Statistical Approaches to DDoS Attack Detection and Mitigation to TCP SYN Flood Attack

K S Ananth
CS22BTECH11029

Pradyumn Kangule
CS22BTECH11048

Nimai Parsa
CS22BTECH11044

Swapnil Bag
ES22BTECH11034

M Bhavesh Chowdary
CS22BTECH11041

Devansh Agarwal
ES22BTECH11010

Syed Abrar
CS22BTECH11058

1 Introduction

DDos (Distributed Denial of Service) attack is a malicious attempt to disrupt the normal functioning of a network, service, or server by flooding the target with an overwhelming amount of messages. In a distributed attack, multiple compromised systems or botnets orchestrate an attack. Through this project we explore a detection strategy by monitoring the parameters of the incoming packets and employ statistical techniques to determine whether a server is under attack.

2 Project setup

It consists of a single server that implements a firewall to detect DDoS attacks. The clients and attackers send requests from their devices using a script. The server and the client are a part of the same subnet (must be connected to a common access point or hotspot for demonstration purposes). The server is run on the IP allocated dynamically on that subnet, and clients send messages to that IP.

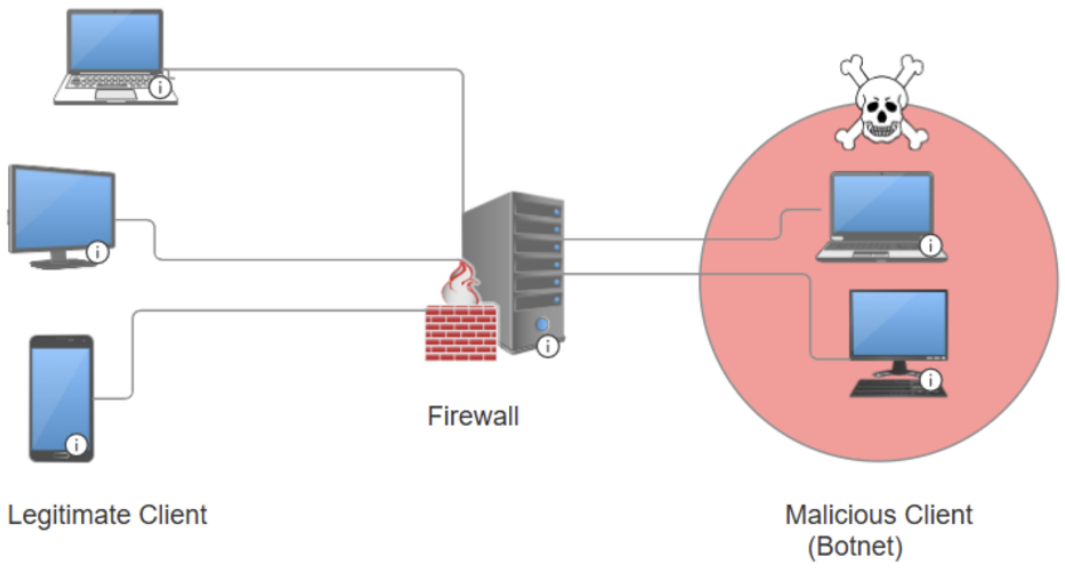


Figure 1: Block diagram for Network

3 Instructions to setup for demonstration

- Run the server.py file first by typing "python3 server.py" in the terminal.
- After the server is set up and running, Run the firewall.py file in another terminal by typing "sudo python3 firewall.py".
- Now both the server and the firewall are setup and running.
- We have two different client files :
  - client_normal.py
  - client_attacker.py
- The client_normal.py generates normal traffic to the server this is achieved by using sleep for a random time after sending each packet.
- The client_attacker.py will make the attack, that is it generates continuous and heavy traffic(continuous packets are bursted to the server with out any sleep between two packets).
- For generating multiple clients from same device with various ips, we are creating virtual ip addresses on the network interface.

4 Entropy-Based Detection:

4.1 Theory

- Entropy is the measure of randomness in information.

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

  In our implementation of a DDoS attack detector, we consider TCP SYN flood attack which exploits the TCP handshake process to the server.
- It tracks the distinct IPs that sent a request with syn bit as 1 over a certain duration and computes the entropy for that window.

$$P(i) = \frac{\text{Count of SYN from IP}_i}{\text{Total SYN counts}}$$

- Under normal traffic conditions, the entropy values are high, because on an average each legitimate client sends connection requests with syn bit 1 are uniformly distributed over a certain range.
- Under attack, malicious clients send an excessive number of connection requests, hence, the window is dominated by only certain ips with syn bits as 1. This causes the entropy to drop below the threshold.
- The firewall then blocks these malicious IPs for a certain duration and then unblocks them
- The threshold is varied dynamically by a moving average formula

$$\text{threshold}_{t+1} = \text{average\_entropy} + (\text{threshold}_t - \text{average\_entropy}) \cdot 0.2$$

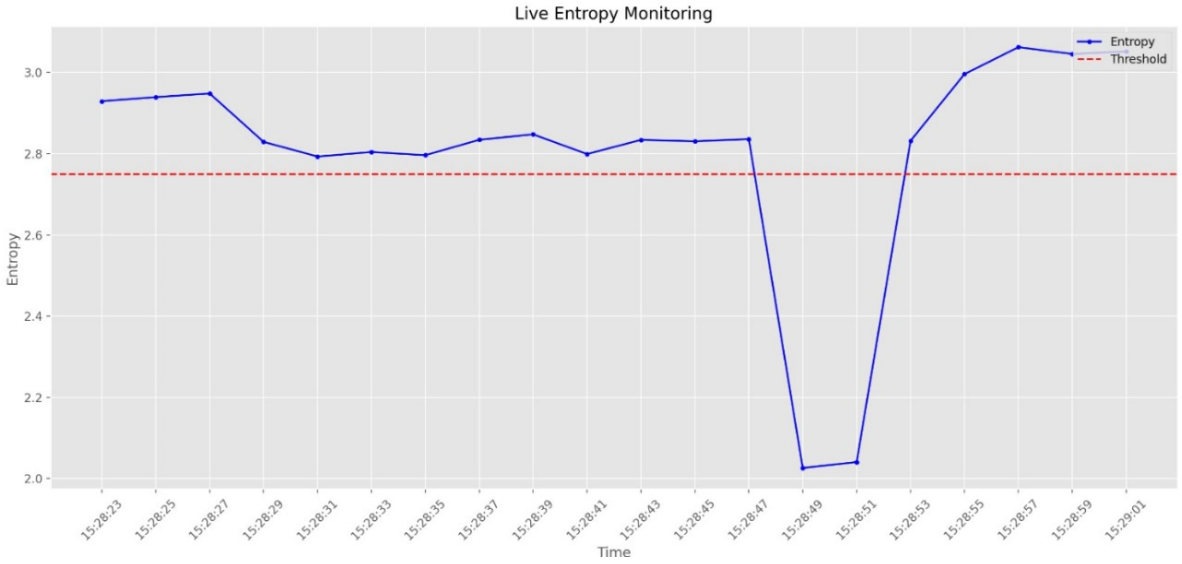


Figure 2: Entropy network statistics plot when the network is under attack

4.2 Pseudocode

Algorithm 1 Monitor Entropy and Detect DDoS

Function MonitorEntropy()
while True **do**
  Sleep for `MONITOR_INTERVAL`.
  `current_time` ← Current time.
  Prune old entries in `ip_syn_tracker`:
    Keep `timestamps` within `TIME_WINDOW`.
    Remove `ip` if no timestamps remain.
  Log `ip_syn_tracker`.
  Compute `counter` ← IP request frequencies.
  `entropy` ← CalculateEntropy(`counter`).
  Enqueue `(current_time, entropy)` for plotting.
  `threshold` ← UpdateThreshold(`entropy`).
  If `entropy < threshold` **and** active IPs > 0:
    Print alert and check IPs:
      If `count > THRESHOLD_SYN_COUNT` **and IP not blocked**:
        Block `ip`.
end while

5 Chi-Square Detection:

5.1 Theory

- The chi-square statistic is calculated using:

$$\chi^2 = \sum \frac{(O_i - E)^2}{E}$$

  where O_i is Observed frequency for each IP (number of SYN packets sent by that IP) and E_i is Expected frequency (average SYN packets per IP).
- Under normal conditions the observed frequencies of SYN packets per IP are evenly distributed and close to the expected frequencies. The chi-square value remains low, indicating no significant deviation from the expected uniform distribution.

- Under attack the observed frequencies become skewed as one or a few IPs send disproportionately high numbers of SYN packets compared to others. This causes the Chi square value to spike

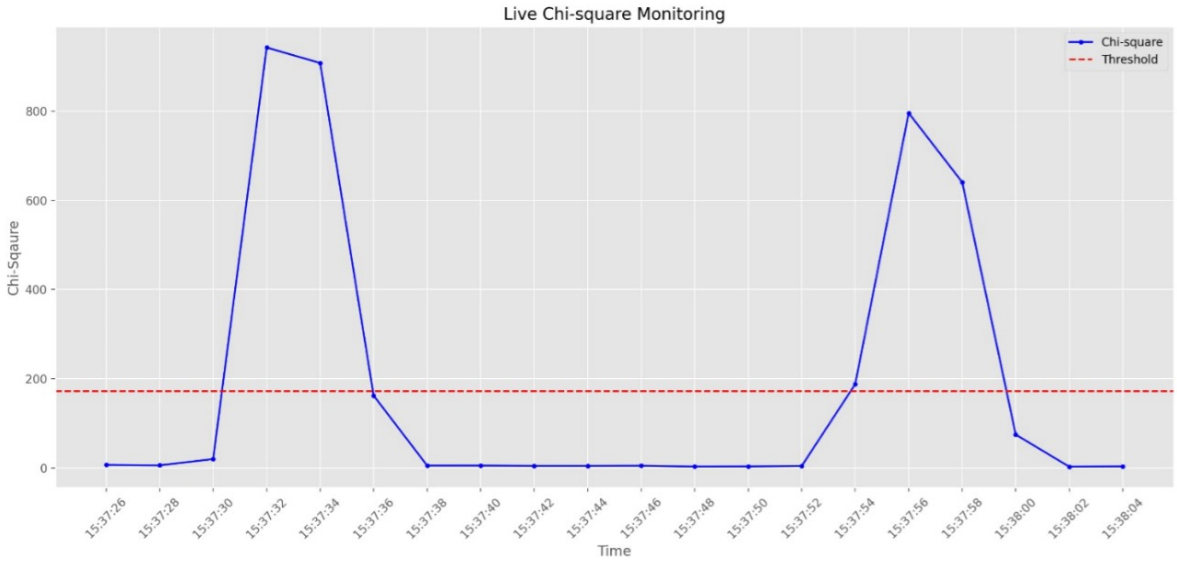


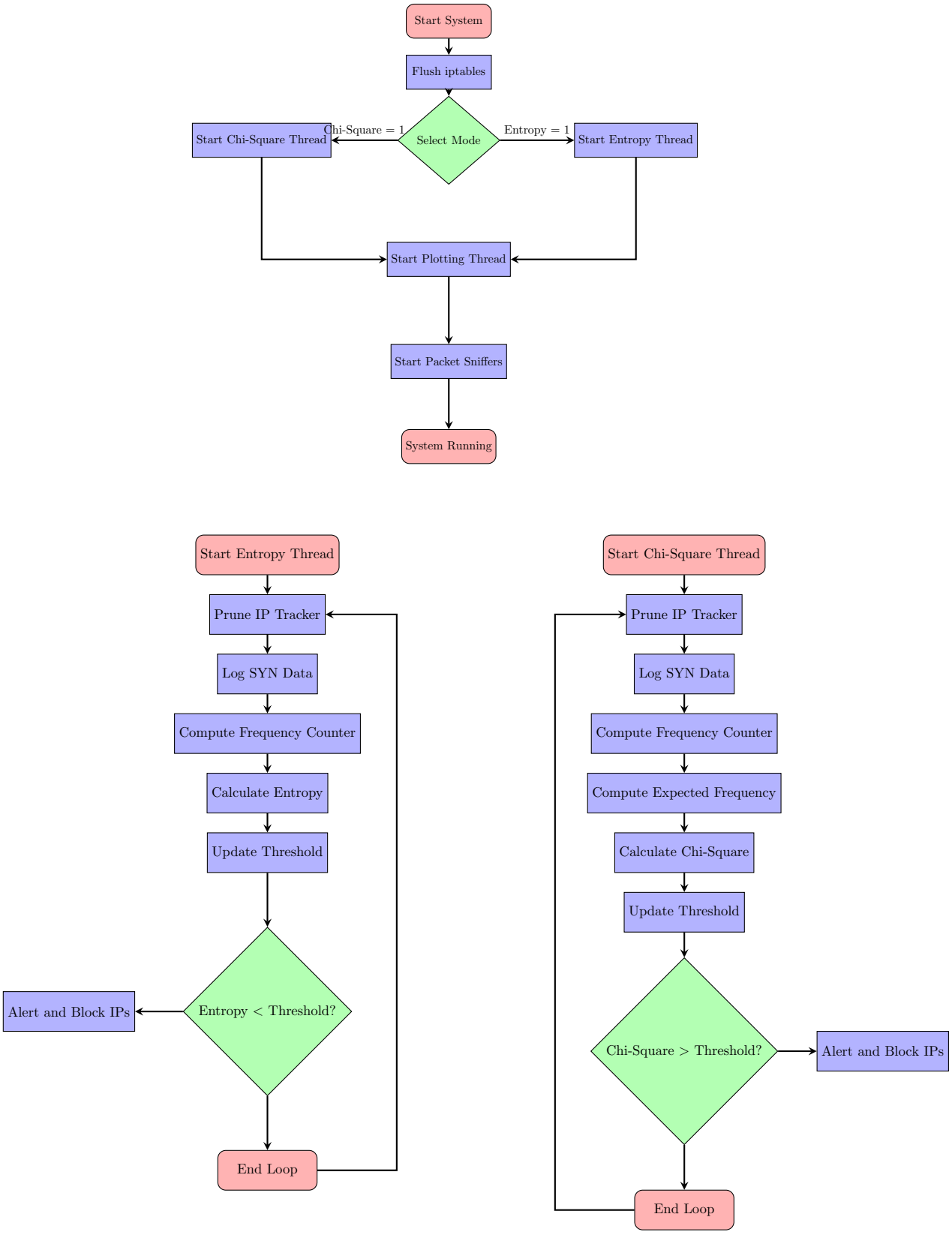
Figure 3: Chi Squared network statistics plot when the network is under attack. The malicious IP gets blocked after a timeout and allowed to reconnect

5.2 Pseudocode

Algorithm 2 Monitor Chi-Square and Detect DDoS

Function MonitorChiSquare()
while True **do**
  Sleep for `MONITOR_INTERVAL`.
  `current_time` ← Current system time.
  Acquire `lock`.
  Prune old entries in `ip_syn_tracker`:
    Keep `timestamps` within `TIME_WINDOW`.
    Remove `ip` if no timestamps remain.
  Log `ip_syn_tracker`.
  Compute `counter` ← Frequency of SYN packets per IP.
  If no active IPs, **continue**.
  Compute expected frequency:
    `total_syn_packets` ← Sum of `counter.values()`.
    `expected_frequency` ← `total_syn_packets` / Number of active IPs.
  Compute Chi-square:
    `chi_square` ←
      $\sum$((`observed_freq - expected_frequency`)^2/`expected_frequency`).
  Enqueue `(current_time, chi_square)` for plotting.
  `threshold` ← UpdateThreshold(`chi_square`).
  Print `chi_square` and updated `threshold`.
  If `chi_square > threshold`:
    Print alert and check IPs:
      If `count > THRESHOLD_SYN_COUNT` **and IP not blocked**:
        Block `ip`.
end while

6 System Block Diagram



System Block Diagram for DDoS Detection and Mitigation

7 Implementation Details

Below, we describe the implementation details of our system.

7.1 System Architecture

The system is designed as a multithreaded application consisting of the following components:

- **Packet Sniffers:** Packet sniffers monitor incoming network traffic on specified interfaces (e.g., `wlo1` and `lo`) and extract SYN packets. These packets are logged in a shared data structure for analysis.
- **Monitoring Thread:** Depending on the chosen mode, either an entropy-based or chi-squared-based monitoring thread analyzes the frequency distribution of SYN packets to detect anomalies.
- **Live Plotting Thread:** This thread visualizes statistical metrics (entropy or chi-square values) in real-time, helping to observe trends in network traffic.
- **Mitigation:** When an anomaly is detected, the system identifies malicious IPs and blocks them using `iptables`.

7.2 Entropy-Based Detection

The entropy-based detection approach evaluates the randomness of the SYN packet distribution across IP addresses:

- **Data Collection:** SYN packet timestamps are tracked in a dictionary, where keys are IP addresses, and values are lists of timestamps.
- **Pruning:** Timestamps outside the predefined time window (`TIME_WINDOW`) are periodically removed.
- **Entropy Calculation:** Using a frequency counter for IP addresses, Shannon entropy is computed to assess the distribution's randomness. A sudden drop in entropy may indicate a DDoS attack.
- **Threshold Update:** A dynamic threshold adjusts over time based on observed entropy values.
- **Mitigation:** If the entropy falls below the threshold and exceeds a minimum, the system raises an alert and blocks suspicious IPs exceeding a packet count threshold (`THRESHOLD_SYN_COUNT`).

7.3 Chi-Squared Test for Detection

The chi-squared approach detects anomalies based on statistical deviation from the expected distribution of SYN packets:

- **Expected Frequency:** The system calculates the expected frequency of SYN packets as the average number of packets per active IP.
- **Chi-Square Calculation:** For each IP, the chi-squared value is computed to compare observed frequencies against the expected frequency. A higher chi-squared value indicates a deviation.
- **Threshold Update:** A dynamic threshold updates with observed chi-squared values.
- **Mitigation:** If the chi-squared value exceeds the threshold, the system flags potential DDoS behavior and blocks attacker IPs.

7.4 System Flow

The system begins by flushing existing `iptables` rules to ensure a clean slate. Based on user selection, the monitoring thread starts using either entropy or chi-squared analysis. Packet sniffers run on separate threads to capture real-time traffic. If an attack is detected, the identified IP addresses are added to the block list, and the corresponding rules are applied to `iptables` to mitigate the attack.

7.5 Additional Features

- **Dynamic Threshold:** We updated the thresholds dynamically, to account for normal fluctuations in network traffic.
- **Multithreading:** We use threading to ensure real-time processing of incoming packets, statistical calculations, and visualization.
- **Shared Data Structures:** A thread-safe data structure (`ip_syn_tracker`) maintains traffic statistics across threads.
- **Plotting:** Real-time plotting provides insights into network traffic, helping in visualization.
- **Blocking:** The `block_ip()` function dynamically adds IP addresses to the firewall for immediate mitigation.

7.6 Code High Level Explanation

- The code works with three threads in total. One is for entropy monitoring, one is for live entropy plots, and one is for sniffing the incoming TCP packets on the wlo1 interface.
- Entropy is calculated every 2 seconds using only the packets that were received in the last 10-second window. A new threshold is calculated based on the average of the previous ten values and the current threshold.
- Now, if the entropy is less than the threshold, the attacker IP is decided using the number of packets sent by that IP in the 10-second window. If some IP is found to have more than a threshold number of packets, the IP will be blocked. This IP will be unblocked again after 5 minutes.
- Similarly, for chi-square, it is calculated every 2 seconds, and a new threshold is calculated using the average last ten values and the current threshold. Now, if the server is under attack, the chi-square value will have a spike, and it will be far above the threshold.
- The code tracks the number of syn bits received from an IP in an interval of 10 seconds by maintaining a dictionary mapping from IP to a list of timestamps.

8 Project Considerations

- There are Machine learning-based DDoS attack detectors. However, we chose a statistical approach. ML models are like black boxes and cannot be very interpretable at times. Statistical models have a lower inference time and are computationally less expensive for low-latency systems and high-traffic scenarios.
- We chose to implement our project in a live network setting instead of using Mininet, as this approach provides a more immersive and hands-on experience mimicking real-life scenarios.
- There are various types of DDoS attacks, including UDP floods, ICMP floods, and application-layer attacks. However, this project specifically focuses on detecting and mitigating TCP SYN flood attacks.
- The unblocking strategy, in real life, depends on various factors and policies of the network operator. In our implementation, we have blocked the malicious IPs and after a timeout, we enable them to reconnect
- This solution is implemented within the same subnet for simplicity and testing. In real-world scenarios, it can be scaled to multiple subnets or distributed networks.

9 References

- https://ieeexplore.ieee.org/document/1194894
- https://en.wikipedia.org/wiki/Denial-of-serviceattack
- https://ieeexplore.ieee.org/document/10235131