राष्ट्रीय प्रौद्योगिकी संस्थान सिक्किम
National Institute of Technology Sikkim
An Institute of National Importance

PRESENTATION ON THE TOPIC

“DETECTION AND MITIGATION OF FLOW TABLE FLOOD ATTACK IN SDN ARCHITECTURE”

UNDER THE GUIDANCE OF:

Prof. Mahesh Chandra Govil
Director NIT Sikkim

SUBMITTED BY:

Avinash Kumar (B200041CS)
Vishal Kumar (B200059CS)
B.Tech 4th Year
Department of Computer Science and Engineering
NIT Sikkim

OUTLINE

INTRODUCTION

- In Software-Defined Networks (SDNs), packet movement follows a distinct sequence.
- Packets are initially received, and the controller determines the route based on policies.
- The controller then installs flow rules in switches, guiding packet forwarding.
- Subsequent packets traverse the established path, with the controller adapting routes dynamically as network conditions change, ensuring efficient and flexible data transmission.
- In a Flow Table Flood attack, attacker overwhelm the flow tables by flooding them with a high volume of bogus flow requests. This exhausts the limited resources of SDN switches, causing legitimate flow entries to be displaced. Consequently, network performance degrades, and critical services may be disrupted.

MOTIVATION

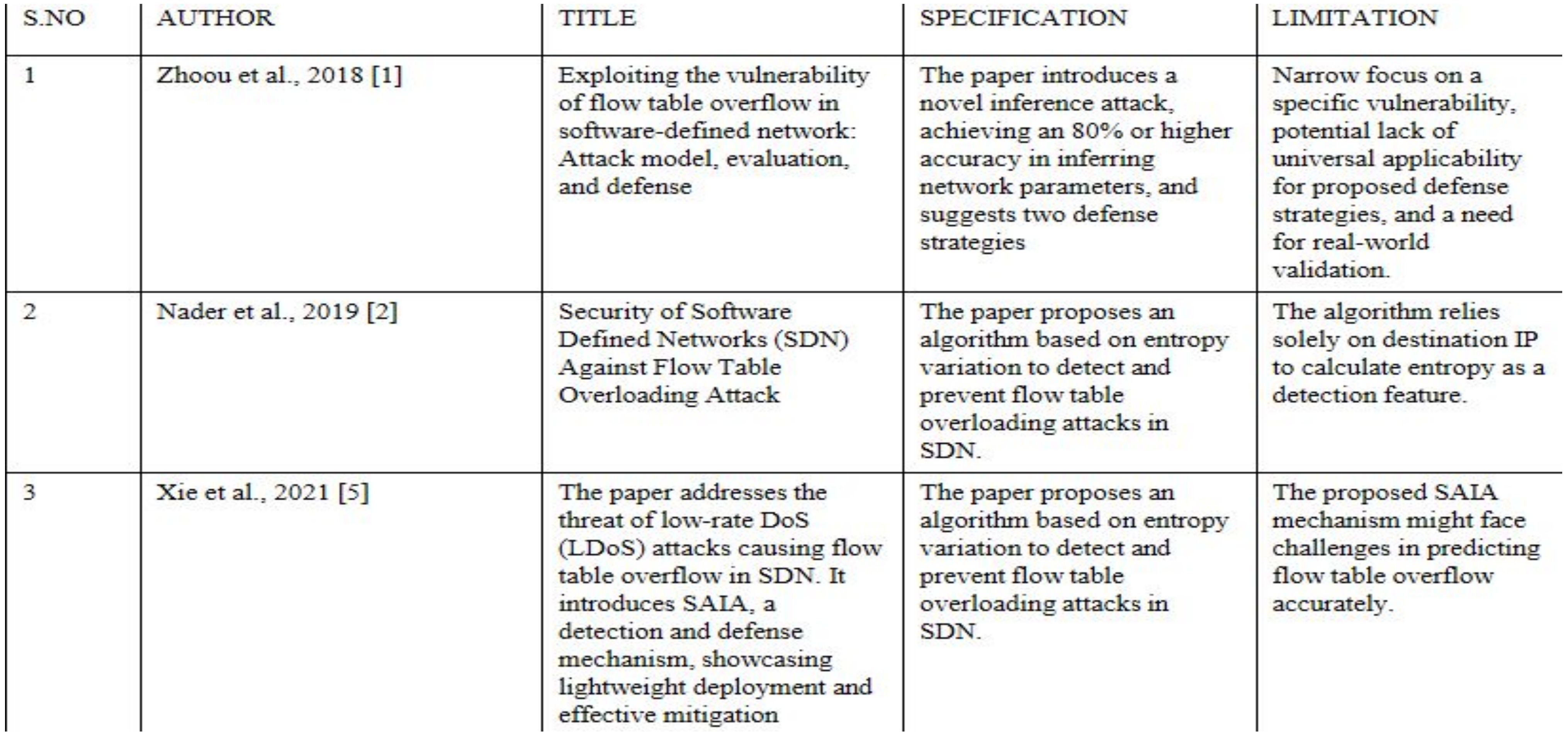
- Existing SDN algorithms for flow table overloading are specific and lack adaptability.
- Rely on static thresholds and single-feature focus, leading to false positives.
- Traditional algorithms struggle to swiftly adapt to changing attack patterns, enabling significant damage.
- Single-feature focus may overlook varied attack characteristics, emphasizing the need for multifaceted detection.
- A robust algorithm ensures a secure, reliable network, countering evolving threats with swift adaptation and multifaceted detection.

PROBLEM STATEMENT

- Current flow table overloading attack detection algorithms face limitations in coverage, adaptability, and responsiveness. Struggling to detect sophisticated attacks and adapt to dynamic conditions leads to false positives and negatives.

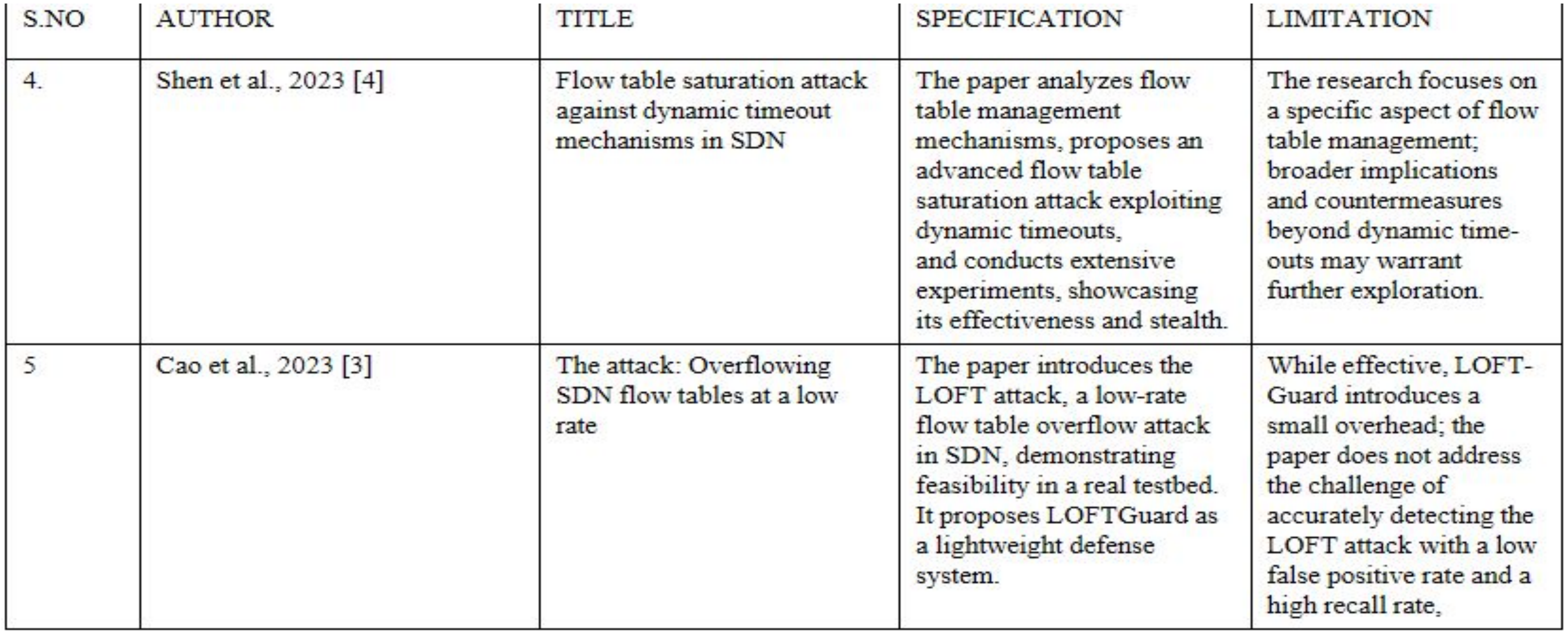
- An improved algorithm is crucial, with broad effectiveness, dynamic adaptability, multiple feature utilization, and rapid responsiveness, enhancing SDN security for a more reliable network infrastructure.

LITERATURE SURVEY



S.NO	AUTHOR	TITLE	SPECIFICATION	LIMITATION
1	Zhoou et al., 2018 [1]	Exploiting the vulnerability of flow table overflow in software-defined network: Attack model, evaluation, and defense	The paper introduces a novel inference attack, achieving an 80% or higher accuracy in inferring network parameters, and suggests two defense strategies	Narrow focus on a specific vulnerability, potential lack of universal applicability for proposed defense strategies, and a need for real-world validation.
2	Nader et al., 2019 [2]	Security of Software Defined Networks (SDN) Against Flow Table Overloading Attack	The paper proposes an algorithm based on entropy variation to detect and prevent flow table overloading attacks in SDN.	The algorithm relies solely on destination IP to calculate entropy as a detection feature.
3	Xie et al., 2021 [5]	The paper addresses the threat of low-rate DoS (LDoS) attacks causing flow table overflow in SDN. It introduces SAIA, a detection and defense mechanism, showcasing lightweight deployment and effective mitigation	The paper proposes an algorithm based on entropy variation to detect and prevent flow table overloading attacks in SDN.	The proposed SAIA mechanism might face challenges in predicting flow table overflow accurately.

LITERATURE SURVEY



S.NO	AUTHOR	TITLE	SPECIFICATION	LIMITATION
4.	Shen et al., 2023 [4]	Flow table saturation attack against dynamic timeout mechanisms in SDN	The paper analyzes flow table management mechanisms, proposes an advanced flow table saturation attack exploiting dynamic timeouts, and conducts extensive experiments, showcasing its effectiveness and stealth.	The research focuses on a specific aspect of flow table management; broader implications and countermeasures beyond dynamic time-outs may warrant further exploration.
5	Cao et al., 2023 [3]	The attack: Overflowing SDN flow tables at a low rate	The paper introduces the LOFT attack, a low-rate flow table overflow attack in SDN, demonstrating feasibility in a real testbed. It proposes LOFTGuard as a lightweight defense system.	While effective, LOFT-Guard introduces a small overhead; the paper does not address the challenge of accurately detecting the LOFT attack with a low false positive rate and a high recall rate,

LITERATURE SURVEY

Sl No.	AUTHORS	TITLE	SPECIFICATION	LIMITATION
6	Mishra, A., et al. 2021[13]	Entropy based features distribution for anti-ddos model in sdn," Sustainability,	The paper introduces a defensive mechanism utilizing entropy variation of destination IP address in SDN-Cloud OSN to detect and block DDoS attacks.	Limited to entropy-based DDoS detection. May not work for larger volume attacks and other topologies. False positives could still occur.
7	Ahalawat, A., et al. (2019)[14]	A hybrid entropy based dos attacks detection system for software defined networks (sdn): A proposed trust mechanism,	DDoS attack overloads systems to deny access. DDoS attackers use a set of compromised users to flood systems.	DDoS's distributed nature makes mitigation difficult. Limits capacity and affects bandwidth, making legitimate use impossible during attack.

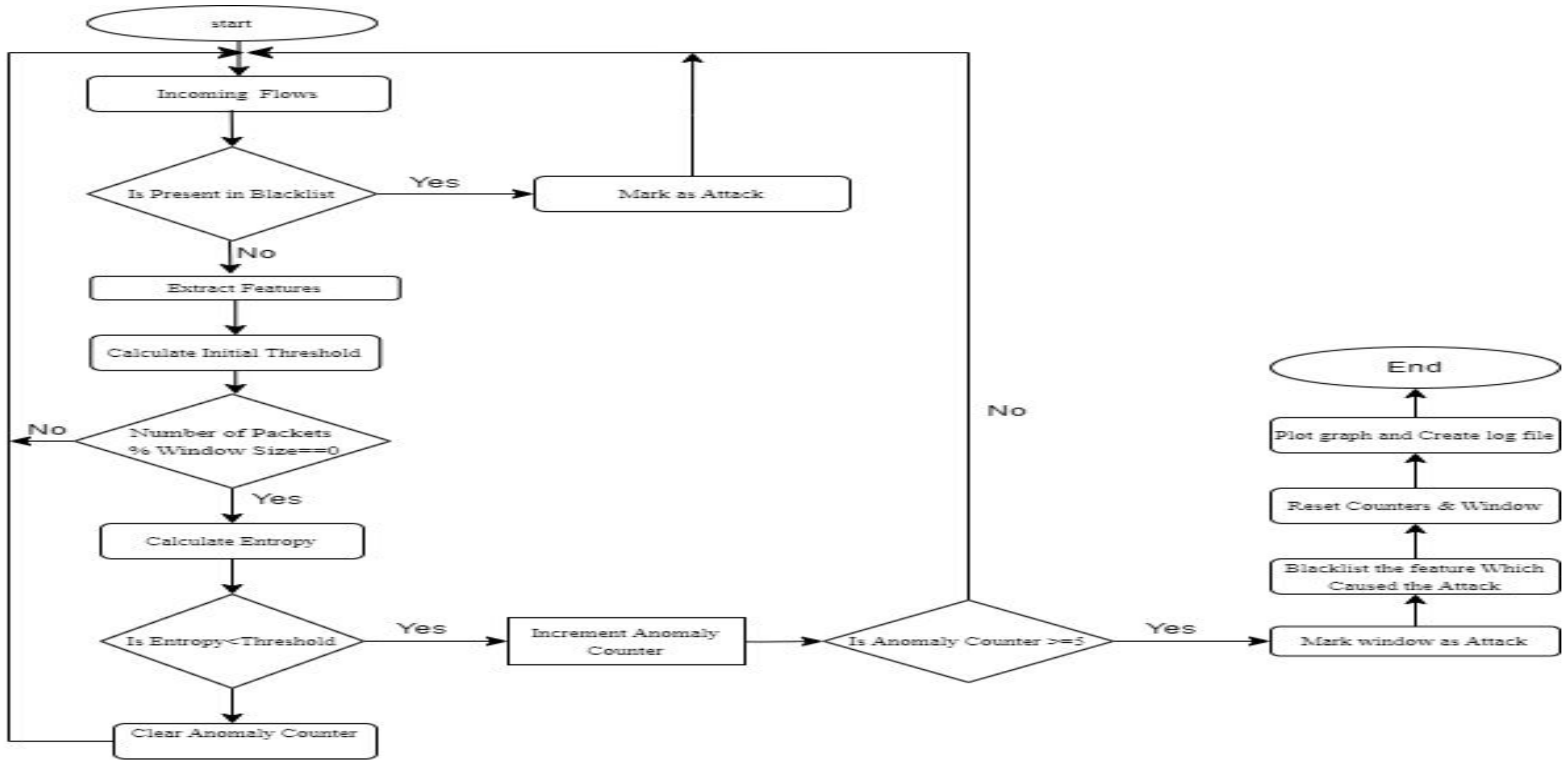
PROPOSED METHODOLOGY

- Multi-Feature Entropy: Calculate entropy based on a combination of features like destination IP, packet size, and flow duration to capture a broader range of attack patterns.

- Adaptive Threshold: Implement an adaptive threshold mechanism that automatically adjusts based on observed traffic patterns. This ensures the algorithm adapts to changing network conditions and maintains optimal detection accuracy.

- Attack Response Mechanism: Develop an integrated response mechanism that triggers mitigation actions upon detecting a potential attack. This could involve rate limiting, dropping suspicious packets, or notifying administrators for further investigation.

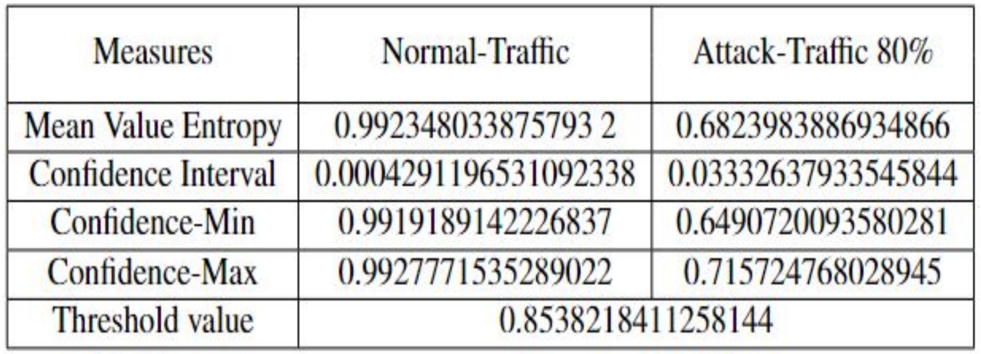
PROPOSED METHODOLOGY CONTINUED..



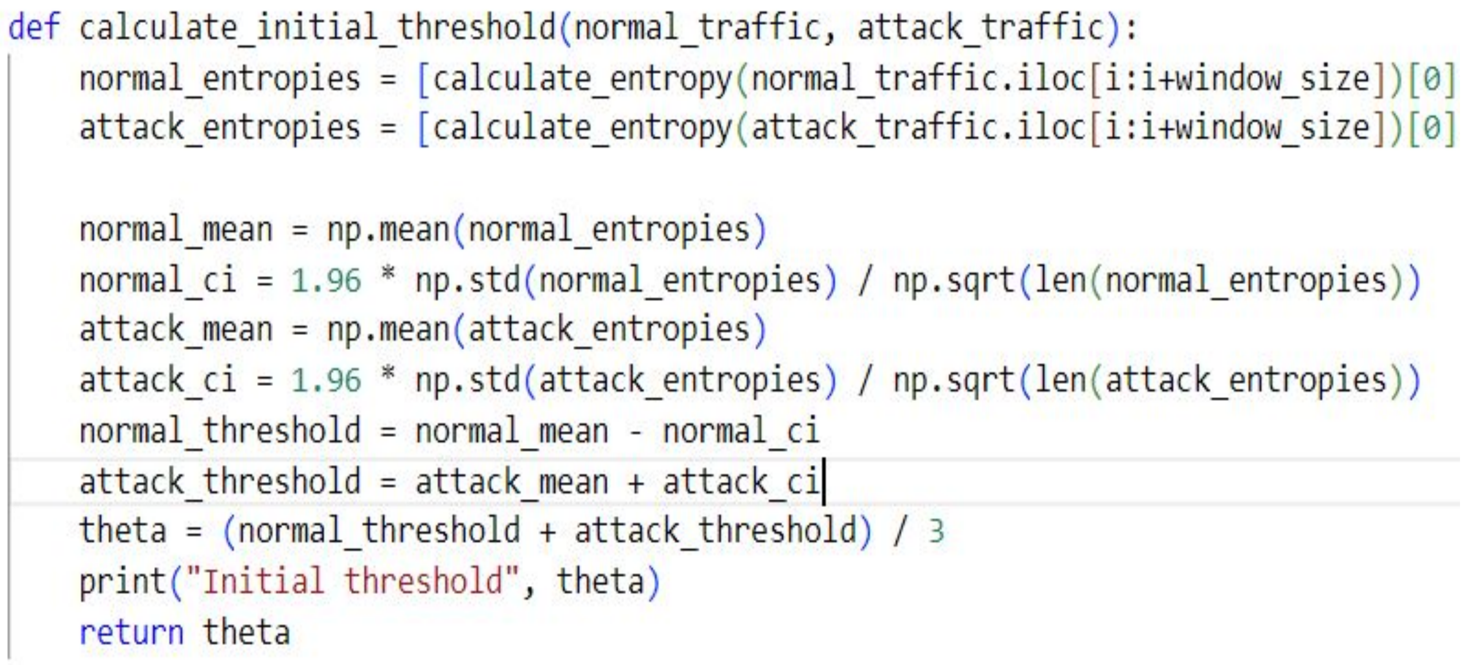
IMPLEMENTATION

Procedure 1: Packet Arrival

Calculation of Initial Threshold:



Measures	Normal-Traffic	Attack-Traffic 80%
Mean Value Entropy	0.992348033875793 2	0.6823983886934866
Confidence Interval	0.0004291196531092338	0.03332637933545844
Confidence-Min	0.9919189142226837	0.6490720093580281
Confidence-Max	0.9927771535289022	0.715724768028945
Threshold value	0.8538218411258144	



```
def calculate_initial_threshold(normal_traffic, attack_traffic):
    normal_entropies = [calculate_entropy(normal_traffic.iloc[i:i+window_size])[0]
    attack_entropies = [calculate_entropy(attack_traffic.iloc[i:i+window_size])[0]

    normal_mean = np.mean(normal_entropies)
    normal_ci = 1.96 * np.std(normal_entropies) / np.sqrt(len(normal_entropies))
    attack_mean = np.mean(attack_entropies)
    attack_ci = 1.96 * np.std(attack_entropies) / np.sqrt(len(attack_entropies))
    normal_threshold = normal_mean - normal_ci
    attack_threshold = attack_mean + attack_ci
    theta = (normal_threshold + attack_threshold) / 3
    print("Initial threshold", theta)
    return theta
```

IMPLEMENTATION

Procedure 2: Window Management

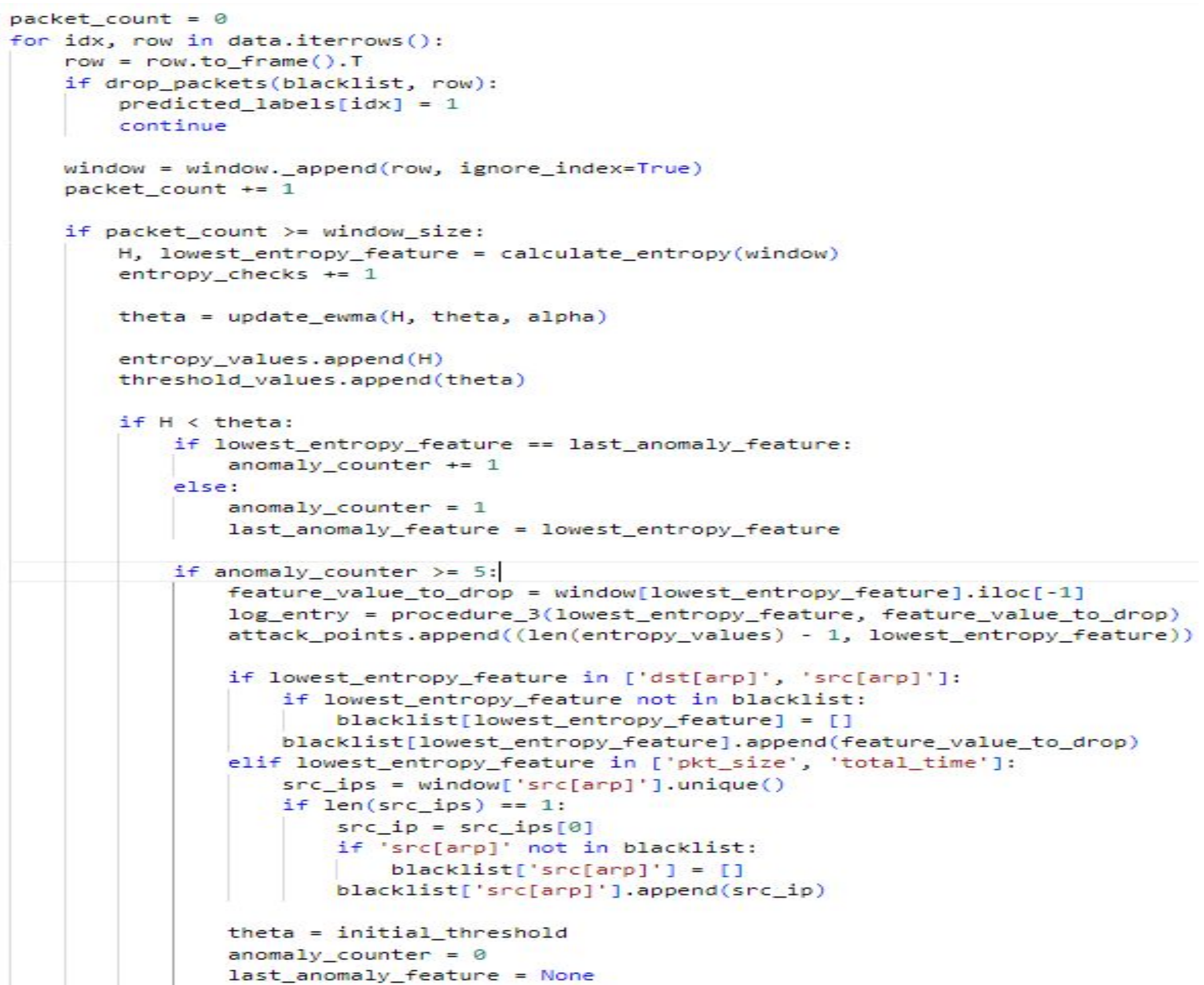
Dynamic Window Size:

Utilizes a sliding window approach to process a continuous stream of network packets. Adjusts window size dynamically to accommodate varying traffic loads and patterns.

Anomaly Detection:

If entropy of window is less than threshold,

Add the feature to Blacklist and call Response mechanism when attack is declared.



```
packet_count = 0
for idx, row in data.iterrows():
    row = row.to_frame().T
    if drop_packets(blacklist, row):
        predicted_labels[idx] = 1
        continue

    window = window._append(row, ignore_index=True)
    packet_count += 1

    if packet_count >= window_size:
        H, lowest_entropy_feature = calculate_entropy(window)
        entropy_checks += 1

        theta = update_ewma(H, theta, alpha)

        entropy_values.append(H)
        threshold_values.append(theta)

        if H < theta:
            if lowest_entropy_feature == last_anomaly_feature:
                anomaly_counter += 1
            else:
                anomaly_counter = 1
                last_anomaly_feature = lowest_entropy_feature

            if anomaly_counter >= 5:
                feature_value_to_drop = window[lowest_entropy_feature].iloc[-1]
                log_entry = procedure_3(lowest_entropy_feature, feature_value_to_drop)
                attack_points.append((len(entropy_values) - 1, lowest_entropy_feature))

                if lowest_entropy_feature in ['dst[arp]', 'src[arp]']:
                    if lowest_entropy_feature not in blacklist:
                        blacklist[lowest_entropy_feature] = []
                    blacklist[lowest_entropy_feature].append(feature_value_to_drop)
                elif lowest_entropy_feature in ['pkt_size', 'total_time']:
                    src_ips = window['src[arp]'].unique()
                    if len(src_ips) == 1:
                        src_ip = src_ips[0]
                        if 'src[arp]' not in blacklist:
                            blacklist['src[arp]'] = []
                        blacklist['src[arp]'].append(src_ip)

                theta = initial_threshold
                anomaly_counter = 0
                last_anomaly_feature = None
```

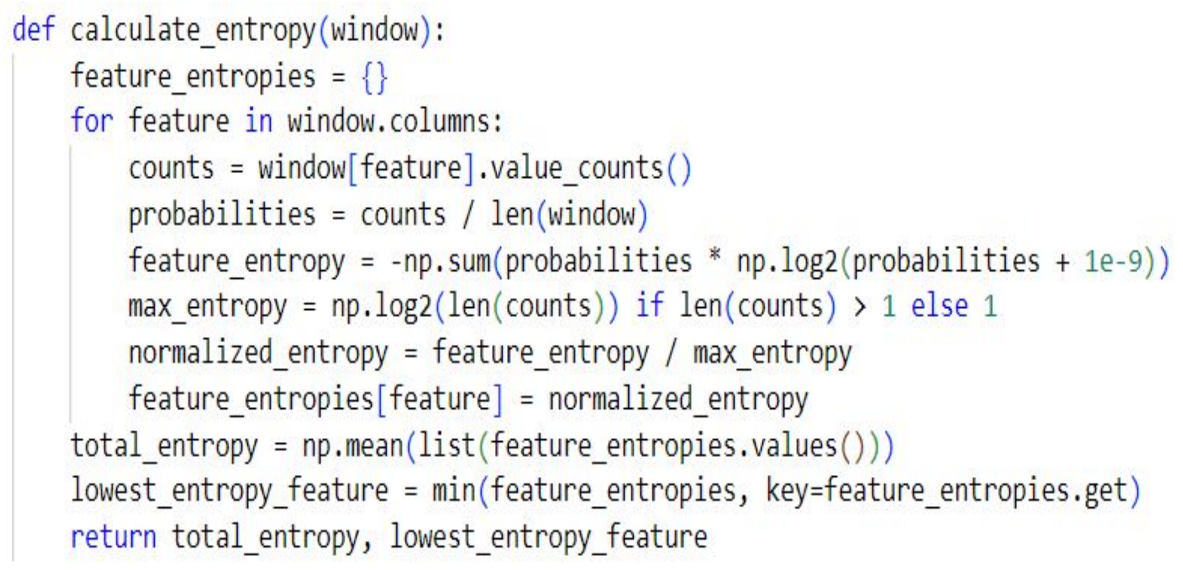
IMPLEMENTATION

Entropy Calculation:

Calculates Shannon entropy to measure the uncertainty of packet feature distributions within the window.

$$H(X) = -\sum P(x_i) \cdot \log_2(P(x_i) + 1 \times 10^{-9})$$

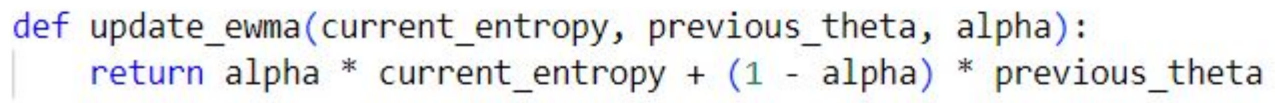
$$H_{\text{total}} = \frac{1}{N_{\text{features}}} \sum_{i=1}^{N_{\text{features}}} H_{\text{normalized}}(X_i)$$



```python
def calculate_entropy(window):
    feature_entropies = {}
    for feature in window.columns:
        counts = window[feature].value_counts()
        probabilities = counts / len(window)
        feature_entropy = -np.sum(probabilities * np.log2(probabilities + 1e-9))
        max_entropy = np.log2(len(counts)) if len(counts) > 1 else 1
        normalized_entropy = feature_entropy / max_entropy
        feature_entropies[feature] = normalized_entropy
    total_entropy = np.mean(list(feature_entropies.values()))
    lowest_entropy_feature = min(feature_entropies, key=feature_entropies.get)
    return total_entropy, lowest_entropy_feature
```

Update EWMA:

Updates threshold using EWMA to adapt to changing network conditions and detect anomalies.



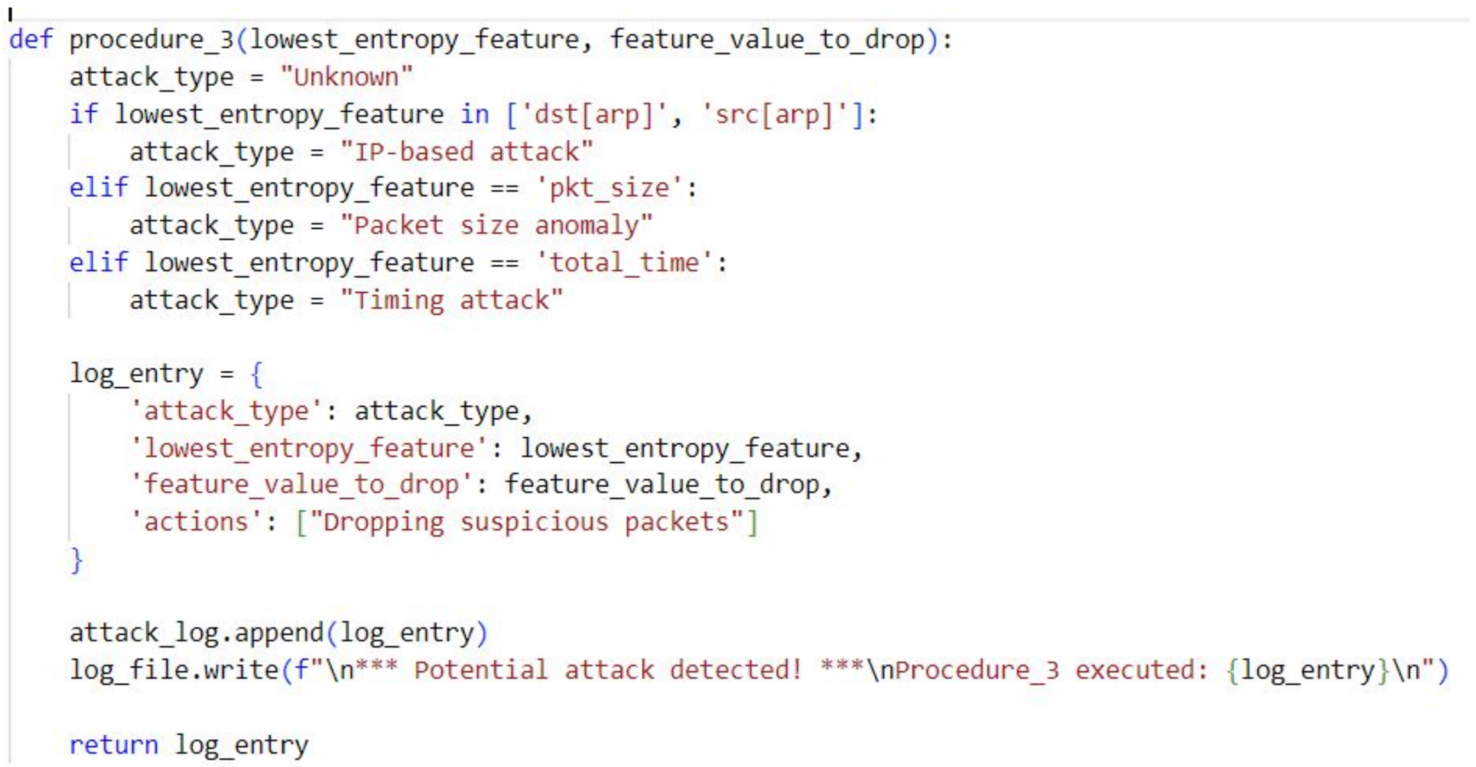
```python
def update_ewma(current_entropy, previous_theta, alpha):
    return alpha * current_entropy + (1 - alpha) * previous_theta
```

IMPLEMENTATION

Procedure 3: Response Mechanism

Classification of Attack :

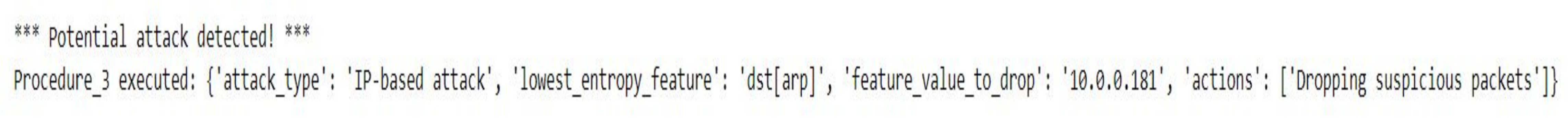
This code classifies attacks based on features and values. It assigns attack types such as IP-based, packet size anomaly, or timing attack. The classification triggers logging and action execution for suspicious packets.



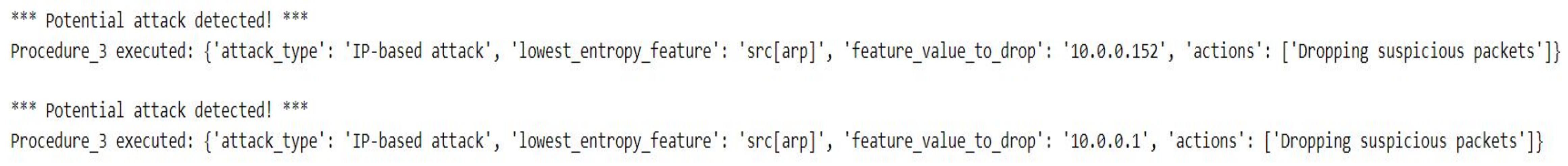
```python
def procedure_3(lowest_entropy_feature, feature_value_to_drop):
    attack_type = "Unknown"
    if lowest_entropy_feature in ['dst[arp]', 'src[arp]']:
        attack_type = "IP-based attack"
    elif lowest_entropy_feature == 'pkt_size':
        attack_type = "Packet size anomaly"
    elif lowest_entropy_feature == 'total_time':
        attack_type = "Timing attack"

    log_entry = {
        'attack_type': attack_type,
        'lowest_entropy_feature': lowest_entropy_feature,
        'feature_value_to_drop': feature_value_to_drop,
        'actions': ["Dropping suspicious packets"]
    }

    attack_log.append(log_entry)
    log_file.write(f"\n*** Potential attack detected! ***\nProcedure_3 executed: {log_entry}\n")

    return log_entry
```

TESTING AND RESULT

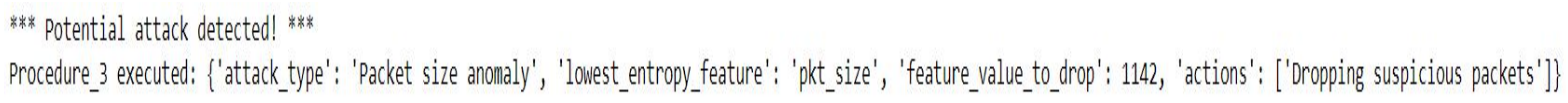
The algorithm was able to detect and classify attacks based on different entropy levels of attributes present in the SDN flow.



```
*** Potential attack detected! ***
Procedure_3 executed: {'attack_type': 'IP-based attack', 'lowest_entropy_feature': 'dst[arp]', 'feature_value_to_drop': '10.0.0.181', 'actions': ['Dropping suspicious packets']}
```

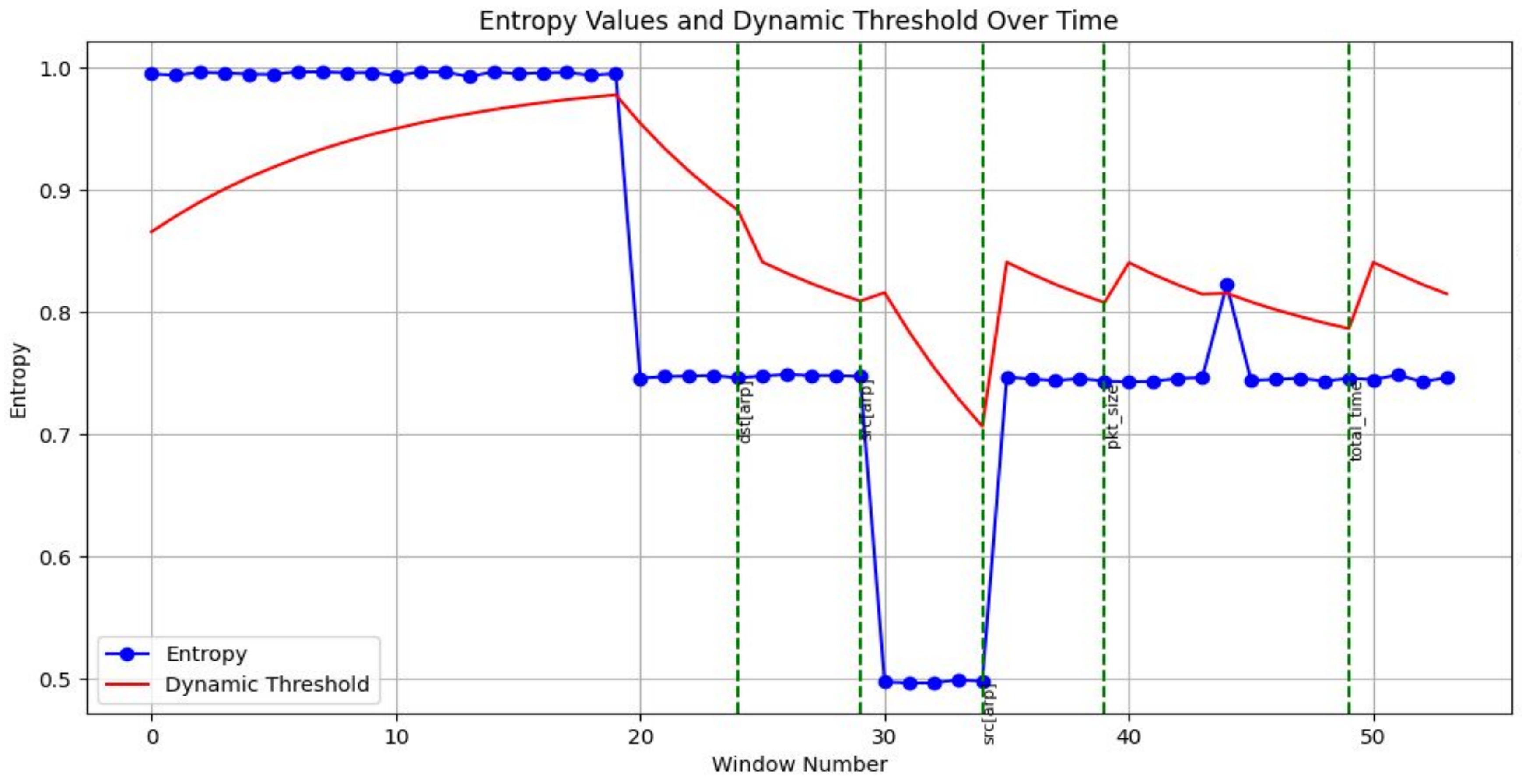


```
*** Potential attack detected! ***
Procedure_3 executed: {'attack_type': 'IP-based attack', 'lowest_entropy_feature': 'src[arp]', 'feature_value_to_drop': '10.0.0.152', 'actions': ['Dropping suspicious packets']}

*** Potential attack detected! ***
Procedure_3 executed: {'attack_type': 'IP-based attack', 'lowest_entropy_feature': 'src[arp]', 'feature_value_to_drop': '10.0.0.1', 'actions': ['Dropping suspicious packets']}
```



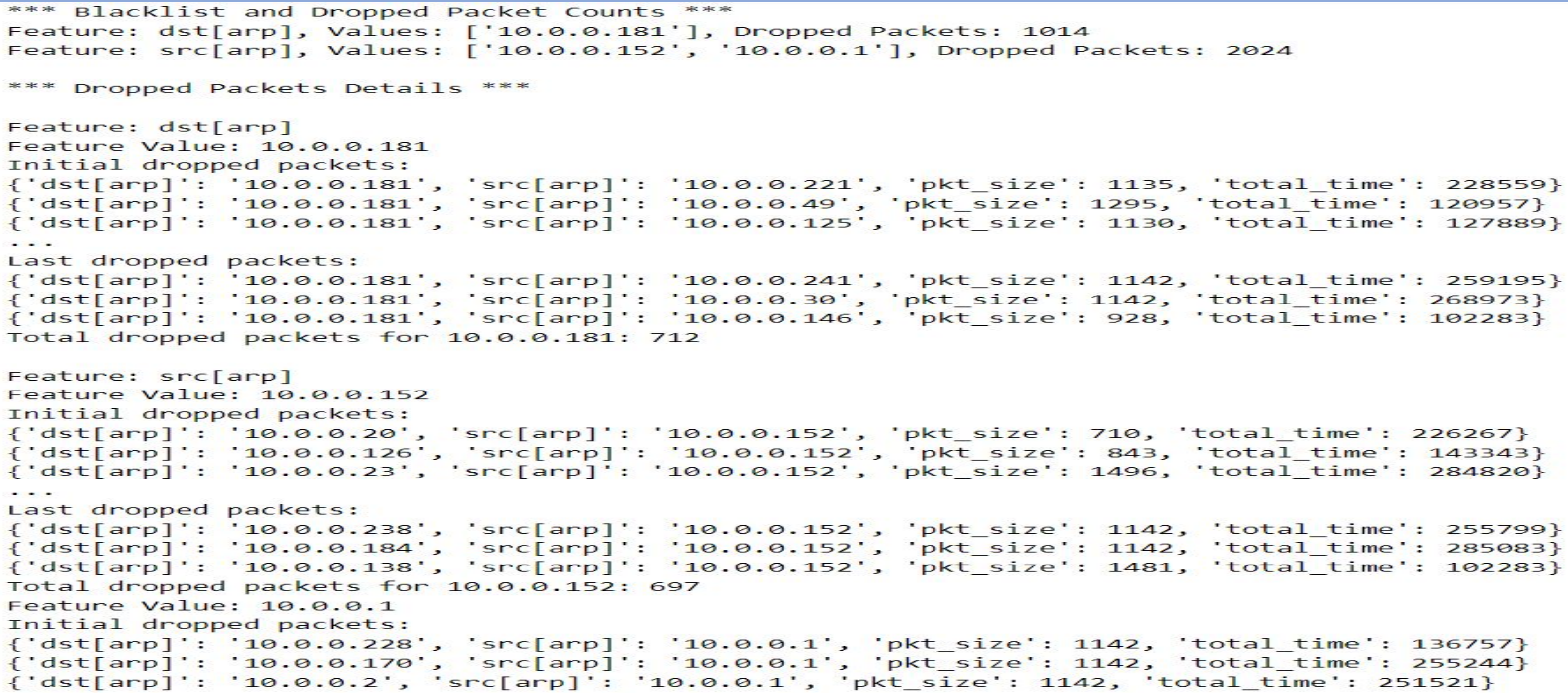
```
*** Potential attack detected! ***
Procedure_3 executed: {'attack_type': 'Packet size anomaly', 'lowest_entropy_feature': 'pkt_size', 'feature_value_to_drop': 1142, 'actions': ['Dropping suspicious packets']}
```

TESTING AND RESULT



Measure	Value
Sensitivity	0.8103
Specificity	0.9238
Precision	0.9115
Negative Predictive Value	0.8466
False Positive Rate	0.0297
False Discovery Rate	0.0322
False Negative Rate	0.1231
Detection Rate	0.9650
F1 Score	0.7889

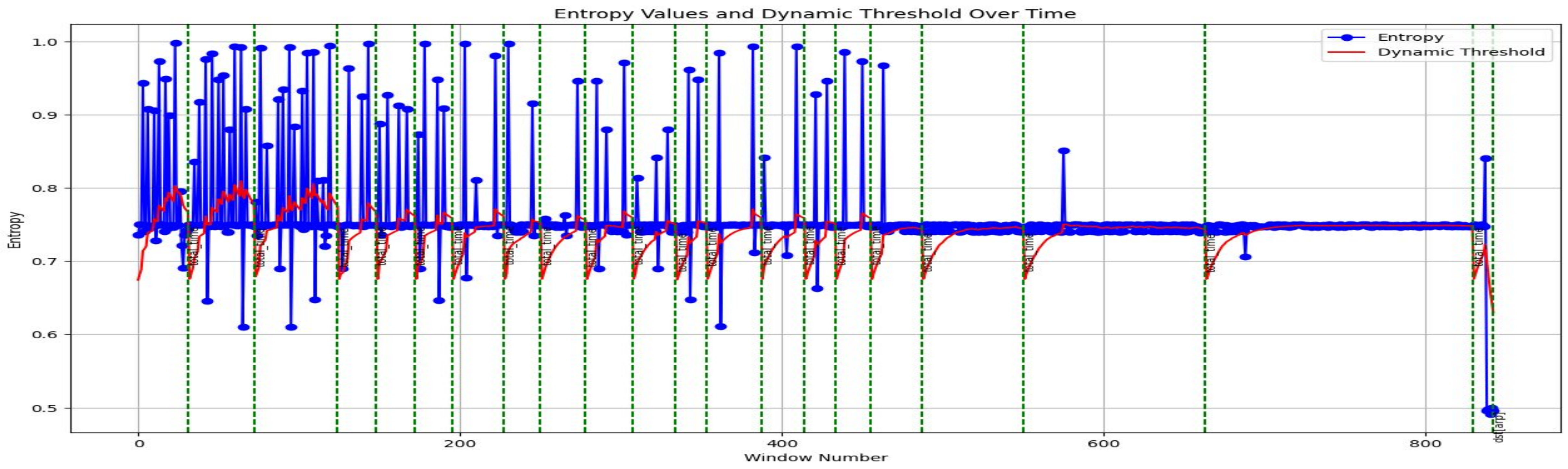
TESTING AND RESULT



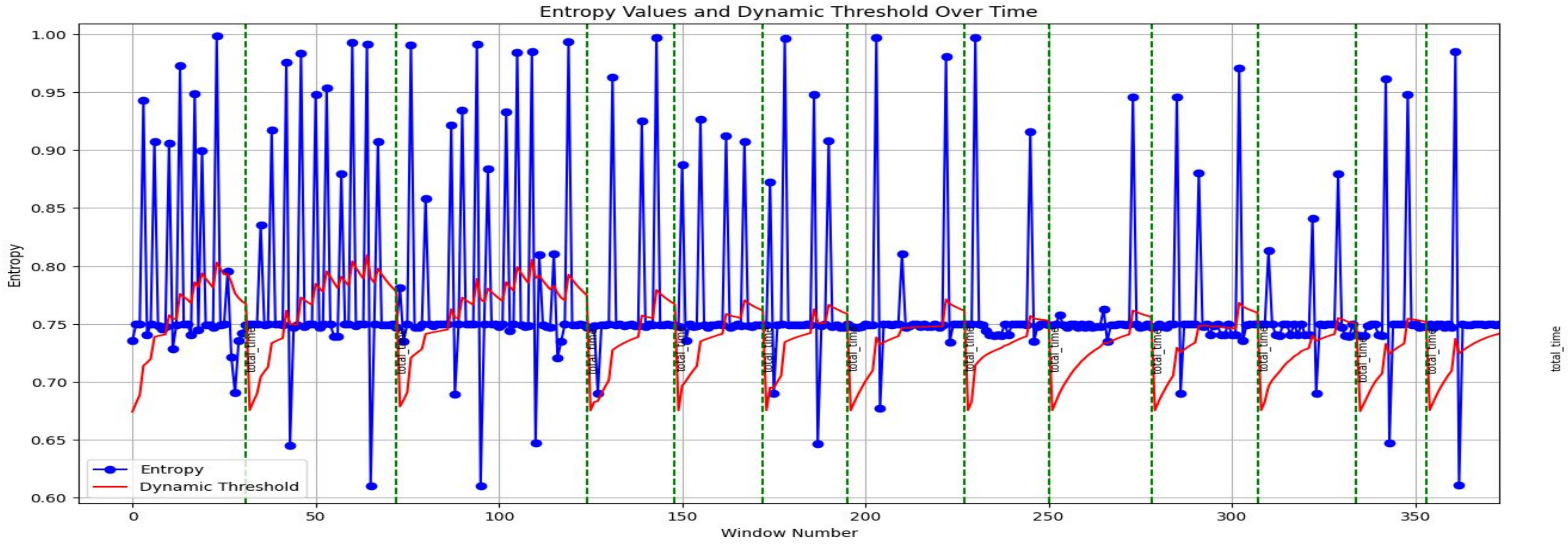
```
*** Blacklist and Dropped Packet Counts ***
Feature: dst[arp], Values: ['10.0.0.181'], Dropped Packets: 1014
Feature: src[arp], Values: ['10.0.0.152', '10.0.0.1'], Dropped Packets: 2024

*** Dropped Packets Details ***

Feature: dst[arp]
Feature Value: 10.0.0.181
Initial dropped packets:
{'dst[arp]': '10.0.0.181', 'src[arp]': '10.0.0.221', 'pkt_size': 1135, 'total_time': 228559}
{'dst[arp]': '10.0.0.181', 'src[arp]': '10.0.0.49', 'pkt_size': 1295, 'total_time': 120957}
{'dst[arp]': '10.0.0.181', 'src[arp]': '10.0.0.125', 'pkt_size': 1130, 'total_time': 127889}
...
Last dropped packets:
{'dst[arp]': '10.0.0.181', 'src[arp]': '10.0.0.241', 'pkt_size': 1142, 'total_time': 259195}
{'dst[arp]': '10.0.0.181', 'src[arp]': '10.0.0.30', 'pkt_size': 1142, 'total_time': 268973}
{'dst[arp]': '10.0.0.181', 'src[arp]': '10.0.0.146', 'pkt_size': 928, 'total_time': 102283}
Total dropped packets for 10.0.0.181: 712

Feature: src[arp]
Feature Value: 10.0.0.152
Initial dropped packets:
{'dst[arp]': '10.0.0.20', 'src[arp]': '10.0.0.152', 'pkt_size': 710, 'total_time': 226267}
{'dst[arp]': '10.0.0.126', 'src[arp]': '10.0.0.152', 'pkt_size': 843, 'total_time': 143343}
{'dst[arp]': '10.0.0.23', 'src[arp]': '10.0.0.152', 'pkt_size': 1496, 'total_time': 284820}
...
Last dropped packets:
{'dst[arp]': '10.0.0.238', 'src[arp]': '10.0.0.152', 'pkt_size': 1142, 'total_time': 255799}
{'dst[arp]': '10.0.0.184', 'src[arp]': '10.0.0.152', 'pkt_size': 1142, 'total_time': 285083}
{'dst[arp]': '10.0.0.138', 'src[arp]': '10.0.0.152', 'pkt_size': 1481, 'total_time': 102283}
Total dropped packets for 10.0.0.152: 697
Feature Value: 10.0.0.1
Initial dropped packets:
{'dst[arp]': '10.0.0.228', 'src[arp]': '10.0.0.1', 'pkt_size': 1142, 'total_time': 136757}
{'dst[arp]': '10.0.0.170', 'src[arp]': '10.0.0.1', 'pkt_size': 1142, 'total_time': 255244}
{'dst[arp]': '10.0.0.2', 'src[arp]': '10.0.0.1', 'pkt_size': 1142, 'total_time': 251521}
```

TESTING AND RESULT

We conducted experiments to assess the performance of the algorithm on a distinct and more extensive dataset, simulating a more realistic network environment. Following this, We generated graphs to visualize the outcomes of the algorithm's execution.



TESTING AND RESULT



CONCLUSION AND FUTURE SCOPE

CONCLUSION

1. Novel SDN algorithm uses entropy and EWMA for robust anomaly detection and mitigation.
2. Simulations show high accuracy and low false-positives, ensuring real-time deployment effectiveness.
3. Enhances SDN security by addressing flow table vulnerabilities with scalable, lightweight solutions.

FUTURE SCOPE

1. Enhance the algorithm by incorporating machine learning techniques to predict and preemptively mitigate flow table overloading attacks.
2. Investigate the integration of our algorithm with other SDN security frameworks for a comprehensive defense mechanism.
3. Explore adaptive parameter tuning to further reduce the algorithm's computational overhead and improve response times.

REFERENCES

1. Y. Zhou, K. Chen, J. Zhang, J. Leng, and Y. Tang, “Exploiting the vulnerability of flow table overflow in software-defined network: Attack model, evaluation, and defense,” Security and Communication Networks, vol. 2018, pp. 1–15, 2018.

2. R. Neamah and W. Bhaya, “Security of software defined networks (sdn) against flow table overloading attack,” Journal of Advanced Research in Dynamical and Control Systems, vol. 11, pp. 752–759, 10 2019.

3. Y. Shen, C. Wu, D. Kong, and Q. Cheng, “Flow table saturation attack against dynamic timeout mechanisms in sdn,” Applied Sciences, vol. 13, no. 12, p. 7210, 2023.

4. J. Cao, M. Xu, Q. Li, K. Sun, and Y. Yang, “The attack: Overflowing sdn flow tables at a low rate,” IEEE/ACM Transactions on Networking, 2022

5. S. Xie, C. Xing, G. Zhang, and J. Zhao, “A table overflow ldos attack defending mechanism in software-defined networks,” Security and Communication Networks, vol. 2021, pp. 1–16, 2021

6. S. A. Mehdi, J. Khalid, and S. A. Khayam, “Revisiting traffic anomaly detection using software defined networking,” in Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2019. Proceedings 14. Springer, 2011, pp. 161–180.

REFERENCES

7. M. J. Santos-Neto, J. L. Bordim, E. A. Alchieri, and E. Ishikawa, “Ddos attack detection in sdn: Enhancing entropy-based detection with machine learning,” Concurrency and Computation: Practice and Experience, p. e8021, 2024.

8. M. A. Aladaileh, M. Anbar, A. J. Hintaw, I. H. Hasbullah, A. A. Bahashwan, T. A. Al-Amiedy, and D. R. Ibrahim, “Effectiveness of an entropy-based approach for detecting low-and high-rate ddos attacks against the sdn controller: Experimental analysis,” Applied Sciences, vol. 13, no. 2, p. 775, 2023.

9. D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, 2014.

10. B. Yuan, D. Zou, S. Yu, H. Jin, W. Qiang, and J. Shen, “Defending against flow table overloading attack in software-defined networks,” IEEE Transactions on Services Computing, vol. 12, no. 2, pp. 231–246, 2016.

11. Z. Li, W. Xing, S. Khamaiseh, and D. Xu, “Detecting saturation attacks based on self-similarity of openflow traffic,” IEEE Transactions on Network and Service Management, vol. 17, no. 1, pp. 607–621, 2019.

12. M. Zhang, J. Bi, J. Bai, Z. Dong, Y. Li, and Z. Li, “Ftguard: A priority-aware strategy against the flow table overflow attack in sdn,” in Proceedings of the SIGCOMM Posters and Demos, 2017, pp. 141–143.

13. R. M. A. Ujjan, Z. Pervez, K. Dahal, W. A. Khan, A. M. Khattak, and B. Hayat, “Entropy based features distribution for anti-ddos model in sdn,” Sustainability, vol. 13,no. 3, p. 1522, 2021.

REFERENCES

14. N. M. AbdelAzim, S. F. Fahmy, M. A. Sobh, and A. M. B. Eldin, “A hybrid entropy based dos attacks detection system for software defined networks (sdn): A proposed trust mechanism,” Egyptian Informatics Journal, vol. 22, no. 1, pp. 85–90, 2021.

15. H. A. AL-OFEISHAT, “Dynamic threshold generalized entropies-based ddos detection against software-defined network controller,” Journal of Theoretical and Applied Information Technology, vol. 102, no. 3, 2024.

THANK YOU