CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Scalable Anomaly-Based Network Intrusion Detection Using a Statistical Model and Data Sketches

Employing Lightweight Statistical Techniques for Scalable Detection of DDoS Attacks in Network Traffic

Master's Thesis in Computer Science and Engineering

Albert Wickman & Daniel Rygaard

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Scalable Anomaly-Based Network Intrusion Detection Using a Statistical Model and Data Sketches

Employing Lightweight Statistical Techniques for Scalable Detection of DDoS Attacks in Network Traffic

Albert Wickman & Daniel Rygaard

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Scalable Anomaly-Based Network Intrusion Detection Using a Statistical Model and Data Sketches
Employing Lightweight Statistical Techniques for Scalable Detection of DDoS Attacks in Network Traffic
Albert Wickman & Daniel Rygaard

Supervisor: Romaric Duvignau, Computer Science and Engineering
Advisor: Georgios Pseiras, Ericsson AB
Examiner: Marina Papatriantafilou, Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Scalable Anomaly-Based Network Intrusion Detection Using a Statistical Model and Data Sketches
Employing Lightweight Statistical Techniques for Scalable Detection of DDoS Attacks in Network Traffic
Albert Wickman & Daniel Rygaard
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The growing frequency and complexity of cyber-attacks, especially Distributed Denial of Service (DDoS) attacks, has made protecting networks a major priority for businesses. Traditional Network Intrusion Detection Systems (NIDS) often struggle to cope with the large volumes of traffic seen in todays networks. These systems can be inefficient, often bogged down by high memory usage and significant computational demands. In this thesis, we propose a solution to these challenges by developing a more efficient, scalable system for detecting anomalies in network traffic. Our approach, the Baseline Configuration, combines a statistical model with probabilistic data structures, such as Count-Min Sketch and HeavyKeeper Sketch, to process high volumes of traffic in real-time while keeping resource consumption to a minimum.

At the core of the Baseline Configuration is a statistical model built around the Interquartile Range (IQR) rule, which adjusts a detection threshold based on changes in network traffic. This helps the system identify abnormal patterns without flagging harmless variations as threats. To make the system even more responsive, we incorporate sliding window techniques, enabling it to continuously monitor traffic in small, manageable time segments. This ensures that the system remains accurate and efficient, even when network traffic spikes.

The performance of the proposed system is tested using different datasets, including traffic data from Ericsson and the Center for Applied Internet Data Analysis (CAIDA). CAIDA is a well-known repository that provides real-world internet traffic traces commonly used for network research. The memory efficiency and processing times are compared to a Hash Map and Priority Queue (HP) Configuration, which uses these data structures instead of the Count-Min and HeavyKeeper sketch. Additionally, the detection accuracy and performance of the Baseline Configuration are compared to a Machine Learning (ML) Configuration which uses the Isolation Forest algorithm. The evaluation results demonstrate that the Baseline Configuration not only provides higher detection accuracy but also operates with significantly lower memory usage and faster response times than the other configurations. The systems ability to adapt to increasing traffic without compromising its performance makes it suitable for large-scale network environments. Through this work, it is shown that combining statistical models with data sketches provides a cost-effective, scalable, and efficient solution for real-time network intrusion detection for DDoS attacks.

Keywords: Cybersecurity, Distributed Denial of Service (DDoS) Attacks, Network

Intrusion Detection Systems (NIDS), Anomaly Detection, Statistical Models, Data Sketches, Count-Min Sketch, HeavyKeeper Sketch, Interquartile Range (IQR), Sliding Window Technique, Real-Time Traffic Monitoring, Scalable Detection Systems, Hash Map and Priority Queue Configuration, Memory Efficiency, Processing Efficiency, Machine Learning (ML), Isolation Forest, Detection Accuracy, Resource Efficiency, Traffic Analysis, Network Traffic Fluctuations, Large-Scale Network Security, Real-Time Network Intrusion Detection.

Acknowledgements

We would like to express our gratitude to our supervisor and examiner from Chalmers, Romaric Duvignau, for his guidance and support throughout the course of our work. We also extend our thanks to Georgios Pseiras, our supervisor at Ericsson AB, for his continuous support and counseling.

Albert Wickman & Daniel Rygaard, Gothenburg, 2025-06-10

Contents

List of Figures **xv**

List of Tables **xvii**

1 Introduction **1**
1.1 Context . . . 1
1.2 Aim . . . 2

2 Background **3**
2.1 Network Intrusion Detection Systems . . . 3
2.2 Interquartile Range Rule . . . 4
2.3 Distributed Denial of Service Attacks . . . 4
2.4 Machine Learning Algorithms . . . 5
2.4.1 Isolation Forest . . . 5
2.5 Data Stream . . . 6
2.6 Windowing . . . 6
2.6.1 Tumbling Windows . . . 6
2.6.2 Sliding Windows . . . 6
2.7 Data Sketches . . . 7
2.7.1 Count-Min Data Sketch . . . 7
2.7.2 HeavyKeeper Algorithm . . . 8
2.8 Datasets . . . 8
2.8.1 CAIDA2007 . . . 8
2.8.2 Ericsson Data . . . 9

3 Method **11**
3.1 Comparative Analysis of Anomaly Detection Methods . . . 11
3.2 Implementation . . . 11
3.2.1 System Architecture . . . 12
3.2.2 Configurations . . . 12
3.2.2.1 Baseline Configuration . . . 13
3.2.2.2 Hash Map Priority Queue (HP) Configuration . . . 13
3.2.2.3 Machine Learning (ML) Configuration . . . 14
3.2.3 Data Collection . . . 14
3.2.4 Data Preprocessing . . . 15
3.2.4.1 Data Aggregation . . . 15

3.2.4.2 Data Structures . . . 15
3.2.5 Anomaly Detection . . . 15
3.2.5.1 Packet Threshold per Time Window . . . 16
3.2.5.2 Packet Threshold per IP Address . . . 16
3.2.5.3 Setting the Threshold . . . 17
3.2.6 Data Pipe Publisher . . . 18
3.3 Metrics . . . 18
3.3.1 Anomaly Detection . . . 19
3.3.2 Threshold Scalability . . . 19
3.3.3 Comparing the Statistical Model to Machine Learning . . . 20
3.3.4 Detection Latency . . . 20
3.3.5 Resource Utilization . . . 21
3.3.6 Counting Element Frequency . . . 21
3.3.7 Sorting . . . 21
3.4 Limitations . . . 22

4 Results **23**
4.1 Baseline Configuration . . . 23
4.1.1 Detection Accuracy . . . 23
4.1.1.1 Accuracy with attacks . . . 23
4.1.1.2 Accuracy with normal data only . . . 24
4.1.2 Rate Limit . . . 25
4.1.3 Evaluating the Threshold . . . 27
4.1.4 Threshold Adaptability . . . 28
4.1.5 Detection Latency . . . 29
4.2 Comparing the Baseline Configuration to the HP Configuration . . . 30
4.2.1 Detection Accuracy . . . 30
4.2.2 Processing Time . . . 31
4.2.3 Memory Utilization . . . 32
4.3 Comparing the Baseline Configuration to the ML Configuration . . . 32
4.3.1 Detection Accuracy . . . 32
4.3.2 Response Time . . . 33
4.4 Summary . . . 34
4.4.1 Detection Accuracy . . . 34
4.4.2 Detection Latency . . . 35
4.4.3 Resource Utilization . . . 36
4.4.4 Rate Limiting . . . 36
4.4.5 Comparison to Machine Learning . . . 36

5 Conclusion **39**
5.1 Summary of the Thesis . . . 39
5.2 Future Work . . . 40
5.2.1 IP Address Filtering . . . 40
5.2.2 Threshold Offset Based on Data Fluctuations . . . 40
5.2.3 Machine Learning Fine Tuning . . . 40
5.2.4 Setting the Optimal Rate Limit . . . 41
5.2.5 Deciding the Optimal Time Window Size . . . 41

5.2.6 Degree of Threshold Flexibility 41
5.2.7 Potential Bottlenecks in the System 41

Bibliography **43**

List of Figures

2.1 Ericsson Dataset 9

3.1 Baseline Configuration System Architecture Diagram 12
3.2 Baseline Configuration System Architecture Diagram 13
3.3 Hash Map & Priority Queue Configuration System Architecture Diagram 13
3.4 Machine Learning Configuration System Architecture Diagram 14
3.5 Threshold based on normal data. 18

4.1 Ericsson normal data and threshold 24
4.2 Unfiltered vs Filtered Packet Counts, Dataset: rand_src_DDoS_108 provided by Ericsson AB 25
4.3 Unfiltered vs Filtered Packet Counts, Dataset: ddostrace.20070804_144936 26
4.4 Impact of a higher time window threshold on normal traffic. Dataset: *normal_traffic* (Ericsson AB) 27
4.5 Impact of a lower time window threshold on normal traffic. Dataset: *normal_traffic* (Ericsson AB) 27
4.6 Threshold adaptability under increasing normal data. Dataset: *normal_traffic* (Ericsson AB) 28
4.7 Threshold adaptability under increasing normal data at 20% every 10 seconds Dataset: *normal_traffic* (Ericsson AB) 29
4.8 Processing Time with attack using Count-Min & HeavyKeeper vs Hash Map & Priority Queue, Dataset: Ericsson & CAIDA DDOS 2007 31
4.9 Memory Usage with attack using Count-Min & HeavyKeeper vs Hash Map & Priority Queue, Dataset: Ericsson & CAIDA DDOS 2007 . . 32
4.10 Maximum response time, with the filled line representing the `detect-timewindow` API using the statistical model and the dotted line representing the `detect-ml` API utilizing the Isolation Forest method. 34
4.11 Correct Detections comparison between the three difference configurations. 35
4.12 Memory Usage summary of the Baseline Configuration and the HP Configuration during an attack, Dataset: Ericsson & CAIDA DDOS 2007 36

List of Tables

3.1 Comparison of Element Counting between Count-Min Sketch and Hash Map, where k is the number of hash functions and w is the number of buckets (width) 21
3.2 Time and Space Complexity for Redis Top K and Merge Sort, where K is the number of elements to be retrieved and k is the number of hash functions 21

4.1 True Positives and True Negatives for different datasets 23
4.2 False Positives and False Negatives for different datasets 24
4.3 Accuracy metrics for Ericsson data without artificial attacks 25
4.4 Summary of detection latency data 30
4.5 Hash Map & Priority Queue with attack performance 30
4.6 Machine learning model with attack performance, Dataset: CAIDA DDOS 2007 33
4.7 Machine learning model without attack performance. Dataset: CAIDA DDOS 2007 33
4.8 Comparison between the Statistical and Isolation Forest (IF) Models 34
4.9 Performance Comparison: Statistical Model vs Machine Learning Model 37

1

Introduction

In recent years, corporate digitization has seen a significant increase in light of the increased usage of the internet by everyday people [1]. As a result, the volume of sensitive data sent over corporate networks has become a significant risk factor when setting up the internal infrastructure. Therefore, network intrusion detection systems, also known as NIDS, have become a vital component of such infrastructures to ensure that malicious attacks can be detected and sometimes even mitigated as soon as possible [2]. The continuous nature of network traffic makes this a challenging task. In other words, the storage of information from a never-ending data stream grows over time. This, in turn, creates a need for constant maintenance and upgrades to the system's memory capacity, which becomes increasingly expensive as the system scales. This creates a need for a NIDS that can scale efficiently as the usage of the systems increases over time. Several tools and techniques have surfaced to counteract this, such as windowing, data sketches, micro-clusters, and more [3]. Such solutions allow for continuous flushing of old redundant information or use minimal space required for the task. We propose using these methods and data structures to record patterns that represent normal traffic flow and raise an alert if the traffic deviates from such patterns. We achieve this by delving into the current state-of-the-art techniques for detecting attacks. The goal was to develop a more memory and computationally efficient solution to scale with the system it protects.

1.1 Context

Currently, preventing DDoS attacks is a hot topic in the research field. As Guptan et al. [4] in their paper on DDoS prevention techniques and Suthar et al. [5] in their survey paper on DDoS detection and prevention show there exist DDoS prevention techniques that can achieve high accuracy. However, both papers highlight a significant tradeoff between high accuracy and memory or computation complexity. This is further reinforced by Abdulganiyu et al. [6] in their survey paper on NIDS in general, where they emphasize that most effective NIDS implementations show much promise from a theoretical standpoint but are arduous to implement in practice due to their high demands on memory capacity or computational load requirements. That is why this paper explores ways to effectively detect and prevent DDoS attacks while maintaining low memory usage and low computational load by leveraging data sketches and statistical approximations.

1.2 Aim

The primary objective of this thesis is to design and implement a scalable anomaly-based network intrusion detection system capable of efficiently detecting distributed denial-of-service attacks while minimizing memory and computational overhead. The effectiveness of this system was evaluated against existing solutions to assess its capability in identifying contemporary network attacks. To achieve this, the research was to explore existing anomaly-based detection techniques and investigate the application of data sketches to optimize memory usage. Furthermore, a statistical model was developed to detect anomalies in network traffic. The integration of the model with data sketches was examined to improve efficiency and detection accuracy. Finally, the proposed system was evaluated based on performance metrics, including false positive rates, memory efficiency, and computational cost.

2

Background

This chapter provides the necessary background knowledge and foundational concepts that are important for understanding the approach and methodology of this study. It begins with an overview of Network Intrusion Detection Systems (NIDS), their importance in modern cybersecurity, and the challenges faced by traditional detection methods in the face of growing and evolving network threats. The chapter also delves into statistical and machine learning techniques, such as the Interquartile Range (IQR) Rule and Isolation Forest, in the context of anomaly detection. Furthermore, the use of data sketches like Count-Min Sketch and HeavyKeeper Sketch, which optimize the detection and tracking of malicious network traffic, is explored.

2.1 Network Intrusion Detection Systems

Network security has become an increasingly critical issue due to the rapid digitization of industries and the growing number of cyber threats. Organizations require robust and scalable network intrusion detection systems (NIDS) to counter unauthorized access and cyberattacks. Traditional rule-based approaches struggle with efficiency when dealing with large-scale network traffic, while contemporary anomaly-based detection methods often suffer from high false-positive rates and excessive memory consumption.

NIDS are usually categorized into signature-based and anomaly-based detection systems. Signature-based systems rely on predefined patterns or signatures of known attacks. While these methods are highly effective at detecting known threats with minimal false positives, they fail to identify novel attacks or zero-day exploits [7]. In comparison, anomaly-based systems leverage machine learning or statistical models to detect deviations from normal network behavior, enabling them to identify unknown threats. However, they often require careful tuning to reduce false positives and improve reliability [8].

As cyber threats evolve, hybrid approaches combining signature-based and anomaly-based detection are becoming more popular. By integrating behavioral analysis, threat intelligence feeds, and AI-driven models, modern NIDS can provide more robust and adaptive security solutions. Additionally, the adoption of cloud-based NIDS allows organizations to deploy scalable, distributed security mechanisms that can efficiently monitor and respond to network threats in real-time [9].

2.2 Interquartile Range Rule

The Interquartile Range (IQR) Rule is a widely used statistical method for detecting outliers in a dataset [10]. It is based on the interquartile range, which measures the spread of the middle 50% of the data, making it a robust alternative to methods that rely on standard deviation and normal distribution assumptions.

The IQR is calculated as the difference between the third quartile (Q_3) and the first quartile (Q_1), which effectively captures the range of the central portion of the data:

$$\text{IQR} = Q_3 - Q_1$$

Using this range, the IQR Rule sets specific thresholds to identify outliers. A data point is considered a **mild outlier** if it falls below or above:

$$x < Q_1 - 1.5 \times \text{IQR}$$
$$x > Q_3 + 1.5 \times \text{IQR}$$

More extreme outliers are identified when a data point falls below or above:

$$x < Q_1 - 3 \times \text{IQR}$$
$$x > Q_3 + 3 \times \text{IQR}$$

One of the advantages of the IQR Rule is its robustness against skewed distributions and extreme values, which makes it very reliable. Unlike methods that assume a normal distribution, the IQR approach adapts well to datasets with non-Gaussian characteristics. This makes it useful in fields for network traffic that can change a lot.

By focusing on the middle 50% of the data and disregarding extreme values, the IQR Rule provides a balanced method to avoid being highly impacted by traffic fluctuations. This approach ensures that statistical conclusions are not disproportionately influenced by a few extreme values.

2.3 Distributed Denial of Service Attacks

A Distributed Denial of Service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming it with a flood of internet traffic from multiple sources. The goal of the attacker is to exhaust the target system's resources, rendering it inaccessible to its actual users. Many DDoS attacks use botnets, which are networks of compromised computers, devices, or servers controlled by the attacker. These devices are often unknowingly hijacked through malware and can be remotely directed to send massive amounts of requests to a target system simultaneously. The reason for this is that the attackers

aim to amplify the scale of their attacks while also making it harder to trace the origin of the attack. Cybercriminals often rent out botnet services in underground markets, making DDoS attacks more accessible and widespread. Mitigating DDoS attacks requires advanced security measures such as traffic filtering, rate limiting, and the use of AI-driven anomaly detection to identify malicious traffic patterns [11], [12]. On October 21, 2016, a massive DDoS attack was carried out by a major DNS provider called Dyn. This DDoS attack disrupted websites big like Twitter, Netflix, and GitHub. The attack used a botnet that hijacked IoT devices with weak credentials to launch a 1.2 Tbps volumetric attack, making it one of the largest DDoS incidents in history [13]. The outage lasted hours and affected millions of users across the U.S. and Europe. [14].

2.4 Machine Learning Algorithms

Machine learning plays a crucial role in modern anomaly detection systems by enabling models to learn patterns from data and identify deviations from expected behavior. Generally, machine learning algorithms can be categorized into supervised and unsupervised learning methods.

- Supervised Learning relies on labeled datasets, where the algorithm learns from input-output pairs. This approach is commonly used for classification tasks, such as distinguishing between normal and malicious network traffic. However, obtaining labeled data for intrusion detection can be costly and time-consuming [15].
- Unsupervised Learning, on the other hand, does not require labeled data. Instead, it identifies anomalies by detecting patterns that deviate from the norm. This makes it particularly useful for cybersecurity applications where attack patterns constantly evolve and labeled data is scarce [16].

Due to the unpredictable nature of cyber threats, unsupervised methods, such as Isolation Forest, are increasingly being used in network intrusion detection systems. These models can efficiently identify anomalies in large-scale network traffic with minimal supervision. [17].

2.4.1 Isolation Forest

Isolation Forest is an unsupervised machine learning algorithm designed for anomaly detection. It operates by constructing an ensemble of binary trees, known as Isolation Trees, which recursively partition the data. At each node within a tree, a feature is randomly selected, and a random split value is chosen within the range of that feature. This process continues until each data point is isolated or a predefined tree depth is reached. The underlying principle is that anomalies, being rare and distinct, are more susceptible to isolation with fewer partitions, resulting in shorter path lengths from the root to the terminating node in the tree. Consequently, the average path length across all trees in the forest serves as an indicator of the normality of a data point, with shorter lengths suggesting a higher likelihood of anomaly.[18]

One of the key advantages of Isolation Forest is its efficiency and effectiveness in identifying anomalies without relying on distance or density measures. This characteristic makes it particularly suitable for high-dimensional datasets and capable of handling data with varied distributions. Moreover, the algorithm has a linear time complexity and a low memory requirement, enhancing its practicality for large-scale applications. These attributes have led to its widespread adoption in various domains, including fraud detection, network security, and system monitoring.[19]

2.5 Data Stream

In the context of network traffic, a data stream refers to a continuous flow of data packets transmitted over a network between two endpoints or multiple points. These endpoints could be devices, applications, servers, or other network entities communicating over a defined protocol.

2.6 Windowing

Windowing, in the context of data streams, is a technique used to divide incoming data into more manageable clusters based on the time in which they are received [20]. In the research of time windows, multiple techniques have emerged, two of which are tumbling windows and sliding windows. The tumbling windows technique works by immediately transitioning to the next window once a specific time period has elapsed. However, in the context of anomaly detection, this poses a problem: an anomaly that begins just before a window ends may bleed over into the next window. As a result, the anomaly could either be detected as two separate anomalies or fail to be detected at all. To address this issue, sliding windows are proposed, where each window overlaps with its neighboring windows, ensuring that no direct split of information occurs.

2.6.1 Tumbling Windows

Tumbling windows are a type of time-based windowing technique where the data stream is divided into fixed, non-overlapping time intervals [20]. Once a window reaches its predefined duration, it closes, and a new window begins. This ensures that each data point belongs to only one window. Tumbling windows are useful for batch processing and computing statistics [21]. However, a key limitation of tumbling windows in anomaly detection is that anomalies occurring near the boundary of a window may be split, which can lead to inaccurate detection results [22].

2.6.2 Sliding Windows

Sliding windows, compared to tumbling windows, provide a more flexible approach by allowing consecutive windows to overlap; this ensures that data points can appear in multiple windows [20]. This technique mitigates the issue of boundary splits seen in tumbling windows, which makes it suitable for anomaly detection and real-time

monitoring [23]. Sliding windows are defined by two parameters: the window size, which determines the total duration of the window, and the slide interval, which specifies how frequently a new window starts. By overlapping windows, sliding windows provide a more continuous analysis of data streams. This ensures that anomalies that span across window boundaries are captured more accurately [24]. However, this increased accuracy comes at the cost of higher computational complexity, as more windows need to be processed simultaneously.

2.7 Data Sketches

Data sketches, also known as probabilistic data structures, are specialized structures that give approximate answers to queries using sublinear memory; this makes them highly efficient for large-scale data processing [25]. In contrast to traditional data structures that store raw data, data sketches rely on probabilistic techniques to estimate key statistics while reducing storage and computational overhead [26].

Data sketches are especially valuable in data streaming where each data point is processed only once. This makes them well-suited for real-time analytics and scenarios with large datasets [27]. In such cases, maintaining exact counts or statistics is impractical due to memory constraints, and using data sketches provides an efficient alternative. These structures allow fast approximations of various metrics such as frequency counting, quantile estimation, and similarity detection across large datasets [28].

2.7.1 Count-Min Data Sketch

The Count-Min Data Sketch is a probabilistic data structure designed for efficiently estimating the frequency of elements in a data stream while using less memory than exact counting methods [29]. It is very useful in scenarios where keeping exact counts is infeasible due to memory constraints and is often used in network security due to large amounts of traffic flows.

The Count-Min Sketch consists of a two-dimensional array (matrix) of counters with w columns and d rows. Each row has an associated hash function that maps incoming elements to a column index. The add operation takes the new element in the stream and updates one counter in each row, incrementing the corresponding cell determined by the respective hash function. The query operation estimates the frequency of an element, to do so the Count-Min Sketch gets the minimum value across all the counters associated with the elements hash mappings. This ensures that the estimate never undercounts but may slightly overcount due to hash collisions [29]. The probability of overestimation is controlled by choosing appropriate values for w and d, which determine the trade-off between accuracy and memory usage. It requires only $O(1)$ time complexity for updates and queries. Its space complexity is $O(w \cdot d)$, where $w = O(1/\epsilon)$ and $d = O(\log(1/\delta))$, making it scalable for large datasets while providing an accuracy guarantee of ϵ with probability $1-\delta$ [29]. Here, δ represents the maximum probability that the estimated frequency will exceed the true frequency by more than ϵ, essentially defining the confidence level of the sketch.

Despite its advantages, the Count-Min Sketch has limitations. The primary drawback is overestimation due to hash collisions, which can be problematic in cases where exact frequency counts are required and low memory usage is not the main focus. Additionally, it does not support deletions (decrementing counts) [30].

2.7.2 HeavyKeeper Algorithm

The HeavyKeeper algorithm or HeavyKeeper data sketch is a probabilistic data structure designed to efficiently track the most frequently occurring elements also known as the top-k elements in a list or data stream. Unlike exact frequency counters, which require storing every element, HeavyKeeper sketches keep an approximate ranking of the highest-frequency items [31].

The idea behind a HeavyKeeper sketch is to process streaming data in a way that keeps track of only the most relevant elements. As new elements arrive in the stream, the sketch updates frequency estimates. This ensures that only the most frequent items are retained while lower-frequency elements are discarded or replaced. This enables efficient querying of the top K elements at any time, without requiring storage proportional to the total number of unique items in the stream [31].

Just like the aforementioned Count-Min Data Sketch, the HeavyKeeper Data Sketch has advantages in scenarios where identifying the most frequent items and memory efficiency is more important than maintaining exact counts for all elements. They are widely used in areas such as search engine query analysis, where they help identify trending searches; network traffic monitoring, where they track the most accessed IP addresses; and database query optimization, where they help in identifying frequently accessed records.

One of the main limitations of the HeavyKeeper sketch is that it can introduce false positiveselements that appear in the Top-K list due to hash collisions or estimation errors. Additionally, it may not provide precise rankings, as frequency approximations have small errors [31].

2.8 Datasets

Datasets of prerecorded segments of network traffic provide a reliable and replayable way of simulating different scenarios that can occur in a live network. These datasets are used to evaluate and observe behaviors in the system proposed in this paper.

2.8.1 CAIDA2007

The CAIDA2007 dataset [32] captures approximately one hour of anonymized network traffic recorded during a distributed denial-of-service (DDoS) attack that occurred on August 4, 2007, between 20:50:08 and 21:56:16 UTC. The attack aimed to overwhelm the targeted server by exhausting its computing resources, thereby disrupting legitimate access. In this study, we utilized the CAIDA2007 dataset as it was readily available to us at the outset of the research. While the CAIDA2019

dataset contains more recent traffic data, the process to gain access to CAIDA2019 typically involves a lengthy approval procedure that can take several months. Consequently, utilizing the CAIDA2007 dataset was a more time-efficient choice, allowing us to proceed with the analysis without significant delays.

2.8.2 Ericsson Data

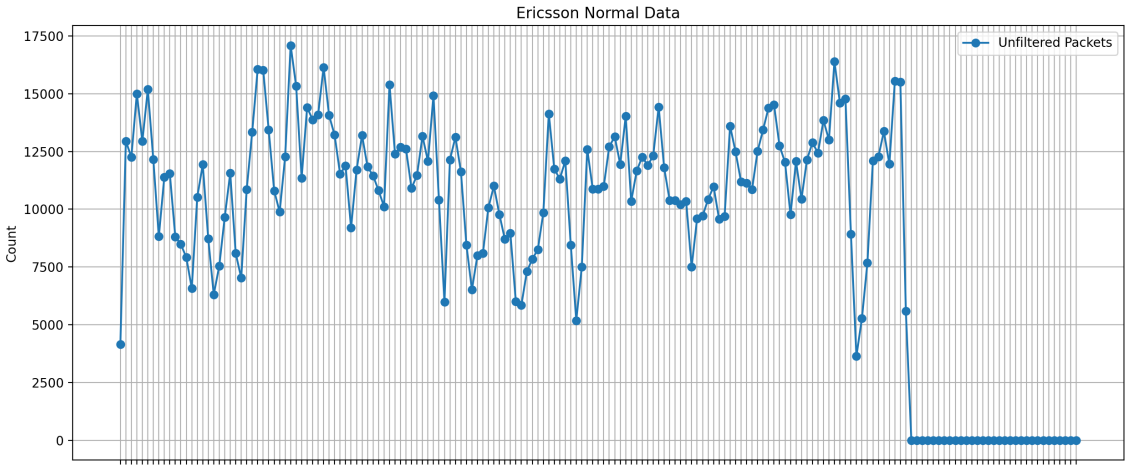


Figure 2.1: Ericsson Dataset

The Ericsson data was provided by Ericsson AB and is a simulation of normal traffic that their internal systems use to evaluate how their it handles traffic. It is synthetically generated data by their internal systems.

3

Method

This section highlights the methodologies used to design, implement, and evaluate the proposed intrusion detection system. The problem tackled by the proposed system is to detect DDoS attacks with high detection accuracy and low latency while also maintaining low memory utilization. To achieve this, a scalable anomaly-based detection framework is combined with a statistical model and data sketches to identify DDoS attacks. The approach presented utilizes a real-time system integration to achieve efficient and accurate detection.

3.1 Comparative Analysis of Anomaly Detection Methods

To lay the groundwork for the NIDS, an analysis of existing anomaly detection methods for DDoS attacks is necessary. This comparison helps identify the strengths and weaknesses of different techniques and improves the understanding of their applicability to Ericssons network environments.

We focus on evaluating methods based on several aspects: scalability, adaptability to dynamic network traffic, detection accuracy, false positive rates, and computational efficiency. We also explore the potential of using data sketches, particularly the Count-Min Sketch as a preprocessing step for handling large-scale network traffic and how they have been used before in similar scenarios. The Count-Min Sketch provides an efficient frequency estimation of packet occurrences, with minimal memory usage.

This comparative study also helps identify gaps in existing research and justifies our area of research for the thesis. We also use Golang for its low latency, high-performance execution as the load increases [33] making it a good choice for a real-time network environment.

3.2 Implementation

In this section, we cover the implementation of the IDS, from data collection to intrusion detection. This includes the technologies and techniques used to achieve the same results.

3.2.1 System Architecture

To achieve anomaly detection, a NIDS is implemented, accompanied by a pilot environment to visualize the performance of the system in real time.

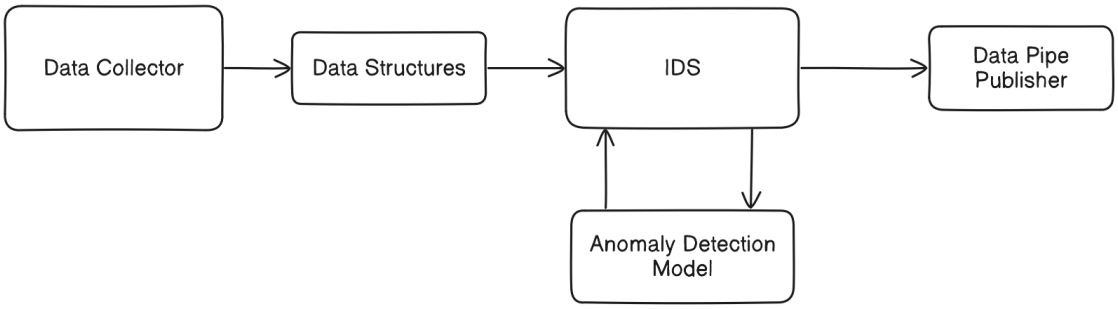


Figure 3.1: Baseline Configuration System Architecture Diagram

The system follows the architecture shown in Figure 3.4 where the core of the system is composed around 5 primary nodes. The data collector collects and preprocesses data before it is stored within the data structures. The IDS regularly queries the data structures, formats the result, and sends the data to an anomaly detection model. The model then responds with true or false depending on if an attack has occurred; if so, the IDS then publishes a report to the data pipe. Additionally, there are external nodes that can subscribe to the data pipe and take action when an attack has occurred. Different configurations are made using this architecture overview; a Baseline Configuration representing the system proposed in this paper, another configuration using machine learning, and one using a hash map and priority queue instead of probabilistic data structures. The two latter ones are used for comparisons in the evaluation.

3.2.2 Configurations

This study involves creating different configurations, each of which utilizes different techniques, such as using a combination of a Hash Map and a priority queue instead of probabilistic data structures, and machine learning instead of the statistical model. This simplifies the evaluation of the proposed solution since each configuration can be used on the same tests and then compared.

3.2.2.1 Baseline Configuration

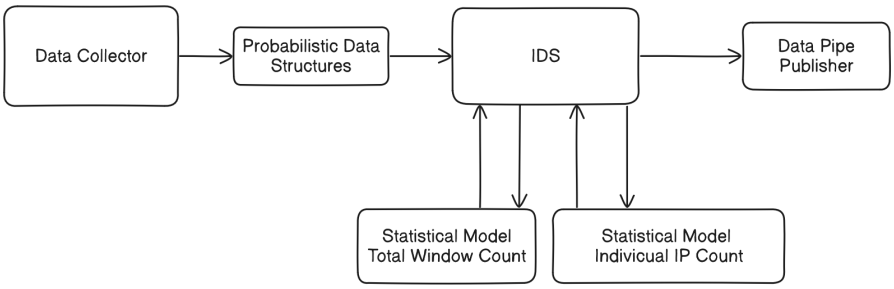


Figure 3.2: Baseline Configuration System Architecture Diagram

The Baseline Configuration serves as the core setup for anomaly detection in this thesis and follows the architecture shown in Figure 3.2. It combines a statistical model with probabilistic data structures such as Count-Min Sketch and the Heavy-Keeper Sketch. These structures are chosen for their low memory usage and efficiency in handling large-scale network traffic. Count-Min Sketch tracks frequency counts, while the HeavyKeeper Sketch keeps track of the most frequent network flows. The detection method uses an Interquartile Range (IQR) based statistical model to dynamically adjust the detection threshold. This ensures the system can accommodate fluctuations in normal traffic without generating false positives. The threshold is calculated using the 75th percentile (Q3) and an IQR offset, allowing it to adapt to varying network traffic levels. Additionally, the sliding window technique is used to process data in real-time, breaking the traffic into smaller, manageable segments. Each window is independently evaluated for anomalies, ensuring fast and efficient detection without overwhelming the system's resources. This configuration is evaluated against other setups, such as the Hash Map & Priority Queue (HP) and Machine Learning (ML) configurations, to demonstrate its effectiveness in terms of accuracy, memory, and processing time.

3.2.2.2 Hash Map Priority Queue (HP) Configuration

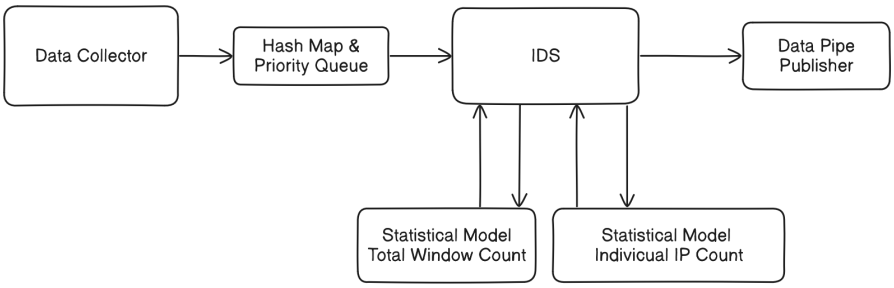


Figure 3.3: Hash Map & Priority Queue Configuration System Architecture Diagram

The Hash Map Priority Queue configuration (HP Configuration) is used for testing a similar data collection method without using probabilistic data structures. The architecture for the configuration can be seen in Figure 3.3. Instead of using the

HeavyKeeper algorithm, it uses a Priority Queue, and for keeping a count of individual IP addresses, it uses a Hash Map.

3.2.2.3 Machine Learning (ML) Configuration

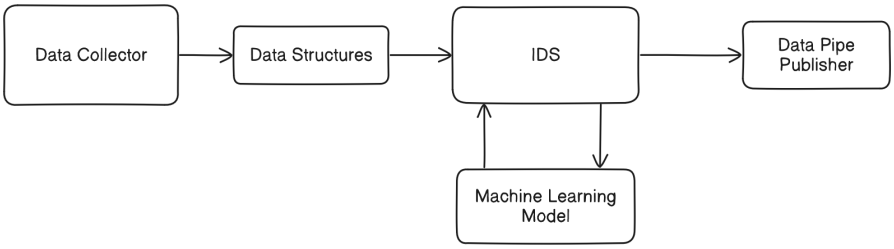


Figure 3.4: Machine Learning Configuration System Architecture Diagram

The Machine Learning Configuration (ML Configuration) serves as an alternative anomaly detection method, based on unsupervised machine learning. In this setup, the system uses the Isolation Forest algorithm to identify anomalous traffic by isolating outliers. It is implemented using the default parameters, without any additional tuning. This is implemented to serve as a baseline comparison against the Baseline Configuration with a statistical model. The Isolation Forest algorithm is specifically chosen because it is unsupervised and does not require labeled data. This makes it more flexible and adaptable for testing across various types of datasets.

3.2.3 Data Collection

To simulate real-world network traffic, a PCAP reader is developed. The component reads network data pre-collected in a PCAP file, extracts the relevant headers, and sends that information to the IDS. The relevant headers needed for detection are the time and source IP address. In order to simulate the time between packets, the process responsible for reading the PCAP file sleeps for a computed delay based on the time header of the previous and next packet.

To clarify, let T_b represent the timestamp of the last packet sent and T_c the timestamp of the next packet to be sent. The intended delay between the two packets is then calculated as D_i.

$$D_i = T_c - T_b \tag{3.1}$$

To emulate the data as closely as possible, we need to account for the processing time that the simulator needs before the process starts sleeping, denoted as D_a. This is done by saving the system time for when the last packet is processed, represented as T_r minus the current system time T_s.

$$D_a = T_s - T_r \tag{3.2}$$

The resulting delay time S is then calculated by taking the intended delay D_i minus the processing time D_a. If the processing time takes longer than the intended delay, it sets the resulting delay to 0 so that the next packet is sent immediately.

$$S = \max(0, D_i - D_a) \tag{3.3}$$

3.2.4 Data Preprocessing

The data processing step is crucial for keeping memory utilization low since this is the step where only the relevant information needed to complete the assigned task is collected and allows us to filter out the rest. Here, sliding windows are utilized to bundle the information into distinct time windows. Each window includes 2 data structures, one of which keeps a sorted list of the topmost prevalent IP addresses while the other keeps a count for each IP address.

3.2.4.1 Data Aggregation

To manage data aggregation, a new process is started to update the time windows according to the sliding windows technique. This process loops ten times each time window to check whether or not it needs to be updated. Each update checks the current windows that should be active based on the current system time and the overlap percentage.

To clarify, each loop iteration fetches a list of the current time windows that should be active based on the system time. If a window is open that does not exist within that list, it is closed. Furthermore, if there does not exist a window containing an entry in the list, a window is opened. When a window is closed, the data from the corresponding sketches and counts are then sent for intrusion detection.

3.2.4.2 Data Structures

As mentioned above, each window contains two data structures. The data structures mentioned depend on the configuration used. To elaborate, within the Baseline Configuration, each time window includes a count-min and HeavyKeeper data sketch. The HeavyKeeper sketch is used to keep track of the top k most prevalent flows in the network. To clarify, it keeps a sorted list of the most frequent IP addresses within the system, while the Count-Min sketch is used to keep the counts of these IP addresses. These combined effectively provide an estimated sorted list of the most frequent IP addresses with an estimation of their counts using sublinear memory complexity within each window. In contrast, the HP Configuration uses a Priority Queue for the sorted list and a Hash Map to keep counts.

3.2.5 Anomaly Detection

The preprocessed traffic data is analyzed using a statistical model, which detects deviations from normal traffic patterns and flags potential DDoS attacks. The statistical model is based on the interquartile range and sets a threshold, which acts as the limit that differentiates normal from malicious behavior. It floats slightly above the normal data to allow for some fluctuation and can change the dynamic threshold based on its behavior.

3.2.5.1 Packet Threshold per Time Window

The system monitors the total number of packets within a fixed time window, in this case, 1 second. The threshold is determined using the 75th percentile (Q3) and an interquartile range-based adjustment, as can be seen in Figure 4.2. If the attempt count exceeds this threshold, the time window is flagged as malicious. The packet thresholds for each IP address are temporarily frozen when a time window is flagged as malicious to prevent updating on malicious DDoS data, which can skew the threshold. The model updates based on incoming JSON data via its API endpoint `/detect-timewindow` and responds with its decision if it exceeds the packet count for that time window. This might look like the following listings. The threshold is included only for illustration.

Request:

```
{
  "timestamp": "2024-03-12T14:00:00Z",
  "total_attempts": 24869
}
```

Response:

```
{
  "timestamp": "2024-03-12T14:00:00Z",
  "isMalicious": true,
  "threshold": 14380
}
```

3.2.5.2 Packet Threshold per IP Address

To detect individual IP addresses with excessive amounts of packets, the model updates based on both normal and malicious data if the time window is flagged as normal. It updates based on malicious IP addresses to prevent wrongly flagged IPs within that time window and to make the threshold more dynamic to increasing amounts of normal data. The threshold is calculated using Q3 (75th percentile) and a standard deviation offset to give some space for smaller spikes that are not DDoS attacks. If an IP surpasses this limit, it is marked as malicious and stored in Redis. This model also updates based on incoming JSON data via its API endpoint `/detect-ips` and responds with its decision for every IP address in the list whether it is malicious or not. This might look like the following listings. The threshold is included only for illustration.

Request:

```
[
  {
    "ip_address": "192.168.1.10",
    "total_attempts": 600
  },
```

```
  {
    "ip_address": "192.168.1.20",
    "total_attempts": 150
  },
  {
    "ip_address": "192.168.1.30",
    "total_attempts": 175
  }
]
```

Response:

```
{
  "threshold": 400
}
```

3.2.5.3 Setting the Threshold

$$Q1 = 25\text{th percentile of normal traffic} \tag{3.4}$$

$$Q3 = 75\text{th percentile of normal traffic} \tag{3.5}$$

$$IQR = Q3 - Q1 \tag{3.6}$$

$$\text{Threshold} = \max(Q3 + 1.5 \times IQR, 200) \tag{3.7}$$

1.5 x IQR is added to the third quartile. Any number greater than this is a suspected outlier. 1.5 x IQR is subtracted from the first quartile. Any number less than this is a suspected outlier [34]. To further adapt to dynamic changes in traffic, we incorporate a standard deviation-based offset to provide a margin of safety.

$$\text{Final Threshold} = \text{Threshold} + 2\sigma \tag{3.8}$$

where σ represents the standard deviation of normal traffic. A base threshold of 200 is used to create a margin of safety in order to avoid having the threshold get stuck at 0. This number is set through trial and error for this particular scenario. It is worth noting that this could be changed to a number that is better suited for another scenario. The IQR Q3 rule is chosen as a statistical approach since the lower chunk of the normal data is not relevant in the context of finding a threshold. Therefore, only the upper part of the normal data is taken into account to mitigate the risk of having the system slowed down.

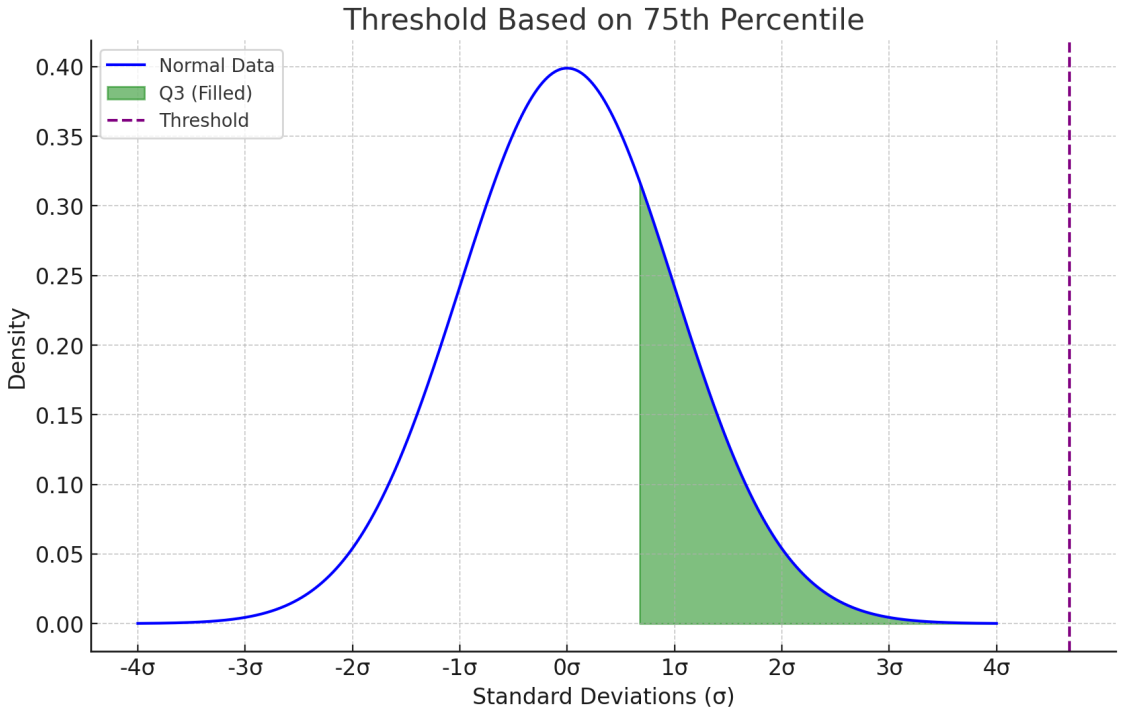


Figure 3.5: Threshold based on normal data.

3.2.6 Data Pipe Publisher

An observer pattern is implemented by using a data publisher to which external systems can subscribe. For example, such external systems can be an Intrusion Prevention system or a Logging system. The data published to the pipe is whether or not an attack is detected and a suggested rate limit for each IP equivalent to the IP threshold.

3.3 Metrics

This section highlights how the evaluation phase of the project was carried out in the results. To assess the efficiency of our proposed NIDS, a series of tests is composed that shows the applicability of such a system if it were to be used in practice. The evaluation is divided into several steps where the first step presents the detection rate of the system, showing how well it achieves its assigned goal. The second step evaluates the threshold scalability, showing that the system works efficiently over time as the usage of the network scales up. Thirdly, we compare the statistical model to a more machine learning-based approach. Lastly, in the third step, the resource utilization is evaluated, highlighting that the system can handle large throughput and does not fail under higher traffic loads.

3.3.1 Anomaly Detection

To emphasize the applicability of our proposed NIDS, it is imperative that the system works for its intended purpose. To assess whether the system can accurately detect attacks, multiple tests are conducted on several datasets. The tests provide the following evaluation metrics:

- **Detection Accuracy:** The ability of the system to correctly classify both attack and normal traffic. It measures the overall correctness of the model.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **False Positive Rate (FPR):** The rate at which normal traffic is incorrectly flagged as anomalous.

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} = \frac{FP}{FP + TN}$$

- **Precision:** The proportion of correctly identified DDoS attacks among all predicted attacks. A higher precision means fewer false positives.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} = \frac{TP}{TP + FP}$$

- **Computational Efficiency:** Measured in terms of CPU load, memory utilization and processing speed.
- **Scalability:** The system's ability to handle increasing network traffic loads without significant performance degradation.

3.3.2 Threshold Scalability

To determine whether or not the system can handle network traffic that increases over time, a scalability test is conducted. This is done in two ways; firstly, a specialized test is conducted where the packet count is artificially increased by a set amount i every nth window:

$$P_t = P_{t-1} + i \cdot \mathbb{1}(t \mod n = 0)$$

To clarify, P_t represents the total number of packets at time t, indicating the system's current load. The previous packet count, denoted as P_{t-1}, reflects the number of packets before the latest update. The term i is a fixed value representing the additional packets introduced at specific intervals, while n defines the frequency of this increase, meaning that every n time steps, the packet count rises by i. The expression $\mathbb{1}(t \mod n = 0)$ is an indicator function that determines whether t is a multiple of n. It evaluates to 1 when the condition is met, triggering the packet increase, and remains 0 otherwise, ensuring that the count only increments at designated intervals.

Secondly, we conduct a similar test but instead of the linear increase, we have an exponential increase where the traffic is increased by a factor of p every nth window:

$$P_t = P_{t-1} \cdot p^{\mathbb{1}(t \mod n=0)}$$

To clarify, P_t represents the packet count at time t, which reflects the total number of packets at that particular point in time. P_{t-1} is the packet count from the previous time step, or the number of packets immediately before the update. The term p is the growth factor, a constant value by which the packet count is multiplied, leading to an exponential increase in traffic. The factor p determines how much the traffic grows during each update.

The value n specifies the interval of time steps at which the packet count is updated. Every nth time window, the packet count is multiplied by p, thus simulating an exponential growth in the packet count. The expression $\mathbb{1}(t \mod n = 0)$ is an indicator function that checks whether t is a multiple of n. It returns 1 when the condition is true, causing the packet count to be scaled by p, and returns 0 otherwise, meaning no multiplication happens at non-multiple time steps.

These tests allow us to observe how the threshold adapts to different increases in traffic and determine if a normal increase in traffic over time falsely triggers the system of an attack.

3.3.3 Comparing the Statistical Model to Machine Learning

Comparing the statistical model to a machine learning model for anomaly detection is important because it allows us to compare their performance in terms of both accuracy and processing time. Statistical models are often simpler and require fewer computational resources, making them faster and more efficient. However, they may not always capture complex patterns or non-linear relationships in data. We compare with machine learning in particular since it is the most widely used method for dynamic anomaly classification. By testing both aspects, we determine which method offers the best trade-off between accuracy and efficiency, ensuring we select the most suitable model for the system.

3.3.4 Detection Latency

The detection latency of an IDS is the time it takes from an attack to start until it is detected by the system. This is determined by saving the exact system time when the attack is initiated in unison with saving the system time of the first occurrence of a detected anomaly:

$$\text{Latency} = T_{\text{detection}} - T_{\text{attack}}$$

This shows the system's worst-case response time for detecting attacks. If the latency is low enough, it indicates that the system is a viable option to utilize in practice.

3.3.5 Resource Utilization

The resource utilization analysis of the system is conducted in two ways: theoretically and practically. Most of the data structures used are well-documented with precise metrics of the memory and lookup complexities. This allows for a theoretical analysis going over the data structures used. The total complexity can then be compared to the same system as if it were to use different data structures. The practical analysis is done through monitoring and logging the resources used by the process that handles data preprocessing and aggregation. As a result, these tests show how robust the system is against overloading the computer due to high traffic load.

3.3.6 Counting Element Frequency

Aspect	**Count-Min Sketch**	**Hash Map**
Space Complexity	$O(k \cdot w)$	$O(n)$, (if all elements are unique)
Time Complexity	$O(k)$	$O(n)$
Accuracy	Approximate	Exact

Table 3.1: Comparison of Element Counting between Count-Min Sketch and Hash Map, where k is the number of hash functions and w is the number of buckets (width)

The Table 3.1 compares the time and space complexities of the Count-Min Sketch and Hash Map in terms of counting element frequencies. In this implementation, it is used to count the number of IP addresses. Count-Min Sketch is a probabilistic data structure that uses multiple hash functions and a fixed-width table to approximate frequencies [35]. Its space complexity is $O(k \cdot w)$, where k is the number of hash functions and w is the width of the table. While its time complexity is $O(k)$, it gives approximate results with a trade-off in accuracy but is more efficient. In contrast, a Hash Map offers exact frequency counts with a space complexity of $O(n)$ (if all elements are unique) and a time complexity of $O(n)$ for both insertion and retrieval [36]. The Hash Map is more accurate but less memory efficient, especially with large data, which is encountered in network traffic.

3.3.7 Sorting

Aspect	**Redis Top K**	**Merge Sort**
Time Complexity	$O(K \cdot \log(k))$	$O(n \cdot \log(n))$
Space Complexity	$O(K)$	$O(n)$

Table 3.2: Time and Space Complexity for Redis Top K and Merge Sort, where K is the number of elements to be retrieved and k is the number of hash functions

The Table 3.2 shows the time complexities of the Redis Top K algorithm and Merge Sort in the context of sorting and retrieving elements. Redis Top K is designed to

efficiently find the top K most frequent elements from a stream of data [37]. Its time complexity is $O(K \cdot \log(k))$, where K is the number of top elements returned. Merge Sort, on the other hand, is a general-purpose sorting algorithm with a time complexity of $O(n \cdot \log(n))$, where n is the number of elements to be sorted [38]. While Merge Sort guarantees sorting of all elements, Redis Top K is optimized for scenarios where only a subset of the data is needed.

3.4 Limitations

The following limitations are encountered:

- The research focuses solely on anomaly-based detection, excluding signature-based methods.
- Due to ethical and privacy constraints, real-world attack datasets are not always available, so open-source benchmark datasets (CICIDS2007, NSL-KDD), Ericsson's internal data and synthetically generated data are used to update the model instead.
- The thesis only focuses on DDoS attacks.
- The implementation focuses on enterprise networks, more specifically Ericsson's networks, excluding large-scale ISP-level traffic.

4

Results

The results are obtained from the experiments conducted to evaluate the performance of the NIDS. The experiments include evaluating the detection accuracy, the rate limit handling, the adaptability of threshold calculations, and the resource utilization of the system. Different metrics such as true positives, true negatives, false positives, and false negatives are used as measurements of the effectiveness of the system in detecting DDoS attacks.

4.1 Baseline Configuration

This section presents the results obtained using the Baseline Configuration, which uses the IQR-based statistical model. It evaluates the system's ability to accurately detect DDoS attacks and handle normal traffic without false alarms. The focus is on key performance metrics such as detection accuracy, rate limiting, threshold evaluation, and detection latency.

4.1.1 Detection Accuracy

To evaluate the applicability of the system this section is dedicated to the results from the detection accuracy tests. Starting with the detection accuracy with normal and attack traffic, and then the accuracy when there is only normal traffic.

4.1.1.1 Accuracy with attacks

To determine the detection accuracy, network traffic containing normal work traffic was replayed through the simulator. For each window, it logs whether or not an attack was detected. A pcap file containing a DDoS attack was then replayed on top of the normal traffic. When the attack was initiated, a field was set to true and logged together with the aforementioned field. If they are the same, we get a true positive or true negative; if not, we get a false positive or false negative. The following results were observed:

Dataset	**Correct Detections**	**True Positives**	**True Negatives**
CAIDA	96.09%	53.35%	42.74%
Ericsson	94.03%	14.93%	79.10%

Table 4.1: True Positives and True Negatives for different datasets

Dataset	False Positives	False Negatives
CAIDA	0.56%	3.35%
Ericsson	1.49%	4.48%

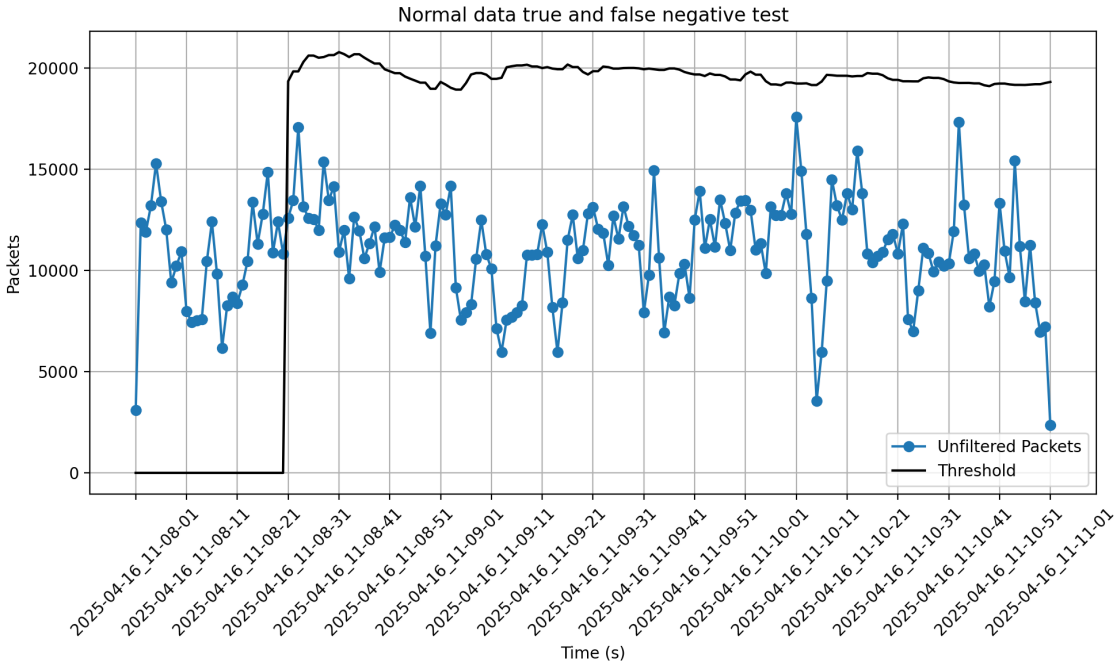
Table 4.2: False Positives and False Negatives for different datasets

4.1.1.2 Accuracy with normal data only

The primary purpose of this test is to verify that our IDS does not mistakenly classify normal network traffic as a DDoS attack. In this case, we are using normal data from Ericsson to assess the accuracy of the IDS under conditions without any artificial attacks.

The results shown in Figure 4.1 indicate that no false positives or false negatives were detected during the test, confirming that the IDS does not misidentify normal data as DDoS attacks. The threshold also gives room for fluctuations within the normal data.

Figure 4.1: Ericsson normal data and threshold



The accuracy metrics, shown in Table 4.3, further illustrate that the system correctly identified all normal data as non-threatening, achieving 100% accuracy with no false positives or negatives.

Metric	Value	Percentage
True Positives	0	0.00%
False Positives	0	0.00%
True Negatives	181	100.00%
False Negatives	0	0.00%
Total Matching Percentage	181	100.00%

Table 4.3: Accuracy metrics for Ericsson data without artificial attacks

4.1.2 Rate Limit

The purpose of incorporating a rate limit into the system is to control the volume of traffic allowed to pass through the network. This helps prevent network congestion and mitigates the impact of DDoS attacks.

Figure 4.2: Unfiltered vs Filtered Packet Counts,
Dataset: rand_src_DDoS_108 provided by Ericsson AB

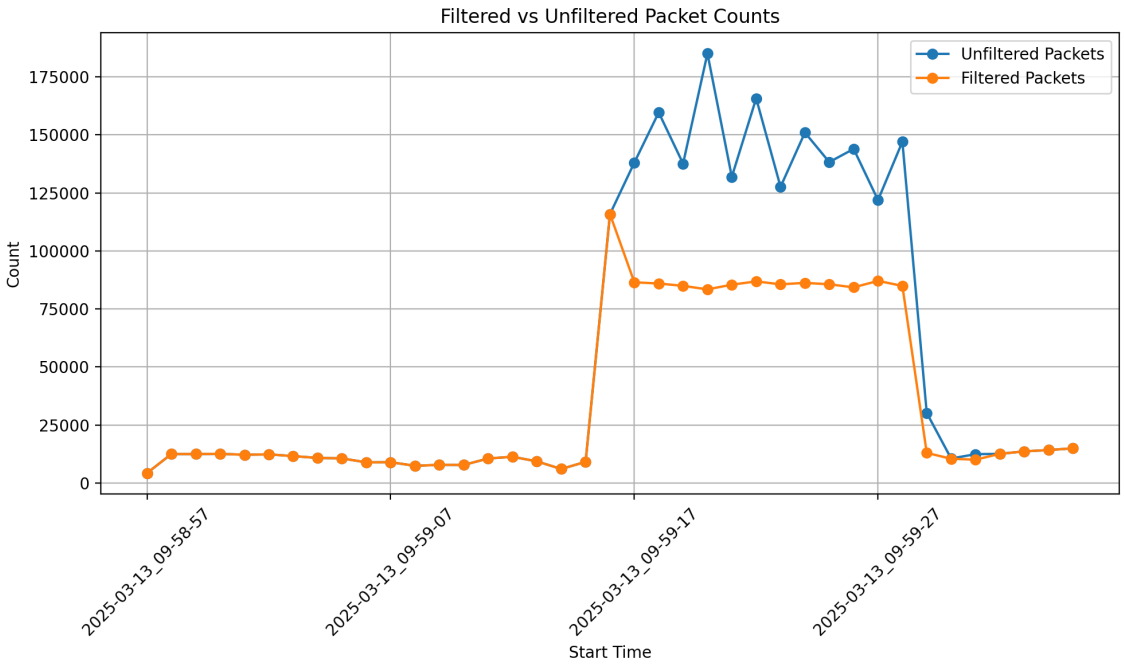


Figure 4.3: Unfiltered vs Filtered Packet Counts,
Dataset: ddostrace.20070804_144936

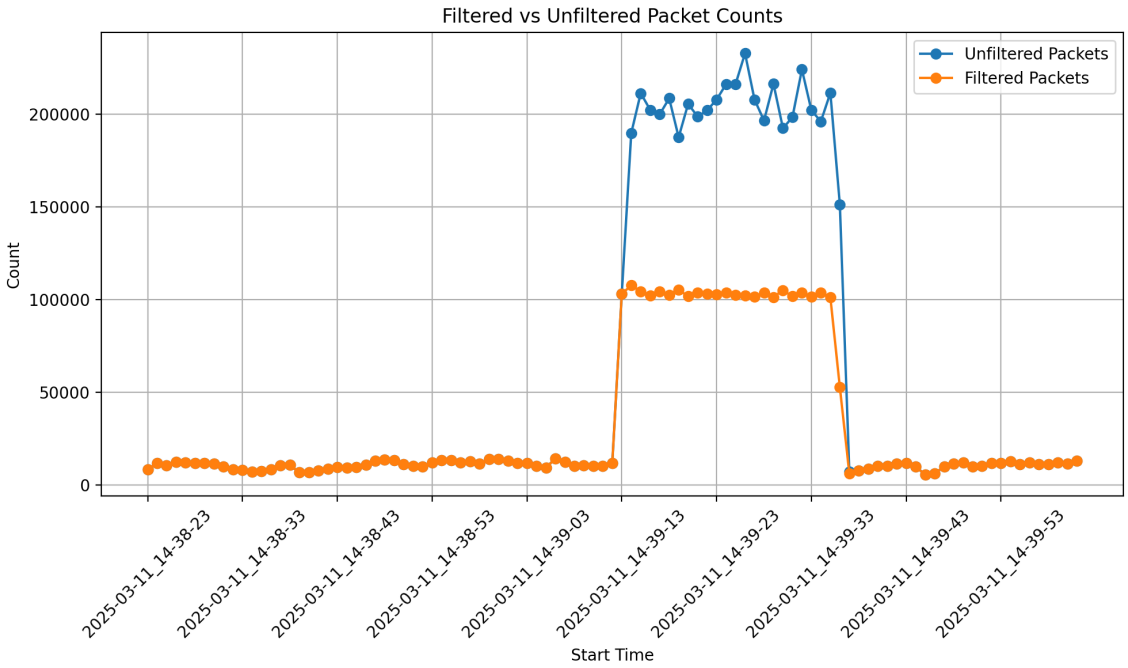


Figure 4.2 and Figure 4.3 present a time-series comparison of unfiltered and filtered packet counts when the NIDS is active. The x-axis represents the time, while the y-axis shows the packet count. The unfiltered packets (blue) show a noticeable spike. The filtered packets (orange) show an increase at the same point, followed by a stabilization at a lower level that is tolerable. This showcases the systems ability to adapt to fluctuations in network traffic and its ability to distinguish between normal and malicious activity.

4.1.3 Evaluating the Threshold

To determine an appropriate time window threshold, we analyze normal traffic behavior using the *normal_traffic* dataset provided by Ericsson AB and evaluate the effect of different threshold calculation methods in the statistical model. This is because we aim to allow for some traffic fluctuations without the system flagging it as an attack.

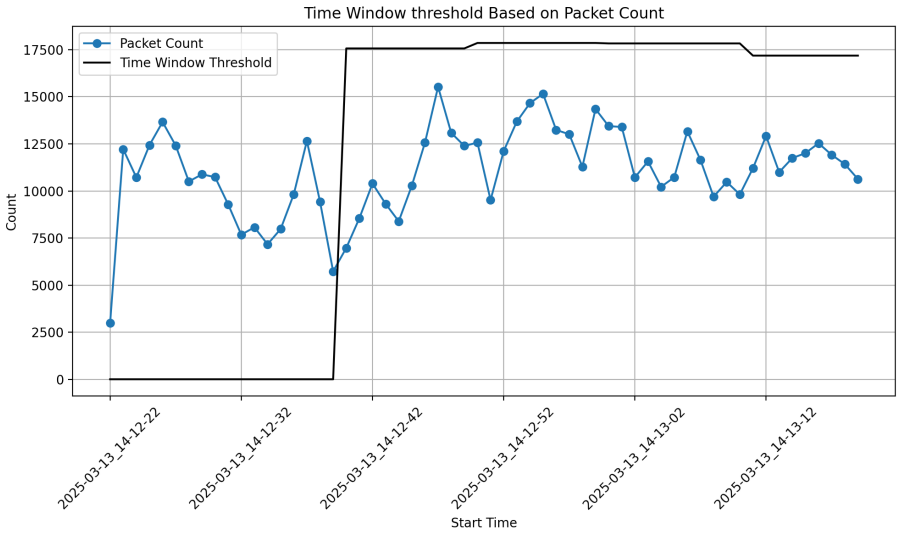


Figure 4.4: Impact of a higher time window threshold on normal traffic. Dataset: *normal_traffic* (Ericsson AB)

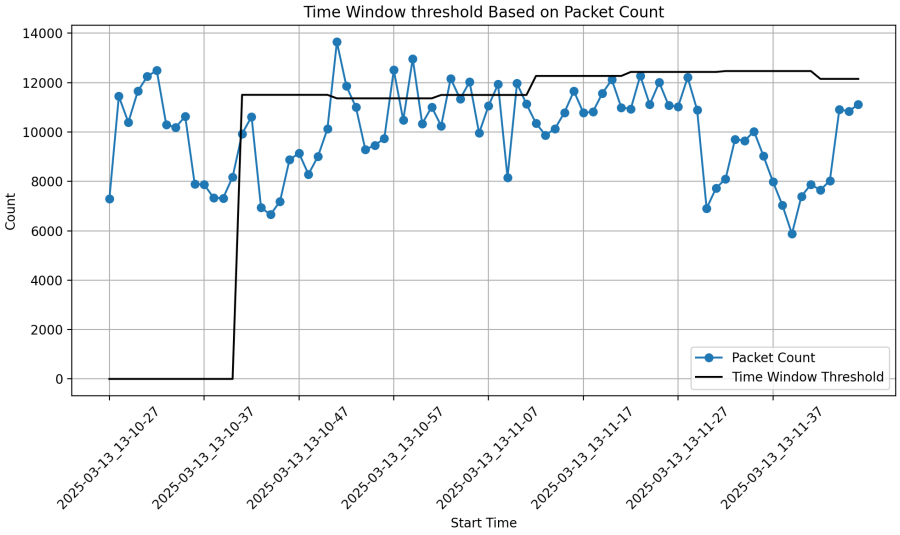


Figure 4.5: Impact of a lower time window threshold on normal traffic. Dataset: *normal_traffic* (Ericsson AB)

As illustrated in Figure 4.4, we apply a threshold calculation based on the Interquartile Range (IQR) method, or more specifically, the IQR 3 rule. This accounts for the

natural variability in normal traffic. An offset of 2 standard deviations is used to make the threshold float above the normal traffic. This combined approach ensures robustness by reducing sensitivity to normal fluctuations while effectively identifying anomalous behavior.

- **Total entries after threshold activation**: 70
- **Count above window threshold**: 0
- **Percentage above window threshold**: 0.00%

On the other hand, a threshold calculation using the 50th percentile, as shown in Figure 4.5, results in:

- **Total entries after threshold activation**: 66
- **Count above window threshold**: 8
- **Percentage above window threshold**: 12.12%

This method calculates the threshold based only on the 50th percentile and not IQR with an offset, making it less responsive to traffic variations, thus having a 12.12% false positive rate in this case since the threshold is too low.

4.1.4 Threshold Adaptability

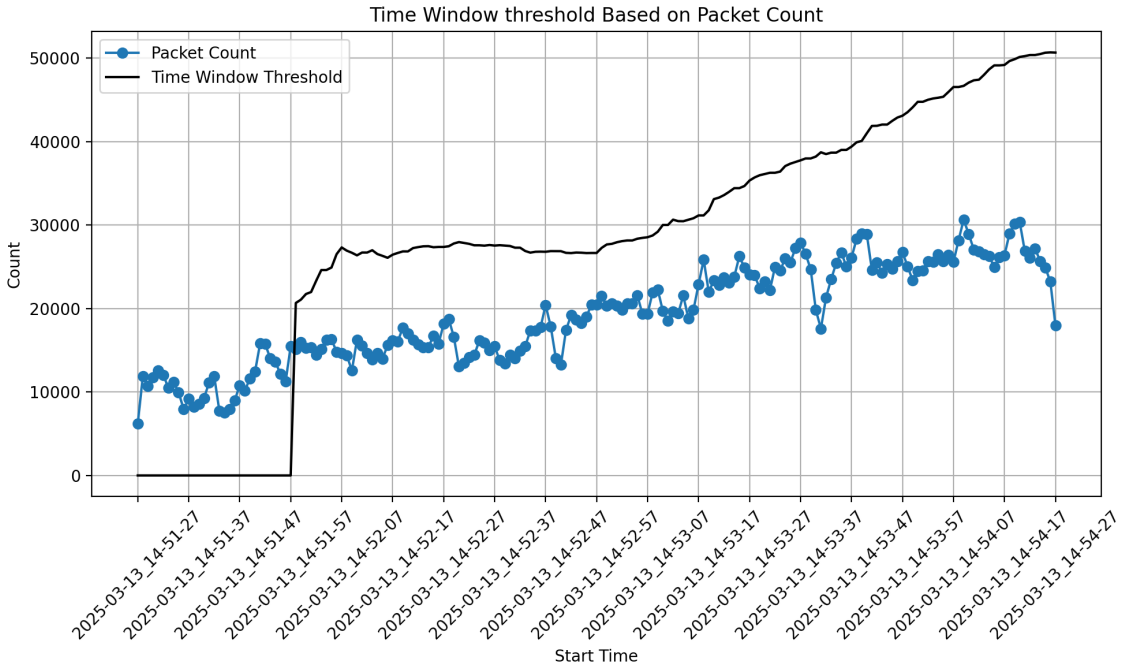


Figure 4.6: Threshold adaptability under increasing normal data.
Dataset: *normal_traffic* (Ericsson AB)

Figure 4.6 demonstrates how the system dynamically adapts its threshold as normal traffic volume increases. The packet count, represented by blue dots, fluctuates but generally trends upward over time. In response, the time window threshold (black line) adjusts accordingly, starting at a lower value and progressively increasing to account for rising traffic. This adaptability ensures that the threshold remains relevant, preventing false positives by accounting for natural fluctuations.

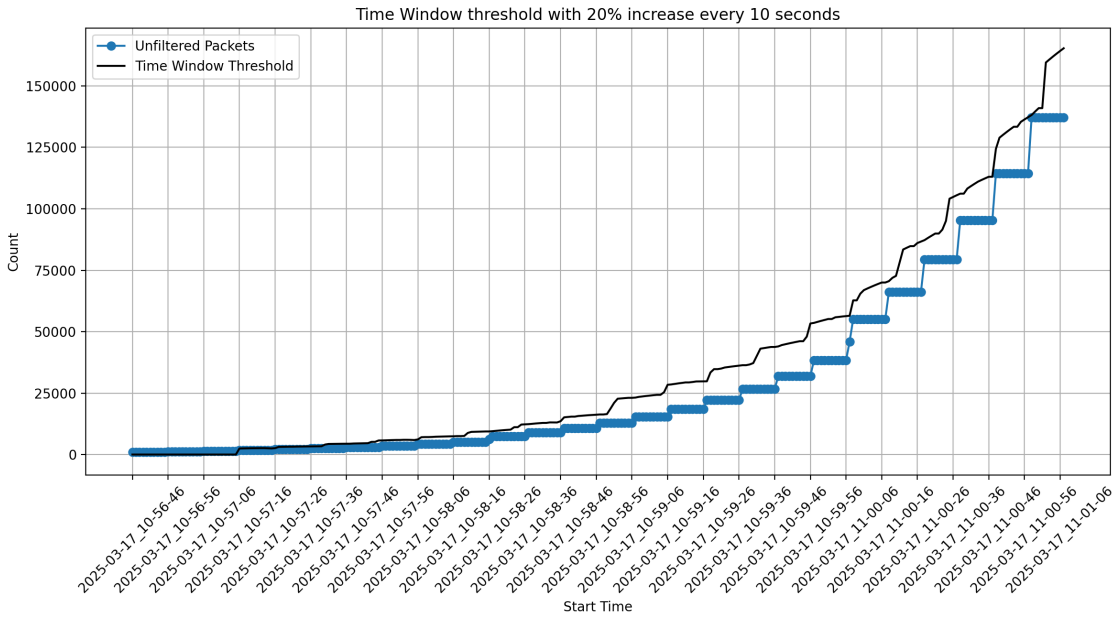


Figure 4.7: Threshold adaptability under increasing normal data at 20% every 10 seconds
Dataset: *normal_traffic* (Ericsson AB)

Figure 4.7 illustrates how the system dynamically adjusts its threshold in response to increasing normal traffic volume at a rate of 20% every 10 seconds. In comparison to Figure 4.6, this scenario shows that the threshold increases but is on the absolute limit of calling the increase an attack instead. This is expected since allowing an increase that is too steep would open the risk of letting attacks pass through.

4.1.5 Detection Latency

The goal of analyzing the detection latency of the system is to show that the absolute worst case time latency of an attack of sufficient size to the point of detection is at most two time windows. The longest time until detection should therefore always be less than 2.4 seconds when accounting for the sliding overlap of the windows.

Statistic	Value (ms)
Mean	602
Median	462
Minimum	155
Maximum	1229
Standard Deviation	320
Samples	30

Table 4.4: Summary of detection latency data

The results of the detection latency test are shown in Figure 4.4. As the data suggests, the worst-case detection latency is 1.229 seconds. This validates the suggestion that the worst-case detection latency is at most 2 windows. Additionally, the mean is 602 ms, which further supports the worst case of approximately 1200, since the mean is most likely to fall towards the midpoint between 0 and 1200.

4.2 Comparing the Baseline Configuration to the HP Configuration

This section compares the performance of HeavyKeeper and Count-Min with traditional data structures such as Hash Maps and Priority Queues. It evaluates key aspects such as processing time and memory utilization to demonstrate how our approach outperforms these alternatives in terms of efficiency and scalability. By examining the differences in these metrics, the section highlights the advantages of using data sketches for handling large-scale network traffic in real-time intrusion detection systems.

4.2.1 Detection Accuracy

Table 4.5: Hash Map & Priority Queue with attack performance

Metric	Value	Percentage (%)
True Positives	11	20.37%
False Positives	2	3.70%
True Negatives	40	74.07%
False Negatives	1	1.85%
Correct Detections	51	94.44%

The Table 4.5 shows the detection performance of the Hash Map & Priority Queue (HP) Configuration when tested with attack data. It showcases the systems ability to identify both attacks and normal traffic. The results indicate that the system achieved 20.37% true positives and 74.07% true negatives, with a correct detection rate of 94.44%. While the false positive rate was 3.70%, the false negative rate was

relatively low at 1.85%, suggesting that the HP configuration is quite effective, but with room for improvement in detecting attacks more accurately.

4.2.2 Processing Time

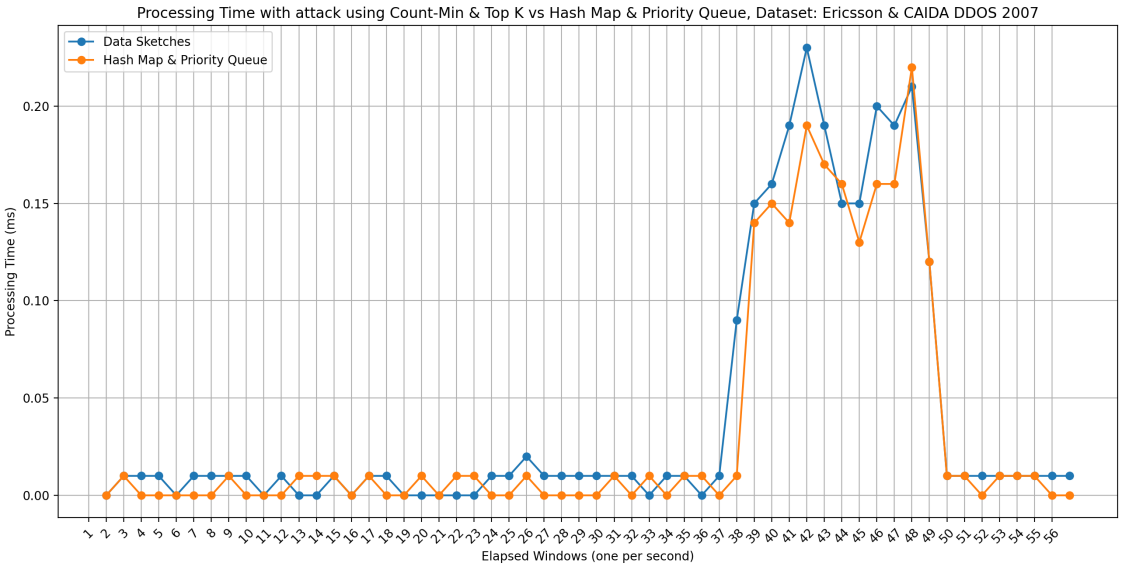


Figure 4.8: Processing Time with attack using Count-Min & HeavyKeeper vs Hash Map & Priority Queue, Dataset: Ericsson & CAIDA DDOS 2007

Figure 4.8 shows the observed processing time when processing network traffic data that includes attacks from datasets provided by Ericsson and the CAIDA DDOS 2007 dataset. The results clearly show that the data sketches maintain a stable and substantially lower processing time even as traffic volumes increase, demonstrating their effectiveness in managing large-scale data streams. In contrast, the Hash Map shows significant growth in processing time proportional to the traffic volume, making it less suited for real-time, high-throughput environments.

4.2.3 Memory Utilization

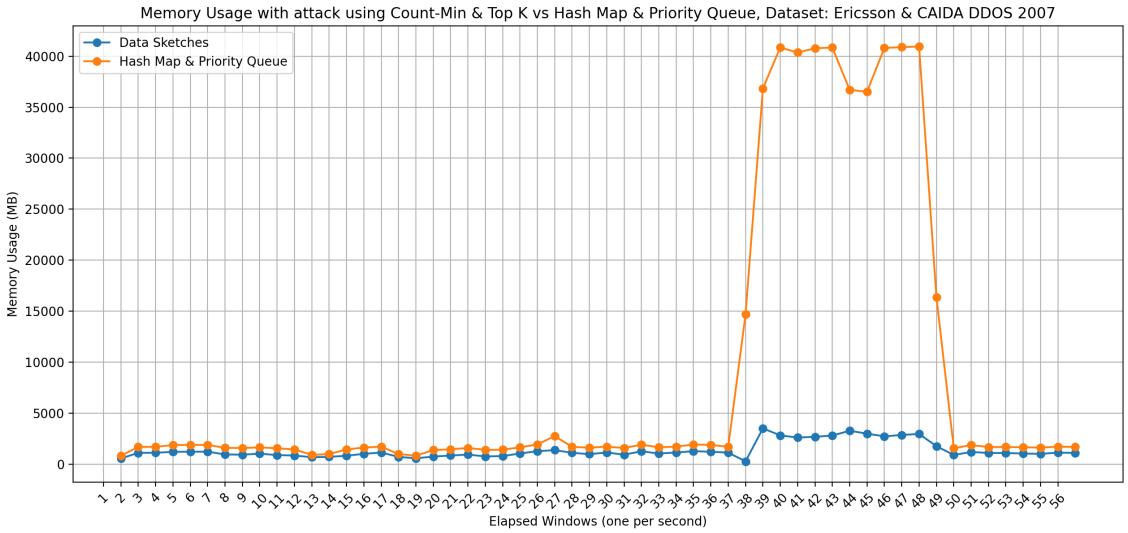


Figure 4.9: Memory Usage with attack using Count-Min & HeavyKeeper vs Hash Map & Priority Queue, Dataset: Ericsson & CAIDA DDOS 2007

Figure 4.9 presents the memory utilization of Data Sketches compared to a Hash Map and Priority Queue during the processing of the same datasets. The Count-Min and HeavyKeeper sketches consistently utilized significantly less memory than the Hash Map and Priority Queue. The Hash Map and Priority Queue, on the other hand, show very high growth in memory consumption, with approximately 10-15 times higher memory usage than the Data Sketches during the attack. This makes it impractical for sustained deployment in high-volume environments. This test demonstrated that it would not be able to work as a viable solution in a similar scenario.

4.3 Comparing the Baseline Configuration to the ML Configuration

In this section, a comparison of the performance between the ML Configuration and the Baseline Configuration is conducted. The goal is to evaluate how each model performs in detecting anomalies and DDoS attacks under the same conditions.

4.3.1 Detection Accuracy

The first table presents the results of the ML Configuration with an attack, while the second table shows the results when there is no attack and only normal data.

Metric	Value	Percentage (%)
True Positives	30	32.97
False Positives	24	26.37
True Negatives	37	40.66
False Negatives	0	0.00
Correct Detections	67	73.63

Table 4.6: Machine learning model with attack performance, Dataset: CAIDA DDOS 2007

Metric	Value	Percentage (%)
True Positives	0	0.00
False Positives	85	46.70
True Negatives	97	53.30
False Negatives	0	0.00
Correct Detections	97	53.30

Table 4.7: Machine learning model without attack performance. Dataset: CAIDA DDOS 2007

When comparing these two scenarios of the ML Configuration to the performance of the Baseline Configuration, seen in 4.1 and 4.2, we see that the ML Configuration is performing worse in the case of an attack. The ML Configuration with attacks only achieves a total correct detection percentage of 73.63%, whereas the Baseline Configuration performed better with a total correct detection percentage of 96.09%. In the case without attacks, the ML Configuration has an even lower percentage of 53.30% when compared to the statistical models performance on the same dataset.

4.3.2 Response Time

Figure 4.10 showcases the response times of two different API endpoints: `detect-timewindow` and `detect-ml`. The endpoint `detect-timewindow` is used to detect malicious time windows by using the IQR-based statistical model and determining a threshold for acceptable traffic load. The endpoint `detect-ml`, on the other hand, uses the Isolation Forest algorithm and does not determine a threshold, but rather learns from incoming data and alerts anomalies that could be potential DDoS attacks. Figure 4.10 illustrates the maximum response time, where the filled line represents the `detect-timewindow`, and the dotted line corresponds to the `detect-ml`. From the graph, we can observe the fluctuations in response times over the course of 4 minutes with one request sent per second. The requests sent to the two endpoints were identical to ensure an accurate comparison, containing a fixed `timestamp` and a `total_attempts` field with a static value of 7000.

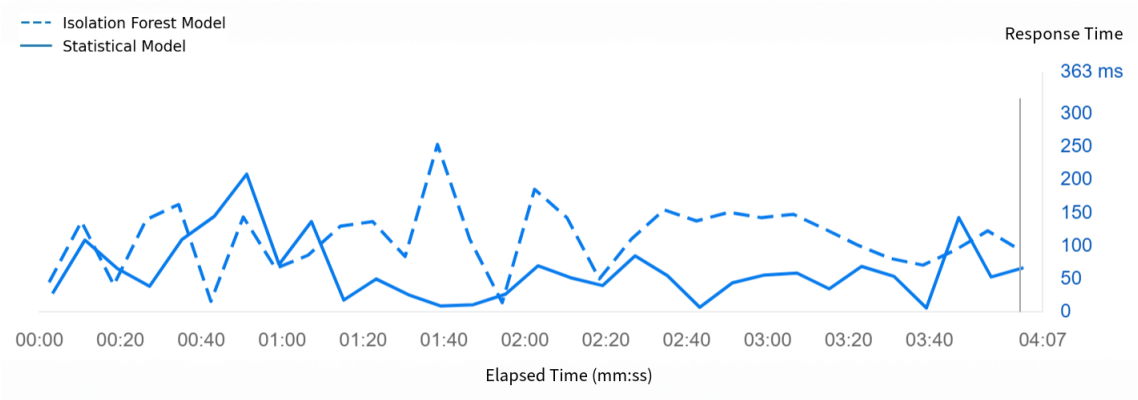


Figure 4.10: Maximum response time, with the filled line representing the `detect-timewindow` API using the statistical model and the dotted line representing the `detect-ml` API utilizing the Isolation Forest method.

Table 4.8: Comparison between the Statistical and Isolation Forest (IF) Models

Metric	**Statistical Model**	**IF Model**	**Delta**
Total requests sent	225	215	10
Requests/second	0.91	0.87	0.04
Avg. response time	17 ms	40 ms	-23 ms
Max time	209 ms	254 ms	-45 ms

4.4 Summary

In this section, an overview is presented of the systems performance across various test scenarios. By comparing different configurations, including machine learning and data structures, we assess the effectiveness of our Baseline Configuration using the statistical model combined with data sketches.

4.4.1 Detection Accuracy

Simulating traffic using the Ericsson normal traffic dataset with the CAIDA DDOS attack recording from 2007, the Baseline Configuration achieved 96.09% detection accuracy. The reasoning for the missing 3.97% is due to the nature of the test. To clarify, the test flagged windows as attack windows as soon as the attack started; however, the attack has a windup time where the first windows are not significant enough to do any damage, which produces false negatives. Even still, they are not detected and are therefore taken into account in the evaluation. Furthermore, the same test was run with Ericsson's own diagnostic test dataset with Ericsson normal traffic, which yielded similar results with a 94.03% detection accuracy.

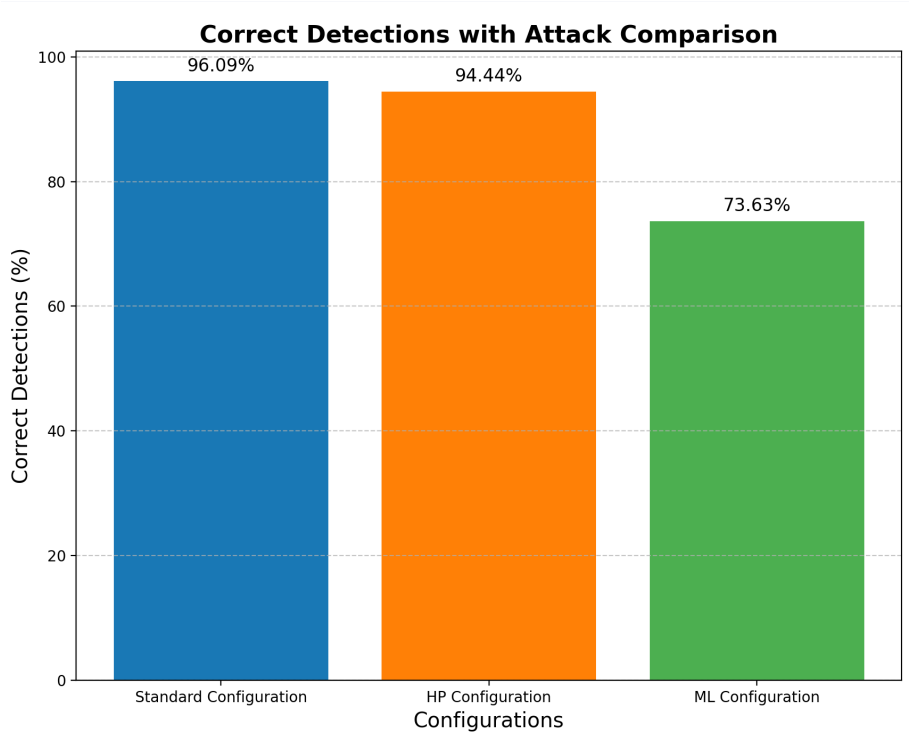


Figure 4.11: Correct Detections comparison between the three difference configurations.

This Figure 4.11 presents a comparison of Correct Detections between the Baseline Configuration, HP Configuration, and the ML Configuration. It shows the accuracy of each configuration in detecting DDoS attacks. The Baseline Configuration outperformed both the Hash Map & Priority Queue and ML Configuration, with the HP Configuration as a close contender. The ML Configuration performed significantly worse than the rest.

4.4.2 Detection Latency

The analysis of detection latency using the Baseline Configuration shows that the worst-case time until detecting an attack of sufficient size is at most two time windows, or less than 2.4 seconds, due to the sliding window overlap. Detection latency statistics indicate a maximum latency of 1.229 seconds, supporting the claim. The mean latency is 602 ms, aligning with the expected worst-case latency of approximately 1.2 seconds.

4.4.3 Resource Utilization

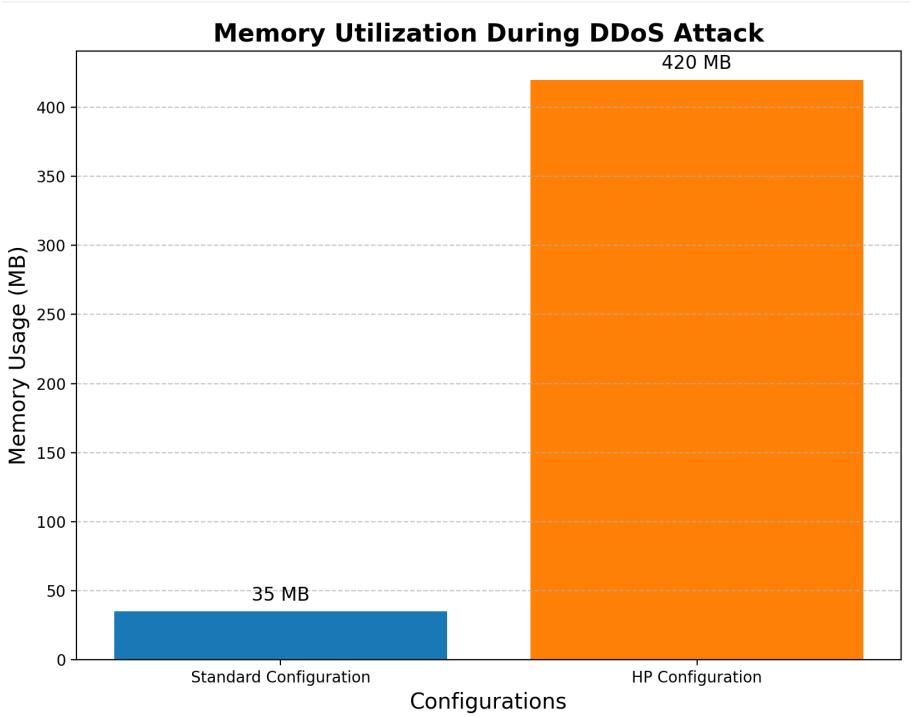


Figure 4.12: Memory Usage summary of the Baseline Configuration and the HP Configuration during an attack, Dataset: Ericsson & CAIDA DDOS 2007

The evaluation shows that the Baseline Configuration drastically outperformed the HP configuration in memory usage when handling attack-laden network traffic (Ericsson & CAIDA DDOS 2007 datasets [32]) as shown in Figure 4.12. The Baseline Configuration maintains low, stable processing times and minimal memory consumption, making it a suitable approach for real-time, high-volume environments. In contrast, the HP Configuration exhibits steep increases in memory usage, rendering it impractical for such scenarios.

4.4.4 Rate Limiting

The evaluation of rate limiting the incoming packets shows that applying such a mitigation strategy will help prevent congestion and allow the system to continue working even if the system is attacked. Figure 4.2 and Figure 4.3 illustrate a time series comparison of unfiltered and filtered packets counts, showing that while unfiltered traffic spikes, it stabilizes at a manageable level.

4.4.5 Comparison to Machine Learning

The comparison between the Baseline Configuration and the ML Configuration focuses on performance. The study found that the Baseline Configuration consistently outperformed the ML Configuration in terms of both processing time and memory

efficiency. The Baseline Configuration achieved a higher detection accuracy (96.09%) compared to the ML Configuration (73.63%) 4.9 under attack scenarios. In normal traffic scenarios, the ML Configuration struggled with a high false positive rate, resulting in only 53.30% correct detections. Moreover, the Baseline Configuration demonstrated superior response times and lower latency across various metrics, with significantly faster average and percentile response times. Overall, the Baseline Configuration proved to be more scalable, resource-efficient, and practical for real-time network intrusion detection than the ML Configuration tested.

Table 4.9: Performance Comparison: Statistical Model vs Machine Learning Model

Metric	**Baseline Config.**	**ML Config.**
Detection Accuracy (Attack)	96.09%	73.63%
Detection Accuracy (No Attack)	100.00%	53.30%
False Positives (No Attack)	0.00%	46.70%
False Negatives (Attack)	3.35%	0.00%
Average API Response Time	17 ms	40 ms
Max API Response Time	209 ms	254 ms
Memory Usage	Low	Medium
Scalability	High	Medium

5

Conclusion

This chapter compiles the key findings, contributions, and implications of the work presented in this thesis. It summarizes the implementation and evaluation of the proposed anomaly-based intrusion detection system, while also highlighting the systems strengths and performance results, and outlines several directions for future research.

5.1 Summary of the Thesis

This thesis presents the design and implementation of a scalable, anomaly-based NIDS that leverages statistical modeling and data sketches to efficiently detect DDoS attacks. The primary goal was to develop a solution that reduces memory usage while maintaining high detection accuracy, scalability, and low detection latency.

The system was evaluated using different datasets, including real-world traffic data from Ericsson and CAIDAs DDoS dataset. The results demonstrated that the system achieved an impressive detection accuracy of 96.09% for DDoS attacks, and a minimal false positive rate of 0.56% for the CAIDA dataset. This high level of accuracy confirms the potential of statistical modeling combined with data sketches like Count-Min and HeavyKeeper in handling large-scale network traffic efficiently.

Furthermore, the comparison between the statistical model and machine learning model, in this case with the Isolation Forest algorithm, revealed that the statistical model provided superior detection accuracy and lower response times. While machine learning models offer flexibility in handling dynamic data, the simpler, more resource-efficient statistical model proved to be more effective in this context when simply checking if the packet count reaches above the threshold. However, if the goal were to detect more complex patterns of packet behaviors, using a machine learning model would most likely outperform the statistical model.

Resource utilization tests showed that the system could handle substantial traffic volumes without excessive memory or CPU consumption and could process the attacks within the time window. This makes it suitable for real-time applications in enterprise environments. Additionally, the scalability tests showed that the system's dynamic threshold adjustment could effectively adapt to varying network traffic conditions without triggering false alarms.

5.2 Future Work

This section brings up several areas for future improvement. These include enhancing filtering techniques, optimizing detection thresholds, integrating machine learning models, and addressing potential system bottlenecks. Further research in these areas can make the system more robust and adaptable for large-scale deployments.

5.2.1 IP Address Filtering

During the course of our research, we explored the approach of classifying the most prevalent IP addresses as either malicious or non-malicious, and subsequently filtering out those identified as malicious. However, this method did not provide any big improvements to the performance of our proposed system. One major limitation was the restriction to banning only the top k IP addresses per time window, which proved insufficient. Increasing k to cover all potential threats would have required an impractically large value, negatively impacting system efficiency. Nevertheless, we believe that with a more optimized and adaptive filtering mechanism, IP filtering could still offer meaningful benefits in future implementations.

5.2.2 Threshold Offset Based on Data Fluctuations

Currently, the system applies a static threshold offset based on two standard deviations (2σ) to account for normal traffic variability. This decision is based on the need to balance detection sensitivity and false positives, and was decided through trial and error. However, traffic fluctuations can vary a lot depending on the time of day, company size, and external events such as conferences. This means that, in certain cases, fluctuations might be well contained within one standard deviation, while in others, more aggressive variations could require offsets greater than 2σ. For example, in highly stable environments, a tighter offset may be more appropriate, whereas noisier environments might require a broader margin. Dynamic adjustment based on real-time data would likely improve the generalizability and practical applicability of the intrusion detection system.

In future work, a deeper analysis of natural traffic fluctuation patterns could be conducted to dynamically adjust the offset based on observed variance rather than relying on a fixed value. This could involve: quantifying typical fluctuation patterns during normal operation and exceptional events or developing an adaptive strategy where the offset is a function of real-time variance measurements rather than a pre-determined multiple of σ.

5.2.3 Machine Learning Fine Tuning

During the evaluation, the Isolation Forest used in the ML Configuration showed potential but underperformed compared to the statistical approach used in the Baseline Configuration. One important factor is that the machine learning model was not fine-tuned. Future work could involve hyperparameter optimization, feature engineering, and more training to enhance its performance. Fine-tuning could lead to

improved anomaly detection, especially in identifying complex patterns that may be difficult to detect using the Baseline Configuration or other statistical methods.

5.2.4 Setting the Optimal Rate Limit

Finding the optimal rate limit is important to mitigate DDoS attacks without interrupting real network traffic. A too restrictive rate limit may block real and authentic user requests, reducing system usability. On the other hand, an overly permissive limit might fail to adequately prevent or reduce the impact of malicious traffic. Future work may include adaptive rate-limiting strategies that dynamically adjust rate limiting based on real-time traffic behavior.

5.2.5 Deciding the Optimal Time Window Size

Setting the optimal time window size is important for ensuring accurate and efficient anomaly detection in NIDS. A shorter time window size allows for quicker detection of attacks but may lead to increased false positives due to normal traffic fluctuations and also false negatives since a DDoS attack can be cut off in the middle by the short time window. In contrast, a larger window size includes more data but has longer alert times since it can only send an alert when the window has been closed and analyzed. Another aspect to consider is that the longer the time window, the more data has to be held by the system. Thus, choosing an appropriate window size involves balancing the trade-offs between detection latency, computational overhead, and detection accuracy, and should be guided by the characteristics of the monitored network environment and its criteria.

5.2.6 Degree of Threshold Flexibility

It is important to consider the optimal degree of threshold flexibility. In the current implementation, thresholds are adaptive to account for natural traffic variations, but a balance must be determined between rigor and flexibility. If the threshold adapts too quickly or is overly dynamic, there is a risk that the system may fail to detect certain attacks, leading to an increase in false negatives. This is because the threshold would increase too quickly, even if network traffic exceeds normal traffic fluctuations. In contrast, if the threshold is too rigid and slow to adapt, normal fluctuations in traffic could be misclassified as malicious activity, which results in a higher rate of false positives.

Future research should explore ways to control the adaptability of the threshold based on the context of the network environment. For example, during periods of expected high variability, a more flexible threshold might be appropriate, whereas during stable periods, a more rigid threshold could be better.

5.2.7 Potential Bottlenecks in the System

One risk is the speed and responsiveness of IPS (Intrusion Prevention System) subscribers. If a subscriber tasked with banning malicious IP addresses is unable to

process alerts and lists of IP addresses quickly enough, it could result in delayed or missed actions. In high-traffic scenarios, even a small delay can allow attacks to continue for longer periods. Another important requirement includes ensuring that the data aggregation and anomaly detection modules can maintain processing speeds that match or exceed incoming traffic rates. The data sketch structures, while highly efficient, must also be correctly dimensioned to avoid collisions or excessive estimation errors at extremely high loads. Furthermore, the communication between the detection system and external subscribers must have low latency and high throughput guarantees.

Bibliography

[1] World Bank. "Global digitalization in 10 charts." Accessed: 2024-11-27, World Bank. (Mar. 2024), [Online]. Available: https://www.worldbank.org/en/news/immersive-story/2024/03/05/global-digitalization-in-10-charts.

[2] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, e4150, 2021. DOI: https://doi.org/10.1002/ett.4150. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ett.4150. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4150.

[3] M. Bahri, A. Bifet, J. Gama, H. M. Gomes, and S. Maniu, "Data stream analysis: Foundations, major tasks and tools," *WIREs Data Mining and Knowledge Discovery*, vol. 11, no. 3, e1405, 2021. DOI: https://doi.org/10.1002/widm.1405. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1405. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1405.

[4] B. B. Gupta, R. C. Joshi, and M. Misra, *Distributed denial of service prevention techniques*, 2012. arXiv: 1208.3557 [cs.CR]. [Online]. Available: https://arxiv.org/abs/1208.3557.

[5] F. Suthar and N. Patel, "A survey on ddos detection and prevention mechanism," *Journal of Advances in Information Technology*, vol. 14, no. 3, 2023.

[6] O. H. Abdulganiyu, T. Ait Tchakoucht, and Y. K. Saheed, "A systematic literature review for network intrusion detection system (ids)," *International Journal of Information Security*, vol. 22, no. 5, pp. 1125–1162, Mar. 2023, ISSN: 1615-5270. DOI: 10.1007/s10207-023-00682-2. [Online]. Available: http://dx.doi.org/10.1007/s10207-023-00682-2.

[7] M. Roesch, "Snort - lightweight intrusion detection for networks," *Proceedings of LISA*, pp. 229–238, 1999.

[8] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Computers Security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[9] A. Sakharkar, R. M. Parizi, Q. Zhang, and K. Gai, "Edge security: Challenges and directions for future research," *IEEE Consumer Electronics Magazine*, vol. 10, no. 1, pp. 57–65, 2021.

[10] J. T. McClave and T. Sincich, *Statistics*, 12th. Pearson Education, 2013, ISBN: 9780321755933.

[11] Cloudflare. "What is a ddos attack?" Accessed: 2025-03-13. (2024), [Online]. Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/.

[12] Kaspersky. "What is a botnet?" Accessed: 2025-03-13. (2024), [Online]. Available: https://www.kaspersky.com/resource-center/definitions/botnet.

[13] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[14] M. Antonakakis, T. April, M. Bailey, *et al.*, "Understanding the mirai botnet," *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 109–123, 2017.

[15] P. P. Sotiris Kotsiantis I. Zaharakis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, no. 3, pp. 249–268, 2007. [Online]. Available: http://www.informatica.si/index.php/informatica/article/view/148.

[16] V. K. Varun Chandola Arindam Banerjee, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, 15:1–15:58, 2009. DOI: 10.1145/1541880.1541882.

[17] Z.-H. Z. Fei Tony Liu Kai Ming Ting, "Isolation forest," *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, 2008. DOI: 10.1109/ICDM.2008.17.

[18] MathWorks, *Anomaly detection with isolation forest*, Accessed: 2025-02-05, 2023. [Online]. Available: https://www.mathworks.com/help/stats/anomaly-detection-with-isolation-forest.html.

[19] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17. [Online]. Available: https://ieeexplore.ieee.org/document/4781136/.

[20] D. Gusev, *Windowing in stream processing: A comprehensive guide*, Accessed: 2025-02-05, 2025. [Online]. Available: https://quix.io/blog/windowing-stream-processing-guide.

[21] J. Gama, *Knowledge Discovery from Data Streams*. Chapman and Hall/CRC, 2010, ISBN: 978-1-4398-0874-8.

[22] A. Bifet and R. Gavaldà, *Adaptive Learning from Evolving Data Streams*. IOS Press, 2009, ISBN: 978-1-60750-019-3.

[23] C. C. Aggarwal, *Data Streams: Models and Algorithms*. Springer, 2007, ISBN: 978-0-387-47534-9.

[24] P. Kranen and T. Seidl, "Stream clustering: A review and experimental comparison," in *Proceedings of the 15th International Conference on Data Mining*, IEEE, 2013, pp. 271–278. DOI: 10.1109/ICDM.2013.45.

[25] G. Cormode, "Sketch techniques for approximate query processing," *Foundations and Trends in Databases*, vol. 4, no. 1-2, pp. 1–120, 2011.

[26] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *Proceedings of the 2007 Conference on Analysis of Algorithms*, 2007, pp. 127–146.

[27] S. Muthukrishnan, "Data streams: Algorithms and applications," *Foundations and Trends in Theoretical Computer Science*, vol. 1, no. 2, pp. 117–236, 2005.

[28] D. Ting, "Streaming locality-sensitive hashing for high-dimensional data," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 937–946.

[29] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," in *Journal of Algorithms*, vol. 55, 2005, pp. 58–75.

[30] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2002, pp. 693–703.

[31] J. Gong, T. Yang, H. Zhang, *et al.*, "HeavyKeeper: An accurate algorithm for finding top-k elephant flows," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, MA: USENIX Association, Jul. 2018, pp. 909–921, ISBN: 978-1-939133-01-4. [Online]. Available: `https://www.usenix.org/conference/atc18/presentation/gong`.

[32] The CAIDA UCSD Project, *The CAIDA UCSD 'DDoS Attack 2007' Dataset*, Accessed: [Insert Date of Access], 2007. [Online]. Available: `https://www.caida.org/catalog/datasets/ddos-20070804_dataset`.

[33] T. Tanadechopon and B. Kasemsontitum, "Performance evaluation of programming languages as api services for cloud environments: A comparative study of php, python, node.js and golang," in *2023 7th International Conference on Information Technology (InCIT)*, 2023, pp. 293–297. DOI: `10.1109/InCIT60207.2023.10413079`.

[34] C. Taylor. "What is the interquartile range (iqr) rule?" Accessed: March 14, 2025. (2024), [Online]. Available: `https://www.thoughtco.com/what-is-the-interquartile-range-rule-3126244`.

[35] G. Cormode and S. Muthukrishnan, "The count-min sketch and its applications," *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 11–17, 2005. DOI: `10.1145/1066157.1066162`.

[36] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd. Addison-Wesley, 1997, ISBN: 978-0201896855.

[37] Redis, *Redis documentation*, 2025. [Online]. Available: `https://redis.io/documentation`.

[38] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd. The MIT Press, 2009, ISBN: 978-0262033848.