****

The Arab American University

Faculty of Engineering and Information Technology

**Entropy-Based Approach for DDoS Mitigation in Software-Defined Networks**

**Prepared by:**

Ali Jafar Ali 202120075

Zaid Faisal Abu Hamed 202112816

Ahmad Mohammad Abu Al-rob 202112144

**Supervisor:**

Dr. Mohammad Hamarsheh

**Date:**

**Fall 2024/2025**

Contents

[Declaration 4](#_Toc202712467)

[Abstract 5](#_Toc202712468)

[List of Tables 6](#_Toc202712469)

[List of Figures 7](#_Toc202712470)

[Acknowledgements 8](#_Toc202712471)

[Chapter One. Introduction 9](#_Toc202712472)

[1.1 Scope 9](#_Toc202712473)

[1.2 Aims and Objectives 9](#_Toc202712474)

[1.3 Problem Statement 9](#_Toc202712475)

[Chapter Two. Background and Literature Review 11](#_Toc202712476)

[2.1 SOFTWARE DEFINED NETWORK 11](#_Toc202712477)

[2.2 DENIAL-OF-SERVICE IN SDN 12](#_Toc202712478)

[2.3 Literature Review 13](#_Toc202712479)

[Chapter Three. Methodology 16](#_Toc202712480)

[3.1 Mininet 17](#_Toc202712482)

[3.2 Controller 17](#_Toc202712483)

[3.3 Detection Using Entropy 20](#_Toc202712484)

[3.4 Testbed 22](#_Toc202712485)

[Chapter Four. System Design 25](#_Toc202712486)

[4.1 Introduction 25](#_Toc202712487)

[4.2 Algorithm 26](#_Toc202712488)

[4.3 Traffic Generation and Integration 28](#_Toc202712489)

[Chapter Five. Results and Analysis 31](#_Toc202712490)

[5.1 Results and Analysis 31](#_Toc202712491)

[5.1.1 Window 20 32](#_Toc202712492)

[5.1.2 Window 50 33](#_Toc202712493)

[5.1.3 Window 70 34](#_Toc202712494)

[5.1.4 Window 100 35](#_Toc202712495)

[5.2 Performance Evaluation 36](#_Toc202712496)

[5.3 The effect of window size on CPU consumption: 41](#_Toc202712497)

[5.4 Final Analysis: 42](#_Toc202712498)

[Chapter Six. Conclusion 43](#_Toc202712499)

[References 45](#_Toc202712500)

[Appendices 46](#_Toc202712501)

# Declaration

This is to declare that the graduation project entitled (Entropy-Based Approach for DDoS Mitigation in Software-Defined Networks) under the supervision of (Dr. Mohammad Hamarsheh) is our own work and does not contain any unacknowledged work or material previously published or written by another person, except where due reference is made in the text of the report.

Date: Fall 2024/2025

Graduation project’s students:

Name: Ali Jafar Ali Signature:

Name: Zaid Faisal Abu Hamed Signature:

Name: Ahmad Mohammad Abu Al-rob Signature:

# Abstract

Software-defined networks (SDN) have provided many advantages to networks by centralizing control and offering flexibility. However, the centralization of control provided by SDN has made them vulnerable to various types of attacks, especially against denial of service (DoS) attacks. These attacks focus on the SDN controller, causing network outages and draining resources. This project proposed the use of an entropy-based detection method to mitigate DoS attacks in SDN environments. The method involves analyzing the randomness of traffic flow by calculating the entropy values ​​of destination IP addresses. A significant decrease from normal entropy levels indicates a potential DoS attack. The system is implemented using Mininet for network simulation and Ryu controller for traffic management.

# List of Tables

[Table 3‑‎0‑1 Comparison of SDN Controllers (Ryu, ODL, ONOS) 19](#_Toc202708803)

[Table ‎0‑1Change TH for Each Window 31](#_Toc202708804)

[Table 5‑‎0‑2 Confusion Matrix 37](#_Toc202708805)

[Table 5‑‎0‑3 Confusion Matrix – Window Size 20 37](#_Toc202708806)

[Table ‎0‑4 Confusion Matrix – Window Size 50 38](#_Toc202708807)

[Table 5‑‎0‑5 Confusion Matrix – Window Size 70 39](#_Toc202708808)

[Table 5‑‎0‑6 Confusion Matrix – Window Size 100 40](#_Toc202708809)

[Table 5‑7 CPU Usage by Window Size 41](#_Toc202708810)

# List of Figures

[Figure 2‑‎0‑1 SDN architecture concept 11](#_Toc202708785)

[Figure 3‑‎0‑1 Network topology design using Mininet (Tree Topology). 22](#_Toc202708786)

[Figure 3‑‎0‑2 Running Ryu Controller inside Ubuntu to connection test. 23](#_Toc202708787)

[Figure 3‑‎0‑3 Running the Mininet network and connecting it to the Ryu controller. 24](#_Toc202708788)

[Figure 3‑‎0‑4 Network Connectivity Verification Using pingall 24](#_Toc202708789)

[Figure 4‑‎0‑1 Flowchart of the Entropy-Based Detection Algorithm 27](#_Toc202708790)

[Figure 4‑‎0‑2 Packet Rate. 29](#_Toc202708791)

[Figure 4‑‎0‑3 Flowchart of the Traffic Generation. 29](#_Toc202708792)

[Figure 5‑1 Entropy Variation Over Time – Window Size 20 32](#_Toc202708793)

[Figure 5‑2 Entropy Variation Over Time – Window Size 50 33](#_Toc202708794)

[Figure 5‑3 Entropy Variation Over Time – Window Size 70 34](#_Toc202708795)

[Figure 5‑4 Entropy Variation Over Time – Window Size 100 35](#_Toc202708796)

# Acknowledgements

In the name of Allah, the Most Gracious and the Most Merciful. We would like to take this opportunity to thank Allah, the merciful for being our guide and for giving us the strength to write this thesis and carry on this project and for blessing us with many great people who have been our greatest support in both our professional and personal lives.

First and foremost, we would like to sincerely thank our project's supervisor Dr. Mohammad Hamarsheh for his guidance, effort, and patience. All thanks to our lecturers and instructors. They have always provided positive encouragement and continuous support.

Finally, all thanks to all the group members for sharing the positivity and invaluable assistance.

# Chapter One. Introduction

## Scope

This project seeks to improve Software-Defined Networking (SDN) security against the DoS Denial of Service attack. The goal is to devise a method for early-stage detection of DoS attacks utilizing equivalency class partitioning and the use of entropy as a lightweight statistical measure. The method’s efficiency and adaptability is to be check by simulating and testing it across several SDN topologies designs which include single, linear, and multilevel controller networks. The goal is to generate a detection mechanism that is scalable, accurate, resource efficient, and most importantly, feasible in a real SDN environment.

## Aims and Objectives

This project aims to improve the security of SDN networks against DDoS attacks by network traffic.

The main objectives can be summarized as follows:

1. Develop an Effective Detection Method: Create an DDoS attack detection solution that is effective and can be applied at an early stage of an attack by analyzing patterns of network traffic and the randomness of the destination IP addresses 's entropy.

2. Compare performance across different window sizes.

3. Optimize for Performance: Achieve minimum resource consumption while providing accurate results to make the method applicable to active SDN environments.

4. Analyze Detection Metrics: Performance testing via confusion matrix to determine detection accuracy.

## Problem Statement

While SDN systems present increased centralization in terms of management and control, this architecture introduces a key security concern in the form of a single point of failure – the centralized controller. This makes SDN frameworks extremely vulnerable to the Denial of Service (DoS) attack, in which the controller is bombarded with traffic until resources are depleted, disrupting the network entirely. The existing systems of detection have shown to be inefficient with little to no scalability and adaptability to different network topologies. While DoS attacks become increasingly sophisticated and severe, there’s a gap for real time, accurate, and minimal effort sophisticated detection while working within high overhead environments.

# Chapter Two. Background and Literature Review

## SOFTWARE DEFINED NETWORK

Software Defined Networking (SDN) is a new technology in the world of networks, and simply separates the function of network management and control Plane from the function of Data Plane, in separate entities, but what is the result of this separation and how does it differ from the current situation?! Networks currently combine these two functions in every device on the network! But by removing the Control Plane function from all devices and focusing it on only one device that manages the network Controller, this will lead to the transformation of network management from running services on each network device individually to merely programming that service on the management and control device Controller only! Which in turn as a controller will activate those services and manage and control them on all required network devices automatically. Therefore, SDN technology is a successful attempt to separate the data plane from the control plane so that the role of network devices (switches/routers) is limited to passing data only, while management, control and services will be in new layers, which are the control layer and the application layer. [4], [5], [12]

SDN divides the network infrastructure into three layers:

Application Layer

Control Layer

Data Layer

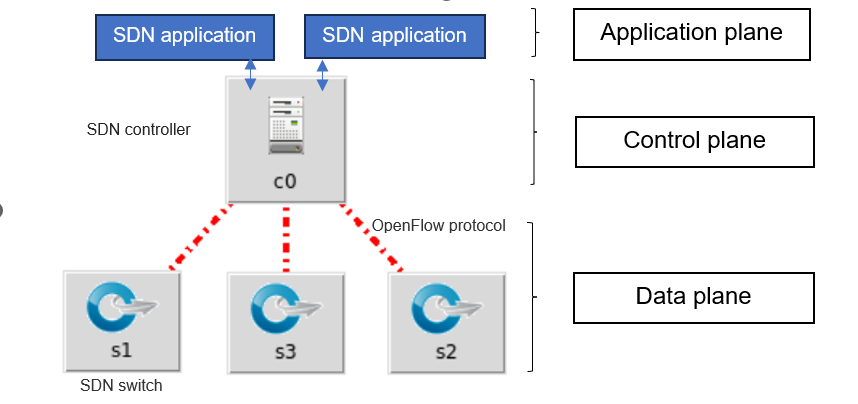


Figure 2‑‑ SDN architecture concept

The advantages of SDN are beneficial in today’s networking world. The main benefit of SDN is the high level of flexibility it provides to an organization to change the behavior of its network as it deems necessary. With centralized management, the entire network is controlled through a single interface; this improves the level of simplicity and visibility of the network. It also has a high scalability; this makes it possible for network administrators to simply add more resources or change the network architecture without much hassle. Moreover, SDN reduces costs as it requires less specialization. [4], [12].

The SDN architecture has some fundamental components. Such as a controller, which is the central system that gives instructions to the network devices that which configuration is to be adopted. Some of them are Open Daylight and ONOS. The network devices including switches and routers are responsible for carrying out the instructions that are handed to them by the controller. This is where applications in SDN are useful in that they use the control layer to implement the network policies and extend the functionality of the network for example virtual firewalls or load balancers. Programming interfaces (APIs) are necessary to enable communication between applications, the controller and network devices in order to ensure that all the components are properly integrated and functioning[4], [6], [7].

## DENIAL-OF-SERVICE IN SDN

A **Denial-of-Service (DoS) attack** is an attempt to disrupt the normal operation of a targeted server, service, or network. It typically works by overwhelming the target with an excessive amount of traffic or exploiting protocol vulnerabilities to consume its available resources. These attacks can lead to service unavailability, degraded performance, or even total system failure. In the context of Software-Defined Networking (SDN), DoS attacks can occur at both the **data plane** and the **control plane**, making them particularly dangerous in centralized network architectures[8], [9].

One of the most common and disruptive types of DoS attacks is the **TCP SYN Flood**. In this attack, the adversary sends a large volume of TCP connection requests (SYN packets) with forged or unreachable source IP addresses. The server, following the standard TCP handshake, responds with a SYN-ACK and waits for a final ACK that never arrives. These half-open connections begin to accumulate, consuming memory and connection resources. As the number of these incomplete sessions grows, the server becomes unable to accept new legitimate connections, resulting in denial of service to actual users.

In our project, we specifically focus on the **SYN Flood attack** within an SDN environment. The attacker sends a high number of TCP SYN packets with spoofed IP addresses to a target within the network. The victim responds and allocates resources to each request, expecting a legitimate handshake completion. However, since the responses never arrive, these pending sessions persist and drain system resources over time. This attack is particularly harmful in SDN networks, where overwhelming the centralized **controller** can degrade the network’s ability to manage and respond, posing a serious threat to overall availability and performance[8], [9], [10].

## Literature Review

* **Implementation of Entropy-Based DDoS Attack Detection Method in Different SDN Topologies:**

This study proposes a lightweight and effective method for detecting DDoS attacks [1] in an SDN environment by calculating the entropy of destination IP traffic. The proposed method has demonstrated its ability to detect attacks across three different SDN network topologies. The experiments were conducted using the Mininet emulator and the OpenFlow protocol, with the RYU controller serving as the SDN control plane.

* **A Collaborative Approach to Detecting DDoS Attacks in SDN Using Entropy and Deep Learning:**

This paper presents a novel approach for DDoS attack detection in software defined networks [2] (SDN) based on entropy and deep learning. As SDN is an approach to the control of networks, the systems are more exposed to DDoS attacks that can cause damage to the network. The primary goal of this study is to improve the accuracy of detecting intrusions by identifying suspicious switches on the basis of entropy and analyzing the traffic flow using deep learning techniques. This two-layer approach is intended to enhance the quality of the attack detection and to contribute to the development of stronger barriers against such attacks.

Previous research has explored various methods for DDoS detection, including machine learning and entropy-based techniques, achieving high accuracy. However, there is often a lack of emphasis on effective feature selection and the utilization of diverse datasets. The study addresses these gaps by employing a robust feature selection process and integrating an SDN-specific dataset to enhance the system’s detection capabilities. This approach ensures a more comprehensive analysis of network traffic and better identification of malicious activities.

Entropy based approach for identifying source IP addresses is explained in the proposed methodology [3] to detect anomalies when the entropy values are less than a certain threshold. This is then followed by the deep learning analysis of the traffic using Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM) networks. It is a real time system for monitoring and classification of traffic and provides a proactive solution to DDoS attacks in SDN.

The system was implemented in a simulated SDN environment to evaluate it, where different machine learning and deep learning techniques were experimented to see how accurately they could identify DDoS attacks. Among the models, the accuracy of the LSTM network was found to be 99.83%, making it the most suitable model for real time detection. The results also show that deep learning is better at identifying the complex patterns in the network traffic than the traditional methods.

In conclusion, the study presents a powerful framework for DDoS detection in SDN, combining statistical and machine learning techniques to achieve high accuracy and reliability. The proposed system not only improves upon existing detection methods but also sets the stage for future research, with plans to test the mechanism in real-world SDN environments using platforms like OpenStack.

* **Entropy-based DoS Attack identification in SDN:**

Mininet was used as a network simulator in the experiment environment while POX as a controller with a basic topology to balance the attacker and the victim comprising 4 switches, 9 hosts and POX controller with 100Mbps bandwidth The victim is located on the network Switch4 whereas the attacker is located on Switch 2 Legitimate packets were generated using scapy while fake SYN packets were generated by hping and normal traffic was generated to obtain entropy values under normal, threshold and abnormal conditions A TCP- SYN attack was simulated to test the behavior of the suggested solution and the destination IP address of entropy was computed with varying sizes of the monitoring window SYN flood and compute the entropy of the destination IP address with different window sizes of monitoring In the experiment a total of 50,000 packets were studied with windows of sizes 100, 80,60,40,20,80,60,60,40,20 The attack was detected after 104 seconds for the window size of 20 packets; the attack was detected after 105 seconds for the window size of 40 packets; the attack was detected after 106 seconds for the window size of 60 packets; and the packet was detected after 108 seconds for the window size of 100 packets. Making less accurate results which is high in positive and negative error rates f1 is less in comparison with other windows which is average compatibility. As you increase the window size, CPU consumption grows. So, based on the previous analysis, test cases were run where an average was calculated between them to find a perfect balance to show that window 60 will give the best performance as it does not put a burden on processing.

# Chapter Three. Methodology



This chapter presents the practical approach used for a Denial of Service (DoS) attack detection project using the entropy algorithm in a Software Defined Networking (SDN) environment[4], [6]. The tools and techniques used, the design of the virtual environment, the method of data execution and logging, and an intelligent approach to analyzing abnormal behaviour are explained. The project relies on building a virtual network environment using the Mininet simulator, integrating the Ryu controller to test the monitoring and analysis of packet flow across the network. The Python natural language and the necessary codes for calculating entropy were used, along with other tools such as hping3 for handling the processing and the Scapy library and its awareness for analysis[10]. This chapter presents new steps for designing the used topology, the automation of each tool within the experimental setup, and explains how to calculate entropy and determine the threshold used by the system to classify packets as normal or attack. Applying this practice can serve as a practical and experimental basis for building smarter and more effective solutions in the field of security networks.

## Mininet

Mininet is an open-source network emulator [7] that allows researchers and developers to simulate complex virtual networks on a single machine. It provides a lightweight, efficient, and easy-to-use environment that includes virtual hosts, switches, controllers, and network links. This enables the testing of various network protocols, routing mechanisms, and security solutions without the need for physical infrastructure. One of Mininet’s core advantages is its native support for the OpenFlow protocol, which makes it especially suitable for Software-Defined Networking (SDN) research and experimentation.

With Mininet, users can easily define and deploy custom network topologies. It offers full control over link parameters such as bandwidth and latency, and allows direct execution of commands on virtual hosts. Moreover, it enables real-time traffic monitoring and interaction with the simulated components, making it ideal for rapid prototyping and performance testing. This flexibility has made Mininet a widely adopted tool in both academic and industrial SDN projects.

In our project, Mininet was used to create a virtual SDN environment based on a **2-depth, 3-branch tree topology**. This topology included several hosts connected through multiple switches to a centralized **Ryu controller**. Within this setup, one host was designated as the **attacker**, while another served as the **victim**, allowing for the controlled generation of both normal and malicious traffic. This testbed provided the foundation for evaluating the proposed entropy-based detection algorithm and measuring its effectiveness in identifying **DoS attacks** in a realistic yet isolated environment.

## Controller

The Ryu controller was chosen as the main component of the control layer [6] within the software-defined network (SDN) environment used in this project. Ryu is an open-source controller written in Python. Its flexible and easily customizable architecture makes it ideal for research and experimental applications that require direct intervention in packet processing and analysis. One of Ryu's most notable features is its full support for the OpenFlow protocol, which is used to communicate between the controller and network components such as switches. Ryu also allows programmers to write their own applications to process incoming packets and interact with data dynamically. Since it is written in Python, Ryu's integration with other analysis tools such as Scapy or entropy calculators is straightforward.

In this project, Ryu's code was modified to collect the destination addresses of packets passing through the network, group them within a specific window, and then calculate their entropy using an algorithm built into the controller. This integration of monitoring and processing makes it possible to detect changes in traffic patterns in real-time, and compare the results of each window against a pre-defined threshold to determine whether an attack exists.

The ease of developing applications within Ryu, along with its support for integration with Mininet, makes it the ideal choice for implementing this type of detection system that relies on analyzing real-time data on the network.

**Comparison of Different SDN Controllers:**

In addition to the Ryu controller, there are several other popular controllers in the software-defined networking (SDN) environment, most notably ONOS and OpenDaylight (ODL). ONOS is an open-source operating system designed to manage large networks, such as those of telecom companies, and features redundancy techniques to ensure high continuity. ODL is also an open-source controller developed to meet the requirements of large enterprises. It supports multiple protocols and is used in complex network projects. Despite the strengths of ONOS and ODL, the Ryu controller remains best suited for academic and research projects that require flexibility and modifiability, especially during the development and testing phases, due to its reliance on the Python language and its ease of integration with the Mininet simulator.

The following table compares the three controllers:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Ryu** | **ODL** | **ONOS** |
| Programming language | Python | Java | Java |
| The ease of use | Research, Development and Experimentation | Large network and service providers | Large organisations and complex project |
| Mininet integration | Easily integrated | Need additional setup | Need additional setup |
| Community and support | Wide | Wide | Wide but more complex |
| Performance | Light and fast | High performance and suitable for large network | Heavy and resource intensive |

Table 3‑‑ Comparison of SDN Controllers (Ryu, ODL, ONOS)

## Detection Using Entropy

In this project, the Shannon Entropy algorithm was adopted as the basis for detecting DoS attacks in a software-defined network environment. This algorithm measures the degree of randomness of the destination IPs of packets passing through the network over a specific period of time, and expresses this distribution as a numerical value known as entropy.

In normal circumstances, packets are directed to multiple destinations, meaning the distribution of destinations is somewhat random, and therefore the entropy value is high. In the case of a DoS attack, the attacker sends a large number of packets to only one target, resulting in a significant decrease in the entropy value due to the high repetition of destination addresses. This change in the value can be considered a strong indicator of an attack.

The entropy in this project was calculated using the following equation:

Where:

: The number of different destinations in the window.

: The probability of destination i recurring within the window (i.e., the number of times it appears ÷ the total number of packets in the window).

The calculation is performed within a time or number window of packets. The controller collects the incoming packets, counts the number of times each destination recurred, and then calculates the entropy based on those recurrences. To classify whether the traffic within the window is normal or attack-related, a threshold is defined against which the comparison is made. This threshold is calculated by taking the average of the first ten entropy values ​​at the beginning of each experiment, the period during which no attack occurred. If any entropy value falls below this average, the window is considered "attack-related."

This approach is computationally simple, yet effective at detecting network distribution changes resulting from attack behavior, making it a suitable choice for early detection systems, especially within an SDN environment that allows complete control and analysis of packet flow.

## Testbed

Testing environment design is a key component of this project, as we worked to build an integrated virtual environment that enables simulation of realistic software-defined network (SDN) attack scenarios. Ubuntu 20.04 was used as the default operating system, due to its stability and broad support for open source networking tools. The operating system was run in VMware Workstation on a machine with 7 GB of RAM and a dual-core processor. This configuration provided a flexible and customizable environment, enabling the researcher to conduct repeated experiments, track data, and accurately analyze performance, without the need for physical infrastructure. This virtual environment provided a suitable platform for conducting advanced experiments in a secure and organized manner.

For the network design, the topology was created using the Mininet simulator. A tree topology with a depth of 2 and a fanout of 3 was chosen, which is suitable for testing because it provides a sufficient number of hosts and switches to represent multiple traffic scenarios. The central controller (Controller) was connected to all switches in the network. One host connected via Switch2 was designated as the attacker (Attacker), while another host connected via Switch4 was designated as the target or victim (Victim). This distribution allows packets to be routed through multiple nodes in the network, accurately demonstrating the impact of attacks at the propagation level and providing a clearer view of the effectiveness of the entropy algorithm in detecting changes in traffic behavior.

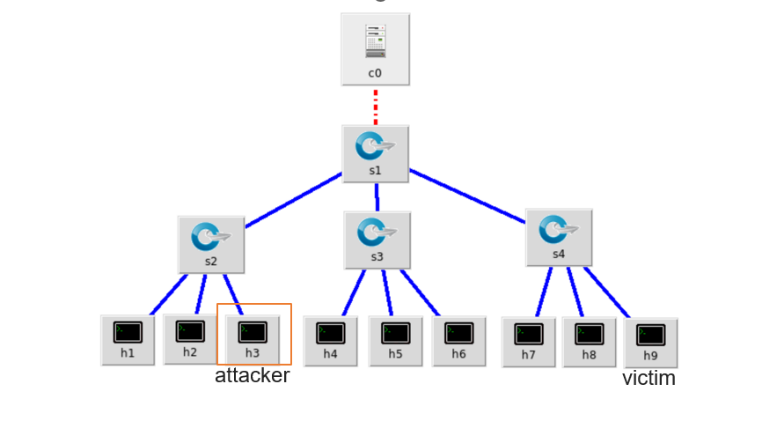


Figure 3‑‑ Network topology design using Mininet (Tree Topology).

For data generation, several software tools were used to implement various stages of the experiment. Initially, natural traffic was generated using Python code that leveraged the Scapy library to send various data packets from one set of devices to another. A SYN Flood DoS attack was then launched using the hping3 tool, sending thousands of concentrated packets to the targeted host during a specified period in the middle of the experiment. Meanwhile, the Ryu controller monitored the flow of packets across the network and calculated the entropy value for each time window, with the results stored in CSV files for later analysis. This environment allowed for the practical application of the mathematical model of the detection algorithm, and provided flexibility in changing settings and testing the system's performance under different conditions in terms of the size, type, and duration of the attack.

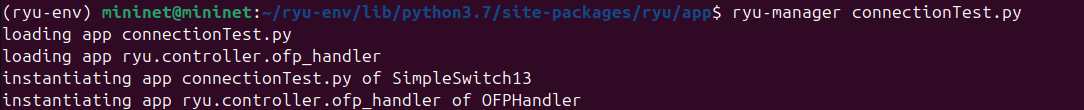
As shown in Figure 3-2 the controller is equipped and is of the type of Ryu and operated

Figure 3‑‑ Running Ryu Controller inside Ubuntu to connection test.

As shown in Figure 3-3the network was created in the Mininet and connected to the controller of the Ryu type:

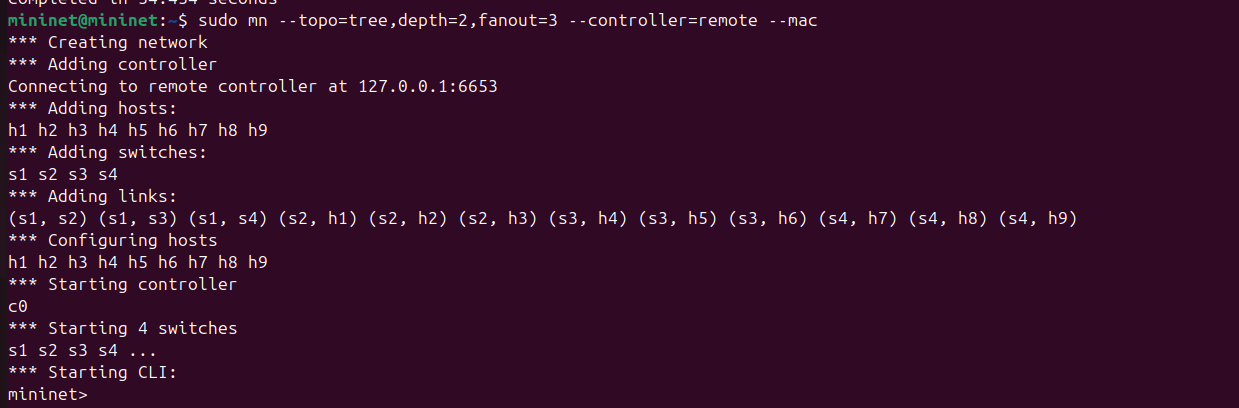


Figure 3‑‑ Running the Mininet network and connecting it to the Ryu controller.

The network was checked using the pingall command to ensure that the network is working correctly and to make sure that it has been linked to the controller:

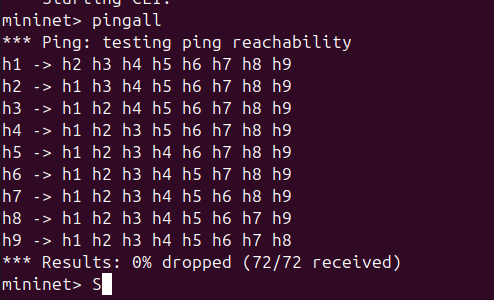


Figure 3‑‑ Network Connectivity Verification Using pingall

# Chapter Four. System Design

## Introduction

The proposed system design is based on achieving intelligent and continuous monitoring of data traffic within a Software-Defined Networking (SDN) environment by integrating an entropy calculation module into the controller components to detect sudden and abnormal changes in routing behavior. This design aims to provide a flexible architecture capable of detecting Denial of Service (DoS) attacks based on packet destination analysis rather than attack signatures or static characteristics.

In this system, components not only operate independently, but interact with each other through a unified mechanism that enables tracking of packet flow from the moment of generation to the classification stage. This is accomplished by distributing roles between infrastructure components (such as virtual switches and servers) and the intelligent controller, where logging, processing, and analysis are performed sequentially within the system in real time.

The entropy calculation module is the analytical heart of the system, relying on a mechanism to segment packets into analytical windows, then calculate the probability of duplicate addresses, and derive the entropy value for each window. When the value drops below a pre-defined threshold, the window is considered suspicious, indicating a possible attack. Thus, the design enables highly accurate dynamic detection without relying on static rules or traditional attack patterns.

This approach, based on statistical analysis and the distribution of roles within the system, makes the proposed design scalable and integrated with other environments that adopt the same SDN architecture, and it serves as a basis for developing more complex solutions in the future.

## Algorithm

The discovery algorithm relies on analyzing the statistical distribution of destination addresses in packets passing through the network, using a measure of Shannon entropy. This algorithm is implemented in real time through the Ryu controller, which analyzes packets within successive analysis windows. The following is a detailed explanation of the algorithm's steps:

Step 1: The traffic is divided into analysis windows ​ , with each window containing a specific number of packets 𝑚 (e.g., 20, 50, 70, 100). Each window contains a set of packets that arrived at the controller sequentially:

Step 2: The frequency of each destination address within the window is calculated, and then the probability of each address appearing is calculated using the relationship:

Where:

is the number of times the destination address appears within the window,

And is the number of packets in the window.

Step 3: The entropy of the window is calculated using the Shannon equation It represents the degree of "distribution" or "randomness" of the destinations within the window.

Step 4: After calculating the entropy of the first windows (for example, the first 10 windows), which are assumed to contain only natural traffic, the threshold value is calculated by averaging the values:

This threshold is later used as a reference for classifying each window.

Step 5: For each subsequent window, the calculated entropy is compared with the threshold θ. If the value is less than the threshold, the traffic within the window is classified as an attack, and if it is equal to or greater than the threshold, it is considered normal.

All resulting values ​​are recorded in a CSV file containing the window number, time, entropy value, and classification, facilitating subsequent analysis and comparison of results across different windows.

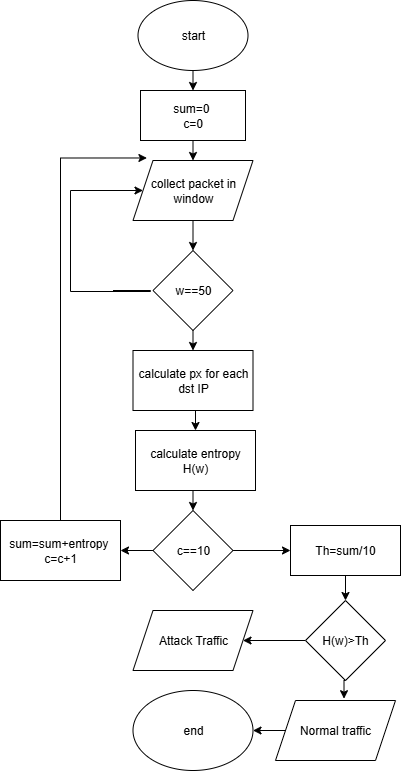


Figure 4‑‑ Flowchart of the Entropy-Based Detection Algorithm

This diagram illustrates the step-by-step process of packet collection, entropy calculation, threshold determination, and detection decision used in the system.

## Traffic Generation and Integration

The process of generating and distributing traffic within the virtual environment is a key element in testing the effectiveness of a DoS detection algorithm using the entropy principle. To achieve this, a realistic scenario was built that included both natural and attack traffic. The timing of each occurrence was precisely controlled within a unified and controlled experiment, allowing for measuring the system's response to sudden changes in packet pattern.

First: Natural Traffic

Natural traffic was generated using Python code based on the Scapy library, which allows for sending customized packets to multiple destinations within the network. These packets mimicked the behavior of real users, being randomly and uniformly distributed across multiple network endpoints to ensure a clear diversity of destination addresses, thus maintaining a high entropy value that reflects the natural distribution of packets.

Second: Attack Traffic (SYN Flood)

To evaluate the system's effectiveness in detecting attacks, a SYN Flood attack was implemented using the hping3 tool. This tool generates a large number of TCP SYN packets and directs them to a specific victim device within the network. As a result, the distribution pattern becomes concentrated toward a single IP address, resulting in a sharp drop in entropy within the time window, a clear indication of an attack.

Third: Combining the two traffic within the experiment

Both the normal and attack traffic were combined into a single, sequential experiment as follows:

From seconds 0 to 100: Generating normal traffic.

From seconds 100 to 150: Executing a SYN Flood attack.

From seconds 150 to 300: Returning to normal traffic.

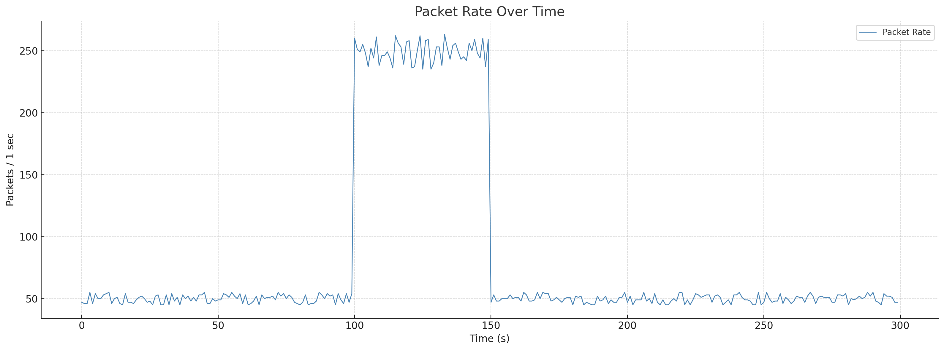


Figure 4‑‑ Packet Rate.

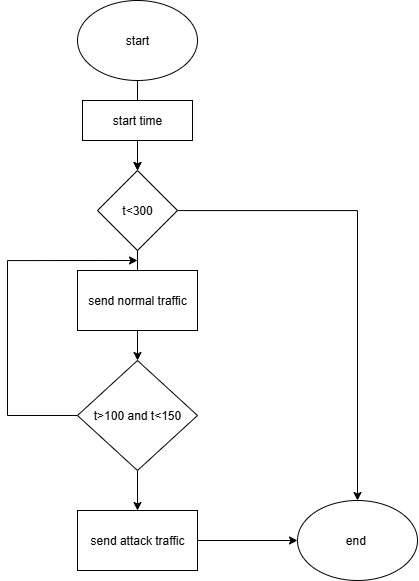


Figure 4‑‑ Flowchart of the Traffic Generation.

This temporal order in the distribution of traffic within the experiment is a crucial factor in testing the performance of the detection algorithm. It provides the system with a realistic opportunity to observe changes in entropy values ​​under various conditions, from a completely normal environment, through a period of intense attack activity, to the system's return to normal. During the attack period, entropy is expected to drop significantly, which is a clear signal that the system can detect. After the attack ends, the system is expected to demonstrate the ability to restore stability and accurately distinguish between normal and malicious traffic.

​

# Chapter Five. Results and Analysis

## Results and Analysis

This section reviews the effect of window size on the detection accuracy and response speed of an entropy-based detection system.

From Table 0-1, which displays the average entropy values ​​for the normal state and the attack state for four different window sizes, a clear pattern in the system's performance can be observed. As the window size increases, the entropy value for the normal state increases gradually, indicating a diverse and random distribution of the normal traffic. Conversely, the entropy value for the attack state was relatively low and nearly constant across the different windows, indicating the uniform nature of the traffic generated by the attack.

The most important indicator in this table is the difference between and H which reflects the ability to distinguish between normal and malicious traffic. The difference was highest in window 100 (1.1438), followed by windows 50 and 70 with values ​​(1.11681and 1.1275, respectively), while the lowest value was in window 20 (0.8292).

These results indicate that larger windows enhance the stability and accuracy of detection, as they allow for a greater amount of data to be collected within each window, making the differences in distribution more pronounced. However, it should be noted that increasing the window size may result in a slight delay in detection speed, which will be analyzed later in the final system performance evaluation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Window Size | Entropy (Normal Traffic) | Entropy (Attack Traffic) | |  | | --- | |  |   Difference |
| 20 | 2.0879 | 1.2587 | 0.8292 |
| 50 | 2.45301 | 1.3362 | 1.11681 |
| 70 | 2.4707 | 1.3432 | 1.1275 |
| 100 | 2.496 | 1.3522 | 1.1438 |

Table ‑Change TH for Each Window

### 5.1.1 Window 20

The threshold value calculated from the first 10 windows was 2.0879.

The window demonstrated excellent early detection of an attack, as the entropy value dropped rapidly at the onset of the attack. However, the increased sensitivity of this window resulted in a higher number of false positives when the attack ended and traffic returned to normal.

The change in entropy over time for the 20-window window is illustrated in Figure 5-1

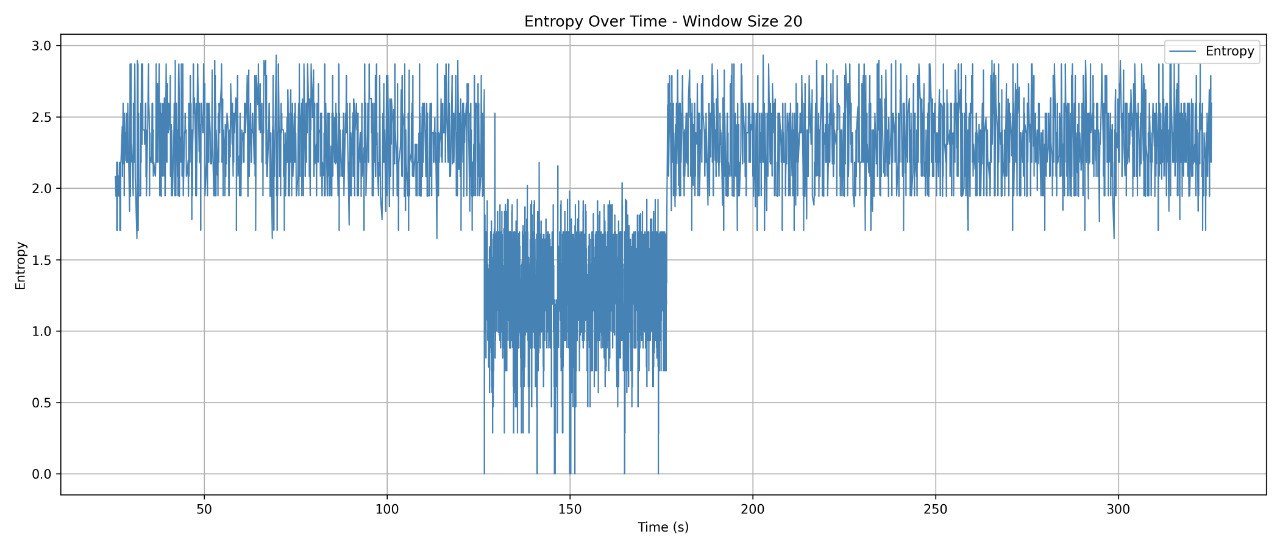


Figure 5‑ Entropy Variation Over Time – Window Size 20

### 5.1.2 Window 50

The calculated threshold was 2.45301.

The 50-day window offered a good balance between early detection and reducing false alarms. The entropy decrease was clear and regular throughout the attack period, reflecting the stability of this window in dealing with changes in traffic.

Figure 5-2 illustrates the behavior of entropy during the 50-window.

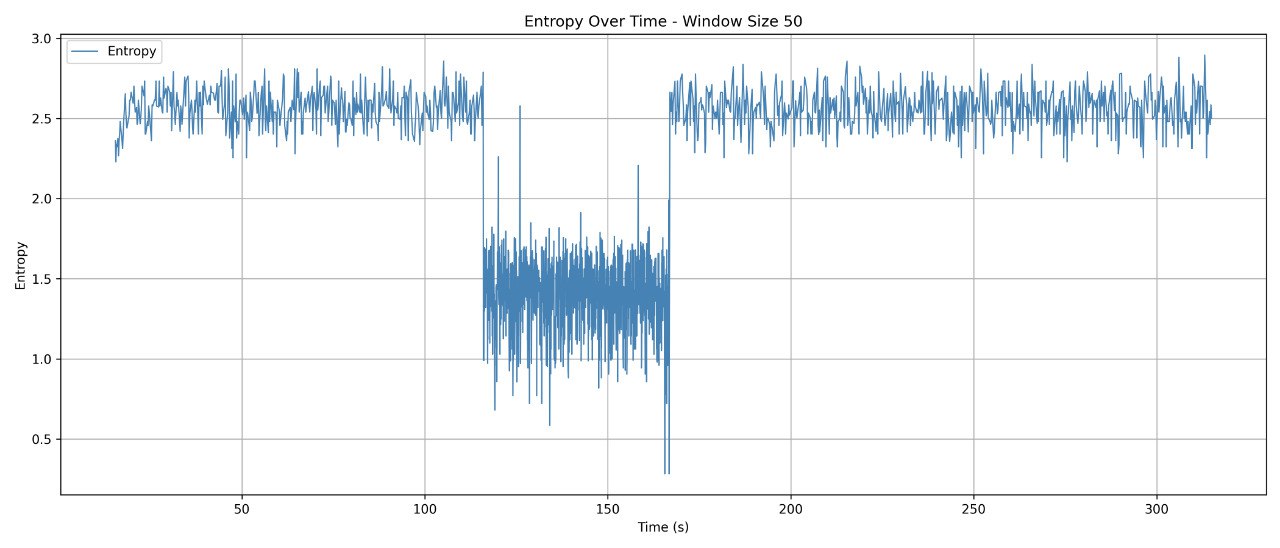


Figure 5‑ Entropy Variation Over Time – Window Size 50

### 5.1.3 Window 70

Calculated threshold: 2.4707.

A window of 70 showed high performance in terms of detection speed, with a clear stability in the entropy curve. Although a small number of false alarms occurred after the attack, the consistency of the decrease during the attack reflects the accuracy of the system at this window size.

The entropy change for this window is shown in Figure 5-0-3

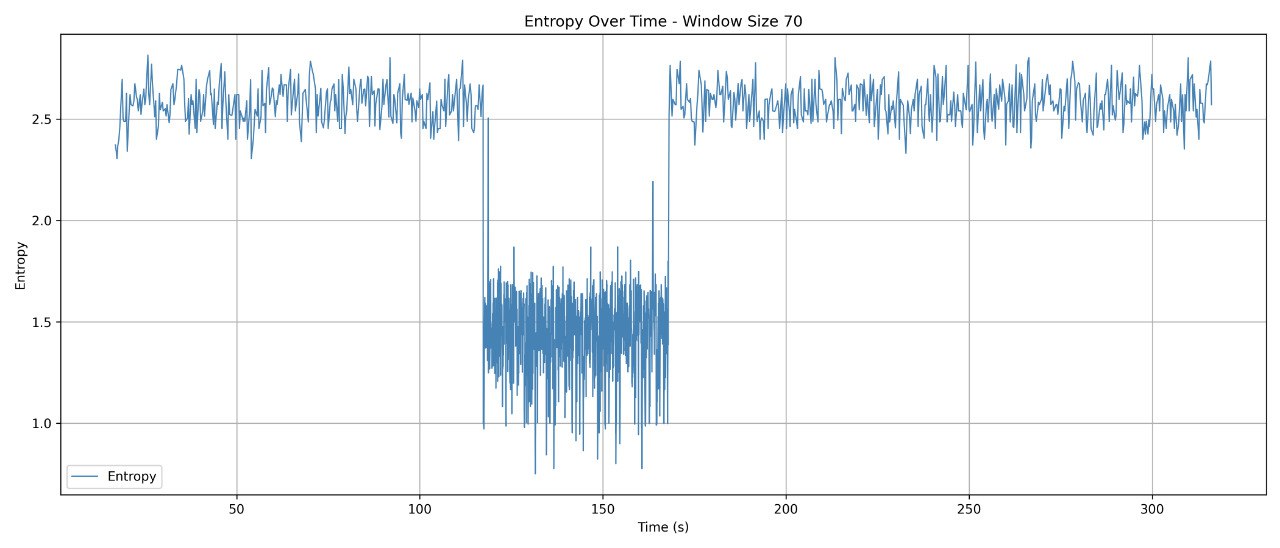


Figure 5‑ Entropy Variation Over Time – Window Size 70

### 5.1.4 Window 100

Calculated threshold: 2.496.

The 100-count window showed clear stability in values ​​but was slower in response compared to other windows. The entropy curve showed a slow decline at the onset of the attack, which could delay detection time. However, it provided consistent performance and minimal fluctuations in values.

Figure 5-4 shows the entropy change for the 100-count window.

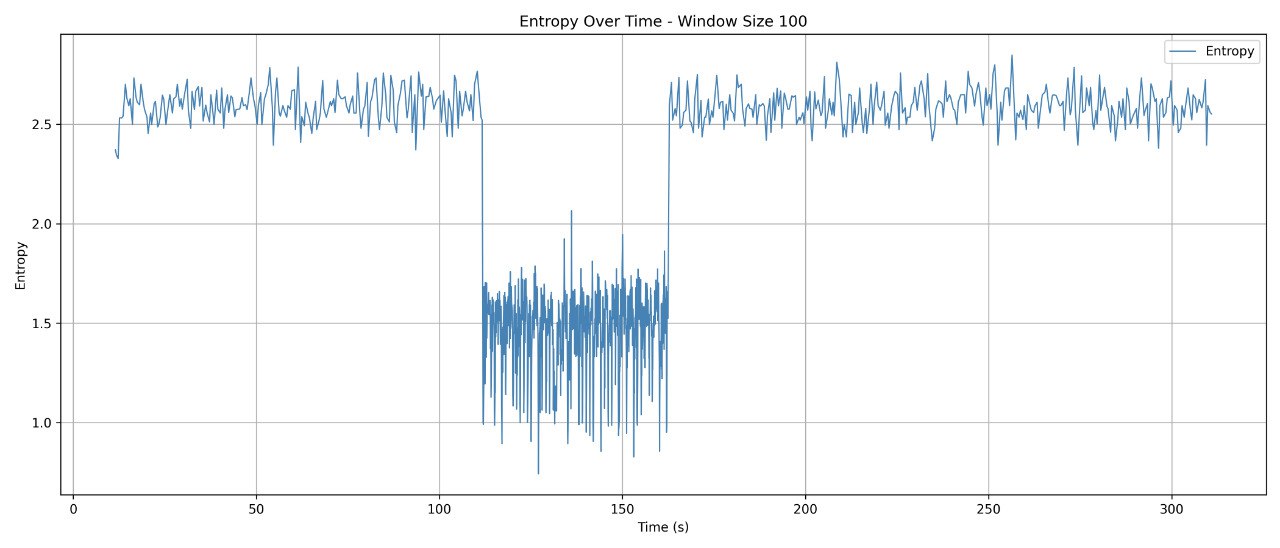


Figure 5‑ Entropy Variation Over Time – Window Size 100

## Performance Evaluation

The evaluation process in this section is based on the concept of the confusion matrix, a fundamental tool for assessing the performance of classification systems. This matrix illustrates how the predicted results align with the actual state of the data. The outcomes are divided into four categories: packets that were correctly identified as attacks, referred to as True Positives (TP); packets that were incorrectly classified as attacks but were actually normal, known as False Positives (FP); packets that were correctly identified as normal, called True Negatives (TN); and finally, packets that were attacks but were not detected, referred to as False Negatives (FN).

Using these four classifications, precise evaluation metrics can be derived, such as the system's overall accuracy, its ability to detect attacks, and its reliability in correctly classifying attacks without raising false alarms. These metrics are used to compare system performance under different settings and determine the optimal time window.

Accuracy: The percentage of correct classifications out of the total number of cases.

Precision: The percentage of packets classified as attacks that were actually attacks.

Recall: The percentage of attack packets detected out of all attack packets.

F1-score: The harmonic means between precision and recall.

The system was evaluated using four different time windows (20, 50, 70, and 100). In each experiment, the threshold was calculated from the first 10 windows and the performance was evaluated during the attack period.

In this section, the performance of the proposed DoS detection system using the entropy algorithm was analyzed through multiple experiments using four different time windows: 20, 50, 70, and 100. In each experiment, the threshold was dynamically calculated based on the average of the first ten entropy windows. The system was then evaluated using a confusion matrix and accuracy, precision, recall, and F1-score metrics.

|  |  |  |
| --- | --- | --- |
|  | Actual attack | Actual normal |
| Predicted attack | TP | FP |
| Predicted normal | FN | TN |

Table 5‑‑ Confusion Matrix

**Window Size: 20**

In this experiment, a window size of 20 was used. The detection threshold was computed as the average entropy of the first 10 windows, which resulted in a threshold of **2.0879**. During the attack period, the average entropy dropped to **1.2587**, indicating a clear deviation from the normal traffic pattern.

The confusion matrix for this window is as follows:

This result indicates that the system successfully detected nearly all attack packets (True Positives = 3883), with only 3 attack packets missed (False Negatives). However, it also produced a large number of false alarms (False Positives = 579), which affected the overall precision.

|  |  |  |
| --- | --- | --- |
|  | Actual attack | Actual normal |
| Predicted attack | 99.9% | **23.3%** |
| Predicted normal | 0.1% | 76.7% |

Table 5‑‑ Confusion Matrix – Window Size 20

Accuracy: 90.87%

Precision: 87.02%

Recall: 99.92%

F1-score: 93.03%

Although the system demonstrated high sensitivity to attacks (as shown by the high recall), the large number of false positives reduced its precision. This reflects a tendency to overclassify traffic as malicious, which may be problematic in environments where minimizing false alerts is critical.

**Window Size: 50**

In this experiment, the performance of a Denial of Service (DoS) detection system based on the entropy algorithm was evaluated using a time window of 50. The threshold was calculated from the average entropy of the first 10 windows, which was 2.4530. During the attack period, the average entropy decreased to 1.3362, indicating a clear anomaly compared to the normal case.

The following table shows the confusion matrix for this experiment. The system successfully identified the majority of the attack packets (TP = 1579), with only two missed cases (FN = 2). The number of false alarms (FP = 181) was also reduced compared to a window of 20, demonstrating an improved balance between sensitivity and specificity.

|  |  |  |
| --- | --- | --- |
|  | Actual attack | Actual normal |
| Predicted attack | 99.87% | 18.1% |
| Predicted normal | 0.13% | 81.9% |

Table ‑ Confusion Matrix – Window Size 50

Accuracy: 92.91%

Specificity: 89.72%

Recall: 99.87%

F1-score: 94.52%

The performance metrics reflect a clear improvement over the smaller window, with an F1 index of 0.9452, demonstrating an effective balance between attack detection and reducing false alarms. These results indicate that using a 50 size window provides higher accuracy without significantly compromising sensitivity, making it suitable for systems that require reliable detection and reduced inconvenience from false alarms.

**Window Size: 70**

In this experiment, a time window of size 70 was used, with the threshold calculated from the average of the first 10 windows, which was 2.4707. The average entropy decreased during the attack to 1.3432, indicating a clear change in the traffic pattern.

The following confusion matrix indicates that the system maintained its high attack detection capability, identifying 1,109 out of 1,111 attack packets, while significantly reducing the number of false positives (only 110), contributing to improved qualitative accuracy.

|  |  |  |
| --- | --- | --- |
|  | Actual attack | Actual normal |
| Predicted attack | 99.82% | 15.56% |
| Predicted normal | 0.18% | 84.44% |

Table 5‑‑ Confusion Matrix – Window Size 70

Accuracy: 0.9384

Precision: 0.9098

Recall: 0.9982

F1 Score: 0.9519

These results demonstrate an excellent balance between precision and recall, with an F1 score of 0.9519, demonstrating the system's high classification effectiveness while minimizing false alarms. This window also achieved stable performance with acceptable resource consumption, making it an ideal candidate for real-world environments.

Window size: 100

In this experiment, a window size of 100 was used, with the threshold determined based on the average entropy of the first 10 windows, which was 2.496. During the attack period, between 111.85 and 162.43 seconds, the average entropy dropped to 1.3522, clearly indicating a change in the traffic pattern as a result of the attack.

The following confusion matrix shows that the system successfully detected all attack packets with a recall rate of 100%, without recording any failures (FN = 0). The system also correctly classified a large percentage of natural packets, with only 72 natural packets misclassified as attacks (FP = 72) out of 495 natural packets.

|  |  |  |
| --- | --- | --- |
|  | Actual attack | Actual normal |
| Predicted attack | 100.0% | 14.55% |
| Predicted normal | 0.0% | 85.45% |

Table 5‑‑ Confusion Matrix – Window Size 100

Accuracy:94.35%

Precision:0.9154

Recall: 1.0000

F1 Score: 0.9558

These results indicate that the system performed exceptionally well when using a window size of 100, maintaining very high sensitivity (Recall = 1.0) while reducing false alarms to reasonable levels. The F1 score in this experiment was 0.9558, reflecting a good balance between detection capability and classification accuracy.

Thus, using this large window proves effective in accurately detecting DoS attacks while maintaining acceptable reliability in terms of the number of false alarms, especially in environments that tolerate larger traffic volumes and require deeper pattern analysis.

## The effect of window size on CPU consumption:

Performance efficiency is an important aspect when designing attack detection systems, especially in environments that require real-time traffic monitoring without significantly impacting system resources. In this study, CPU consumption was measured during system operation using four different windows (20, 50, 70, and 100) to evaluate the impact of window size on the computational load resulting from analysis and calculation operations.

As shown in Table 0-6, there is a direct relationship between window size and CPU consumption. The larger the window size, the more time the system needs to calculate entropy and aggregate data, leading to a significant increase in resource consumption.

For example, when using a window size of 20, CPU1 and CPU2 averaged approximately 31.1% and 29%, respectively. When using a window size of 100, CPU consumption increased to 39.1% and 39.5%, representing a significant difference in system load. Despite this increase, performance remained within acceptable limits, confirming the feasibility of using these windows in real-world environments, provided sufficient computing power is available.

These results highlight the importance of balancing detection effectiveness with implementation efficiency. Choosing an appropriately sized window not only ensures accurate attack detection but also maintains the stability and overall performance of the system.

|  |  |  |
| --- | --- | --- |
| Window size | Cpu1 | Cpu2 |
| 20 | 31.1% | 29.0% |
| 50 | 34.6% | 33.9% |
| 70 | 36.9% | 35.5% |
| 100 | 39.1% | 39.5% |

Table 5‑ CPU Usage by Window Size

## Final Analysis:

After an in-depth analysis of the performance results from the four experiments using different window sizes (20, 50, 70, and 100), and by examining confusion matrices and quantitative evaluation indicators such as precision, recall, F1 value, and accuracy, along with CPU consumption, it is clear that window size plays a crucial role in determining the effectiveness and efficiency of the system.

While small windows such as 20 demonstrated high detection ability, they suffered from a large number of false alarms, reducing the system's reliability in real-world situations. Conversely, large windows such as 100, despite achieving full recall, were accompanied by a high rate of false alarms and an excessive burden on system resources, which can lead to reduced overall performance, especially in high-frequency environments.

The results highlight that a window size of 70 represents the optimal balance between performance, precision, and consumption. It achieved a good F1 value compared to other windows, demonstrating an excellent balance between qualitative accuracy and detection capability, while maintaining moderate resource consumption. This is reinforced by the fact that the system was able to significantly reduce the number of false positives without compromising detection effectiveness, as reflected in the high percentage of correct classification rates for both attack and natural packets.

Therefore, it can be confidently stated that a window size of 70 represents the optimal choice for use in entropy-based DoS detection systems. Not only does it achieve high classification performance, but it does so with a measured efficiency that allows it to be integrated into real-world applications. These results support the recommendation to adopt this window size as the default standard in the design of detection mechanisms that rely on statistical analysis of traffic behavior.

# Chapter Six. Conclusion

This research project represents a research attempt to design and implement an effective system for detecting Denial of Service (DoS) attacks in a Software-Defined Networking (SDN) environment, relying on entropy analysis as a means of detecting abnormal changes in traffic behavior. A test environment was built using the Mininet platform, and a controller was developed using a Ryu controller to calculate entropy values ​​based on transmitted packets and analyze these values ​​to detect any anomalies that might indicate an attack.

The research methodology included conducting a series of experiments using four different window sizes (20, 50, 70, and 100). Performance was analyzed by calculating confusion matrices and associated statistical indicators, such as precision, recall, and the F1 function, along with monitoring computing resource (CPU) consumption. The results revealed that using entropy as a detection mechanism provides an accurate and rapid response to attacks, with all experiments demonstrating a high ability to detect attack packets, with varying false alarm rates and CPU consumption.

Of all the windows tested, a window size of 70 was found to be the optimal choice, offering the best balance between detection accuracy, false positives, and resource efficiency, making it best suited for real-world applications that require high performance without sacrificing efficiency.

**Recommendations:**

The entropy algorithm is recommended for SDN environments as a lightweight and effective DoS detection tool.

The window size should be adjusted based on the nature of the network, with 70 being the ideal starting point.

The system could be developed in the future to support detection of other types of attacks (such as DDoS, port scanning) by incorporating multiple features and machine learning.

**Future work:**

Extend the framework to support real-world environments or large-scale networks.

Use machine learning or deep learning models to improve predictive power.

Develop an automated response system that activates when an attack is detected to protect network resources in real time.

# References

1. Implementation of Entropy-Based DDoS Attack Detection Method in Different SDN Topologies Mayadah A. Mohsina\*, Ali H. Hamadb
2. A Collaborative Approach to Detecting DDoS Attacks in SDN Using Entropy and Deep Learning Narayan D.G., Heena W, and Amit K *KLE Technological University, Hubballi, Karnataka, India*
3. Entropy-based DoS Attack identification in SDN Ranyelson N. Carvalho, Jacir L. Bordim and Eduardo A. P. Alchieri Department of Computer Science University of Brasília (UnB) Brasília, Brazil
4. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). **OpenFlow: Enabling innovation in campus networks**. ACM SIGCOMM Computer Communication Review, 38(2), 69–74.  
   https://doi.org/10.1145/1355734.1355746
5. Lantz, B., Heller, B., & McKeown, N. (2010). **A network in a laptop: Rapid prototyping for software-defined networks**. Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IX).  
   https://doi.org/10.1145/1868447.1868466
6. Ryu SDN Framework. (n.d.). **Ryu Documentation**. Retrieved from:  
   <https://ryu.readthedocs.io>
7. Mininet Team. (n.d.). **Mininet: An Instant Virtual Network on Your Laptop (or Other PC)**. Retrieved from:  
   <http://mininet.org>
8. Mirkovic, J., & Reiher, P. (2004). **A taxonomy of DDoS attack and DDoS defense mechanisms**. ACM SIGCOMM Computer Communication Review, 34(2), 39–53.  
   https://doi.org/10.1145/997150.997156
9. Zhang, Y., & Lee, W. (2000). **Intrusion detection in wireless ad-hoc networks**. Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, 275–283.
10. Chen, Y., Hwang, K., & Ku, S. (2007). **Collaborative detection of DDoS attacks over multiple network domains**. IEEE Transactions on Parallel and Distributed Systems, 18(12), 1649–1662.
11. Scapy Documentation – **Scapy: Packet Manipulation Tool**.  
    <https://scapy.readthedocs.io>
12. Open Networking Foundation. (2012). **Software-Defined Networking: The New Norm for Networks**. ONF White Paper.

# Appendices

**Appendix A – GitHub Repository**

The full source code, scripts, and result files for this project are available on the following GitHub repository:  
🔗 https://github.com/zaidfaisl/entropy-dos-detection.git

The repository includes the entropy detection controller, traffic generation scripts, and experiment automation tools.

**Appendix B – Key Code Snippets**

**1. Entropy Calculation Function**

def calculate\_entropy(window):

freq = Counter(window)

total = len(window)

return -sum((count / total) \* math.log2(count / total) for count in freq.values())

This function implements Shannon entropy to evaluate the randomness of destination IP addresses in each traffic window.

**2. Dynamic Threshold Calculation**

if len(entropy\_history) == max\_windows and threshold is None:

threshold = sum(entropy\_history) / max\_windows

The system dynamically establishes a detection threshold using the average entropy of the first N windows, assumed to represent normal traffic.

**3. Traffic Generation Snippet (Scapy)**

from scapy.all import \*

import random

for i in range(100):

dst = random.choice(["10.0.0.1", "10.0.0.2", "10.0.0.3"])

pkt = IP(dst=dst)/TCP(dport=80)

send(pkt)

This code sends randomized TCP packets to simulate natural traffic flow in the testbed.

**4. Attack Launch Snippet (hping3)**

sudo hping3 -S -p 80 -i u4000 -c 10000 10.0.0.9

Launches a SYN Flood attack to generate high-volume targeted traffic.

**Appendix c– CPU Usage per Window**

Table below show CPU consumption during detection for each window size. The increase in processing load correlates with larger window sizes due to increased packet volume per entropy cycle.

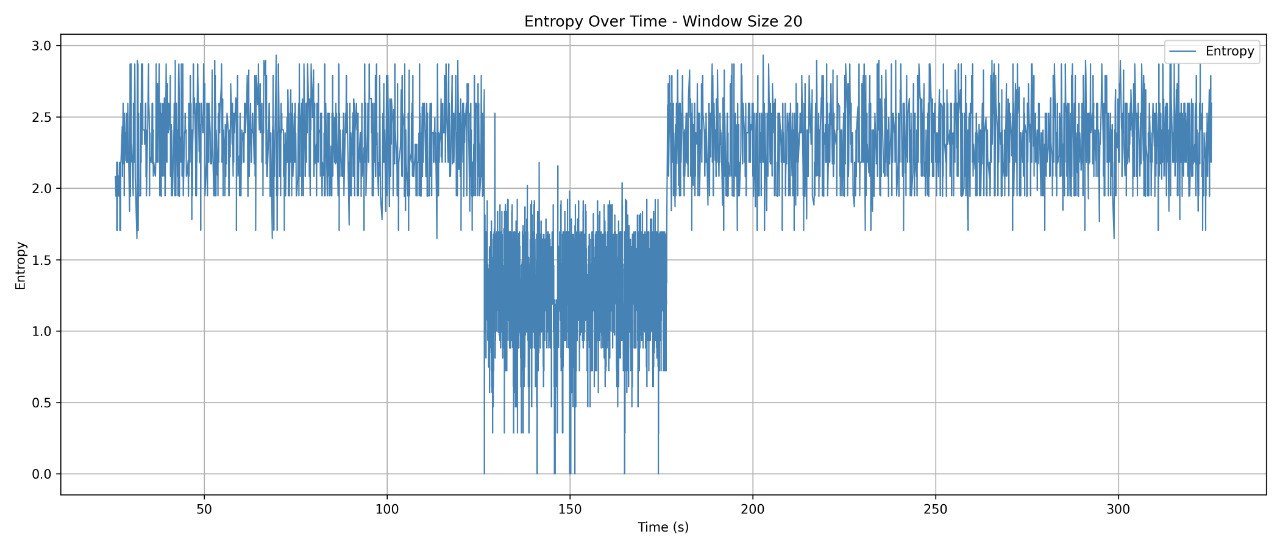
|  |  |  |
| --- | --- | --- |
| Window size | Cpu1 | Cpu2 |
| 20 | 31.1% | 29.0% |
| 50 | 34.6% | 33.9% |
| 70 | 36.9% | 35.5% |
| 100 | 39.1% | 39.5% |

A clear trend can be observed: as window size increases, entropy calculation becomes more computationally expensive.

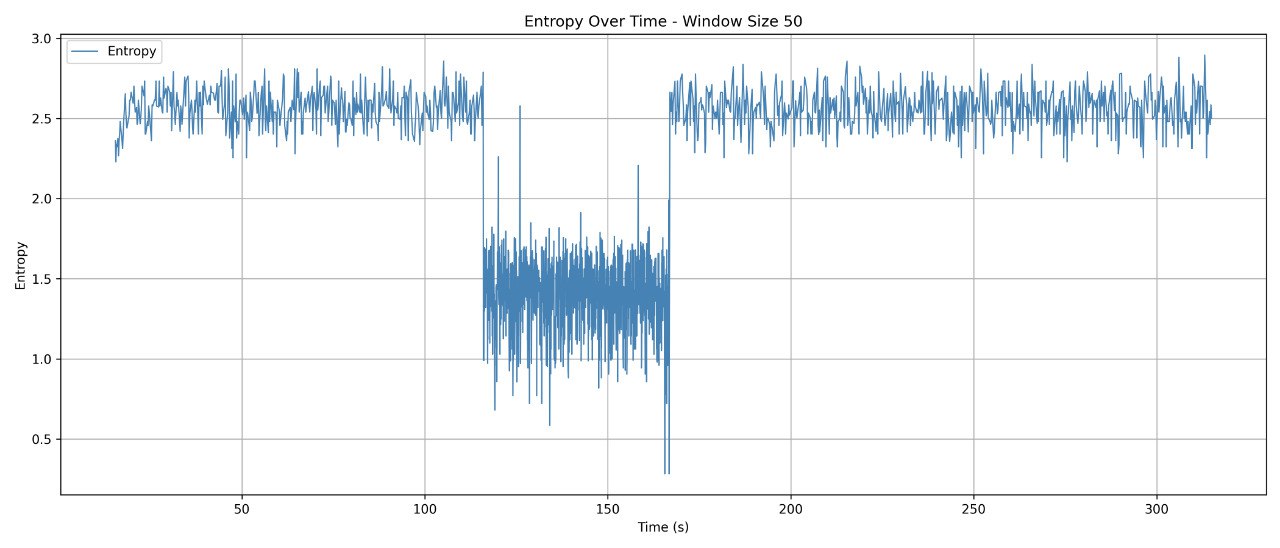
**Appendix D– Entropy Curves per Window (Previously in Results Chapter)**

To reinforce the system’s behavior and detection timing, the following graphs previously included in the results section are referenced again here:

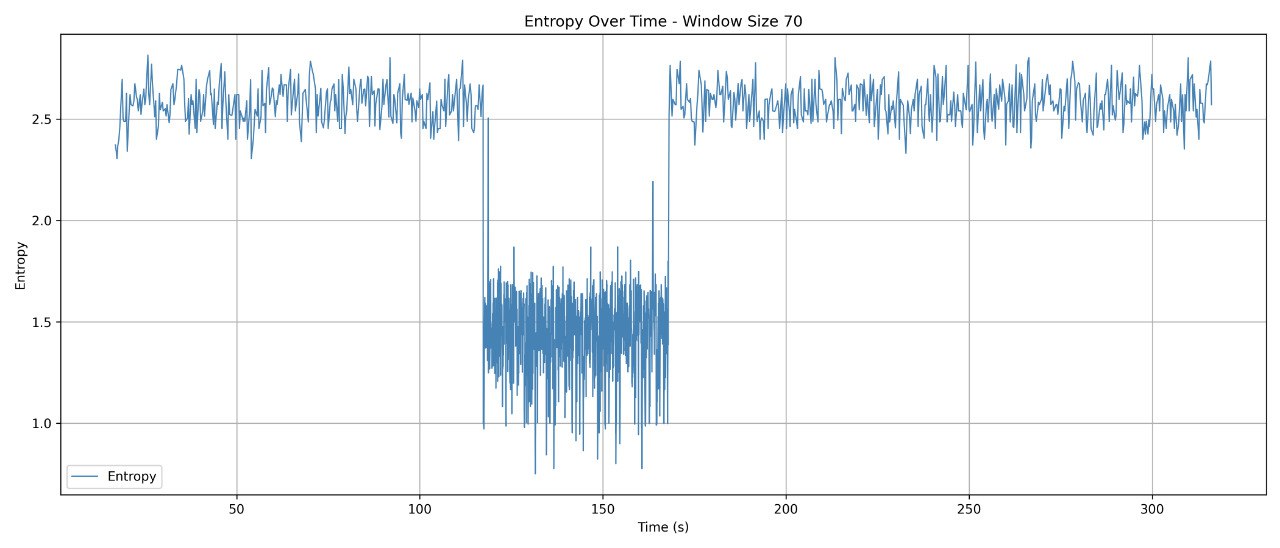
* Entropy over time – Window 20



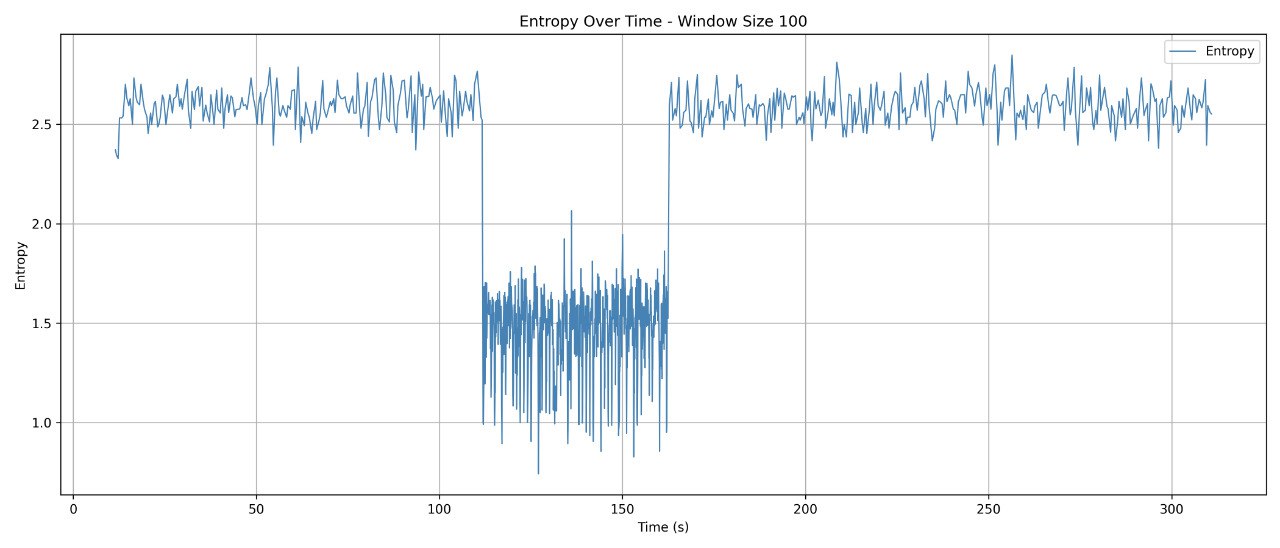
* Entropy over time – Window 50



* Entropy over time – Window 70



* Entropy over time – Window 100



These graphs visualize the sharp entropy drop during the attack window and the system’s ability to detect anomalies.

**Appendix E: Summary of Confusion Matrix Results**

| **Window Size** | **TP** | **FP** | **TN** | **FN** | **Precision** | **Recall** | **F1-score** | **Accuracy** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **20** | **3883** | **579** | **1909** | **3** | **0.8702** | **0.9992** | **0.9303** | **0.9087** |
| **50** | **1579** | **181** | **819** | **2** | **0.8972** | **0.9987** | **0.9452** | **0.9291** |
| **70** | **1109** | **110** | **597** | **2** | **0.9098** | **0.9982** | **0.9519** | **0.9384** |
| **100** | **779** | **72** | **423** | **0** | **0.9154** | **1.0000** | **0.9558** | **0.9435** |

The following table summarizes the confusion matrix results obtained for each window size during the evaluation of the entropy-based DoS detection system. It includes the main classification outcomes (TP, FP, TN, FN) and the corresponding performance metrics (Precision, Recall, F1-score, and Accuracy) for comparative analysis.