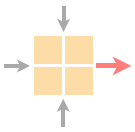
ETH zürich



Networked Systems
ETH Zürich — seit 2015

Automating morphing DDoS attacks with ML techniques

Semester Thesis

Author: Adrien Nelson Rey

Tutor: Dr. Muoi Tran, Eric Joles

Supervisor: Prof. Dr. Laurent Vanbever

September 2024 to January 2025

Abstract

This project introduces a novel machine learning-powered morphing DDoS attack methodology that optimizes both efficiency and adversarial effort while leveraging network feedback. By analyzing the feedback, the attack identifies the active defense mechanism and uses this intelligence to adapt its flooding strategy to maximize impact with minimal computational and operational cost. Simulations demonstrate that the attack achieves comparable or superior effectiveness in disrupting legitimate traffic while significantly reducing the burden on the attacker. Furthermore, experiments under a constrained adversary model, using realistic IP subsets, confirm the robustness and adaptability of the attack, even with limited resources.

Contents

Chapter 1

Introduction

This report begins with an introduction outlining the motivation and objectives of the study. The background chapter provides the necessary context, including an overview of morphing attacks, the simulation environment, and the defense mechanisms under study. The design chapter describes the methodology behind the attack, detailing how it extracts information about the victim's defenses strategy and how it optimizes its flooding strategy. The evaluation chapter assesses the effectiveness of the attack against various defenses, comparing it to prior techniques. It also analyze its performance under a more realistic adversarial constraints. Finally, the report concludes with a discussion on the implications of the findings and potential directions for future research. Additionally, the appendices include an analysis of the "Giant Cluster" problem, a weakness in ACCTurbo discovered during this project.

1.1 Motivation

Distributed Denial of Service (DDoS) attacks represent a significant and persistent threat to the availability and reliability of online systems. Over the years, attackers have continuously adapted their techniques to evade detection and countermeasures, while defenders have responded with increasingly sophisticated strategies. This arms race has led to a critical need for understanding and analyzing the evolving landscape of both offensive and defensive techniques in cybersecurity.

The motivation for this project stems from the dual perspective of advancing our understanding of attack strategies and highlighting the limitations of current defense mechanisms. While many traditional DDoS attacks rely on brute-force flooding to overwhelm a target, these methods are becoming less effective against modern defenses. Newer techniques, such as morphing attacks, which dynamically adjust their characteristics to bypass filtering, have proven more effective but often at a higher computational or operational cost for the attacker.

Additionally, recent advancements in machine learning and network analysis have opened opportunities for more intelligent attacks. By leveraging feedback from the network itself, attackers could tailor their strategies to the specific defense mechanisms in place, making them both highly effective and resource-efficient. This capability poses a significant threat to existing defenses.

The motivation behind this project is to explore the feasibility and effectiveness of a machine learning-driven DDoS attack model that integrates feedback from the target network to adaptively optimize its flooding technique. The proposed model seeks to demonstrate how such attacks can maintain high disruption levels while minimizing adversarial effort.

1.2 Thesis Objectives

The primary objective of this thesis is to design and implement a novel adaptive morphing attack that improves upon the prior work presented in [6]. This new attack leverages machine learning (ML) techniques to analyze network feedback and gain precise insights into the defense strategies employed by the victim. With this intelligence, the attack dynamically adapts its behavior, optimizing its ability to bypass defenses effectively.

The aim is for this new attack to achieve comparable or superior results to its predecessor in terms of its impact on the victim's traffic while also addressing critical efficiency and cost concerns for the attacker. By minimizing the computational overhead, reducing the number of probing packets, and lowering the mutation cost of flood packets, the proposed attack aims to become a more feasible option for real-world adversaries.

This work serves as a proof of concept, demonstrating not only the viability of such an advanced adaptive attack but also its potential to pose significant challenges to state-of-the-art defenses.

1.3 Related Work

The study of Distributed Denial of Service (DDoS) attacks and their defenses has been a focal point in cybersecurity research, driven by the evolving interplay between attackers and defenders. Traditional DDoS attacks, such as SYN or UDP flooding are for long time now well studied [9]. In response, defenses such as Jaqen [7] which uses signature-based traffic analysis, and ACCTurbo [8] that use a clustering-based approaches, have emerged to dynamically identify and mitigate malicious traffic based on its patterns.

Recent research has also developed attack technique incorporating adaptive strategies, including network feedback, to outmaneuver these defenses. Adaptive Morphing DDoS attacks have been proposed to iteratively modify packet characteristics, aiming to bypass detection [6]. While effective, these approaches often incur significant costs, including high probing traffic and computational overhead. This project builds on these advancements to present a proof-of-concept attack that efficiently leverages machine learning to analyze network feedback, identify deployed defenses, and optimize flooding techniques accordingly.

Chapter 2

Background

In this chapter, we begin by explaining the concept of morphing DDoS attacks and providing an overview of their current state. Next, we introduce the tools and datasets utilized in this project to evaluate the newly developed attacks. Finally, we provide a brief overview of the functioning of three DDoS defenses selected for testing the effectiveness of these attacks, no defense, ACCTurbo and Jaqen Heavy Hitter.

2.1 Morphing DDos attack

A Distributed Denial of Service (DDoS) attack leverages multiple systems to flood a target, such as a server, website, or a network link, with excessive traffic. This overwhelms the target's resources, including bandwidth, memory, and processing power, rendering it incapable of handling legitimate connections and denying access to legitimate users.

DDoS attacks are becoming increasingly prevalent, with tools like the Mirai botnet making them more accessible and widespread. This trend shows no signs of slowing in the coming years. However, DDoS attacks are not a new phenomenon and have undergone numerous evolutions over the decades. One of the latest development in this ongoing arms race is the emergence of Morphing DDoS Attacks.

Morphing DDoS attacks stand out by focusing on altering the structure and characteristics of the flood packets as well as changing the attack techniques. By continuously modifying packet attributes, such as header values, payload sizes, or timing, morphing attacks aim to evade detection systems that rely on identifying consistent patterns or static signatures in the traffic. Morphing DDoS adaptability is a real challenge for traditional defense mechanisms.

2.1.1 Dynamic Morphing DDoS Attacks

Dynamic morphing DDoS attacks employ frequent and unpredictable changes in packet characteristics and attack vector to avoid detection, ensuring that their traffic remains difficult to classify. This constant variation allows dynamic morphing attacks to slip past defenses that rely on identifying static or repetitive traffic patterns.

2.1.2 Adaptive Morphing DDoS Attacks

Adaptive morphing DDoS attacks take this concept further by incorporating real-time feedback of the network into their strategy. These attacks monitor the target's responses during an ongoing attack and adjust the attacks dynamically to bypass mitigation measures. For example, if certain

packet types are blocked by the defense system, the attacker can adapt by generating packets with modified attributes, ensuring continued impact.

As part of [6], an implementation of Adaptive Morphing DDoS Attacks was developed, which serves as a baseline for this thesis. From this point onward, we will refer to these attacks as "*adaptive morphing attacks*". This implementation uses two different traffic: probing traffic and attack traffic. Probing traffic consists of a small number of packets sent to gather feedback about the target system (packet drop rates and latency). Attack traffic, on the other hand, is the main volumetric stream designed to overwhelm the target.

The process works as follows:

- **Probing and Feedback:** Probing packets are sampled randomly, with specific features modified to identify performance under different conditions. After a feedback delay, the attacker receives data indicating whether each packet was dropped and its latency.
- **Analysis and Adjustment:** The attacker evaluates feedback every 5 seconds, analyzing for each features which values have the lowest drop rates. These preferred values are used to construct attack packets for the next interval.
- **Attack Traffic Morphing:** During the attack phase, packets are dynamically morphed based on the analyzed feedback. Each feature is set to a value within its corresponding "low-drop" range.

The packet features used in probing and then mutate for this attacks are : Source IP, Source Port, Destination IP, Destination Port, TTL, Packet length, and protocol. For completeness each byte of the IPs are dealt as a unique feature and are then mutated independently.

2.2 Netbench packet simulator

NetBench [5] is a packet simulator originally developed for evaluating routing in static network topologies using discrete event simulation. It models physical network components such as links, output ports, and network devices, enabling the simulation of custom network configurations with arbitrary data. This flexibility provides a robust testing environment to analyze various protocols, systems, and their effects on network performance.

NetBench was first employed by the creators of ACCTurbo[1] to develop a framework that simulated attacks on their defense mechanism.

Netbench was then used and improved by [6]. It utilized NetBench to simulate various attacks and defenses across different network configurations. In this project, new specific attacks and defenses have been added to netbench. Additionally, a PCAP extractor was required, which was initially developed during the ACCTurbo project and further refined as part of [6].

This thesis makes use of NetBench in a similar manner to simulate a network and evaluate the efficiency of newly developed attacks. It leverages prior work in NetBench by reusing and modifying the previously implemented DDoS defenses. Furthermore, it employs the previously developed attacks as a baseline for comparing the performance and impact of the new attacks. Additionally, this thesis utilizes and enhances the attacker feedback feature originally developed in [6].

2.3 DataSet

In order to make our experiments realistic, we will replay some existing traffic captures trough the different simulations. Fortunately NetBench allow background/benign traffic to be supplied in PCAP format. In this thesis, we decide to use the CIC-IDS2017 dataset [10] as benign traffic sources, as was used in [6].

The CIC-IDS2017 dataset was developed by the Canadian Institute of Cybersecurity and is a useful resource for intrusion detection and prevention research. It was generated using a laboratory setup that mirrors real-world conditions, featuring both attack and victim networks. Interestingly, this dataset includes data traces without any DDoS attacks, making it particularly suited for this thesis. These benign traces serve as a reliable source of background traffic during NetBench's simulation.

2.4 DDOS Defenses

In this project, three different DDos defenses are used to evaluate the effectiveness of different attacks. The first is a baseline setup which lacks any specialized DDos protection, representing a simple router using FIFO as its forwarding policy. The second, Jaqen, is a defense designed to block packets based on known malicious packet features, serving as a representative of signature-based defense. Lastly, ACCTurbo leverages clustering techniques to prioritize or deprioritize packets, serving as a representative for distance-based DDos defenses. Each defense is discussed in detail in the following sections.

2.4.1 No defense

When a link is protected by no defense, the forwarding policy employed is straightforward and follow a First-In, First-Out (FIFO) strategy. In this setup, the router operates with a finite waiting queue. As packets arrive, they are enqueued in the order of their arrival and forwarded when they reach the head of the queue.

However, the simplicity of this approach comes with a significant limitation. If the queue becomes completely filled, any newly arriving packets that cannot fit into the queue are immediately dropped. This lack of prioritization or filtering makes the no-defense scenario a baseline for evaluating DDoS defenses and attacks.

2.4.2 Jaqen Heavy Hitter

The Jaqen Heavy Hitter defense [7] addresses the limitations of traditional DDoS mitigation methods, which either depend on expensive, rigid hardware, or slower software-based solutions. It fully integrates detection and defense capabilities within programmable switches. It operates directly in-line with traffic flow, allowing for real-time detection and counteraction.

At the core of Jaqen's operation is the use of a bloom filter to track the packet signatures going through the router. Each time a packet arrives, Jaqen computes its signature, typically derived by hashing specific packet features. The bloom filter maintains a count for each unique signature, incrementing the count whenever a matching packet arrives. Periodically (determined by a timeout interval), Jaqen checks the bloom filter to identify any signatures exceeding a predefined packet count threshold during the last time interval. If a signature surpasses the threshold, Jaqen blocks all packets with that signature for the next interval. Packets that are not blocked are then added to a FIFO queue equivalent to the no defense case.

This mechanism is highly effective at mitigating volumetric DDoS attacks, by promptly identifying and suppressing high-volume malicious traffic. However, the system's performance heavily depends on the configuration of the timeout and threshold values. Incorrect tuning could lead to either ineffective defense or the blocking of legitimate traffic.

2.4.3 ACCTurbo

ACCTurbo [1] is a DDoS defense mechanism designed to mitigate pulse-wave and other volumetric attacks in real-time. It is itself an adaptation and reproduction of Aggregate-based Congestion Control (ACC) [8] in order for it to operate at line rate. This allow ACCTurbo to react to attacks within a second, thanks to its use of online clustering and programmable scheduling.

The core principle of ACCTurbo lies in its ability to observe and categorize incoming traffic into clusters. Each packet is assigned to the closest cluster based on some selected features of the packet. The distance between a packet and a cluster is calculated using the Manhattan distance. For nominal features, a mismatch results in a distance of 1, while for ordinal features, the numeric difference is used. This clustering process enables ACCTurbo to identify and manage traffic aggregates efficiently.

This clusters are then use for the programmable scheduling phase. Periodically the control plane monitors cluster statistics (such as throughput, packet rate, and/or size) and applies a ranking algorithm to prioritize or deprioritize clusters. Clusters containing too much bandwidth will be suspected to be part of an attack and will be deprioritized, ensuring that legitimate traffic from unaffected clusters maintains priority. Within each cluster/priority level, packets are added to a FIFO queue like in the no-defense case.

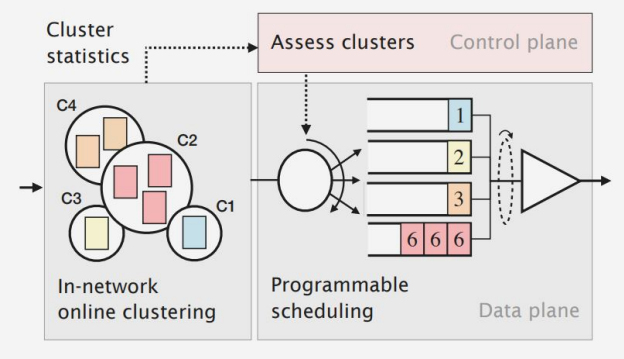


Figure 2.1: ACCTurbo Architecture.[2]

Despite its strengths, ACCTurbo has limitations. It is tailored to volumetric attacks and relies on the assumption that attacks manifest as traffic aggregates with similar features. Attackers capable of breaking similarity at the packet or aggregate level could bypass this defense. Additionally, the simplifications made to ensure the system runs at line rate introduce new challenges, such as the emergence of giant clusters, a phenomenon analyzed in detail in Appendix A.

Nevertheless, ACCTurbo's line-rate performance and generality make it a powerful tool against volumetric DDoS attacks. It is a good representative of distance-based defense which rely on similarity of packet to counter DDos.

.

Chapter 3

Design

This chapter first presents the threat models and assumptions underlying the attacks. Then it provides an overview of the attack and describes the various modules that compose it.

3.1 Threat Model: Adversary Capabilities

The purpose of this threat model is to establish the boundaries of the adversary's capabilities, serving as a foundation for the design and evaluation of the adaptive attacks developed in this thesis. By explicitly defining these limits, we ensure that the attacks remain realistic in some way. Furthermore, this threat model does not assume any capabilities beyond those demonstrated by adversaries in prior work on adaptive morphing DDoS attacks [6].

3.1.1 Packet Crafting Control

The adversary has complete control over the headers and content of the packet. This enables them to arbitrarily modify critical packet attributes.The adversary can adjust:

- **Source IP Address**
- **Source Port**
- **Destination IP Address**
- **Destination Port**
- **Protocol**
- **Packet Length**
- **Time-to-Live (TTL)**

Why This Is Realistic: Modern tools for packet crafting, such as Scapy [3], allow attackers to generate tailored packets with minimal effort, enabling seamless and efficient changes to any packet parameter stated before. Additionally, botnets like Mirai with thousands of devices allow an attacker to use a wide range of different source IP.

3.1.2 Probing and Feedback

The adversary can send **probing packets** and receive feedback on these probes. This feedback include whether the packets were dropped or successfully delivered and the lattency. This feedback loop enables the adversary to analyze the target's current state and tailor their attack accordingly.

Why This Is Realistic: Attackers can leverage multiple network protocols to obtain feedback. For example, the ICMP protocol is commonly used to send Echo Requests (ping) and receive Echo Replies, providing insights into whether packets are being dropped or delayed. Additionally, attackers can exploit stateful services running over protocols like TCP, where connection establishment and responses from services (e.g., SYN/ACK in TCP handshake) offer feedback on whether packets are reaching the target or being filtered by defenses.

3.1.3 Adaptive Behavior

The adversary, based on the feedback obtained from the probing packets, can dynamically adjust their attack strategy. This includes modifying packet features and changing the rate of attack. The adversary can also alter these parameters mid-attack to respond to the changing network conditions or adapt to countermeasures implemented by the target.

Why This Is Realistic: It is relatively easy for an attacker to adjust their attack in real time.

3.2 Overview of the attacks

The idea behind this new attack is built around two main phases. In the first phase, called the **reconnaissance phase**, the attacker uses feedback from the network to identify the type of defense in place and gather key information about how it is configured. In the second phase, called **the attack phase**, the attacker uses this information to launch the most effective flood attack designed to bypass the identified defense. This approach ensures the attack is both targeted and efficient.

The attack operates in a series of repetitive phases, each lasting 5 seconds, referred to as the "**5s attack phases**" During each phase, the attacker employs a dedicated module tailored to the conditions of the previous phase. This module is responsible for two primary tasks:

- **Sending Flooding Packets:** The module determines the packet(s) used to flood the target network during the phase
- **Sending Probing Packets:** The module selects the probing packets that will be sent during the phase

Throughout the phase, the flooding packet may mutate every second. At the end of the 5-second phase, feedback from the probing packets should have been received. This feedback allows the module to compute statistics based on the packet drop rates and latencies, providing valuable information about the network's state. Using this data, the module decides which module should be activated for the next attack phase and with which parameters, adapting the attack strategy accordingly.

An overview of the attack flow is depicted in Figure 3.1 and each module will be shortly explained here. The precise implementation and functionality of each module will be elaborated in the next section.

At the first phase of the attack, the **Defense Classifier** module is executed. The purpose of this module is to analyze feedback from the network in order to identify the DDos defense mechanism active on the targeted router. By the end of this phase, the module classifies the defense and selects the subsequent module accordingly:

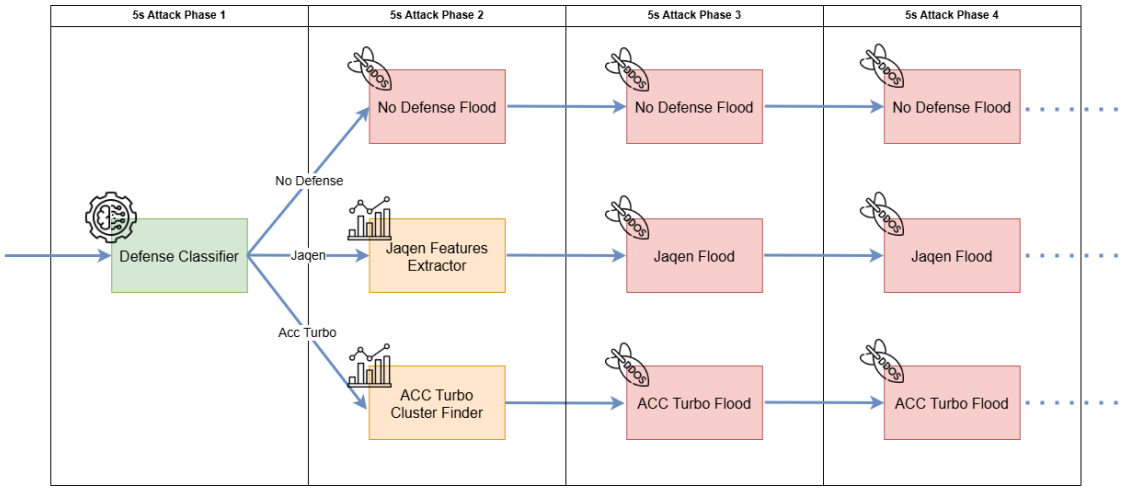


Figure 3.1: Module during the 4 first Attack Phases

- No-Defense: If no defense mechanism is detected, there is no need to continue the recon phase and the No-Defense Flood Module is selected.
- Jaqen Defense: If Jaqen is identified, the Jaqen Features Extractor Module becomes the next step.
- ACCTurbo Defense: If ACCTurbo is classified, the attack proceeds with the ACCTurbo Cluster Finder Module.

The **No-Defense Flood** Module performs a basic volumetric DDoS attack, designed to flood a router that lacks any active defense mechanism. It operates in a straightforward manner without further adaptations.

The **Jaqen Features Extractor** Module assumes that Jaqen is the defense mechanism in use and analyzes feedback to deduce which packet features are used by Jaqen to compute its packet signatures. This information is critical for fine-tuning the next phase of the attack. Following this, the Jaqen Flood Module uses the insights gained from the previous module to mutate the identified packet features strategically. Its goal is to bypass the Jaqen defense mechanism.

If ACCTurbo is identified, the **ACCTurbo Cluster Finder** Module is used to find where the clusters used by ACC Tubo are located. This module analyzes feedback to identify the clusters utilized by ACCTurbo, preparing the attack for the next phase. Based on the findings of the cluster finder, the **ACCTurbo Flood** Module directs the flood traffic to overwhelm all found clusters simultaneously.

3.3 Defenses Classifier

The Defense Classifier is designed to identify the defense mechanism deployed on the network by observing how different types of packets are treated.

3.3.1 Idea

The main idea to classify the defense consists of three main steps:

1. **Flood the Network:** A packet with fixed features is used to flood the network. This will triggers the defense mechanism which should try to mitigate this attacks.

2. **Send Probing Packets:** Alongside the flood, specific probing packets are sent to gather detailed statistical insights. Each probing packet is crafted to serve a distinct purpose:

   - **Baseline Packet:** Identical to the flood packet, its drop rate is used as the baseline for other probing packets.
   - **Nearby Packet:** Slightly modified compared to the flood packet (e.g., each feature incremented by 1), it provides information on whether packets similar to the flood packet are also affected by the defense.
   - **Far Packet:** A packet with features significantly different from the flood packet, located in a distant "corner" of the feature space. This packet tests whether the defense mechanism targets dissimilar traffic.
   - **Opposite Far Packet:** A second distant packet, located in the opposite corner of the feature space. If this packet and the previous one show similar drop rates, it suggests that the defense is not using similarity-based technique.

3. The results from these probing packets, particularly their drop rates, are used as signature for the defense. This signature reflects how the defense distinguishes between traffic types, enabling the attacker to classify the defense and plan subsequent attacks accordingly.

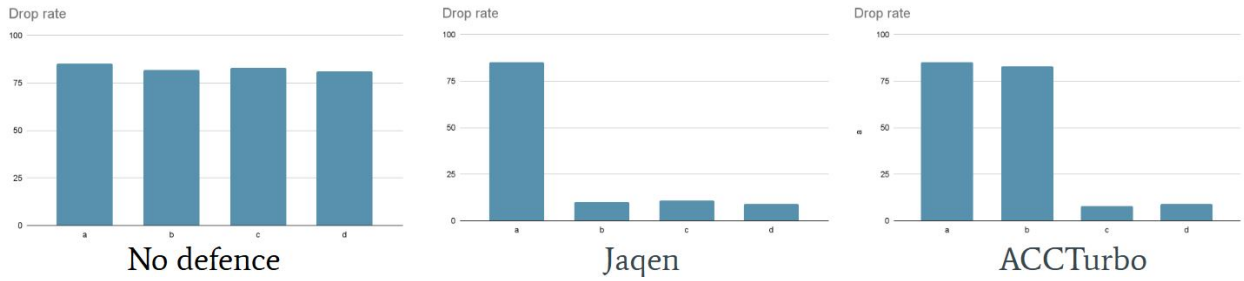


Figure 3.2: Example of signature for the different defenses

The expected signatures are shown in Figure 3.2 and represent the behavior of the three defenses as follow:

- **No Defense:** When no defense mechanism is in place, the expected drop rates for all four probing packet types should be nearly identical. This is because, without prioritization or packet filtering, all incoming packets are processed in the same queue. Consequently, they experience similar treatment and drop rates.

- **Jaqen Heavy Hitter:** Jaqen is based on a signature-based mechanism, hashing specific packet features to decide whether a packet matches an attack profile. Only packets with the exact same features as the flood packet will match the signature and be dropped. Probing packets with even the slightest changes in the features used in the hash (except in cases of hash collisions, which are ignored here for simplicity) will not match the signature and thus will not be dropped. The expected signature is then a high drop rate for the flooding packet and a low drop rate for all other probing packets.

- **ACCTurbo:** ACCTurbo clusters traffic based on feature similarity and assigns priorities to these clusters. The expected signature will then show:
  - High drop rate for the flooding packet as it occupies a cluster identified as malicious and assigned a low priority
  - High drop rate for the nearby packet since it likely falls into the same cluster and experiences an equivalent drop rate.
  - Low drop rate for the both far packets since these packets are unlikely to fall into the same cluster. They will instead be assigned to a cluster with a higher priority.

The goal of our defense classifier is then to identify this different signatures to deduce the defense used on the victim network.

3.3.2 Implementation

The purpose of the defense classifier is to distinguish between the three defense mechanisms based on the signatures of drop rates described earlier. However, the absolute drop rate values can vary significantly across experiments, influenced by factors such as network setup and attacker bandwidth. Therefore, the classifier should focus on the similarity between the distributions of drop rates rather than their exact values.

When feedback on drop rates is received, the classifier processes the data by first computing the variance of the drop rates across the four probing packet types. It then calculates all pairwise covariances between the four distributions. This process generates a total of seven features: one variance and six covariances, which collectively represent the similarity and correlation between the drop rate distributions.

To understand this representation of the defenses ,features extracted from 100 simulation were plotted pairwise, as shown in Figure 3.3. The visualization highlights that the data seems to form distinct clusters corresponding to the three defense types. However, these clusters are not linearly separable.

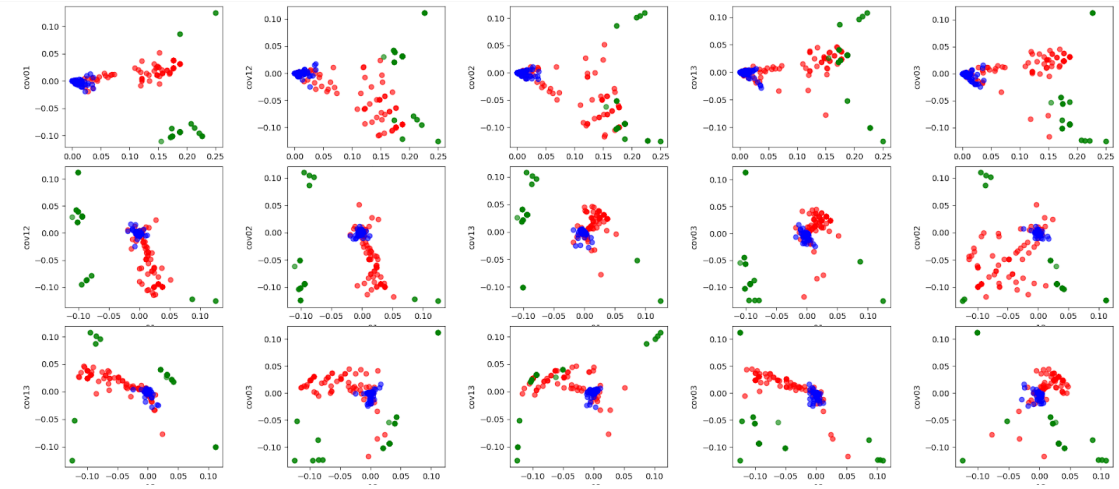


Figure 3.3: Distribution of the variance and the different covariance pairwise in two dimension

Three machine learning algorithms were then tested on a small datasets for their suitability in classifying non-linear separeted classes. The first was **K-Nearest Neighbors** (KNN), a simple

algorithm that assigns classifications based on proximity in feature space. The second was **Random Forest**, a learning method that constructs multiple decision trees to capture complex patterns. The third was a **semi-autoencoder**, a small neural network designed to capture intricate relationships within the data for classification tasks. While all three models performed well, Random Forest was chosen due to its better accuracy.

To train the Random Forest classifier, a dataset was generated. The dataset consisted of 5880 samples collected from 1960 simulations. These simulations were conducted across four distinct network configurations, as described in Table 3.1, and included 70 types of benign traffic combined with five different defense mechanisms: No Defense, ACC5, ACC10, JaqenFast, and JaqenSlow. Before training, the data underwent preprocessing to exclude outliers, remove duplicates, and balance the number of samples across the three defense types, resulting in approximately 1400 samples per defense type.

Table 3.1: Network configurations.

Configuration name	Throughput in Mbps	Link Delay in ns	Buffer Size
ENV1	50	200000	50 packets
ENV2	50	200000	200 packets
ENV3	200	50000	50 packets
ENV4	200	50000	200 packets

With this processed dataset, the Random Forest model was trained to effectively classify defenses based on the variance and covariance features.

3.3.3 Assessment

To assess the performance of our machine learning-based defense classifier, we tested it across 100 simulations using the same network configurations as in the training phase but introducing new benign traffic patterns not included in the training set. The classifier achieved an overall accuracy of **86%**, with precision scores of **100%** for Jaqen, **90%** for No Defense, and **69%** for ACC Turbo. The results are summarized in the confusion matrix shown in Figure 3.4.

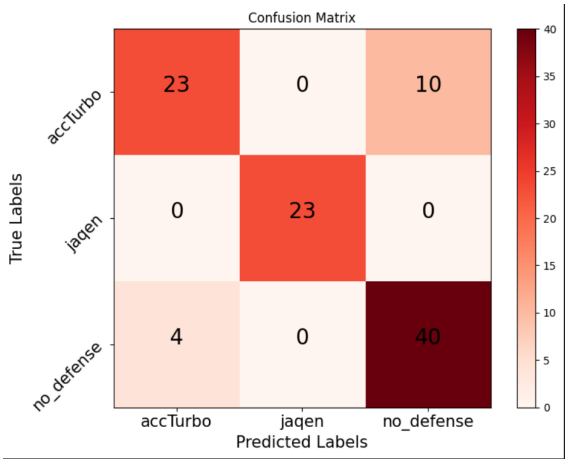


Figure 3.4: Confusion Matrix of the results of the defense classifier for 100 simulations

While this performance may appear satisfactory, a closer look reveals a significant issue: nearly

one-third of ACC Turbo samples are misclassified as No-Defense. This misclassification deserved further investigation.

Upon analysis, we observed that in some cases, the ACC Turbo defense produced an unexpected signature. Instead of the anticipated drop pattern (where only the first two probing packets are significantly dropped) all four probing packets were dropped at similar rates. This result seemed counterintuitive, as the third and fourth probing packets, which are positioned far from the flood packet in feature space, should fall outside the primary cluster. For all four packets to exhibit equivalent drop rates implies that a single cluster covers the entire feature space.

This phenomenon aligns with a issue in ACCTurbo already suspected in [6] and that is discussed in detail in Appendix A. In short, ACC Turbo occasionally experiences a "giant cluster" event where one cluster expands uncontrollably, covering the entire feature space and even overlapping with other clusters. Under these conditions, every incoming packet is assigned to this "giant cluster". As a result, the packets are treated identically and are processed in a single FIFO queue.

For an attacker, this misclassification of ACCTurbo as No-Defense can be considered as an attack vector. When ACCTurbo enters a giant cluster state, it behaves as No-Defense. In fact, this scenario is even more favorable for the attacker, as ACCTurbo's smaller queue sizes (divided among clusters) lead to higher packet drop rates compared to a system with No-Defense. Consequently, deploying the No-Defense flood strategy in this scenario remains optimal.

Recognizing this behavior, we refined our evaluation by treating ACC Turbo with a giant cluster as a distinct category. With this adjustment, the classifier's accuracy improved to 93%, with precision scores of 100% for Jaqen, 90% for No Defense, and 88% for ACC Turbo.

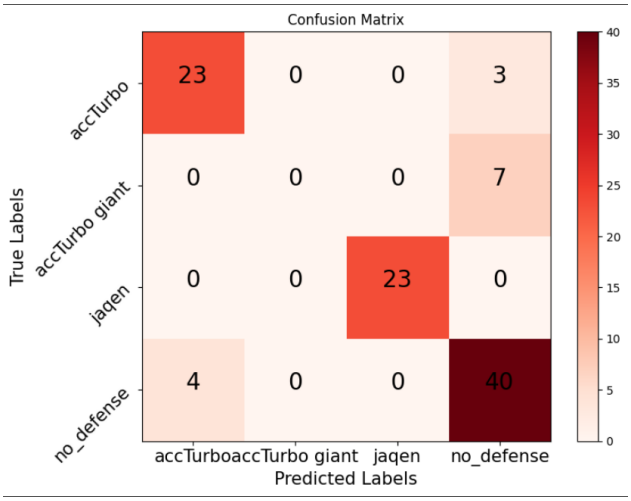


Figure 3.5: Confusion Matrix of the results of the defense classifier for 100 simulations with giant cluster classification added

To further assess the robustness of our classifier, we tested it on four new network configurations: two with very limited capabilities and two with extensive capabilities. This network configuration can be found in Table 3.2. Evaluating our classifier on a network setup different from the one used during the training phase allows us to determine whether our model can generalize to new environments. These setups represent environments significantly different from the training set. The results, shown in Figure 9, demonstrate 96% accuracy, with precision scores of 100% for Jaqen, 91% for No Defense, and 98% for ACC Turbo.

These findings validate our decision to rely on variance and covariance features instead of absolute drop rate values. The classifier proves to be highly effective across diverse network con-

Table 3.2: Network configurations.

Configuration name	Throughput in Mbps	Link Delayin ns	Buffer Size
ENV5	200	200000	50 packets
ENV6	200	200000	200 packets
ENV7	50	50000	50 packets
ENV8	50	50000	200 packets

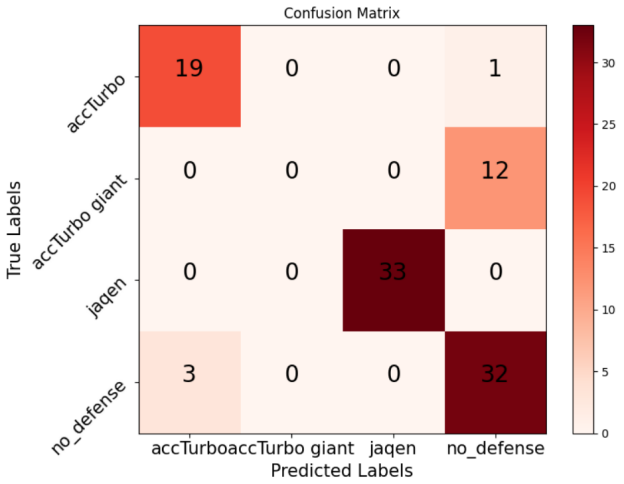


Figure 3.6: Confusion Matrix of the results of the defense classifier for 100 simulations on different network configurations

figurations, as it focuses on the relationships between drop rate distributions rather than their absolute magnitudes.

This evaluation highlights the effectiveness of our defense classifier in accurately identifying the defense mechanism in use. By using statistical similarity features, the model adapt well to different network conditions. This ensures that attackers can reliably determine the active defense mechanism to later deploy the most effective flooding strategy.[1]

3.4 Jaqen Parameters Extractor

Jaqen Parameters Extractot purpose is to determine set up of the Jaqen algortihm running on the defense. The primary objective is to identify which packet features (among Source IP, Source Port, Destination IP, Destination Port, TTL, Protocol, and Length) are utilized by Jaqen to compute its signature. Knowing this information will help an attacker to better mutate its attack.

3.4.1 Design

The design of the Jaqen Parameter share some similarity with the Defense Classifier. Like the previous one the module begins by continuously flooding the network with a specific packet type to

[1]This experiment can be reproduced by running the *./run_ml_morphing_ddos_defense_classifier.sh* script. The results are stored in the *netbench/projects/morphing_ddos_with_ml/ML_defense_classifier* folder.

trigger the defense mechanism. Under normal circumstances, packets sharing the same signature as the flooding packet are blocked by Jaqen.

Next, the module sends probing packets, each identical to the flooding packet except for one altered feature. For each feature, a separate probing packet is generated, varying that feature while keeping all others constant. For example, one probing packet may alter the Source IP, while another modifies the TTL, and so forth.

The critical feedback lies in the drop rate of these probing packets. If a particular feature is used in Jaqen's signature computation, changing that feature alters the packet's signature. As a result, the probing packet would no longer match the flooding packet's signature and would not be blocked by Jaqen's blocking mechanism, leading to a low drop rate.

Conversely, if a feature is not used in the signature computation, altering it does not affect the packet's signature. In this case, the probing packet still matches the flood packet's signature and is dropped, resulting in a high drop rate.

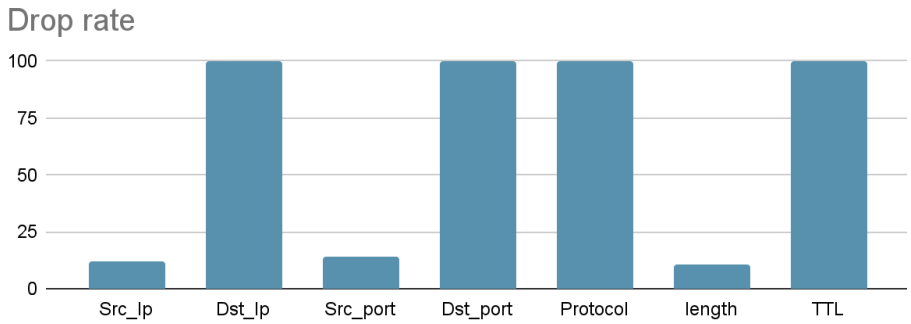


Figure 3.7: Drop rate of the probing packets against a specific Jaqen set up

Figure 3.7 illustrates an example drop rate distribution for a simulation where Jaqen utilized Source IP, Source Port, and Packet Length to compute the signature. In the figure, packets that modified these specific features exhibit significantly lower drop rates, clearly distinguishing them from packets altering features not used in the signature.

Analyzing the feedback and statistics from the probing packets could have been approached using machine learning techniques, like in the previous module. Initially, this was considered a viable solution for the Jaqen Parameter Extractor. However, after examining data from multiple simulations, it became evident that the statistics are so self-evident that a straightforward if/else construction can accurately identify the features used in the signature computation. This simpler approach avoids unnecessary complexity while maintaining high precision in the results.

3.4.2 Assessment

To assess the effectiveness of the Jaqen Parameter Finder module, we implemented a custom Jaqen defense within the Netbench. Previous implementations of Jaqen in Netbench were limited, with one version computing the signature exclusively on the source IP and another version using a fixed set of features: source IP, destination IP, source port, destination port, protocol. To provide a more versatile testing setup, we extended this implementation to enable Jaqen to compute signatures over a configurable subset of the seven packet features: source/destination IP/port, TTL, protocol, and length.

For each Netbench run, this new Jaqen implementation randomly selects a subset of features from the seven to compute the signature. Additionally, we varied the "speed" configuration of

Jaqen, randomly assigning it to either a "fast" Jaqen configuration (1.5-second timeout and a threshold of 5000 packets) or a "slow" Jaqen configuration (4.5-second timeout and a threshold of 15,000 packets).

We conducted 100 simulations, each with different benign traffic profiles. Over the 100 evaluation simulations, the Jaqen Parameter Finder achieved **100% accuracy** in identifying the packet features used for signature computation. This result demonstrates the module's ability to extract the correct subset of features regardless of the random configurations or benign traffic variations.[2]

3.5 AccTurbo cluster Finder

The ACC Turbo Cluster Finder module is designed to analyze and identify as many distinct clusters as possible within the defense mechanism of ACC Turbo. This is crucial for optimizing an attacker's flooding strategy. ACC Turbo employs a prioritization mechanism across different clusters, flooding only one cluster would have no effect on other cluster. By identifying more clusters, an attacker can more effectively distribute their attack.

3.5.1 Idea

The design of the ACC Turbo Cluster Finder module follow a strategy similar to the previous modules. First, it continues flooding the network with a specific packet to trigger ACC Turbo's defense mechanism. Under normal operation, this flood packet will populate a specific cluster, deprioritizing it and causing its queue to fill up. As a result, packets going in this cluster are more likely to experience drops, while packets associated with other clusters should face minimal or no drops.

The second step involves sending random probing packets and measuring their drop rates. If a probing packet experiences a significantly lower drop rate compared to the flood packet, it indicates that the packet belongs to a different cluster. The challenge then becomes estimating the likelihood of a random packet falling into a distinct cluster.

This probability can be mathematically expressed as the ratio of the size of the cluster to the total size of the feature space:

$$P(\text{new cluster}) = \frac{\text{Cluster Size}}{\text{Feature Space Size}}$$

To better understand this, we conducted 100 simulations and computed this probability by measuring the size of the second-largest cluster in each simulation, assuming the flood packet is associated with the largest cluster. Figure 3.8 show the results of these experiments.

The findings reveal that in approximately 55% of simulations, the probability of discovering a new cluster remains feasible, staying above 1 in 1,000,000. However, in the remaining 45% of simulations, the probability becomes exceedingly low. This divergence can be attributed to ACC Turbo's behavior.

- In some simulations, multiple clusters grow to substantial sizes, ensuring better balance across clusters.
- In other simulations, ACC Turbo creates a dominant "giant cluster" (detailed in Appendix A) that dwarfs the remaining clusters. These smaller clusters become overspecialized, optimized for specific benign traffic patterns, such as:

[2]This experiment can be reproduced by running the *./run_ml_morphing_jaquen_features_extractor.sh* scrypt. The results are stored in the *netbench/projects/morphing_ddos_with_ml/jaqen_features_extractor* folder.

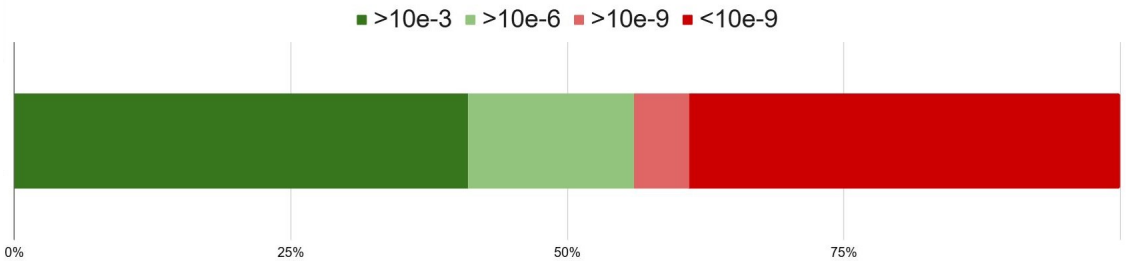


Figure 3.8: Proportion of the second cluster over the space size for 100 simulations

- A cluster that only accepts packets destined for a single or a few specific IP addresses.
- A cluster that only accepts traffic from a narrow /24 subnet of source IPs.

This behavior directly impacts the probability of discovering a new cluster, as overly specialized small clusters offer fewer opportunities for random probing packets to identify distinct clusters. However, we can still confidently try to capture other clusters for more than 50% of the simulation.

3.5.2 Design

The main goal of this phase is to identify a packet that belongs to a new cluster. Instead of randomly generating and testing packets until one ends up in a new cluster, this module leverages techniques used in previous adaptive morphing attack implementations to systematically explore the feature space. This technique is briefly summarized here; however, further details and a comprehensive explanation can be found in [6].

The process begins by dividing the range of each packet feature into n bucket ranges. For each bucket of a given feature, the module sends a probing packet with a random value for the other features and a value within the bucket range for the selected feature. This process is repeated for every feature, with probing packets sent across all buckets, and the feedback from these packets is awaited.

Once the feedback is received, the module computes the drop rate for each bucket for each feature. Based on the results, the bucket with the lowest drop rate is identified for each feature. A value is then selected within this low-drop-rate bucket for each feature, and these values collectively form a new candidate packet that hopefully will end in a new cluster.

The module proceeds to test this candidate packet by sending it multiple times and computing its drop rate. Depending on the observed drop rate, the next steps are as follows:

- If the drop rate is above 0%: This indicates that the packet is likely in a cluster experiencing congestion, meaning it belongs to a cluster already identified. In this case, the module restarts, continuing its search for a new cluster.
- If the drop rate is 0%: This implies the packet belongs to a cluster not experiencing congestion, signifying that it resides in a new cluster. The module adds this packet to the list of flood packets, meaning the attacker will now split bandwidth between the previously identified flood packets and the newly discovered packet targeting the new cluster.

Once a new cluster is found, the module restart the process, now flooding two clusters, in an attempt to discover new clusters, iteratively expanding the attack to target as many clusters as possible.

As a quick remark, this module is not only effective at identifying new clusters but also serves as a potent attack mechanism, as it simultaneously floods all known clusters while searching for additional ones. This will be further developed later in section 3.6.3

3.5.3 Assessment

To assess the performance of the ACC Turbo Cluster Finder, we conducted 100 simulations using 100 distinct benign traffic. Each simulation ran for 200 seconds. During the first 25 seconds, no modules were launched to allow the defense mechanism to stabilize. Following this setup period, we ran the cluster finder iteratively every 5 seconds, resulting in 35 iterations per simulation. After these iterations, we assessed how many clusters had been identified.

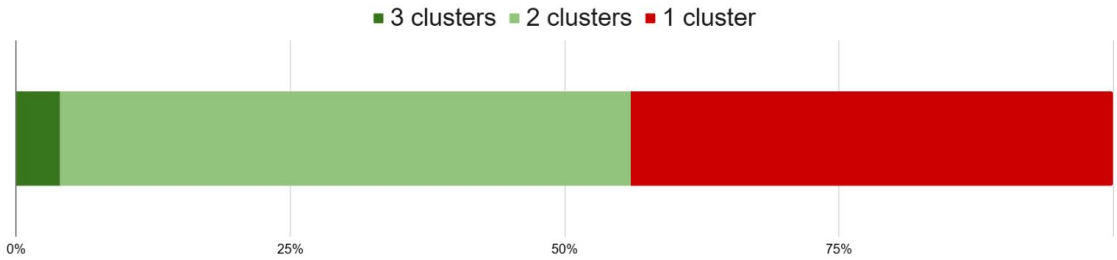


Figure 3.9: Number of clusters found after simulations

The results, summarized in Figure 3.9, show that in 56% of the simulations, more than one cluster was successfully identified, while 44% of the simulations failed to identify additional clusters. This outcome aligns well with the expectations set by the probabilities shown in Figure 3.8, where we anticipated that approximately 56% of the simulations may yield more than one cluster and for the rest it would unlikely to find a new cluster. These results highlight the feasibility of discovering additional clusters, even when the chance of randomly falling into one is as low as 1 in a million.

Interestingly, the results reveal that it is challenging to identify more than two clusters and seemingly impossible to discover more than three clusters in most cases. A deeper analysis of the cluster distribution shows that in the majority of simulations, two to three clusters remain highly specialized and very small, making them exceedingly difficult to detect.

Despite these limitations, the cluster finder remains effective as an attack strategy. If the objective is to identify additional clusters, the module can simply be run iteratively for a longer period.

In conclusion, the ACC Turbo Cluster Finder performs well in identifying clusters, achieving solid results in over half of the simulations. Although it can be challenging to find more than two or three clusters in certain cases, it is effectiveness as both a discovery and attack module. To enhance its performance, we suggest that instead of treating the cluster finding phase as a one-off step followed by an attack, running the module iteratively throughout the entire attack would yield better results.

3.6 Flood attacks

Having gathered sufficient information about the defense mechanisms through the preceding modules, we are now ready to launch flood attacks optimized to bypass these defenses. These attacks are designed to be highly effective in overwhelming the targeted system while maintaining efficiency and simplicity, ensuring they remain feasible for the attacker to execute with minimal overhead.

3.6.1 No defense flood

This module is employed when the defense classifier determined that no defense mechanisms are active on the victim's side. In such scenarios, the attacker can execute a straightforward volumetric attack without the need to adapt the flood packets' characteristics. The strategy is simple: use the entire available bandwidth to flood the network with a constant, unmutated packet. By saturating the victim's resources, this attack maximizes the impact while minimizing complexity for the attacker.

3.6.2 Jaqen flood

This module is activated when Jaqen is identified as the defense mechanism and the relevant packet features used in its signature computation have been extracted using the Jaqen Extractor. In this attack, we leverage the capacity for mutation to ensure that the flood packets continuously evade Jaqen's filtering mechanism. The primary objective is to mutate the flood packet every second so that it no longer matches the current Jaqen signature and bypasses its defenses.

Rather than randomly mutating all packet features, which would impose unnecessary complexity on the attacker, we focus on selectively mutating only one feature known to contribute to the signature. This approach guarantees that the mutated packet avoids detection while minimizing computational effort.

To further define the process, we prioritize certain features for mutation based on their ease of modification. The chosen order is as follows:

1. Source Port
2. TTL (Time to Live)
3. Length
4. Protocol
5. Source IP
6. Destination Port
7. Destination IP

This prioritization reflects the relative cost of mutating each feature. For example, changing the source port or TTL involves minimal effort, requiring only minor packet adjustments. In contrast, altering the source or destination IP may necessitate more complicated spoofing techniques, which are more challenging for the attacker.

By adhering to this feature hierarchy, the attacker can optimize the "mutating cost," a concept explored in greater detail in chapter 4, enabling a balance between attack effectiveness and resource efficiency.

3.6.3 AccTurbo flood

As described earlier, we opted not to design a dedicated ACC Turbo flood module. Instead, we rely on the continuous and iterative operation of the ACC Turbo Cluster Finder. This approach serves as both an effective attack strategy and an adaptive method to maximize disruption on ACC Turbo.

The first reason for this decision is that the cluster finder inherently functions as an optimal attack for the current state of knowledge regarding ACC Turbo's cluster. A direct flood attack against ACC Turbo would aim to distribute the attacker's bandwidth across all known clusters to overwhelm them simultaneously. This is precisely what the cluster finder do. By flooding each identified cluster as they are discovered, the cluster finder ensures that all known clusters are effectively targeted.

The second reason is that by iteratively running the cluster finder, we maintain the potential to discover new clusters over time, even if the probability of finding certain small or specialized clusters is minimal. As observed in Section 3.5.1, some clusters in ACC Turbo configurations can be extremly small and, consequently, unlikely to be discovered through random probing. However, by continuously searching, we can eventually stumble upon these clusters, even with a very low probability.

In summary, the iterative operation of the cluster finder does not hinder the attack's effectiveness, even in the worst case, and holds the potential to enhance the attack by identifying and targeting additional small clusters. For these reasons, we use the ACC Turbo Cluster Finder as the core mechanism for our ACC Turbo flood attack.

Chapter 4

Evaluation

In this chapter we will assess the performance of the newly developed attack methods in comparison to the adaptive morphing DDoS attacks introduced in a previous project [6]. Specifically, we want to determine whether our novel approaches can compete with or surpass the older methods in two critical aspects:

- **Efficiency**: At what point does the attack effectively flood the network and disrupt traffic?
- **Adversarial Effort**: How easy or burdensome is the attack for the adversary to execute in terms of computational and/or operational resources?

To achieve this, we will first define the metrics used to evaluate these two dimensions of performance. Then, for each defense type, we will analyze and compare the results of the new attack methods against the baseline provided by the adaptive morphing DDoS attack. This approach will allow us to draw clear conclusions on the relative strengths and weaknesses of the methods.

4.1 Metrics

4.1.1 Efficiency Mettrics

To evaluate the efficiency of our attacks, we focus on how much they disrupt the victim's legitimate traffic. Specifically, we compute the percentage of background traffic dropped during the attack. This numerical metric provides a clear indicator of the impact of the attack on legitimate users, effectively showing how much of the benign traffic fails to reach its destination.

In addition to this numerical assessment, we visualize the drop rate of each packet type (benign, flood & probing) over time during the simulation. These graphs give a detailed view of the attack's effects throughout its execution, helping us understand how the attack evolves as it progresses. This combination of quantitative metrics and visual insights allows for a comprehensive evaluation of the attack's efficiency.

To provide a thorough comparison, in addition to evaluating our new attacks against the previous adaptive morphing DDoS attacks, we also use a simple volumetric flood attack as a baseline. This baseline attack, being the simplest possible, involves flooding the network with the full attacker bandwidth without any packet mutation. By including this straightforward approach, we establish a reference point to better assess the improvements offered by the two other advanced attack strategies.

4.1.2 Adversarial Effort Mettrics

In this section we detail the metrics used to evaluate the effort required from an adversary to carry out the attack effectively. The effort is measured through three metrics: **Mutating Cost**, **Number of Probing Packets Sent**, and **Time of Computation**. These metrics provide a comprehensive view of the burden placed on an attacker to execute and maintain the attack.

Mutating Cost

The Mutating Cost metric represents the computational and logistical burden for an attacker to mutate specific features of the flood and probing packets. Not all features of a packet are equally easy to mutate; therefore, the mutating cost accounts for the relative difficulty of altering each feature.

For each feature, we define a binary variable x_i

- $x_i = 1$: The feature is mutated.
- $x_i = 0$: The feature remains unchanged.

The mutating cost is then calculated using the formula:

$$MutatingCost = \sum_{i=1}^{n} w_i * x_i$$

Where:

- n : The total number of features, generally 7(src/dst_ip, src/dst_port, ttl, protocol, length)
- w_i : The weight assigned to the i-th feature, representing the effort required to mutate it.
- x_i : The mutation indicator for the i-th feature (1 for mutation, 0 otherwise).

The weights are assigned as follows, based on the complexity of mutating each feature:

- TTL, length, protocol, src_port, dst_port: Weight = 1 (Straightforward to mutate).
- src_ip: Weight = 2 (Requires coordination to use alternate IPs from a botnet).
- dst_ip: Weight = 3 (Requires careful targeting within a small subnet to bypass the target router).

For each simulation, we compute two mutating costs:

1. Mutating Cost for Flooding Packets: Measures the cost of maintaining the attack by continually mutating the flooding packets.
2. Mutating Cost for Probing Packets: Measures the cost of discovering defense characteristics or clusters through mutated probing packets.

This metric allows us to assess the adversary's burden in both maintaining the flood and adapting their probing strategy.

Number of Probing Packets Sent

The Number of Probing Packets Sent metric evaluates the effort required by the attacker to gather information about the defense mechanism. While probing packets are a small fraction of the overall traffic compared to flooding packets, they impose a distinct burden on the attacker. Unlike flooding packets, which require no further action once sent, probing packets necessitate ongoing management. The attacker must maintain a state in memory for each probing packet while awaiting feedback. Once feedback is received, it must then be processed and aggreagte to the corresponding statisitcic.

Reducing the number of probing packets needed to achieve the same results benefits the attacker by lowering computational and memory demands. Consequently, this metric plays a crucial role in assessing the effort involved in executing the attack.

Time of Computation

The Time of Computation metric assesses the processing effort required by the attacker once feedback from probing packets is received. This metric measures the duration taken to analyze the feedback, compute necessary data, and decide the next step in the attack process. While the computation time in our simulations is relatively small compared to the overall duration of an attack cycle, it remains an essential factor.

The significance of this metric lies in its ability to highlight the processor cost for an attacker. This metric serves as a basis for comparing the computational efficiency of different algorithms. By evaluating these variations, we can determine which approaches are better suited for adversaries in terms of processing demands, making it a key component of our overall assessment.

4.2 Against no defense

For efficiency, the results shown in Table 4.1 indicate that all three attacks achieve roughly the same flooding effectiveness, with around 80% of legitimate traffic being dropped. This outcome is expected, as against an undefended network, even the simplest attack, such as a volumetric flood, is already optimal.

Table 4.1: Percentage of dropped background traffic

Simple Flood	Adaptive Morphing	ML Morphing
82.36	80.51	83.23

The attacker effort metrics, however, reveal more nuanced differences. For mutating cost, as detailed in Table 4.2, the adaptive morphing attack incurs a significant burden. At each cycle, it has a probing mutating cost of 8000 and a flood mutating cost of 10. This high cost is attributed to its strategy of mutating every feature of the packet in its quest to find the "best" packet for flooding. This approach, while thorough, is entirely unnecessary against a network with no defense since any packet is equally effective.

In contrast, the ML-powered attacks demonstrate far greater efficiency in terms of adversarial effort. These attacks only require packet mutation during the classifier phase. Once the defense is identified as nonexistent, no further mutations are performed, drastically reducing the mutating cost for both probing and flood packets.

The same pattern is observed for the number of probing packets sent and computation time. While the adaptive morphing attack continuously sends probing packets and performs computa-

Table 4.2: Attacker Effort Metrics for attacks against no defense on c cycles

	Adaptive Morphing	ML Morphing
Flood Packet Mutating Cost	10 * c	0
Probing Packet Mutating Cost	8000 * c	30
Probing Packets Sent	800 * c	400
Time of computation [μs]	57416 * c	1129

tionally intensive statistics updates in every cycle, the ML morphing attack, after classification, ceases these activities. It defaults to a simple flood attack, avoiding unnecessary overhead.

Thus, against no defense, our new attacks achieve comparable efficiency while imposing significantly lower costs on the attacker. Moreover, the longer the attack persists, the greater the advantage of reduced adversarial costs for our new method becomes, as the adaptive morphing attack's continual effort compounds over time.

4.3 Against Jaqen

We evaluated the performance of the attack against two instances of Jaqen Heavy Hitter: one with a fast Jaqen cycle of 1.5 second and the other with a slower Jaqen cycle of 4.5 second. Both instance use the same threshold bandwidth to block traffic.

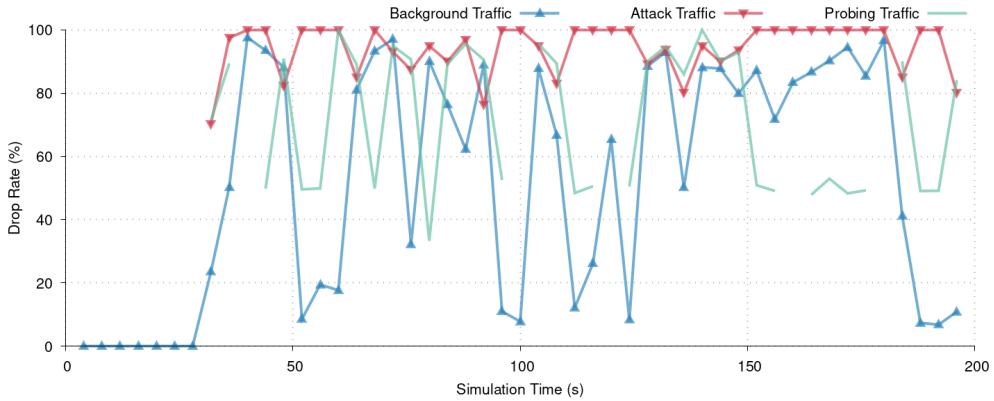
For efficiency, as shown in Table 4.3, both the adaptive morphing attack and the ML-powered attack achieve roughly the same results, significantly outperforming the simple volumetric flood. This outcome was expected, as a normal flood will quickly be blocked by Jaqen due to the static nature of the flood packet's signature. In contrast, both morphing attacks regularly mutate the flood packet, which prevents Jaqen from recognizing the signature and blocking it.

Table 4.3: Percentage of dropped background traffic

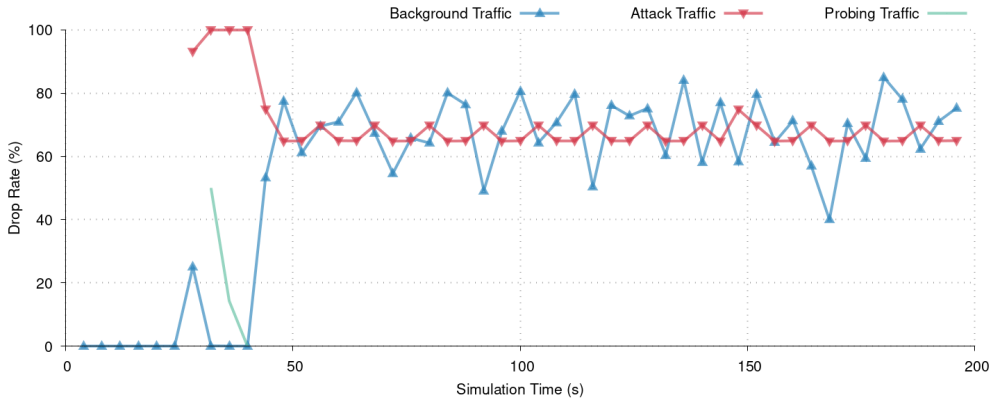
	Simple Flood	Adaptive Morphing	ML Morphing
Jaqen Slow	0.07	76.43	75.42
Jaqen Fast	0.20	59.52	68.38

Notably, when facing the Jaqen slow cycle, both attacks achieve similar drop rates. However, when faced with the fast cycle of Jaqen, the ML-powered attack performs better. This can be explained by the fact that the adaptive morphing attack must wait for feedback before mutating the next flood packet, which can take up to 3 seconds. This delay increases the likelihood that the flood will be blocked by the fast Jaqen defense. In contrast, the ML-powered attack, which no longer requires feedback after the initial classifier phase, can efficiently mutate its packets at a higher rate and bypass Jaqen more effectively, even under fast cycle Jaqen. That can easily be seen in Figure 4.1. The Adaptive Morphing attack experiences a low drop rate, periodically dipping below 20%, because its flooding packet is blocked by Jaqen. In contrast, the ML-powered morphing attack maintains a much more stable drop rate, never falling below 40%, demonstrating its superior ability to bypass the Jaqen defense even during fast cycles.

Regarding attacker effort, the differences between the two morphing attacks are again pronounced. As previously mentioned, the adaptive morphing attack continuously sends probing traffic and recalculates statistics in every cycle. In contrast, the ML-powered attack uses the first two cycles to determine that Jaqen is active and to identify which features are used in its signature



Adaptive Morphing Attack



ML Morphing Attack

Figure 4.1: Packets drop rate for both attacks against Jaqen Fast

computation. After that, it no longer needs to perform further computations, enabling the attack to mutate packets more efficiently.

Table 4.4: Attacker Effort Metrics for attacks against Jaqen on c cycles

	Adaptive Morphing	ML Morphing
Flood Packet Mutating Cost	10 * c	1.068 * c
Probing Packet Mutating Cost	8000 * c	30 + 10
Probing Packets Sent	800 * c	400 +350
Time of computation [μs]	57416 * c	1129 + 0.025 + c * 0.102

Another notable difference lies in the mutation cost of the flood packets. The adaptive morphing attack mutates every feature of the packet at each cycle, resulting in a high mutation cost. On the other hand, our new attack optimizes the flood packet mutation by only modifying the features that incur the least cost, leading to a mean mutation cost of 1.068 compared to the adaptive morphing attack's cost of 10.

In conclusion, our attacks demonstrate the same impressive results as the adaptive morphing DDoS attack ,and even better results against fast Jaqen, but with significantly lower costs for the attacker. This efficiency improvement makes our attacks not only highly effective but also much

more feasible for adversaries to carry out.

4.4 Against AccTubo

We evaluated the performance of the attack against two instances of ACCTurbo: one with 5 clusters and the other with 10 clusters.

The efficiency results, as shown in the Table 4.5, reveal that both morphing attacks perform significantly better than a simple volumetric flood attack. This was expected since a non-mutating flood packet will always end up in the same cluster, causing it to be deprioritized and effectively blocked by ACCTurbo.

Table 4.5: Percentage of dropped background traffic

	Simple Flood	Adaptive Morphing	ML Morphing
Acc5	10.41	37.73	57.15
Acc10	10.29	46.60	52.76

When comparing the two morphing attacks, we observe that the new attack method outperforms the Adaptive Morphing attack. The reason for this improvement lies in the approach each attack takes. In Adaptive Morphing attacks, the method typically floods one cluster at a time, gradually switching from one cluster to another. While this approach is effective, it is not optimal, as it alternates between clusters, sometimes blocking a cluster with a significant amount of benign traffic, and other times blocking a cluster with less background traffic. This will cause an oscillation that can be seen in Figure 4.2, where the drop rate fluctuates as the attack progresses.

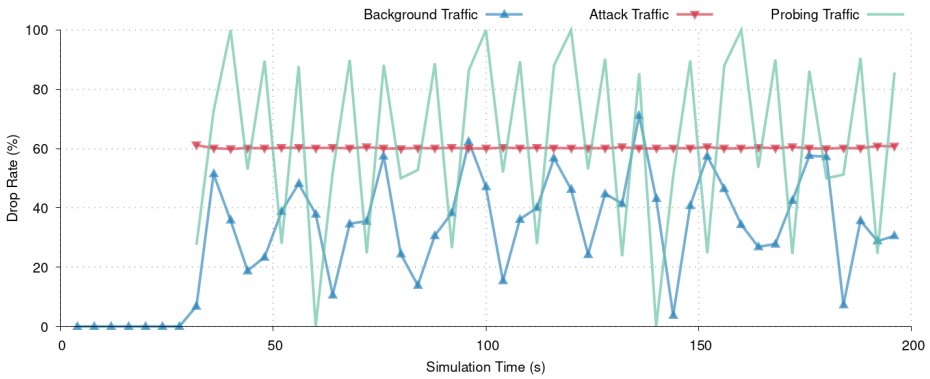


Figure 4.2: Packets drop rate for Adaptive Morphing Attack against ACCTurbo with 5 clusters

In contrast, the new attack method never stops flooding a discovered cluster. As the attack progresses, more clusters are discovered and flooded, leading to a more persistent and efficient DDoS attack that targets larger portions of the victim's traffic. This ability to continuously target and flood multiple clusters simultaneously is the key advantage of the new attack.

On the adversary cost side, both attacks achieve roughly the same cost metrics. This is due to the fact that the new attack is built on techniques used in the Adaptive Morphing attack, with only a slight overhead introduced. The additional computational cost stems from the need to verify whether a newly discovered packet is indeed part of a new cluster. However, this overhead appears minimal compared to the rest of the attack and does not significantly affect the overall performance.

Table 4.6: Attacker Effort Metrics for attacks against Jaqen on c cycles

	Adaptive Morphing	ML Morphing
Flood Packet Mutating Cost	10 * c	(10 if find new cluster else 0) * c
Probing Packet Mutating Cost	8000 * c	30 + 8010 * c
Probing Packets Sent	800 * c	400 + 820 * c
Time of computation [μs]	57416 * c	1129 + 61451 * c

In conclusion, while the new attack may incur a slight overhead compared to the Adaptive Morphing attack, it significantly improves the efficiency of the DDoS, flooding a larger portion of the target's traffic. The new attack's ability to continuously target multiple clusters makes it more effective and damaging, positioning it as a better option for the adversary.

4.5 Lowering the Threat models

This section explores the implications of lowering the adversary's capabilities by limiting their access to a subset of source IPs. In the previous experiments, the adversary was assumed to have significant capabilities. However, this assumption represents an idealized scenario that may not always align with real-world constraints.

One of the most notable limitations in real-world scenarios is the ability to spoof source IPs for probing packets. While spoofing the source IP for flood packets is trivial—requiring only a modification of the packet header—this is not the case for probing packets. Probing packets require feedback to be received by the source machine. If the source IP is spoofed with ip that attacker does not have control, no feedback will be returned, rendering the probing packet ineffective.

To reflect this limitation, we adjust our threat model in the following experiments by restricting the attacker to a subset of all possible source IPs. This reduced threat model aims to more accurately represent a realistic adversary's capabilities.

4.5.1 Mirai Ips

To select the subset of source IPs for our adversary under the reduced threat model, we aimed to use a realistic distribution that reflects plausible adversarial capabilities. For this purpose, we chose to base our experiments on IPs associated with machines compromised by the Mirai botnet.

The Mirai botnet is a well-known botnet that primarily targets IoT devices, compromising them to launch distributed denial-of-service (DDoS) attacks. Its relevance lies in its widespread impact and the realistic network footprint it provides, making it an ideal basis for modeling adversarial capabilities in our experiments. The botnet's scanning techniques are well-documented, and its compromised [4] devices span a wide range of IPs distributed across different geographical and network locations.

To obtain the list of IPs, we leveraged data from Mirai Tracker[11], a website dedicated to monitoring devices infected by the Mirai botnet. Mirai Tracker identifies compromised machines by deploying honeypots that mimic vulnerable devices. When these honeypots are scanned by Mirai instances—using its characteristic scanning techniques, which leave a distinct fingerprint (as detailed in [Paper 8])—the associated IPs are logged.

We scraped the Mirai Tracker website to extract a dataset of approximately 20000 IPs detected over the last three months. This dataset provides a realistic representation of IPs compromised by Mirai, reflecting both their distribution and their potential use by an adversary in launching

attacks.

4.5.2 Adapted Attacks

To accommodate the reduced threat model where the adversary can only use a subset of IPs, we adapted our attack methodology. This primarily involves modifying how source IPs are chosen during the attack to ensure they fall within the constraints imposed by the subset.

The first step in this adaptation was the development of a new malicious traffic generator tailored to the reduced adversary model. This generator ensures that all source IPs used in both probing and flooding traffic are selected exclusively from the predefined botnet IP list. Unlike the unconstrained model, where any IP could be spoofed, the generator adheres strictly to the allowed subset, maintaining compliance with the threat model's limitations.

The overall strategy of the attacks remains unchanged. The adversary still probes defenses, mutates packets, and targets clusters as before. However, when selecting source IPs for these operations, the generator identifies the "closest" IP within the botnet list to the one that would have been used in the ideal case. This adaptation ensures that the attack retains its effectiveness as much as possible, even with the constrained set of IPs.

To determine the closest IP from the botnet list, we implemented a distance function inspired by the clustering logic of ACCTurbo. Specifically, the distance between two IPs, represented as $i_0.i_1.i_2.i_3$ and $j_0.j_1.j_2.j_3$, is calculated using the formula:

$$\text{Distance}(i, j) = |i_0 - j_0| + |i_1 - j_1| + |i_2 - j_2| + |i_3 - j_3|$$

This metric evaluates the similarity between IPs based on their numerical components, ensuring that the replacement IP chosen from the botnet is as close as possible to the ideal IP.

In practice, this adaptation affects both probing packets and flood traffic. For probing packets, the closest IP is selected to ensure that feedback is received and defenses are probed effectively despite the constraints. For flood traffic, the generator substitutes the closest botnet IPs for the original ones, maintaining a realistic attack profile while adhering to the limitations.

4.5.3 Evaluation

We evaluated the performance of the attacks under the constrained adversary model compared to the unconstrained adversary model across various scenarios: *no-defense*, *Jaqen (Fast and Slow)*, and *ACCTurbo (with 5 and 10 clusters)*. The results of this evaluation are summarized in Table **??**.

	Non-constrained Adversary	Constrained Adversary
No Defense	83.21%	83.29%
Jaqen Slow	76.40%	75.42%
Jaqen Fast	63.38%	66.72%
AccTurbo 5	72.29%	57.15%
AccTubo 10	52.76%	55.25%

Table 4.7: Benign traffic drop rate caused by constrained and unconstrained adversary against the different defenses

Overall, the constrained adversary performs almost identically to the unconstrained adversary in most scenarios, with the mean drop rate being approximately the same. This demonstrates that

the reduced access to source IPs does not significantly hinder the efficacy of the attacks in the majority of cases. Furthermore, the classification of the defense mechanisms and the parameter extraction process continues to function flawlessly, ensuring that the attacks remain well-targeted and effective.

The sole exception occurs in the ACCTurbo configuration with 5 clusters. Upon closer inspection, the difference arises because the unconstrained adversary discovers one additional cluster that the constrained adversary does not. This particular cluster is relatively small and restrictive in terms of the source IPs it accepts. Unfortunately, the constrained adversary's subset of source IPs does not contain any addresses that would map to this cluster, making it impossible for the constrained adversary to include it in its attack.

Chapter 5

Outlook

One major consequence of this work is the demonstration that machine learning-powered adversarial strategies can significantly reduce the effort required by attackers while maintaining, or even improving, attack efficiency. This finding underscores the importance of designing defenses that are not only effective against traditional volumetric attacks but also resilient against adaptive and resource-efficient adversaries. Second important consequence of this project is that it serves as a proof of concept that with ML and some network feedback, it is easy for an adversary to deduce what defense or algorithm is used on a remote server, even if they are not supposed to know it.

Looking to the future, several avenues for research emerge from this work. A natural extension would involve adapting the attacks developed here to operate against more complex and dynamic defense setups. For instance, future studies could explore how these methodologies would act against adaptive defenses that vary their strategies in real-time or employ multiple techniques simultaneously, such as combining rate-limiting, signature-based filtering, and clustering.

Additionally, the parameter extraction capabilities demonstrated in this project could be expanded. Currently, the attacks extract only a subset of defense parameters necessary for the specific attack. Further research could focus on extracting additional information, such as the cycle length of Jaqen or the exact clustering algorithm parameters used by ACCTurbo. This deeper understanding could enable even more finely tuned attacks, such as dynamically adjusting mutation rates to match a defense's reaction time or selectively avoiding feature mutation based on the clustering space.

The discovery of the "giant cluster" problem within ACCTurbo defenses also opens new directions for both defensive and offensive research. On the defensive side, a deeper analysis of this phenomenon could lead to a better understanding of this phenomena and maybe to the discovery of analogous vulnerabilities. Maybe leading to a rework or improvement of ACCTurbo to address these weaknesses. On the adversarial side, it would be intriguing to investigate whether an attacker could exploit this phenomenon intentionally. For example, an adversary might craft packets designed to manipulate the defense's clustering algorithm, causing it to consolidate traffic into a single large cluster, thereby rendering the defense less effective. Developing and evaluating attack strategies based on this concept could expose critical vulnerabilities in clustering-based defenses.

Finally, there are broader opportunities to apply the techniques and insights from this work beyond the scope of DDoS attacks. For example, the parameter extraction and adaptive strategies using intel gathered by ML could be adapted to other contexts, such as evading intrusion detection systems.

In conclusion, this project opens up numerous possibilities for both advancing defensive strategies and exploring new dimensions of adversarial behavior, ensuring that the ongoing arms race

between attackers and defenders continues to evolve.

Chapter 6

Summary

In this work, we introduced and evaluated a novel machine-learning-powered DDos attack capable of dynamically adapting to various defense mechanisms. Unlike previous approaches, our method leverages intelligent classification and parameter extraction to optimize flooding strategies while significantly reducing adversarial effort. Through comprehensive assessments against no-defense, Jaqen, and ACCTurbo configurations, we demonstrated that our attacks achieve comparable or superior efficiency compared to previous methods, with a marked reduction for the adversary costs.

Additionally, we extended our evaluation to a constrained adversary model using a realistic subset of source IPs derived from Mirai botnet data. The results reveal that our attacks remain highly effective even under such limitations, showcasing their robustness and practical applicability.

Bibliography

[1] Alcoz, A. G., Strohmeier, M., Lenders, V., and Vanbever, L. Aggregate-based congestion control for pulse-wave DDoS defense. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 693–706.

[2] Alcoz, A. G., Strohmeier, M., Lenders, V., and Vanbever, L. Aggregate-based congestion control for pulse-wave ddos defense. Presentation at SIGCOMM '22: ACM SIGCOMM 2022 Conference, 2022. Presented at SIGCOMM 2022.

[3] Biondi, P., and the Scapy community. Scapy: Packet crafting for python. `https://scapy.net`, 2003–2024. Accessed: 2024-12-13.

[4] Griffioen, H., and Doerr, C. Could you clean up the internet with a pit of tar? investigating tarpit feasibility on internet worms, 05 2023.

[5] Kassing, S. NetBench. GitHub repository, 2017. Available at `https://github.com/ndal-eth/netbench`. Last accessed on December 12, 2024.

[6] Kernbach, N. Exploring morphing ddos attacks. Master thesis, NSG ETH Zurich, Zurich, Switzerland, 2024. October 2023 to May 2024. Supervised by Prof. Dr. Laurent Vanbever and tutored by Dr. Muoi Tran.

[7] Liu, Z., Namkung, H., Nikolaidis, G., Lee, J., Kim, C., Jin, X., Braverman, V., Yu, M., and Sekar, V. Jaqen: A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches. In *30th USENIX Security Symposium (USENIX Security 21)* (Aug. 2021), USENIX Association, pp. 3829–3846.

[8] Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., and Shenker, S. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev. 32*, 3 (July 2002), 62–73.

[9] Mirkovic, J., and Reiher, P. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev. 34*, 2 (Apr. 2004), 39–53.

[10] Sharafaldin, I., Lashkari, A. H., Ghorbani, A. A., et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp 1* (2018), 108–116.

[11] @zbetcheckin. Mirai tracker. `https://mirai.security.gives`. Accessed: 2024-12-26.

Appendix A

The "Giant Cluster" Problem

The "giant cluster" problem is an issue specific to ACCTurbo that may arise in certain simulations. In essence, it occurs when one of ACCTurbo's cluster grows disproportionately large, spanning across the entire features space and even overlapping other clusters. When this happens, all traffic go through the giant cluster, rendering the clustering mechanism ineffective.

This phenomenon was initially hypothesized in previous works [6], which noted that ACC-Turbo's ever-growing cluster mechanism lead to such an imbalance. However, the problem was not rigorously studied until now. In this appendix, we formally investigate the conditions under which the giant cluster problem arises and propose a simple fix to mitigate its occurrence.

A.1 Apparition

Apparition of the Giant Cluster Problem

To understand the conditions under which the giant cluster problem arises, it is essential to examine certain peculiarities of ACCTurbo. These characteristics, while essential for enabling ACCTurbo to operate at line rate compared to the original ACC, also contribute to the emergence of this issue.

Firstly, ACCTurbo employs an optimization where different clusters are never merged. This ensures computational simplicity but also means that a single cluster can grow without being constrained by the presence of others. Secondly, the Manhattan distance is used to determine which cluster an incoming packet belongs to. While computationally efficient, this distance metric does not account for overlapping regions in high-dimensional spaces, which exacerbates the potential for one cluster to dominate. Finally, at the initialization phase of ACCTurbo, the first packets arriving define the starting points of the clusters. This means that early traffic patterns can disproportionately influence the clustering structure. These three optimizations collectively create conditions where one cluster may grow unconstrained, eventually overlapping other clusters and spanning the entire features space. A simple example on how cluster overlap another one can be seen on Figure A.1 The problem becomes more pronounced as the dimensionality of the feature space increases. In the NetBench implementation of ACCTurbo, a dimensionality of 13 is used, further amplifying the likelihood of this phenomenon. As a result, the giant cluster problem is more likely to occur under such high-dimensional conditions, particularly in scenarios with uneven traffic distributions.

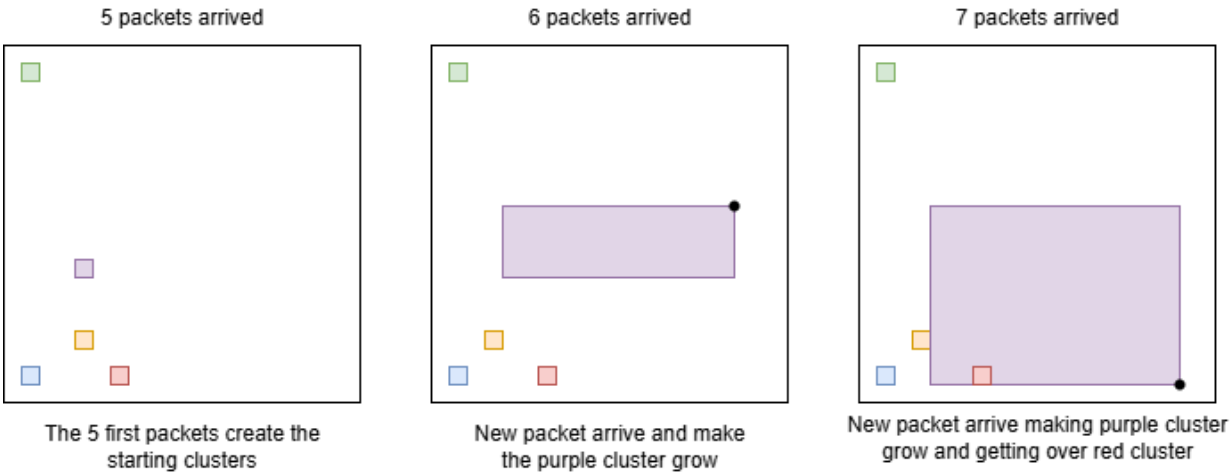


Figure A.1: Example of a run of accTubo (with 5 clusters and two dimensions) in which a cluster overlap another one

A.2 Occurence

Having established the conditions under which the giant cluster phenomenon arises, it is essential to understand how frequently it occurs and whether it is a rare or common event. To assess this, we conducted a series of simulations using ACCTurbo with varying numbers of clusters: 5, 10, 15, and 20. For each configuration, we ran 100 simulations, each lasting 200 seconds, using different traffic. At the end of each simulation, we examined the cluster structure to determine if one cluster overlapped all other clusters, thereby identifying the presence of a giant cluster. The results of

Number of clusters	5 clusters	10 clusters	15 clusters	20 clusters
Proportion of giant clusters	40	11	8	6

Table A.1: Number of simulations experiencing a giant cluster event over 100 simulations

this analysis are summarized in Table A.1. These results reveal that the giant cluster problem is a relatively frequent issue, particularly when the number of clusters is small. For ACCTurbo configured with 5 clusters, a giant cluster appeared in 40% of the runs. As the number of clusters increases, the occurrence of the phenomenon decreases significantly. Doubling the cluster count to 10 reduces the frequency of giant clusters to just 10%. This trend suggests that increasing the number of clusters is an effective way to mitigate the problem, although it does not completely eliminate it.

A.3 Consequences

When a giant cluster emerges in ACCTurbo, the defense mechanism effectively collapses. All incoming traffic is mapped to the same cluster, and packets are processed in a First-In, First-Out (FIFO) manner. This cancel the prioritization and separation of traffic that ACCTurbo is designed to provide, rendering it essentially equivalent to having no defense at all.

Moreover, the situation is exacerbated by the way queue space is allocated in ACCTurbo. Normally, the total queue capacity is divided among all clusters. When a giant cluster forms, this

division remains unchanged, but all traffic is map into a single cluster's queue. This results in a shorter effective queue for the system as a whole compared to an equivalent no-defense router. Consequently, the network becomes even more vulnerable to flooding attacks, as it takes less effort for an adversary to fill the queue and disrupt legitimate traffic.

In summary, the formation of a giant cluster is a severe weakness for ACCTurbo. Not only does it eliminate the defense's intended functionality, but it also reduces the overall resilience of the system, making it easier for attackers to overwhelm the network.

A.4 ACCTurbo+

Acknowledging the giant cluster problem, we proposed a quick and simple fix for ACCTurbo to address this issue in our project. This improved version is called **ACCTurbo+**. Several potential solutions were considered, including merging clusters, improving cluster initialization, and periodically resetting clusters. Each of these options was evaluated for its feasibility and impact on performance:

1. *Merging clusters:* While merging clusters could potentially address the giant cluster problem, it was dismissed due to the significant computational overhead it would introduce. ACCTurbo is designed to operate at line rate, and adding complex merging logic would compromise its efficiency.

2. *Improved cluster initialization:* Another approach considered was better initialization of clusters to prevent uneven growth from the outset. However, this would violate a core assumption of ACCTurbo: its ability to classify traffic automatically and independently, without relying on prior knowledge of legitimate traffic patterns.

3. *Periodic resets:* We decided to pursue periodic resetting of clusters, a strategy already suggested in prior work [6]. However, instead of resetting clusters at fixed time intervals, we opted for a more dynamic approach.

Dynamic Giant Cluster Detection and Reset Mechanism

In ACCTurbo+, we detect giant clusters during runtime by monitoring the distribution of traffic across clusters. At each ACCTurbo cycle, we calculate the percentage of total traffic handled by each cluster. If, in the most recent cycle, more than 99.9% of the traffic is being handle by a single cluster, it is identified as a giant cluster, triggering a reset.

Improved Reset Mechanism

Rather than performing a naive reset by deleting all clusters and allowing new traffic to recreate them from scratch, we implemented an improved reset mechanism in ACCTurbo+. When a reset is triggered, each cluster collapses to its central point. This approach has several advantages:

- **Preservation of priority:** By retaining the central points, the system preserves the memory of past events, including prioritization of traffic. This ensures that if a reset occurs during an ongoing attack, the malicious traffic continues to be deprioritized.

- **Smoother recovery:** Unlike a full reset that would disrupt the classification system entirely, collapsing clusters to their central points minimizes disruption and allows the system to recover more gracefully.

With these enhancements, ACCTurbo+ provides a practical and efficient solution to the giant cluster problem, ensuring robust performance even under challenging conditions.

Testing ACCTurbo+ demonstrated a significant reduction in the occurrence of giant clusters. However in 5% of cases persistent giant clusters seems to appeared. These persistent cases occur

when the reset mechanism fails to prevent the repeated formation of an oversized cluster. While this highlights room for improvement, the current mitigation strategy is considered adequate for the goals of this project. Future work could address this residual issue by exploring more sophisticated techniques.