the number of worker instances.

In this study, each worker instance is allocated 5 Gigabyte (GB) of RAM. Both instances need to run an executor, but one of them must also accommodate the driver program of the application. This driver program is configured to use the default amount of memory, which is 1 GB. To account for this, the executors are configured

to use only 4 GB of RAM, ensuring enough RAM for appropriate memory allocation.

After deploying the Spark application on the cluster, it can be monitored and managed through Spark's web interface.

## 3.3 System Evaluation

With a comprehensive understanding of the relevant evaluation metrics in place, as we examined in Section 2.5, the next step is to formulate an evaluation plan to answer the research questions. In this section, we present our evaluation plan to assess the effectiveness of the recommendation system by addressing RQ1, RQ2, and RQ3 of this study.

### 3.3.1 Evaluation Plan

First, it is important to obtain a representative dataset that reflects the class proportions as observed in real-world situations, encompassing both DDoS attacks and benign traffic.

Subsequently, this dataset can be used to conduct the evaluation. The first RQ to address is RQ1, which is planned out with the following objective, methodology, and findings:

RQ1 **Detection Performance of Each Feature**

**Objective:**
Evaluate the effectiveness of every implemented feature in detecting each attack type within the dataset and identify potential underperforming features.

**Methodology:**
The performance of each feature in detecting each attack type is assessed using the AUPRC. This metric provides insight into the feature's ability to differentiate between purely benign intervals and intervals containing attack traffic. Based on the AUPRC values, a selection of only well-performing features can be made. The focus is on identifying features that are not effective in detecting attack types in general. These features with lower AUPRC values, indicating poor performance, are excluded from the selection to enhance the overall detection performance.

The evaluation then advances to address RQ2 and RQ3, which requires us to determine specific cost scenarios for undetected attacks and falsely retained traffic. However, conducting an in-depth cost analysis is beyond the scope of this study, as these costs may also vary greatly based on user preferences. For instance, frequent false alarms increase storage requirements, while undetected attacks reduce the potential for forensics analysis. Therefore, we decided to assess the system based on the following three general scenarios:

1. **Equal Costs** - A situation where the costs of undetected attacks and falsely retained traffic are equal.

2. **Higher Costs for Undetected Attacks** - A situation where the costs of undetected attacks is 10 times higher than that of falsely retained traffic.

3. **Higher Costs for Falsely Retained Traffic** - A situation where the costs of falsely retained traffic is 10 times higher than that of undetected attacks.

While this approach does not delve into a detailed cost analysis, it aims to offer a general indication of the system's expected performance for various conditions related to falsely retained traffic and undetected attacks.

For addressing RQ2, only the features that remain after answering RQ1 are considered. The plan to answer RQ2 is outlined with the following objective, methodology, and findings:

RQ2 **Optimal Sensitivity Configuration for Various Cost Scenarios**

**Objective:**
Determine the optimal sensitivity configuration for various cost scenarios related to falsely retained traffic and undetected attacks.

**Methodology:**
An analysis is conducted with a dataset that includes multiple attack types. The generalized F-score will be employed to identify the optimal operating point for each scenario. For each feature, the specific $F_\beta$-score for every cost scenario is calculated across the threshold spectrum. The point with the maximum score indicates the optimal threshold for that feature in the considered cost scenario. These results reveal the optimal sensitivity configuration for each feature in each cost scenario. This information is crucial for maximizing the detection performance while minimizing costs associated with falsely retained traffic and undetected attacks.

To address RQ3, we need to provide recommendations based on the combined detection of the features and then act on them to assess retention performance and the reduction of the storage size that can be achieved. Therefore, we decided to use an OR configuration, where we store traffic when one or more features exceed their thresholds. In this configuration, if at least one of the features triggers a detection, the interval is marked as suspected of containing attack traffic. The plan to answer RQ3 is outlined with the following objective, methodology, and findings:

RQ3 **Retention Performance and Potential Storage Reduction for Various Cost Scenarios**

**Objective:**
Assess the retention performance and potential reduction in storage size that can be achieved by utilizing the recommendation system with optimal sensitivity configurations for detecting DDoS attacks in various cost scenarios.

**Methodology:**
The system's retention performance is evaluated by generating recommendations based on the identified optimal sensitivity configurations for each cost scenario, and calculating their precision and recall values. The recall percentage indicates the portion of attack traffic that is correctly retained, while the

precision percentage indicates the portion of all retained data that actually belongs to attack traffic.

The storage reduction for each cost scenario is obtained by calculating the ratio between the initial size of the dataset and the size when only the intervals marked as containing attack traffic are retained. These represent storage sizes in the case where all network traffic were captured and stored (i.e., no recommendation system in place) and the storage size if the traces for non-suspect intervals are immediately discarded.

This evaluation indicates the retention performance and potential reduction in storage size that can be achieved across various cost scenarios when employing the recommendation system.

When the entire evaluation has been addressed, we can discuss our results and conclude by answering the Main RQ of our study.

## 3.4 Dataset

This section provides an overview of the dataset utilized in this study, encompassing both its requirements and acquisition process.

### 3.4.1 Dataset Requirements

The dataset must meet specific criteria to be suitable for this study. First of all, it must be in the PCAP format, which captures packets along with their corresponding timestamps as observed on the network interface. This format enables the creation of flows as input for the system using the specified export configuration, and it also facilitates the comparison of storage sizes, both with and without the system in use, for effective evaluation of the system's contribution.

Additionally, the dataset should include traffic related to various network attack types, particularly within the DDoS category, along with corresponding ground truth labels. This facilitates effective evaluation of the system based on its detection performance across various attack scenarios.

Furthermore, the dataset should reflect realistic proportions of normal network traffic and instances of attacks, encompassing intervals of solely benign traffic and intervals with the addition of traffic from ongoing attacks. Without these distinctive intervals, the system's retention recommendations would be irrelevant anyway. Maintaining realistic proportions facilitates evaluation of the system based on data that represents the traffic encountered in a real-world network environment, thereby enhancing the significance of evaluation results and reducing bias caused by skewed representation in the dataset.

### 3.4.2 Dataset Acquisition

While datasets from other studies are available online, they appeared unsuitable for evaluating our system due to their excessive focus on attacks. This led to an unrealistic overrepresentation of attack traffic and a deficiency of benign intervals. For instance, in the dataset created by Sharafaldin et al. [122], various types of attacks occur in rapid succession, and the dataset does not include any prolonged intervals without attacks. Consequently, relying on such datasets would likely skew the evaluation of our system's performance.

To address this limitation, we opted to create our own dataset. Fortunately, we were able to obtain a dataset in the PCAP format from another study [123] named ISCXIDS2012 [124], that contains 24 hours of benign labeled traffic. This benign dataset exclusively comprises benign traffic, making it free from the presence of any attack traffic. Therefore, this dataset can serve as a solid foundation in the form of background traffic, providing a realistic representation of the benign traffic encountered in real-world networks.

Since it lacked any instances of attacks, we opted to incorporate self-generated attacks into this dataset. This approach allows us to control the frequency, duration, and therefore overall presence of attack traffic in our dataset. We based these characteristics of our self-generated attacks on the DDoS characteristics from the latest extensive report on DDoS attacks by the NBIP [6], to make our attacks representative of those observed in real-world scenarios. The attack traffic was generated within an isolated simulation environment and subsequently merged with the background traffic. Acknowledging that this method introduces some unrealistic artifacts, since the background traffic does not precisely mimic its behavior during an actual attack as they did not truly coexist, we consider this approach reasonable within the limited time frame of this study.

This combined dataset allows us to create a PCAP encompassing both normal traffic and relevant attacks in realistic proportions along with ground truth labels, thereby enabling a comprehensive evaluation of our system. In the following subsections, we will provide an overview of the background traffic in more detail, delve into the generation of the attack traffic, and explain how attack and background traffic are merged and labeled. The scripts used to simulate DDoS attack traffic to create the synthetic dataset are publicly available in the following GitLab repository, hosted by the University of Twente:
`https://gitlab.utwente.nl/BasBleijerveld/scalable-ddos-traffic-retention/-/tree/main/dataset`

### 3.4.3 Benign Background Traffic

The ISCXIDS2012 dataset [124] comprising benign background traffic was obtained from the research conducted by Shiravi et al. [123]. The authors employed a systematic approach to generate their synthetic dataset based on profiles containing

abstract representations of events and behaviors observed on the network. These profiles were composed based on the network behavior of users and servers observed within their network of the Information Security Centre of Excellence (ISCX) at the University of New Brunswick.

For the generation of benign traffic, the authors created profiles for the following selection of protocols: HTTP, Simple Mail Transfer Protocol (SMTP), Post Office Protocol 3 (POP3), Internet Message Access Protocol (IMAP), SSH, and File Transfer Protocol (FTP). Network traffic for other protocols, such as Network Basic Input Output System (NetBIOS) and DNS, was indirectly generated as a consequence of utilizing the aforementioned protocols and also due to layer 2 protocols. While the exact protocol composition of this dataset from 2012 may not fully reflect current internet trends, our analysis with the recommendation system focuses on underlying network behavior and related entropy values rather than specific protocols.

Shiravi et al. [123] deployed agents on devices within a physical testbed network to mimic user activity based on their created profiles. To ensure a more authentic representation and avoid deterministic behavior, these agents utilized pseudorandom noise based on the observed characteristics of the underlying protocol distributions. This approach facilitated the creation of a synthetic dataset that closely mirrored real-world network behaviors. Their dataset spanned multiple days, including some days with the addition of attacks. However, we only used the PCAP data from the first day, which solely consisted of benign traffic.

The underlying network architecture of the testbed environment used by Shiravi et al. [123] is shown in Figure 3.6. Their architecture consisted of 21 interconnected Windows workstations, divided into four different LANs to effectively build a truly interconnected network. Besides that, a fifth LAN housed servers responsible for web services, email, DNS resolution, and Network Address Translation (NAT). The NAT server acted as the gateway to the Internet for the entire network. The main server was responsible for hosting the network's website, providing email services, and acting as an internal name resolver. Additionally, a secondary server was responsible for internal ASP.NET applications. The main and NAT servers ran on Linux, while the secondary server operated on Windows Server. A sixth LAN provided the means to conduct non-disruptive monitoring and maintenance of workstations and servers, as the traffic to and from this LAN was not captured. A single switch was used to provide the necessary switching and also mirroring of traffic. All connections were explicitly set to 10 Megabits per second (Mbps) to reduce the probability of packets being dropped by both the switch and the capturing devices. An Ethernet tap was used to send the mirrored traffic to multiple devices, which were responsible for capturing the network traffic and creating the PCAP data, but they also facilitated intrusion detection and monitoring purposes.

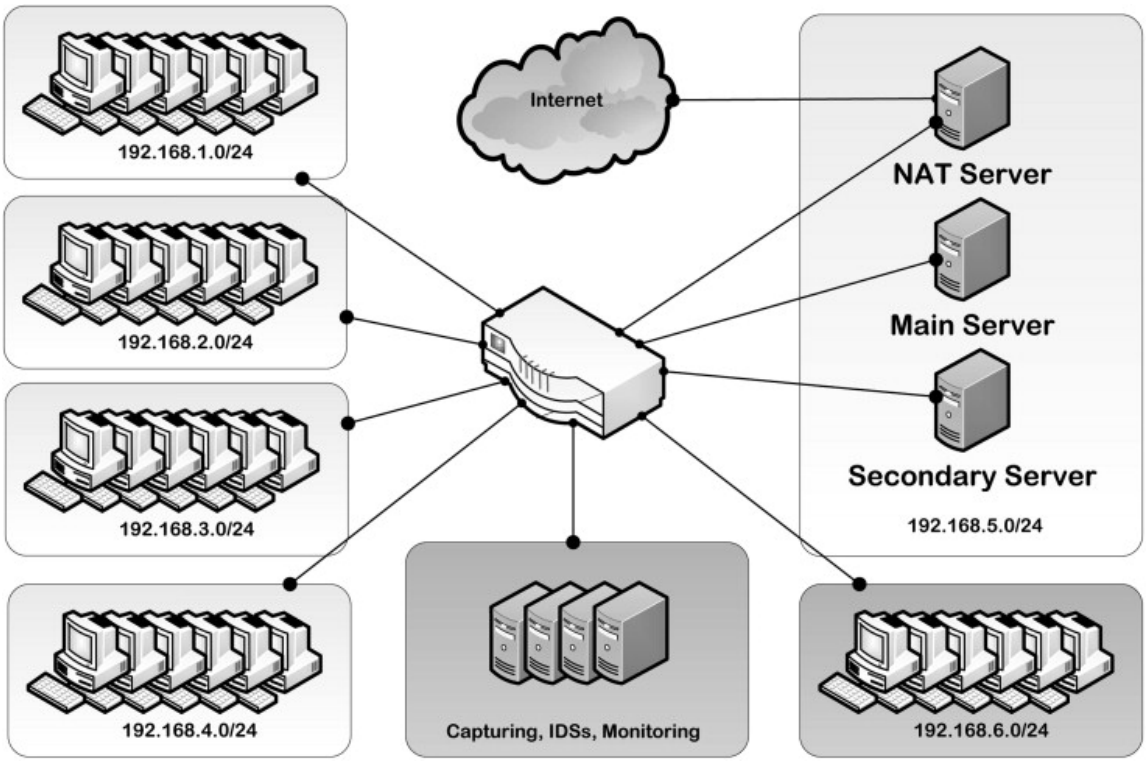



**Figure 3.6:** Network Architecture of the Testbed Environment [123]

### 3.4.4 Attack Traffic

In this subsection, we provide an overview of the selected types of attacks. The chosen attacks are intended to enable a comprehensive evaluation of our system, encompassing a range of attack variants commonly encountered in the field, each with slightly diverse characteristics. After this overview, we elaborate on how the traffic from these attacks is generated within our isolated simulation network.

#### Selection of Attacks

According to the latest extensive report on DDoS attacks by the NBIP [6], three main groups are classified: UDP flood, TCP flood, and UDP amplification. Therefore, we have included common attack variants that span across these groups and created the following selection:

- **UDP Flood**
  - *Spoofed UDP Flood Targeting a DNS Server (Port 53)* - Mimics the characteristics of a spoofed or botnet-driven attack targeting a DNS server running on port 53, which is the default port for DNS traffic. The attack packets use random source addresses and ports while having a fixed payload size.
  - *Spoofed UDP Flood Targeting Random Ports* - Mimics the characteristics of a spoofed or botnet-driven attack targeting random ports of the

victim, mirroring the unpredictable nature of attacks directed indiscriminately across various ports. Like the previous variant, attack packets use random source addresses and ports while having a fixed payload size.

- **TCP Flood**
  - *Spoofed TCP SYN Flood Targeting a Web Server (Port 80)* - Mimics the characteristics of a spoofed or botnet-driven attack targeting a web server operating on port 80, which is the default port for HTTP traffic. In this variant, packets with the TCP SYN flag are used to overwhelm the web server with TCP connection requests. The attack packets use random source addresses and ports and do not include any payload.
  - *Single Source TCP SYN Flood Targeting a Web Server (Port 80)* - Mimics the characteristics of a non-spoofed and single-source attack targeting a web server operating on port 80, which is the default port for HTTP traffic. Like the previous variant, packets with the TCP SYN flag are used to overwhelm the web server with TCP connection requests. However, in this variant the source addresses are not spoofed so the attackers IP address is used, while the packets still originate from random ports and do not include any payload. Since the source of the attack is not distributed, it is technically considered a DoS attack. The attacker is configured to not respond to the SYN-ACK packets it receives from the victim, with the aim of keeping the connection requests pending for as long as possible to exhaust the victim's available resources.
- **UDP Amplification**
  - *DNS Amplfication* - Mimics the characteristics of a DNS amplification attack, which is according to the NBIP report [6] the most prevalent form of DDoS attacks. In this variant, the attacker exploits vulnerable DNS servers to amplify the volume of traffic directed at the target. By sending small requests with spoofed source addresses of the victim to open DNS resolvers, the attacker can trick the servers into responding with large responses to the victim's IP address. The goal of this flood of amplified traffic is to overwhelm the victim. The attacker sends packets using random source ports and requests DNS records that result in the largest amplification factor possible, thereby maximizing the impact of the attack.

In addition to these volumetric attacks, it is also interesting to evaluate the detection performance against low and slow attacks, as these are characterized by their stealthy nature, designed to evade detection mechanisms and avoid triggering alarms. Therefore, an attack of this type, called Slowloris, has also been added to our dataset.

- **Low and Slow Attack**
  - *Slowloris (Single Source)* - In this low and slow attack, the attacker establishes multiple simultaneous connections to the target web server. In our case, a single source is used for its execution, so it is technically considered a DoS attack. Unlike the TCP SYN Flood, which only requests the connections, the Slowloris attack truly opens them. After which it starts sending partial HTTP requests at a slow pace to keep the connections open indefinitely. The attacker aims to exhaust the victim's resources, preventing the web server from properly handling legitimate requests.

To determine the frequency and duration of attacks generated for our dataset, we based our decisions on statistics from the NBIP report [6]. According to this report, the participants of the NaWas experienced an average of less than one attack per day. Since the obtained background traffic dataset covers 24 hours, we decided to allocate one attack per day.

Regarding the attack duration, the analysis in the NBIP report showed that 47.40% of attacks lasted less than 15 minutes and 40.20% lasted between 15 minutes and one hour. Based on this distribution, we opted for an attack duration of 20 minutes, which approximately corresponds to the average duration of the vast majority of attacks observed on the NaWas platform.

### Attack Traffic Generation

To generate representative attack traffic, we created an isolated simulation network to replicate attack scenarios that appear realistic from the victim's perspective. This makes it possible to capture solely traffic related to attacks, closely resembling real attack scenarios, which can be used to construct our dataset.

It is worth noting that our detection system we aim to evaluate does not rely on the Media Access Control (MAC) addresses. Therefore, we opted to not pursue a realistic MAC address configuration, in order to simplify the network setup and attack traffic generation process.

The selection of attacks can be categorized into three groups: spoofed, non-spoofed and reflection attacks. Each group utilizes a slightly different simulation network setup to generate the attacks, although the fundamental components remain similar across all of them. Thus, each network consists of at least two Docker [90] containers: an attacker and a victim, which are connected through an Open vSwitch (OVS) [125].

The victim serves as a replica of the main server present in the background traffic, as this server or some of its specific services are used as the target in the attack scenarios. Therefore, it is assigned the same IP address as the main server had in the background traffic, and it is also configured to provide a DNS server and web server. A simple deployment of dnsmasq [126] is used for the DNS server, and Apache HTTP Server [100] is employed as the web server to host a mock-up

webpage created with Lorem Generator [127]. Additionally, tcpdump [32] is used to capture the attack traffic as observed from the victim's perspective, and this data is stored in a PCAP file. It is important to create the attack traffic PCAP with the same snapshot length as the background traffic, otherwise tcpreplay [109] refuses to replay the merged files.

For the attackers, we created three different containers, each tailored to a distinct attack method. The first container is configured with hping3 [102], enabling the execution of UDP and TCP flood attacks. The second container is equipped with a Python tool to carry out a Slowloris attack [128]. For the third attacker, we created a custom Python script to craft and send spoofed DNS queries. This third container also hosts a dnsmasq [126] server that listens on multiple IP addresses, serving as various reflectors for the DNS amplification attack.

An OVS is used to facilitate the configuration of the link connected to the victim. Although the study of the background traffic [123] does not explicitly specify that the 10 Mbps links are symmetric, we assumed that they are. Therefore, in our simulation, the link to the victim is configured to be limited to 10 Mbps in both directions. Additionally, a 20 millisecond delay is applied to the link in both directions to prevent unrealistically fast speeds, since all components of the simulation network are running on the same physical machine. Furthermore, the background traffic employs an Maximum Transmission Unit (MTU) of 1500 bytes, which is commonly the default setting. It is also the default in our simulation network, and therefore, it does not require explicit configuration.

In the spoofed attack scenarios, a gateway container is introduced to enable the victim to establish routes to the broader Internet. This allows the victim to send replies in response to the spoofed attack packets it receives. Since the simulation network is isolated, the gateway is not connected to the real Internet. Therefore, the gateway is configured to act as the entire IP version 4 (IPv4) address space itself by responding to these packets. We considered this approach to be realistic, given the exhaustion of IPv4 addresses [129], making it likely that there are actual devices behind most of the IP addresses.

By using Bash scripts, we precisely defined and automated the setup and execution tasks for the six selected attack scenarios. Each script starts with creating and configuring the simulation network, followed by initiating the traffic capturing process on the victim, and then executing the attack in question. Afterwards, all components of the network are neatly removed again.

The following outlines a few notable configurations for some of the attack scenarios. Both UDP flood attacks are configured to utilize packets with additional payload to completely fill up the MTU size of 1500 bytes. The attack container of the non-spoofed TCP SYN flood is configured to block outgoing TCP RST packets, thereby preventing the attacker from responding to the SYN ACK packets it received from the victim. We determined through testing in our simulation environment that 1024 sockets proved to be sufficient to exhaust the victim's Apache HTTP Server [100]. Therefore, we execute the Slowloris attack using 1024 sockets, which makes the

Apache Server unavailable for additional connections.

For the DNS amplification attack there are several configurations that require explanation. In a Cloudflare blog [130], it was reported that on average, a DNS amplification attack utilized 7100 DNS reflector servers, generating an average bandwidth of around 3.4 Gigabits per second (Gbps). However, this level of intensity is too extreme for our simulation environment. Therefore, we scaled down the numbers to match the capacity of our simulation network. The number of reflectors was scaled down proportionally based on the ratio between our 10 Mbps link and the reported average bandwidth of 3.4 Gbps, which is 3481.6 Mbps. The resulting value is calculated as $7100 \cdot \frac{10}{3481.6}$, which equals approximately 20 reflector servers, and therefore this is the number of reflector servers we have used.

Another important configuration to mention is Extension Mechanisms for DNS (EDNS) [131], which plays a crucial role in DNS amplification attacks as it enables responses larger than the original limit of 512 bytes. We have opted to use an EDNS payload size of 4096 bytes in our attack scenario, aligning with the maximal answer size used by most EDNS implementations [132]. Records that result in responses larger than this limit are truncated and retried over TCP for the full answer, making them less attractive to attackers who rely on spoofing. Therefore, in our attack scenario, we do not request records that exceed this EDNS limit. Support for EDNS is signaled through a pseudo record type called OPT in the query, so this OPT record is added to the spoofed DNS queries that are crafted by our Python script. The large responses will result in packet fragmentation in our simulation network, due to the MTU size of only 1500 bytes.

The main culprit for DNS amplification right now is the ANY response [132], resulting in the largest amplification. Dropping full responses to ANY queries is proposed by Request for Comments (RFC) 8482 [133] to reduce the possible amplification. However, van der Toorn et al. [132] fear that as RFC 8482 gains more adoption, domains created for DDoS attacks will shift their efforts to a single record type to obtain a size nearly equal to their ANY response size. Likely, this record type will be TXT. Furthermore, TXT records are only a single record type and therefore easier to configure within our simulation environment than realistic ANY responses. For these reasons, TXT records are used in our DNS amplification attack scenario.

Our custom Python script requests a distinct TXT record from each DNS server address. The responses are all just below the maximum EDNS size, but do have slightly varying sizes. This mimics the behavior that the attacker requests the largest records available at each unique DNS server.

Using these methodologies, we successfully generated and captured network traffic for the various attack types. This attack traffic data now needs to be merged with the background traffic to create a dataset that can be used for the evaluation of our system.

### 3.4.5 Merging Process and Ground Truth Labels

The merging process first consists of a few steps to align the background and attack traffic PCAP files. To do this, the timestamps of the first packet in both the background and attack traffic PCAP files are extracted using the command-line version of Wireshark [37], called TShark. These timestamps serve as a reference point for subsequent adjustments. Next, based on these reference points, the timestamps of both the background and attack traffic PCAP files are adjusted to start at epoch (time 0) using the editcap tool, which is also part of the Wireshark suite. This initial adjustment ensures that both PCAP files share a common starting time. Subsequently, the timestamps of the attack traffic are adjusted again to start at an offset of 12 hours. This adjustment aligns the attack traffic with the midpoint of the 24-hour background traffic when merged. Once the timestamps are properly aligned, Wireshark's mergecap tool is employed to combine both PCAP files. This merging process creates a unified dataset that incorporates both the background and attack traffic.

For the ground truth labels, it is important to consider the entire duration of the attack PCAP file, which can be easily seen using Wireshark's capinfos tool. This is important because some traffic in the attack capture might be slightly delayed due to being part of the aftermath, potentially extending the presence of traffic associated with an attack. Consequently, the attack presence is labeled as starting from 12 hours after the beginning of the merged PCAP file and continues for the whole duration of the attack traffic PCAP file. This approach provides accurate time-based labels about the presence of attack traffic within the merged dataset.

# Chapter 4

# Results and Analysis

In this chapter, we present the results of our study, organized into four sections. First, we provide a visualization of the raw data to provide a general understanding of what the data looks like. Next, we delve into the detection performance of each feature, addressing RQ1. Following that, we determine the optimal sensitivity configuration for various cost scenarios related to falsely retained traffic and undetected attacks, addressing RQ2. Finally, we assess the retention performance and potential storage reduction for various cost scenarios, addressing RQ3. The code used for the evaluation and analysis is publicly available in the following GitLab repository, hosted by the University of Twente:
`https://gitlab.utwente.nl/BasBleijerveld/scalable-ddos-traffic-retention/-/tree/main/evaluation`

## 4.1 Visualisation of the Raw Data

First, it is helpful to provide an overview of the system's output to give a clear understanding of what the raw data looks like. The results consist of data points representing each feature's modified Z-score for each time window. Since we replayed network traffic from the dataset not at the same time of day, we used some preprocessing on the raw data to align it properly.

Given that there are six separate datasets, each corresponding to a different attack type, and eight features, this generates 48 figures. To avoid presenting an overwhelming amount of figures, we provide three diverse scatter plots to illustrate the general idea.

Each attack starts at 12:00 and lasts 20 minutes, but this may be slightly longer depending on the presence of any residual traffic in the aftermath of the attack. The modified Z-score values can change in the negative or positive direction, depending on whether the data distribution becomes more concentrated or dispersed, as indicated by the underlying entropy. These directional changes could be further examined in future research, such as by fingerprinting different attack types based

on the specific directions of their changes. However, in this study, we refrain from focusing on these directional changes and use the absolute values instead. This approach simplifies the analysis, as it only requires a single threshold value.

Figure 4.1, Figure 4.2, and Figure 4.3 depict these scatter plots, they display the plot with normal values on the left and the plot with absolute values on the right.

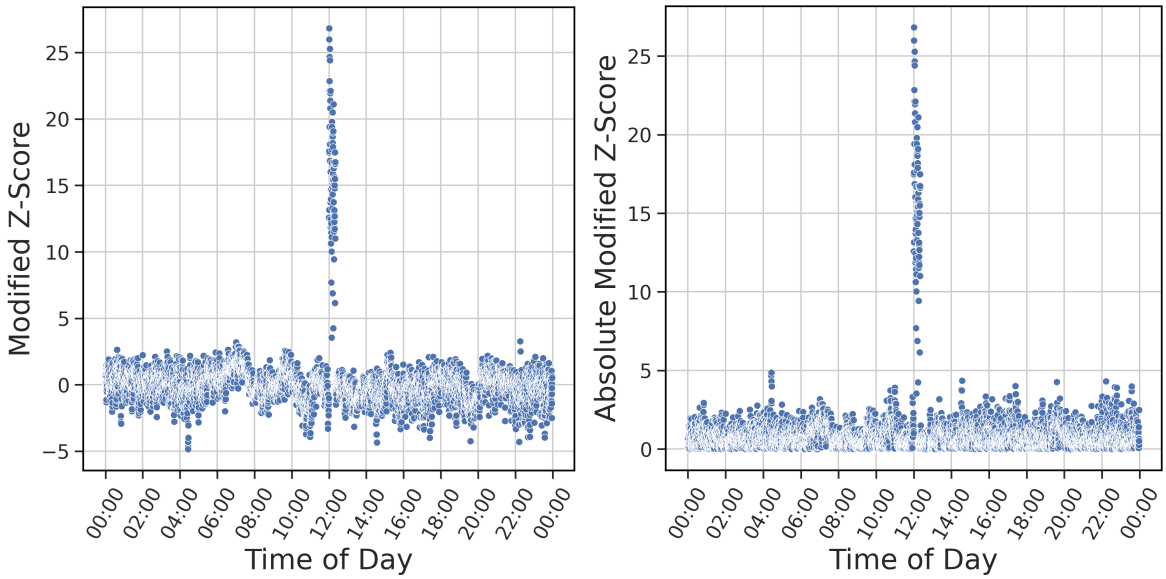



**Figure 4.1:** Modified Z-scores (Normal and Absolute) Showing Major Changes in the Source Address Entropy During a Spoofed UDP Flood Targeting a DNS Server (Port 53) at 12:00

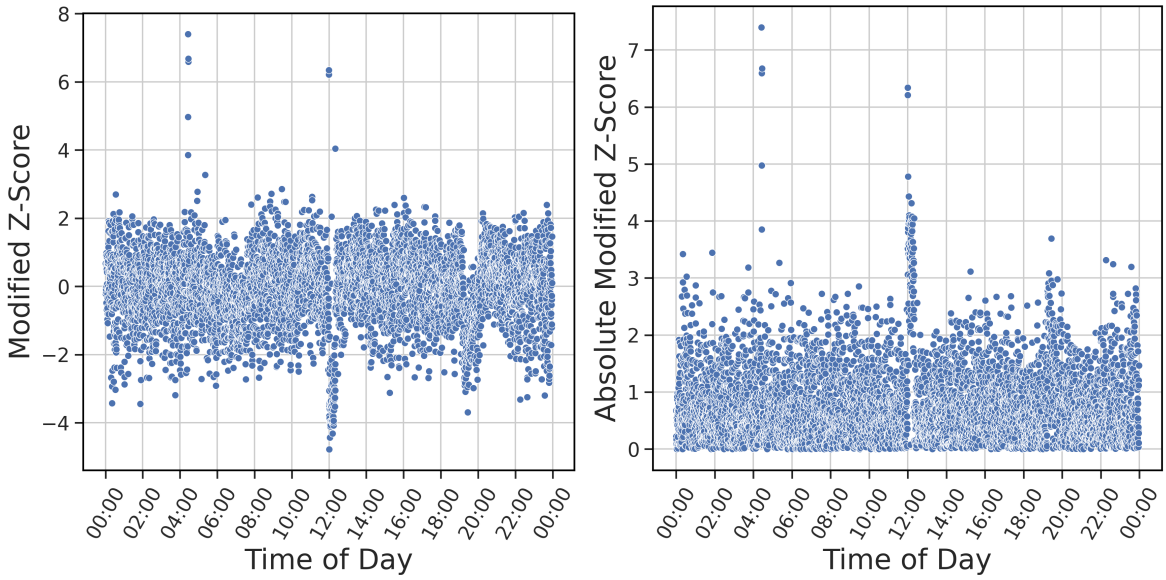



**Figure 4.2:** Modified Z-scores (Normal and Absolute) Showing Moderate Changes in the Destination Port Entropy During a Spoofed UDP Flood Targeting Random Ports at 12:00

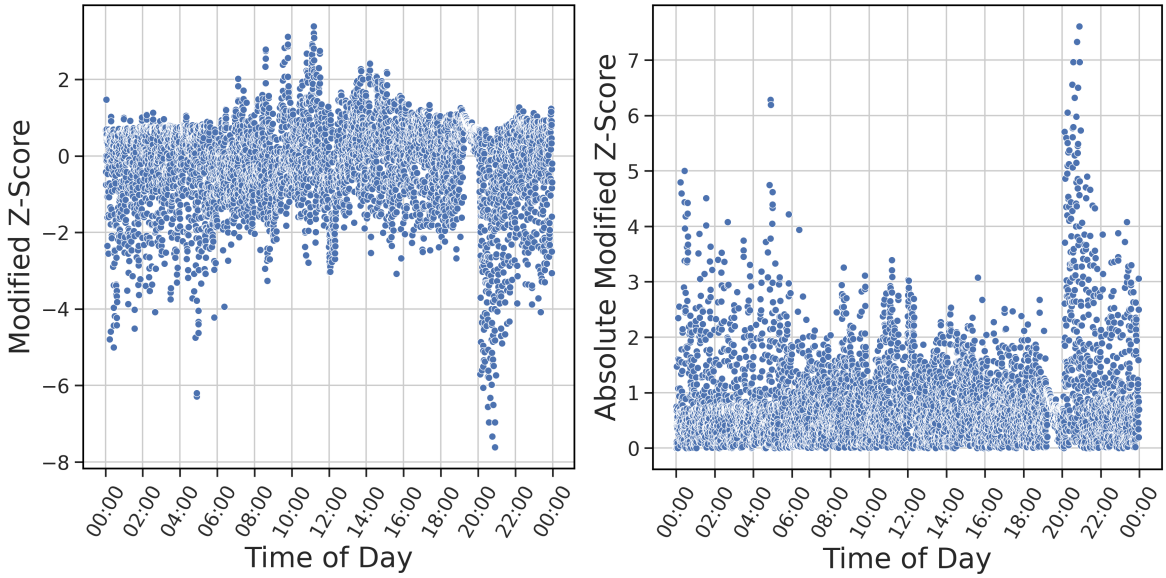



**Figure 4.3:** Modified Z-scores (Normal and Absolute) Showing Minor Changes in the Protocol Entropy During a Slowloris Attack (Single Source) at 12:00

Figure 4.1 shows a significant change in the source address entropy during the presence of spoofed UDP flood attack traffic targeting a DNS server. In Figure 4.2, a comparatively smaller change is depicted in the destination port entropy during the presence of spoofed UDP flood attack traffic targeting random ports. Furthermore, Figure 4.3 shows that no distinctive changes are visible in the protocol entropy during the presence of Slowloris attack traffic.

## 4.2 Detection Performance of Each Feature

In this section, we address RQ1 by evaluating the detection performance of each implemented feature on every attack type in the dataset to identify potential under-performing features. Therefore, it is essential to have the ground truth data available, which we derived from the start time and duration of the attack traffic. Subsequently, we generated a PR curve for every attack and feature combination, resulting in 48 PR curves. To avoid presenting an overwhelming amount of figures, we provide PR curves for the same three attack and feature pairs as before in Section 4.1, to illustrate the general idea.

Figure 4.4, Figure 4.5, and Figure 4.6 depict these PR curves and also show curves for a perfect classifier and for the baseline. The baseline is based on the ratio between intervals with attack traffic and intervals with only benign traffic. In the legend of the PR curve plots, the AUPRC value is shown. This value indicates the general performance of the feature in distinguishing between the attack and benign intervals.

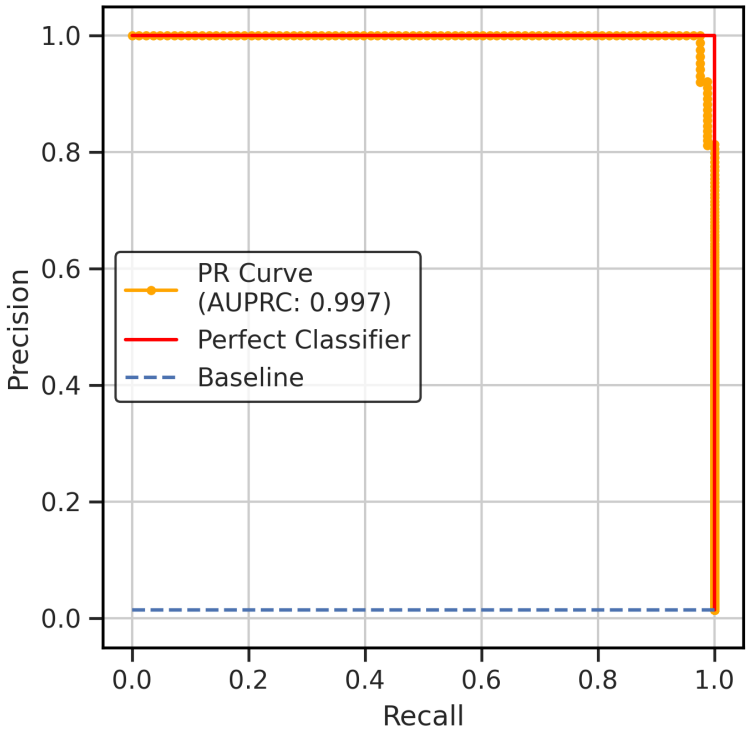



**Figure 4.4:** Precision-Recall (PR) Curve for the Source Address Entropy Feature on Detecting a Spoofed UDP Flood Targeting a DNS Server (Port 53)

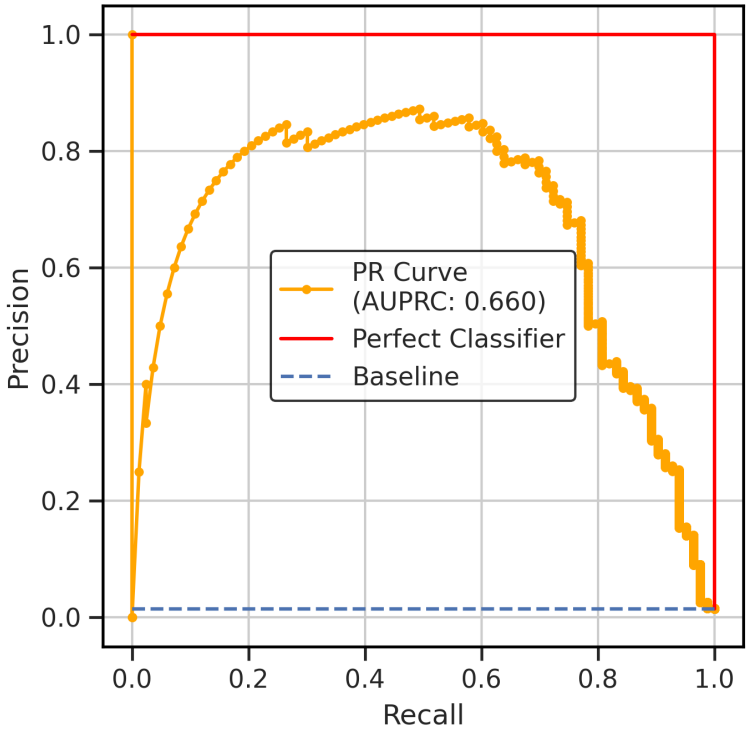



**Figure 4.5:** Precision-Recall (PR) Curve for the Destination Port Entropy Feature on Detecting a Spoofed UDP Flood Targeting Random Ports

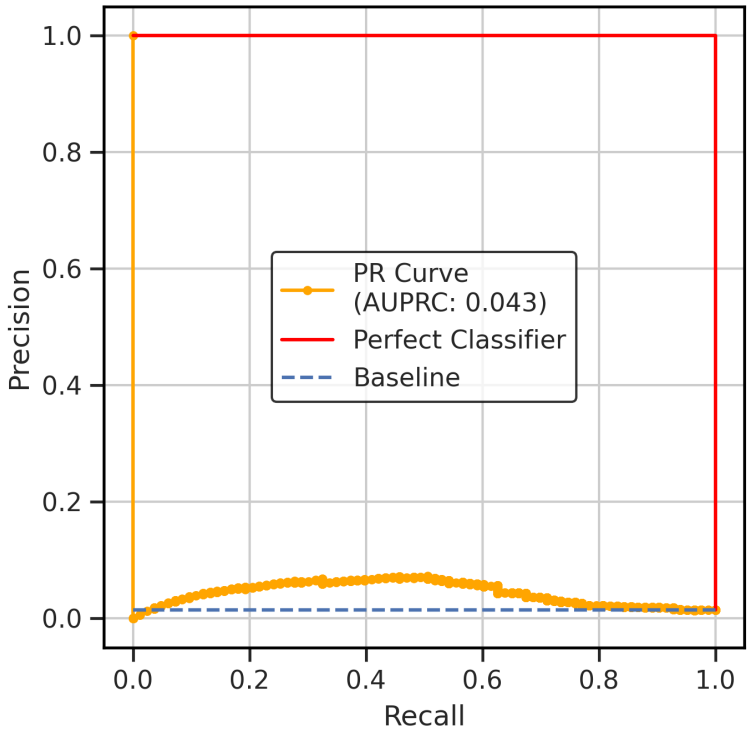



**Figure 4.6:** Precision-Recall (PR) Curve for the Protocol Entropy Feature on Detecting a Slowloris Attack (Single Source)

The AUPRC values of Figure 4.4, Figure 4.5 and Figure 4.6 quantitatively confirm what the corresponding scatter plots in Figure 4.1, Figure 4.2 and Figure 4.3 already illustrated. Features with a clearer distinction during attack traffic in the scatter plots generate PR curves closer to a perfect classifier, resulting in higher AUPRC values, which indicates they are better suited for use in detection.

These plots show the performance of only these three attack and feature pairs, which are chosen to display what diverse performances look like while avoiding an overwhelming number of PR curves. Thus, to display the performance of all 48 feature and attack combinations, we present their AUPRC values in a comprehensive overview using a heatmap, as shown in Figure 4.7.

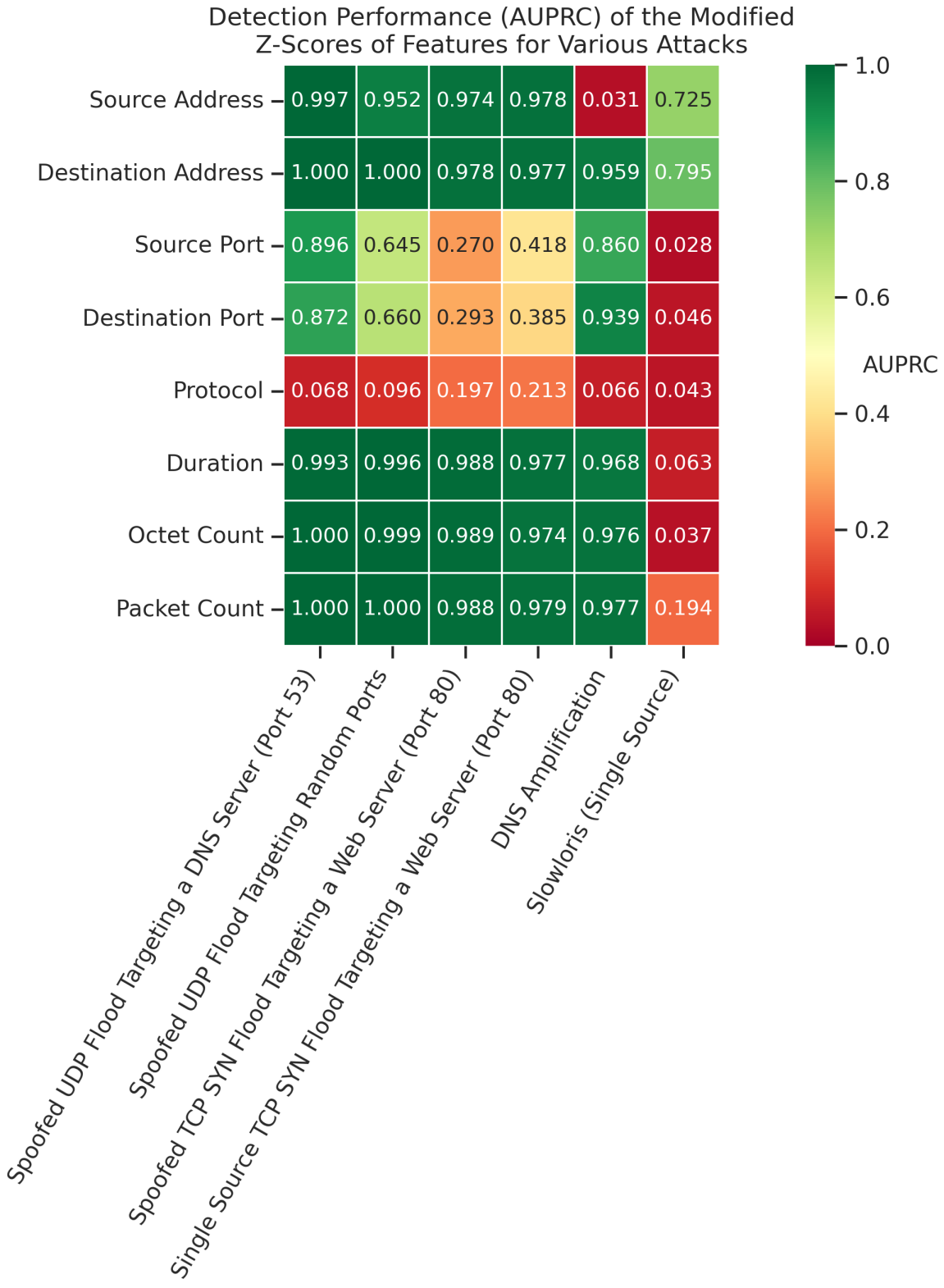



**Figure 4.7:** Heatmap Depicting the Detection Performance (Measured by AUPRC) of the Modified Z-scores of Each Feature for Various Attack Types

This heatmap analysis, as depicted in Figure 4.7, presents several findings. No-

tably, it reveals that the feature based on protocol entropy performs poorly in detecting every included attack in general. Additionally, the heatmap shows that almost every feature performs poorly in detecting the Slowloris attack, except for the features based on the source or destination address entropy.

Since we want to exclude features that underperform in general, we assessed their average detection performance. Figure 4.8 showcases a heatmap with the average detection performance of features across all included attack types. This heatmap is used to identify features that consistently underperform in detecting attacks.

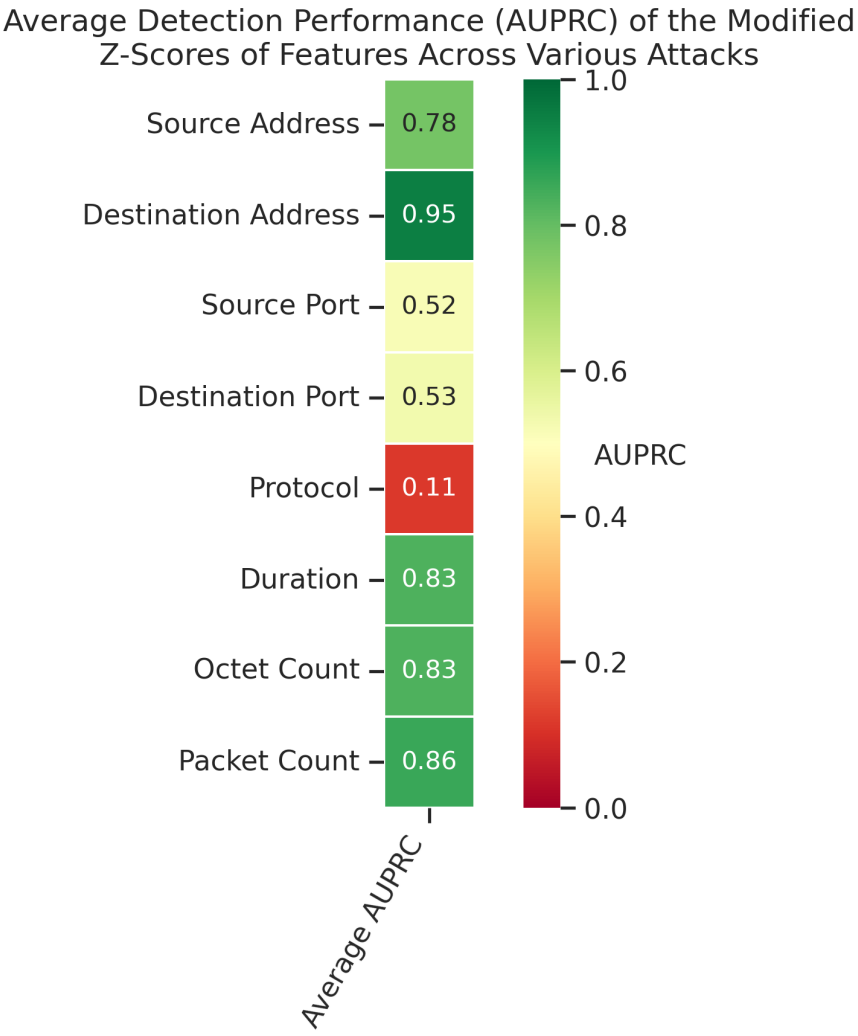



**Figure 4.8:** Heatmap Depicting the Average Detection Performance (Measured by AUPRC) of the Modified Z-scores of Each Feature Across the Various Attack Types

These results in Figure 4.8 reveal poor detection performance for features based on source port, destination port and protocol entropy. Therefore, we decided to exclude these three features to enhance the overall detection performance, thereby

answering RQ1.

## 4.3 Optimal Sensitivity Configuration for Various Cost Scenarios

In this section, we address RQ2 by determining the optimal sensitivity configuration for the three diverse cost scenarios related to costs of falsely retained traffic and undetected attacks, as defined in Section 3.3.1. The sensitivity configuration is optimized for each feature's performance on the whole dataset, encompassing all attack types included in this study.

Therefore, we first created the PR curve for each feature across the entire dataset. Subsequently, we used the generalized F-score to determine each feature's optimal point of operation depending on the cost scenario.

In the scenario where undetected attacks and falsely retained traffic have equal costs, we used a $\beta$ value of 1, i.e. the $F_1$-score. For the scenario where the cost of falsely retained traffic is 10 times that of undetected attacks, precision is given more weight and we used a $\beta$ value of 0.1, i.e. the $F_{0.1}$-score. When the cost of undetected attacks is considered 10 times that of falsely retained traffic, more weight is given to recall, so we used a $\beta$ value of 10, i.e. the $F_{10}$-score.

The optimal point of operation for each cost scenario is indicated by a specific threshold, along with its related performance, expressed in terms of precision and recall. These points can be visualized by marking them on the PR curve of each feature. As this looks quite similar for every feature, we only show it for two features to provide the general idea. The PR curves with these marked optimal points for the source address and octet count features are depicted in the left plots of Figure 4.9 and Figure 4.10, respectively. Furthermore, to visualize the precision, recall, and $F_\beta$-score values across the threshold spectrum, we have included the plots on the right. These plots display the curves depicting precision, recall, and $F_\beta$-scores against the threshold values, with vertical lines indicating the threshold values that result in the maximum $F_\beta$-scores. These thresholds represent the optimal sensitivity configurations for each of the considered cost scenarios.

Table 4.1 presents the optimal thresholds for every feature in each of the three cost scenarios ($F_1$-score, $F_{0.1}$-score, and $F_{10}$-score), thereby answering RQ2.

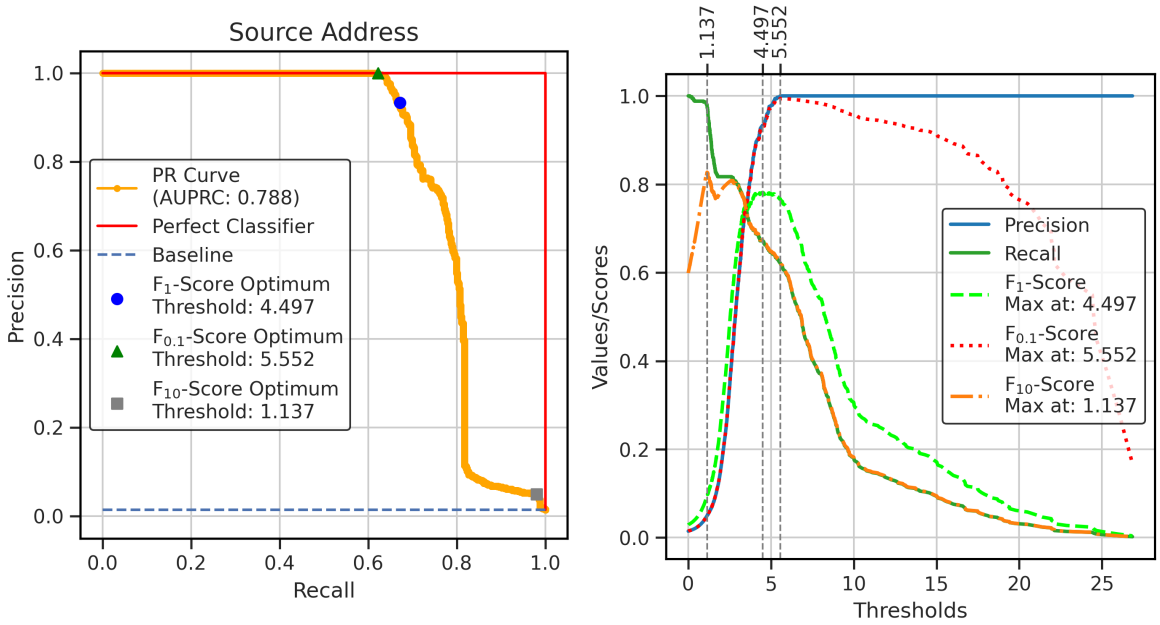



**Figure 4.9:** PR Curve and Threshold Performance for the Source Address Feature. **Left:** PR Curve with the Optimal Points of Operation for Each Cost Scenario. **Right:** Precision, Recall, and $F_\beta$-score Values Across the Threshold Spectrum, with Vertical Lines Indicating the Optimal Thresholds for a Maximum $F_\beta$-score.

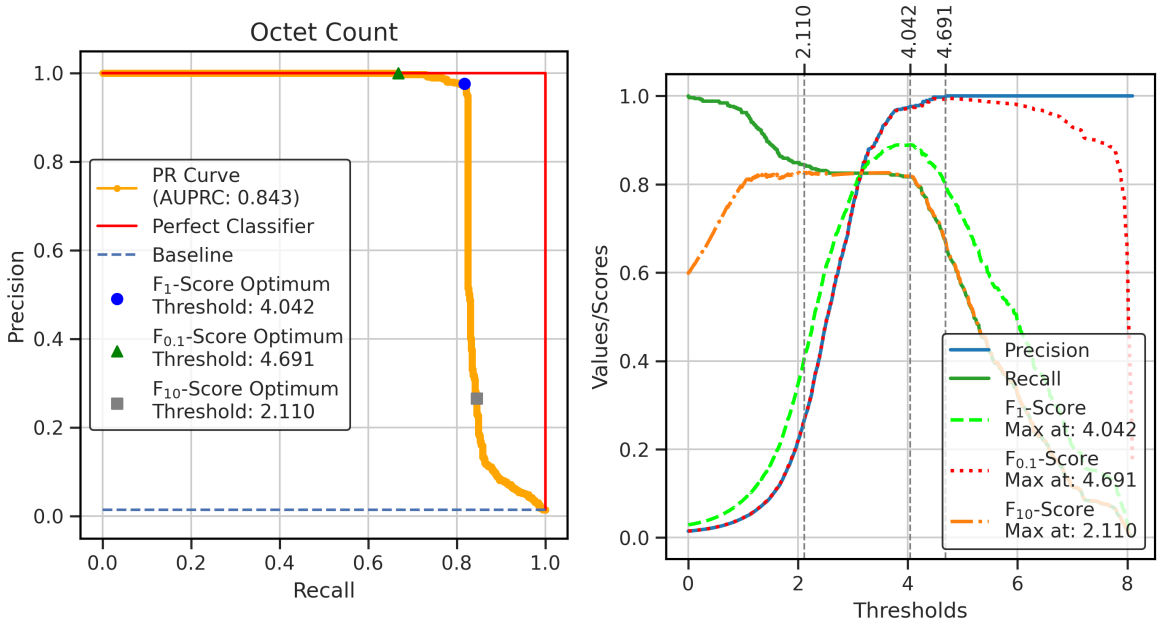



**Figure 4.10:** PR Curve and Threshold Performance for the Octet Count Feature.
**Left:** PR Curve with the Optimal Points of Operation for Each Cost Scenario.
**Right:** Precision, Recall, and $F_\beta$-score Values Across the Threshold Spectrum, with Vertical Lines Indicating the Optimal Thresholds for a Maximum $F_\beta$-score.

**Table 4.1:** Optimal Thresholds for Every Feature in Each of the Three Cost Scenarios ($F_1$-score, $F_{0.1}$-score, and $F_{10}$-score), i.e. the Optimal Sensitivity Configurations.

| | **Source Address** | **Destination Address** | **Duration** | **Octet Count** | **Packet Count** |
|---|---|---|---|---|---|
| **$F_1$-score** | 4.496752 | 5.553165 | 5.468874 | 4.042449 | 4.678176 |
| **$F_{0.1}$-score** | 5.551978 | 5.553165 | 5.468874 | 4.690727 | 5.962065 |
| **$F_{10}$-score** | 1.137378 | 2.954688 | 1.608295 | 2.110243 | 1.641807 |

## 4.4 Retention Performance and Potential Storage Reduction for Various Cost Scenarios

In this section, we address RQ3 by applying the optimal sensitivity configurations for each cost scenario, and then determining the retention performance and potential storage reduction that can be achieved. The retention performance metrics, precision and recall, are determined based on the produced recommendations compared to the ground truth. Storage reduction is assessed by applying the retention recommendations to the dataset and comparing the size of the retained data with the initial

size. We do this for each attack type separately and for the combined total dataset.

First, we consider the cost scenario in which both precision and recall are equally weighted ($F_1$-score). The achieved precision, recall, and reduction percentages are shown in Table 4.2, and these results are visualized in Figure 4.11.

**Table 4.2:** Precision, Recall, and Reduction Percentages for the $F_1$-score Scenario (rounded to 2 decimals)

| | **Precision %** | **Recall %** | **Reduction %** |
|---|---|---|---|
| **Spoofed UDP Flood Targeting a DNS Server (Port 53)** | 96.51 | 100.00 | 89.22 |
| **Spoofed UDP Flood Targeting Random Ports** | 90.22 | 100.00 | 87.08 |
| **Spoofed TCP SYN Flood Targeting a Web Server (Port 80)** | 94.32 | 98.81 | 85.39 |
| **Single Source TCP SYN Flood Targeting a Web Server (Port 80)** | 93.26 | 97.65 | 82.94 |
| **DNS Amplification** | 89.13 | 96.47 | 89.02 |
| **Slowloris (Single Source)** | 72.41 | 25.30 | 98.57 |
| **Total Dataset** | 91.39 | 86.48 | 88.47 |

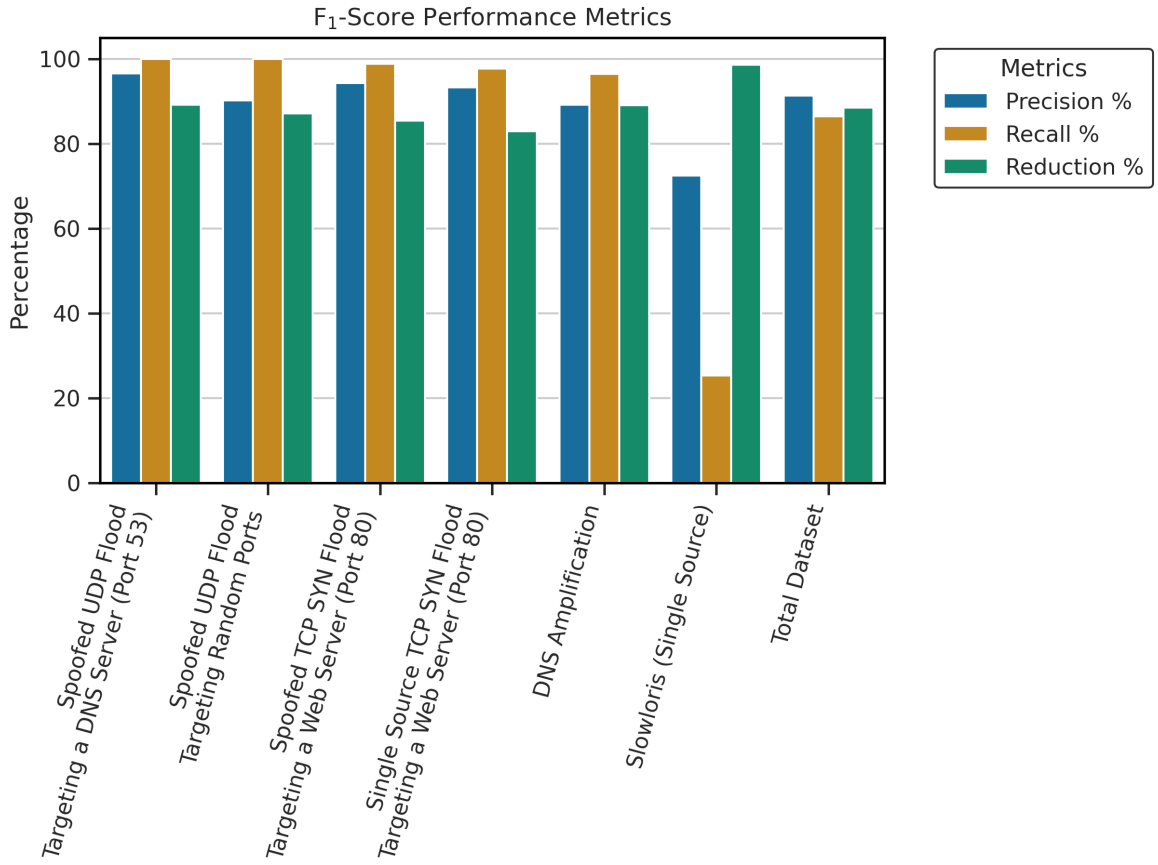



**Figure 4.11:** Bar Chart of Precision, Recall, and Reduction Percentages for the $F_1$-score Scenario

These results for the optimal $F_1$-score sensitivity configuration indicate a significant storage reduction across all attack types, each showing a reduction of over 80%. The system performs well in identifying and retaining relevant data for most attack types, achieving high precision and recall. However, the Slowloris attack type stands out with a notably low recall of 25.30% and a precision of 72.41%. These reflect the challenge of detecting this type of attack, which is designed to operate under the radar. For the combined total dataset, this sensitivity configuration results in a precision of 91.39%, a recall of 86.48%, and an overall data reduction of 88.47%. The results show that this configuration performs well on most attack types, but the low recall on the Slowloris attack indicates a clear weakness.

The results for the cost scenario where precision outweighs recall by a factor of ten ($F_{0.1}$-score) are shown in Table 4.3 and visualized in Figure 4.12.

**Table 4.3:** Precision, Recall, and Reduction Percentages for the $F_{0.1}$-score Scenario (rounded to 2 decimals)

| | **Precision %** | **Recall %** | **Reduction %** |
|---|---|---|---|
| **Spoofed UDP Flood Targeting a DNS Server (Port 53)** | 100.00 | 100.00 | 89.42 |
| **Spoofed UDP Flood Targeting Random Ports** | 100.00 | 100.00 | 87.48 |
| **Spoofed TCP SYN Flood Targeting a Web Server (Port 80)** | 100.00 | 98.81 | 85.60 |
| **Single Source TCP SYN Flood Targeting a Web Server (Port 80)** | 100.00 | 97.65 | 83.28 |
| **DNS Amplification** | 98.80 | 96.47 | 89.43 |
| **Slowloris (Single Source)** | 100.00 | 22.89 | 99.32 |
| **Total Dataset** | 99.77 | 86.08 | 88.85 |

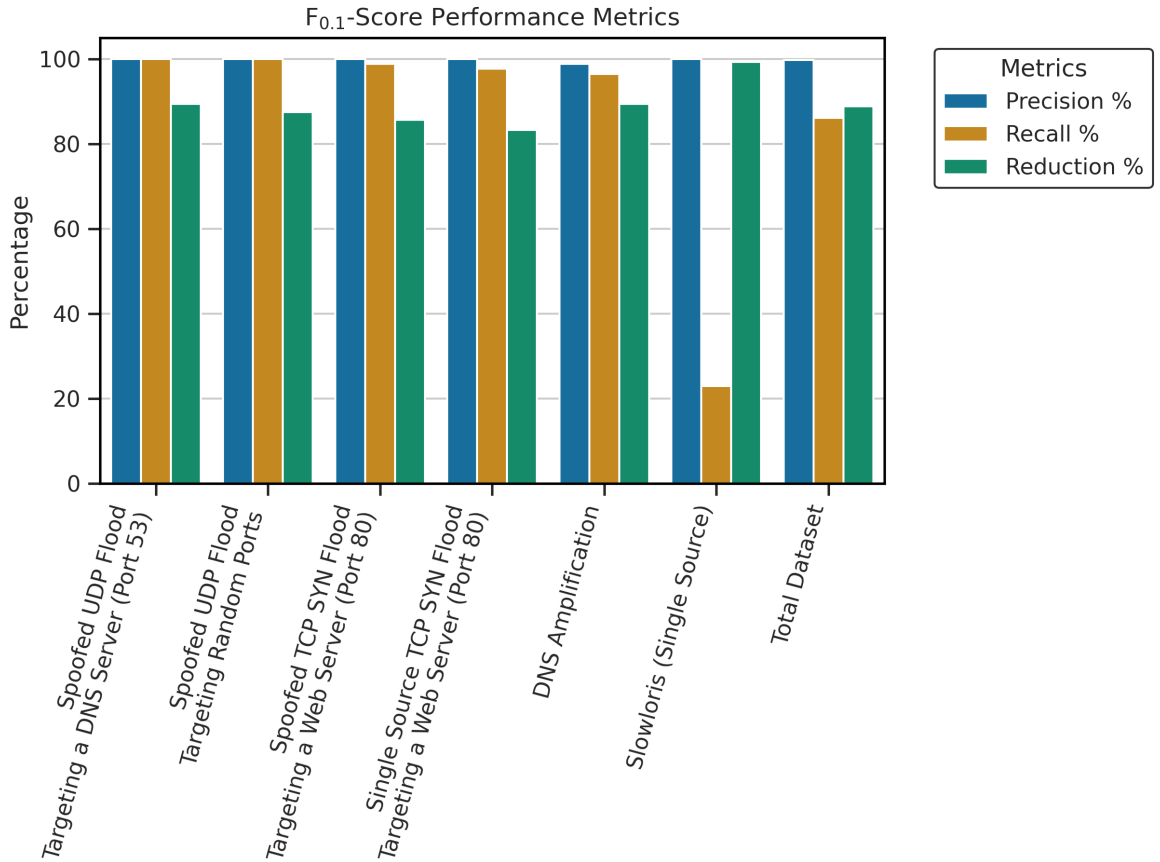



**Figure 4.12:** Bar Chart of Precision, Recall, and Reduction Percentages for the $F_{0.1}$-score Scenario

As expected, the precision increased in this $F_{0.1}$-score scenario, resulting in a

higher reduction percentage due to the stricter thresholds for data retention. However, since the precision was already high in the $F_1$-score scenario, these changes are marginal. Most attack types now achieve near-perfect precision with a minimal gain in storage reduction, at the cost of a slightly lowered recall.

The results for the last cost scenario, where recall has ten times more weight than precision ($F_{10}$-score), are shown in Table 4.4 and visualized in Figure 4.13.

**Table 4.4:** Precision, Recall, and Reduction Percentages for the $F_{10}$-score Scenario (rounded to 2 decimals)

| | **Precision %** | **Recall %** | **Reduction %** |
|---|---|---|---|
| **Spoofed UDP Flood Targeting a DNS Server (Port 53)** | 3.42 | 100.00 | 30.43 |
| **Spoofed UDP Flood Targeting Random Ports** | 3.45 | 100.00 | 29.57 |
| **Spoofed TCP SYN Flood Targeting a Web Server (Port 80)** | 3.40 | 98.81 | 30.53 |
| **Single Source TCP SYN Flood Targeting a Web Server (Port 80)** | 3.38 | 98.82 | 28.52 |
| **DNS Amplification** | 3.34 | 97.65 | 29.37 |
| **Slowloris (Single Source)** | 3.31 | 97.59 | 33.09 |
| **Total Dataset** | 3.38 | 98.81 | 30.19 |

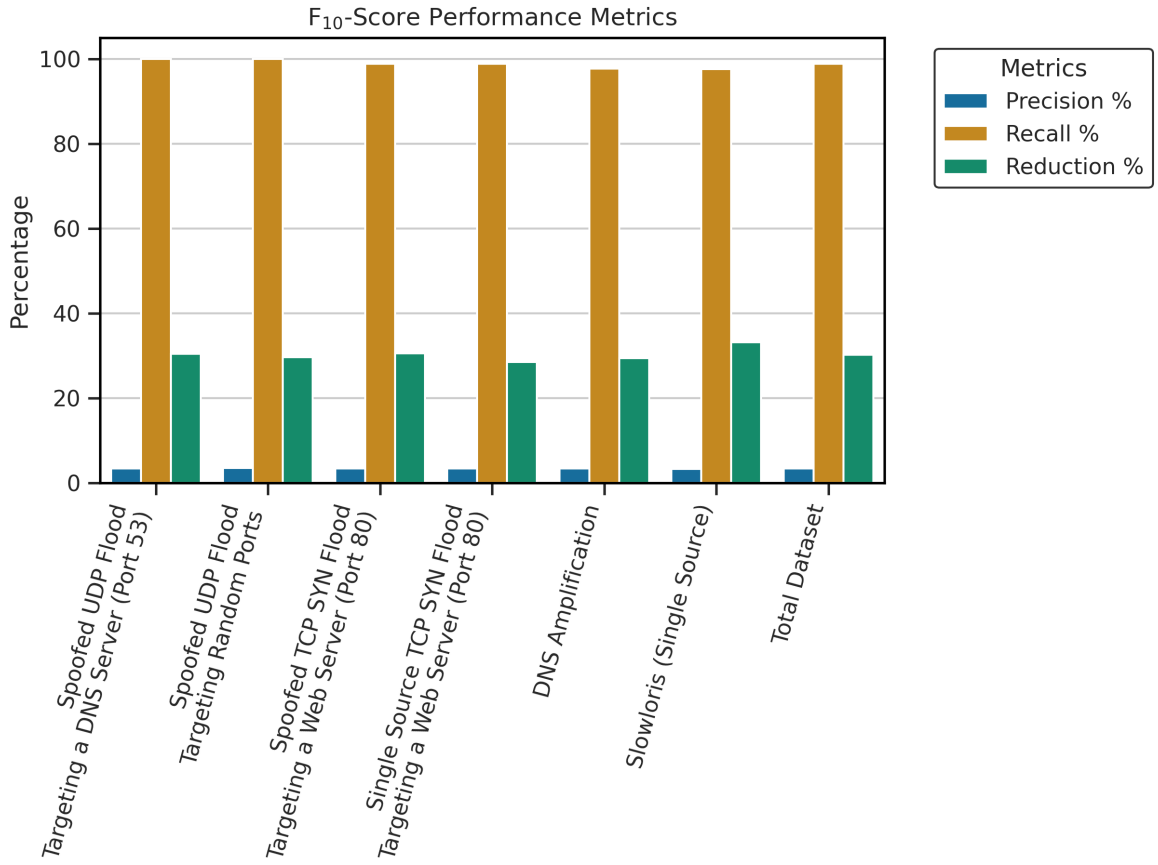



**Figure 4.13:** Bar Chart of Precision, Recall, and Reduction Percentages for the $F_{10}$-score Scenario

This $F_{10}$-score scenario is particularly interesting, as it highlights the trade-off between achieving a high recall to retain as much attack traffic as possible, and maintaining precision. Unfortunately, the precision drops significantly across all attack types, with values slightly above 3%. Despite this, a substantial reduction of around 30% is still achieved. For the total dataset, these sensitivity configurations achieve a recall of 98.81%, with a precision of 3.38% and a reduction of 30.19%. These results demonstrate that prioritizing recall significantly reduces the precision. Nonetheless, a notable reduction in the retained data size can still be achieved.

To clearly show the differences between the results for the various cost scenarios, we included Table 4.5 and Figure 4.14, which present these results side by side.

**Table 4.5:** Precision, Recall, and Reduction Percentages on the Total Dataset for Each Cost Scenario ($F_1$-score, $F_{10}$-score, and $F_{0.1}$-score) (rounded to 2 decimals)

| | **Precision %** | **Recall %** | **Reduction %** |
|---|---|---|---|
| **$F_1$-score** | 91.39 | 86.48 | 88.47 |
| **$F_{0.1}$-score** | 99.77 | 86.08 | 88.85 |
| **$F_{10}$-score** | 3.38 | 98.81 | 30.19 |

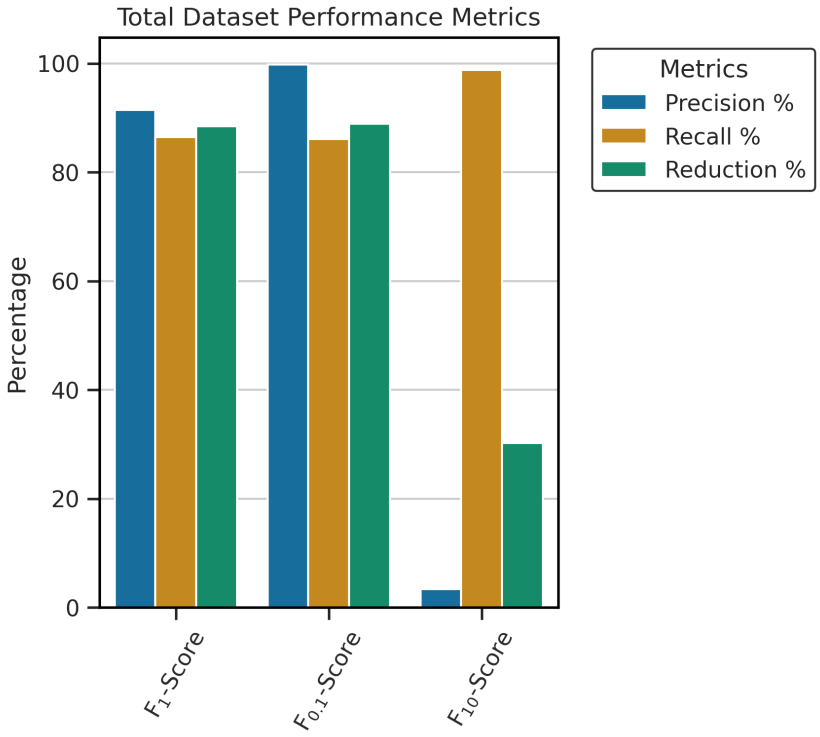



**Figure 4.14:** Bar Chart of Precision, Recall, and Reduction Percentages for Each Cost Scenario ($F_1$-score, $F_{0.1}$-score, and $F_{10}$-score)

As can be seen, the differences between the $F_1$-score and $F_{0.1}$-score scenarios are small. However, in the $F_{10}$-score scenario, which is more focused on retaining attack traffic, there is a clear improvement in recall but a significant decrease in precision, resulting in a lower reduction. Nonetheless, achieving a 30% reduction in storage while retaining nearly all relevant attack traffic data can still be beneficial.

Table 4.5 and Figure 4.14 present the retention performance and potential storage reduction that can be achieved for the three cost scenarios ($F_1$-score, $F_{0.1}$-score, and $F_{10}$-score), thereby answering RQ3.

# Chapter 5

# Discussion and Conclusion

In this study, we presented the design, implementation, deployment, and evaluation of a scalable analysis system aimed at reducing storage requirements for retaining DDoS attack traffic data by providing near real-time retention recommendations. Our findings confirm the feasibility of achieving significant reductions in storage requirements through the deployment of this system, thereby answering our Main RQ: *Is it feasible to reduce storage requirements for retaining DDoS attack traffic data through the deployment of a scalable analysis system that offers near real-time retention recommendations?*

### Interpretation of Results

Our evaluation results show substantial storage reduction across various cost scenarios associated with falsely retained traffic and undetected attacks. The system effectively identified and retained relevant data for the majority of attack types within our dataset. However, the Slowloris attack posed a challenge, highlighting the inherent difficulties in detecting such stealthy attacks. Overall, the results are promising for the adoption of this system in network environments where substantial data storage reductions are desired with minimal impact on the retention of attack traffic data.

### Comparison to Related Work

Compared to the related work discussed in Section 2.6, our system offers some distinct advantages, particularly in terms of scalability and integration with existing network infrastructure:

- **4EMA** [67] - This method, which focuses on DDoS detection using long-term and short-term trends, achieved a precision of 90% and a recall of 76%, which are only slightly lower than our results. However, 4EMA relies on a single-system architecture for its analysis, which limits its scalability, making it less suitable for large-scale network environments.
  In contrast, our system not only improves detection performance but it also uses distributed computing to scale effectively for large networks.

- **D-FACE** [93] - This system is distributed and scalable, while it also achieved high precision (97%) and recall (93%) values. However, it requires custom calculations to be performed by the network infrastructure, which may complicate the deployment and compatibility across diverse network environments.
  In contrast, although our system resulted in a slightly lower detection performance, it leverages the widely supported IPFIX protocol, enabling an easier integration with various enterprise network devices.
- **Online Entropy-Based DDoS Detection** [98] - This system focuses on detecting individual attack connections and it achieved perfect detection rates (100% precision and 100% recall). While these results are impressive, they may reflect the simplicity of the evaluation and might not indicate effectiveness in more complex real-world attack scenarios. Besides that, this system is designed for a single machine, which limits its scalability for large-scale network environments.
  In comparison, our system detects time windows with attack traffic rather than individual connections. Our approach sacrifices some granularity, resulting in less storage reduction, but it does provide a broader view of the attacks by retaining entire windows. And, unlike this system from related work, our system is designed with distributed computing in mind, enabling effective scaling for large networks.
- **Smart Collection and Storage Method** [105] - This method optimizes storage by selectively retaining traffic based on predefined filters, storage formats, and retention durations. The authors reported storage savings of up to 29.97% for full packet capture. While effective, this approach is more general and lacks specificity, as it relies on a static retention policy based on traffic types rather than actively analyzing current network activity. This restricts its potential for more targeted storage reduction, such as the retention of only DDoS attack traffic.
  In contrast, our system makes retention decisions in near real-time using an analytical approach based on the observed network activity. This allows for greater potential storage savings, as shown by our results, where storage reduction range from 30.19% to 88.85%, depending on the importance ratio between precision and recall. However, it is important to note that our system is specifically designed for the retention of DDoS attack traffic and it is not intended for broader traffic retention like this other method.

### Implications

The implications of our research are both theoretical and practical.

Theoretically, our study contributes to the advancement of selective retention methods by demonstrating that such a method can be made scalable and easy to integrate into existing network infrastructure. Our findings show that a scalable re-

tention recommendation system can significantly reduce storage requirements while still retaining most of the relevant DDoS attack traffic data.

Practically, the adoption of such a system could lead to substantial cost savings for organizations managing large volumes of network traffic. By implementing a scalable retention recommendation system like ours, network operators can optimize their data retention strategies, thereby reducing their storage requirements and the related costs while retaining most of the relevant DDoS attack traffic data.

### Limitations

Despite the valuable insights provided by our study, a couple of limitations should be acknowledged.

One limitation of our study is that we relied on a self-generated synthetic dataset for the evaluation of our system due to the absence of a suitable existing dataset. While our dataset allowed us to evaluate the system, it may not have fully captured the complexities of actual network traffic, which limits the generalizability of our results. Moreover, the dataset primarily focused on DDoS attacks, and we thereby excluded other types of network anomalies that could have potentially impacted the system's performance.

Another limitation of our study is the simplified deployment setup, in which we used Spark's standalone mode with two worker instances on the same machine. While this setup demonstrates the distributed processing capabilities, it does not represent a true distribution across multiple devices in a cluster. Besides that, we employed only a single exporter, a single collector, and a single Kafka instance. Although these components can be scaled horizontally, this was not tested in practice within the scope of our research. This limited deployment setup could impact the system's performance, and this may not accurately reflect how it would perform in a more distributed and scaled-up environment.

### Future Work

Future research should aim to address the aforementioned limitations of our study. A more comprehensive evaluation should be conducted with a diverse and realistic dataset that includes a wider range of attack types and network anomalies. This will improve the generalizability of the results compared to our evaluation with a self-generated synthetic dataset. Additionally, it is important to deploy the system on a real cluster environment, rather than the simplified setup used in this study. Evaluating the system when it is deployed across multiple devices and scaled up with multiple exporters, collectors and Kafka instances will provide valuable insights into its performance and scalability when configured for large-scale environments.

Besides addressing the limitations, it is also important that future work addresses some of the issues that we identified in the current implementation. One issue is that the intermediate buffer for the entropy calculation can become exceptionally large when dealing with a substantial number of attribute variations, particularly in the

case of IP addresses and flow size variants. Solutions like address truncation or size bucketing can be considered to limit these variations within safe ranges, although each introduces a trade-off between the buffer size and the ability to distinguish certain variants. Additionally, there is an issue when the MAD equals zero, as this can cause a division by zero in the modified Z-score calculation. This should also be addressed in future iterations, as it currently results in the system reporting NaN.

For future work that further develops the retention recommendation system, several enhancements could be explored to improve its performance. Initially, we aimed to use bidirectional flows, as Nychis et al. [45] recommended bidirectional flows to mitigate bias in the attribute distributions. However, due to an unexpected implementation of bidirectional flows by the developers of Softflowd and the lack of a suitable alternative, we switched to unidirectional flows, which may have negatively impacted our results. Therefore, future work should implement and evaluate the system with bidirectional flows to investigate the potential improvement of the results. In addition, optimizing system parameters such as the timeout duration, window size, and sliding interval could potentially further improve performance. Future research could also focus on refining the detection techniques by exploring more advanced data-driven approaches like machine learning, investigating alternative entropies such as the Renyi [52] and Tsallis [53] entropies, or by utilizing flow attributes other than the common ones used in this study. A generous safety margin for the arrival times of flow records was assumed in this study. Future research can focus on providing better insights into these arrival times to configure this margin more tightly. A final suggestion for future work is to investigate the use of multiple features and their directions of change, whether they are more dispersed or concentrated, as this could be used to even fingerprint and classify different attack types.

## Conclusion

In conclusion, this study demonstrates the feasibility of leveraging a scalable analysis system to significantly reduce storage requirements for retaining DDoS attack traffic by providing near real-time retention recommendations. The findings of this research provide valuable insights for organizations that are looking to improve their data retention strategies, by presenting this system as a promising solution for further development and potential adoption. To facilitate this advancement, we have made the code of our system publicly available in the following repository, hosted by the University of Twente:
`https://gitlab.utwente.nl/BasBleijerveld/scalable-ddos-traffic-retention`

# Bibliography


[1] O. Yoachimik. (2023, January) Cloudflare DDoS threat report for 2022 Q4. The Cloudflare Blog. Accessed: 19-03-2023. [Online]. Available: https://blog.cloudflare.com/ddos-threat-report-2022-q4/

[2] R. Hunt and S. Zeadally, "Network forensics: An Analysis of Techniques, Tools, and Trends," *Computer*, vol. 45, pp. 36–43, July 2012.

[3] S. Garfinkel. (2002, April) Network forensics: Tapping the internet. O'Reilly Network. Accessed: 21-04-2023. [Online]. Available: https://web.archive.org/web/20140610075438/http://www.oreillynet.com:80/pub/a/network/2002/04/26/nettap.html

[4] N. Karie and S. Karume, "Digital Forensic Readiness in Organizations: Issues and Challenges," *The Journal of Digital Forensics, Security and Law*, vol. 12, pp. 43–53, January 2017.

[5] HTTPS encryption on the web. Google Transparency Report. Accessed: 30-11-2023. [Online]. Available: https://transparencyreport.google.com/https/overview

[6] (2020) DDoS data rapport 2020. Nationale Beheersorganisatie Internet Providers. Accessed: 27-03-2024. [Online]. Available: https://www.nbip.nl/wp-content/uploads/2021/04/NBIP-Rapport-DDoS-data-2020.pdf

[7] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, pp. 39–53, April 2004.

[8] S. T. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, pp. 2046–2069, March 2013.

[9] Low Orbit Ion Cannon. Accessed: 05-04-2023. [Online]. Available: https://sourceforge.net/projects/loic/

[10] R-U-Dead-Yet. Accessed: 05-04-2023. [Online]. Available: https://sourceforge.net/projects/r-u-dead-yet/

[11] M. Sauter, ""LOIC Will Tear Us Apart": The Impact of Tool Design and Media Portrayals in the Success of Activist DDOS Attacks," *American Behavioral Scientist*, vol. 57, pp. 983–1007, July 2013.

[12] M. Deseriis, "Hacktivism: On the Use of Botnets in Cyberattacks," *Theory, Culture & Society*, vol. 34, pp. 131–152, July 2017.

[13] What is a DDoS Botnet? Cloudflare Learning Center. Accessed: 12-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/

[14] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proceedings of the 26th USENIX Conference on Security Symposium*. USENIX Association, August 2017, p. 1093–1110.

[15] J. Santanna, R. Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Granville, and A. Pras, "Booters - An Analysis of DDoS-as-a-Service Attacks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, June 2015, pp. 243–251.

[16] What is IP spoofing? Cloudflare Learning Center. Accessed: 12-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/ip-spoofing/

[17] Ping (ICMP) flood DDoS attack. Cloudflare Learning Center. Accessed: 13-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/

[18] UDP flood attack. Cloudflare Learning Center. Accessed: 13-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/udp-flood-ddos-attack/

[19] SYN flood attack. Cloudflare Learning Center. Accessed: 13-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/

[20] HTTP flood attack. Cloudflare Learning Center. Accessed: 14-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/

[21] What is a DNS flood? | DNS flood DDoS attack. Cloudflare Learning Center. Accessed: 14-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/dns-flood-ddos-attack/

[22] NTP amplification DDoS attack. Cloudflare Learning Center. Accessed: 14-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/ntp-amplification-ddos-attack/

[23] Memcached DDoS attack. Cloudflare Learning Center. Accessed: 14-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/memcached-ddos-attack/

[24] Memcached 1.5.6 Release Notes. GitHub. Accessed: 14-04-2023. [Online]. Available: https://github.com/memcached/memcached/wiki/ReleaseNotes156

[25] What is a low and slow attack? Cloudflare Learning Center. Accessed: 14-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/ddos-low-and-slow-attack/

[26] Slowloris DDoS attack. Cloudflare Learning Center. Accessed: 14-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/

[27] R U Dead Yet? (R.U.D.Y.) attack. Cloudflare Learning Center. Accessed: 14-04-2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/r-u-dead-yet-rudy/

[28] L. Cottrel. (2001, March) Passive vs. Active Monitoring. Stanford Linear Accelerator Center. Accessed: 25-04-2023. [Online]. Available: https://www.slac.stanford.edu/comp/net/wan-mon/passive-vs-active.html

[29] RIPE NCC. RIPE Atlas. Accessed: 25-04-2023. [Online]. Available: https://atlas.ripe.net/

[30] NLNOG. NLNOG Ring. Accessed: 25-04-2023. [Online]. Available: https://ring.nlnog.net/

[31] J. Svoboda, I. Ghafir, and V. Prenosil, "Network Monitoring Approaches: An Overview," *International Journal of Advances in Computer Networks and Its Security– IJCNS*, vol. 5, pp. 88–93, October 2015.

[32] tcpdump & libpcap. The Tcpdump Group. Accessed: 03-05-2023. [Online]. Available: https://www.tcpdump.org/

[33] Pcap4J. Accessed: 03-05-2023. [Online]. Available: https://www.pcap4j.org/

[34] Scapy. Accessed: 03-05-2023. [Online]. Available: https://scapy.net/

[35] WinPcap. Riverbed Technology. Accessed: 03-05-2023. [Online]. Available: https://www.winpcap.org/default.htm

[36] Npcap. The Nmap Project. Accessed: 03-05-2023. [Online]. Available: https://npcap.com/

[37] Wireshark. The Wireshark Team. Accessed: 03-05-2023. [Online]. Available: https://www.wireshark.org/

[38] W. John, S. Tafvelin, and T. Olovsson, "Passive Internet Measurement: Overview and Guidelines based on Experiences," *Computer Communications*, vol. 33, pp. 533–550, March 2010.

[39] sFlow. sFlow.org Consortium. Accessed: 04-05-2023. [Online]. Available: https://sflow.org/

[40] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX," *IEEE Communications Surveys and Tutorials*, vol. 16, pp. 2037–2064, April 2014.

[41] S. Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy," Chalmers University of Technology, Tech. Rep. 99-15, March 2000.

[42] H. Debar and J. Viinikka, "Intrusion Detection: Introduction to Intrusion Detection and Security Information Management," in *Foundations of Security Analysis and Design III: FOSAD 2004/2005 Tutorial Lectures*, A. Aldini, R. Gorrieri, and F. Martinelli, Eds. Springer Berlin/Heidelberg, 2005, pp. 207–236.

[43] S. Kumar, "Survey of Current Network Intrusion Detection Techniques," Washington University in St. Louis, pp. 1–18, 2007.

[44] P. Bereziński, B. Jasiul, and M. Szpyrka, "An Entropy-Based Network Anomaly Detection Method," *Entropy*, vol. 17, pp. 2367–2408, April 2015.

[45] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang, "An Empirical Evaluation of Entropy-Based Traffic Anomaly Detection," in *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 151–156.

[46] R. Clausius, *The Mechanical Theory of Heat: With Its Applications to the Steam-engine and to the Physical Properties of Bodies*. London, UK: J. Van Voorst, 1867.

[47] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[48] T. Cover and J. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: Wiley, 2006.

[49] J. David and C. Thomas, "Detection of Distributed Denial of Service Attacks based on Information Theoretic Approach in Time Series Models," *Journal of Information Security and Applications*, vol. 55, p. 102621, 2020.

[50] A. Lakhina, M. Crovella, and C. Diot, “Mining Anomalies Using Traffic Feature Distributions,” in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM ’05. New York, NY, USA: Association for Computing Machinery, 2005, p. 217–228.

[51] T. Maszczyk and W. Duch, “Comparison of Shannon, Renyi and Tsallis Entropy Used in Decision Trees,” in *Artificial Intelligence and Soft Computing – ICAISC 2008*, L. Rutkowski, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds. Springer Berlin/Heidelberg, 2008, pp. 643–651.

[52] A. Rényi, *Probability Theory*. Amsterdam: North-Holland, 1970.

[53] C. Tsallis, “Possible Generalization of Boltzmann-Gibbs Statistics,” *Journal of Statistical Physics*, vol. 52, pp. 479–487, 1988.

[54] Y. Xiang, K. Li, and W. Zhou, “Low-Rate DDoS Attacks Detection and Traceback by Using New Information Metrics,” *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 2, pp. 426–437, 2011.

[55] A. Ziviani, A. T. A. Gomes, M. L. Monsores, and P. S. Rodrigues, “Network Anomaly Detection using Nonextensive Entropy,” *IEEE Communications Letters*, vol. 11, no. 12, pp. 1034–1036, 2007.

[56] B. Tellenbach, M. Burkhart, D. Sornette, and T. Maillart, “Beyond Shannon: Characterizing Internet Traffic with Generalized Entropy Metrics,” in *Passive and Active Network Measurement*. Springer Berlin/Heidelberg, 2009, pp. 239–248.

[57] Single Moving Average. NIST/SEMATECH e-Handbook of Statistical Methods. Accessed: 19-06-2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc421.htm

[58] G. No and I. Ra, “An Efficient and Reliable DDoS Attack Detection Using a Fast Entropy Computation Method,” in *2009 9th International Symposium on Communications and Information Technology*, 2009, pp. 1223–1228.

[59] ——, “Adaptive DDoS Detector Design Using Fast Entropy Computation Method,” in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2011, pp. 86–93.

[60] What is Exponential Smoothing? NIST/SEMATECH e-Handbook of Statistical Methods. Accessed: 19-06-2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc43.htm

[61] Single Exponential Smoothing. NIST/SEMATECH e-Handbook of Statistical Methods. Accessed: 19-06-2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm

[62] Forecasting with Single Exponential Smoothing. NIST/SEMATECH e-Handbook of Statistical Methods. Accessed: 19-06-2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc432.htm

[63] Double Exponential Smoothing. NIST/SEMATECH e-Handbook of Statistical Methods. Accessed: 19-06-2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc433.htm

[64] C. C. Holt, "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages," *International Journal of Forecasting*, vol. 20, no. 1, pp. 5–10, 2004.

[65] Forecasting with Double Exponential Smoothing(LASP). NIST/SEMATECH e-Handbook of Statistical Methods. Accessed: 19-06-2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc434.htm

[66] Triple Exponential Smoothing. NIST/SEMATECH e-Handbook of Statistical Methods. Accessed: 20-06-2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc435.htm

[67] P. Bojović, I. Bašičević, S. Ocovaj, and M. Popović, "A Practical Approach to Detection of Distributed Denial-of-Service Attacks Using a Hybrid Detection Method," *Computers & Electrical Engineering*, vol. 73, pp. 84–96, 2019.

[68] Á. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading Real-time DDoS Attack Detection to Programmable Data Planes," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 19–27.

[69] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras, "Towards Real-Time Intrusion Detection for NetFlow and IPFIX," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 227–234.

[70] Detection of Outliers. NIST/SEMATECH e-Handbook of Statistical Methods. Accessed: 06-07-2023. [Online]. Available: https://www.itl.nist.gov/div898/handbook/eda/section3/eda35h.htm

[71] R. Van de Meent, M. Mandjes, and A. Pras, "Gaussian Traffic Everywhere?" in *2006 IEEE International Conference on Communications*, vol. 2, 2006, pp. 573–578.

[72] B. Iglewicz and D. Hoaglin, *Volume 16: How to Detect and Handle Outliers*, ser. The ASQC Basic References in Quality Control: Statistical Techniques, P. Mykytka, Edward F., Ed. ASQC Quality Press, 1993.

[73] H. Majed, H. N. Noura, O. Salman, M. Malli, and A. Chehab, “Efficient and Secure Statistical DDoS Detection Scheme,” in *Proceedings of the 17th International Joint Conference on e-Business and Telecommunications - WINSYS*, INSTICC. SciTePress, 2020, pp. 153–161.

[74] A. Blaise, M. Bouet, V. Conan, and S. Secci, “Detection of zero-day attacks: An unsupervised port-based approach,” *Computer Networks*, vol. 180, p. 107391, 2020.

[75] T. Jirsik, M. Cermak, D. Tovarnak, and P. Celeda, “Toward Stream-Based IP Flow Analysis,” *IEEE Communications Magazine*, vol. 55, no. 7, pp. 70–76, 2017.

[76] Stream4Flow. GitHub. Accessed: 07-09-2023. [Online]. Available: https://github.com/CSIRT-MU/Stream4Flow

[77] M. Čermák, D. Tovarňák, M. Laštovička, and P. Čeleda, “A Performance Benchmark for NetFlow Data Analysis on Distributed Stream Processing Systems,” in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 919–924.

[78] M. Cermak and P. Celeda, “Stream-Based IP Flow Analysis,” in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 736–741.

[79] IPFIXcol. GitHub. Accessed: 07-09-2023. [Online]. Available: https://github.com/CESNET/ipfixcol

[80] Apache Kafka. The Apache Software Foundation. Accessed: 07-09-2023. [Online]. Available: https://kafka.apache.org/

[81] Apache Spark. The Apache Software Foundation. Accessed: 07-09-2023. [Online]. Available: https://spark.apache.org/

[82] Elastic Stack. Elastic. Accessed: 07-09-2023. [Online]. Available: https://www.elastic.co/elastic-stack/

[83] Logstatsh. Elastic. Accessed: 07-09-2023. [Online]. Available: https://www.elastic.co/logstash

[84] Elasticsearch. Elastic. Accessed: 07-09-2023. [Online]. Available: https://www.elastic.co/elasticsearch/

[85] Kibana. Elastic. Accessed: 07-09-2023. [Online]. Available: https://www.elastic.co/kibana

[86] M. Cermak, M. Laštovička, and T. Jirsik, “Real-time Pattern Detection in IP Flow Data using Apache Spark,” in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 521–526.

[87] Apache Kafka Documentation. The Apache Software Foundation. Accessed: 11-09-2023. [Online]. Available: https://kafka.apache.org/documentation/

[88] Apache Spark Documentation. The Apache Software Foundation. Accessed: 11-09-2023. [Online]. Available: https://spark.apache.org/docs/latest/

[89] Kubernetes. Cloud Native Computing Foundation. Accessed: 12-09-2023. [Online]. Available: https://kubernetes.io/

[90] Docker. Docker Inc. Accessed: 13-09-2023. [Online]. Available: https://www.docker.com/

[91] Docker Documentation. Docker Inc. Accessed: 14-09-2023. [Online]. Available: https://docs.docker.com/

[92] Kubernetes Documentation. Cloud Native Computing Foundation. Accessed: 14-09-2023. [Online]. Available: https://kubernetes.io/docs

[93] S. Behal, K. Kumar, and M. Sachdeva, “D-FACE: An Anomaly Based Distributed Approach for Early Detection of DDoS Attacks and Flash Events,” *Journal of Network and Computer Applications*, vol. 111, pp. 49–63, 2018.

[94] (2000) MIT Lincoln Laboratory LLSDDoS 1.0 Dataset. Accessed: 27-06-2024. [Online]. Available: https://archive.ll.mit.edu/ideval/data/2000/LLS_DDOS_1.0.html

[95] (2007) The CAIDA UCSD "DDoS Attack 2007" Dataset. Accessed: 27-06-2024. [Online]. Available: https://www.caida.org/catalog/datasets/ddos-20070804_dataset

[96] M. Arlitt and T. Jin. (1998, August) 1998 World Cup Web Site Access Logs. Accessed: 27-06-2024. [Online]. Available: https://ita.ee.lbl.gov/html/contrib/WorldCup.html

[97] BoNeSi - The DDoS Botnet Simulator. GitHub. Accessed: 27-06-2024. [Online]. Available: https://github.com/Markus-Go/bonesi

[98] L. D. Tsobdjou, S. Pierre, and A. Quintero, “An Online Entropy-Based DDoS Flooding Attack Detection System With Dynamic Threshold,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1679–1689, 2022.

[99] Graphical Network Simulator-3. Accessed: 08-07-2024. [Online]. Available: https://www.gns3.com/

[100] Apache HTTP Server. Apache Software Foundation. Accessed: 12-04-2024. [Online]. Available: https://httpd.apache.org/

[101] httperf - HTTP Performance Measurement Tool. Accessed: 08-07-2024. [Online]. Available: https://manpages.ubuntu.com/manpages/focal/en/man1/httperf.1.html

[102] hping. Accessed: 12-04-2024. [Online]. Available: https://web.archive.org/web/20221105010555/http://www.hping.org/

[103] Intrusion Detection Evaluation Dataset (CIC-IDS2017). Canadian Institute for Cybersecurity. Accessed: 08-07-2024. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[104] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *4th International Conference on Information Systems Security and Privacy (ICISSP)*, Portugal, January 2018.

[105] A. Horneman and N. Dell, "Smart Collection and Storage Method for Network Traffic Data," Carnegie Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-2014-TR-011, September 2014, Accessed: 10-07-2024. [Online]. Available: https://doi.org/10.1184/R1/6583826.v1

[106] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An Overview of IP Flow-Based Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[107] FlowMon. Progress Software Corporation. Accessed: 16-10-2024. [Online]. Available: https://www.flowmon.com

[108] softflowd. GitHub. Accessed: 20-12-2023. [Online]. Available: https://github.com/irino/softflowd

[109] tcpreplay. GitHub. Accessed: 20-12-2023. [Online]. Available: https://github.com/appneta/tcpreplay

[110] IPFIXcol2. GitHub. Accessed: 21-12-2023. [Online]. Available: https://github.com/CESNET/ipfixcol2

[111] Bitnami Apache Kafka Docker Image. Docker Hub. Accessed: 22-12-2023. [Online]. Available: https://hub.docker.com/r/bitnami/kafka

[112] Spark Release 3.5.0. The Apache Software Foundation. Accessed: 03-01-2024. [Online]. Available: https://spark.apache.org/releases/spark-release-3-5-0.html

[113] A. Chu and J. Lim. (2023, August) Multiple Stateful Operators in Structured Streaming. Databricks Engineering Blog. Accessed: 03-01-2024. [Online]. Available: https://www.databricks.com/blog/multiple-stateful-operators-structured-streaming

[114] A. Azera. (2023, October) Time Window Aggregation in Separate Streams Followed by Stream-Stream Join Not Returning Results. Apache Spark Issue SPARK-45637. [Online]. Available: https://issues.apache.org/jira/browse/SPARK-45637

[115] scala-collection-contrib. GitHub. Accessed: 05-01-2024. [Online]. Available: https://github.com/scala/scala-collection-contrib

[116] Tuning - Spark 3.5.0 Documentation. The Apache Software Foundation. Accessed: 30-05-2024. [Online]. Available: https://spark.apache.org/docs/3.5.0/tuning.html

[117] Spark Standalone Mode - Spark 3.5.0 Documentation. The Apache Software Foundation. Accessed: 08-01-2024. [Online]. Available: https://spark.apache.org/docs/3.5.0/spark-standalone.html

[118] sbt. Scala Center. Accessed: 08-01-2024. [Online]. Available: https://www.scala-sbt.org/

[119] sbt-assembly. GitHub. Accessed: 08-01-2024. [Online]. Available: https://github.com/sbt/sbt-assembly

[120] Apache Hadoop. The Apache Software Foundation. Accessed: 08-01-2024. [Online]. Available: https://hadoop.apache.org/

[121] Amazon S3. Amazon Web Services. Accessed: 08-01-2024. [Online]. Available: https://aws.amazon.com/s3/

[122] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8.

[123] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012.

[124] Intrusion Detection Evaluation Dataset (ISCXIDS2012). Canadian Institute for Cybersecurity. Accessed: 08-07-2024. [Online]. Available: https://www.unb.ca/cic/datasets/ids.html

[125] Open vSwitch. Linux Foundation. Accessed: 12-04-2024. [Online]. Available: https://www.openvswitch.org/

[126] S. Kelley. dnsmasq. Accessed: 12-04-2024. [Online]. Available: https://thekelleys.org.uk/dnsmasq/doc.html

[127] Lorem Generator. Accessed: 12-04-2024. [Online]. Available: https://loremgenerator.io/

[128] G. Yaltirakli. Slowloris. GitHub. Accessed: 12-04-2024. [Online]. Available: https://github.com/gkbrk/slowloris

[129] Internet Corporation for Assigned Names and Numbers. (2009) Global Policy for the Allocation of the Remaining IPv4 Address Space. Accessed: 12-04-2024. [Online]. Available: https://www.icann.org/resources/pages/remaining-ipv4-2012-02-25-en

[130] M. Majkowski. (2017, May) Reflections on reflection (attacks). The Cloudflare Blog. Accessed: 12-04-2024. [Online]. Available: https://blog.cloudflare.com/reflections-on-reflections/

[131] J. da Silva Damas, M. Graff, and P. A. Vixie, "Extension Mechanisms for DNS (EDNS(0))," RFC 6891, Apr. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6891

[132] O. van der Toorn, J. Krupp, M. Jonker, R. van Rijswijk-Deij, C. Rossow, and A. Sperotto, "ANYway: Measuring the Amplification DDoS Potential of Domains," in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021, pp. 500–508.

[133] J. Abley, Ólafur Guðmundsson, M. Majkowski, and E. Hunt, "Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY," RFC 8482, Jan. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8482