ISI Journal of Information Systems and Informatics

Published By Asosiasi Doktor Sistem Informasi Indonesia

Towards Self-Defending SDN Infrastructures: Real-Time Honeypot-Enabled Botnet Detection Using ONOS

Nyamwaga M. Kaare[1], Anael E. Sam[2]

[1,2]The Nelson Mandela African Institution of Science and Technology, Tanzania

Received:
November 4, 2025
Revised:
December 3, 2025
Accepted:
January 6, 2026
Published:
February 12, 2026

Corresponding Author:
Author Name*:
Nyamwaga M. Kaare
Email*:
kaaren@nm-aist.ac.tz

DOI:
10.63158/journalisi.v8i1.1375

Abstract. Modern Software-Defined Networks (SDNs), while benefiting from centralized programmability, remain vulnerable to fast-evolving botnet attacks. This paper presents and evaluates a lightweight ONOS-based honeypot and decoy framework designed to detect and automatically block multi-vector botnet behaviors in real time. The system integrates honeypot-exposed Telnet, SMB, and DNS services with threshold-, entropy-, signature-, and correlation-based inspection within a tree topology (depth = 2, fanout = 4) consisting of five OpenFlow switches and 50 hosts. Quantitatively, the system achieved 100% detection of all signature-based attacks (55/55), 100% blocking of distributed UDP scans (50/50), and 0% false positives on benign decoy access. Median detection latency ranged between 1–3 seconds. True positives (TP), false negatives (FN), false positives (FP), and true negatives (TN) were measured using ground-truth attacker lists built into automated test scripts, yielding precision and recall of 1.00 across all malicious scenarios. This work demonstrates that combining deception with SDN-level flow automation enables effective and computationally efficient botnet defense without machine learning. A key limitation is that all evaluations were conducted exclusively in a controlled Mininet simulation, which may not fully represent real-world traffic dynamics. Future work will validate the system on physical SDN deployments and evaluate its robustness under production workloads.

Keywords: Botnet Detection, Flow Automation, Honeypot, Network Security, ONOS, Software-Defined Networking

1. INTRODUCTION

In this era of cloud computing, the Internet of Things (IoT), and increased connection of smart devices, most people rely on Internet [1], [2]. As a result, network designs have become more complex and vulnerable to advanced cyber-attacks such as those launched by botnets [3], [4], [5]. Botnets are large collections of synchronized malicious devices that carry out widespread assaults on targeted systems. Since 2003, botnets have been present in many forms and have gained widespread recognition as one of the most major and destructive threats [6]. The longevity of botnets may be attributed to their ability to adapt and evolve via constant updates to their infrastructure and algorithms. Hence, the duration for which a certain botnet detection system may maintain its effectiveness and usefulness is a crucial factor in its design [7]. Consequently, honeypot has been used as a method for examining the characteristics and actions of different types of assaults, including those launched by botnets. It employs deception to mislead attackers into believing they are interacting with a genuine system, with the goal of monitoring their actions and delaying their efforts to attack the system [8].

Moreover, the introduction of Software-Defined Network (SDN) addressed the challenge of managing complex networks by separating them into distinct data and control planes [9]. The control plane is responsible for decision-making and centralized management of network activities. On the other hand, the data plane is where the actual forwarding of data packets occurs depending on the decisions made by the control plane. This enables more flexibility, programmability, and agility in network management [10], [11].

Prior research indicates that the botnet may be detected at both the recruiting and execution stages. However, the responsibility of preventing the network from botnet attacks still lies with the network administrator [12]. Conversely, research conducted by the Information Systems Audit and Control Association (ISACA) highlights a significant scarcity of security experts who possess the necessary training and expertise, with a growing need for such individuals. Furthermore, the current cybersecurity workforce is insufficiently manned, posing challenges in promptly addressing attacks 24/7, throughout the year [13]. Machine Learning (ML) and statistics are often used in these studies [14] [15], [16], [17], [18], [19], which require large training datasets and offline analysis. These approaches can be slow to adapt and may not operate in real time, imposing computational overhead and requiring human tuning.

Hassan et al. [20] propose an entropy-based and machine-learning framework for DDoS detection in SDN, using traffic randomness and k-means clustering to classify network behavior across datasets such as CIC-IDS2017. Although effective, their approach relies on periodic entropy computation, feature extraction, and trained models, making it resource-intensive and less suited for real-time response. Their findings highlight the ongoing need for adaptive, lightweight SDN defenses capable of addressing multi-vector attacks with faster, controller-level mitigation.

Fan et al [21] suggested an SDN model for the network data controller of hybrid honeypots. The controller sends potentially interesting traffic to monitoring stations called "honeypots". It utilizes OpenFlow switches, Snort, and Ryu SDN to filter traffic and implement a TCP connection handover mechanism. The design of the data controller has been greatly eased by SDN technologies, in particular, the capability to programmatically monitor and govern network data flows. But because of the limited virtual environments used for testing, several performance problems have been found.

A method developed by Ichise et al [22] aims to identify botnet DNS traffic through the creation of a mechanism. Any DNS communication that is discovered to use an invalid name server is immediately flagged as malicious and blocked. However, DNS traffic is the only topic of investigation here. It was suggested by Achleitner et al [23] to utilize decoys and honeypots in SDN as part of a reconnaissance deception system to stall an attacker during the scanning phase. As the system produces unique virtual views for every single node in the network, it is not suited to identifying malicious behaviours that are disseminated over the network as a whole.

Ja'fari et al [24] in their study offered an innovative way to block intelligent. In the intelligent blocking strategy, the loaders are distinguished from the bots once the connection between the members of the botnet is determined. This allows the loaders to be blocked. After that, only the loaders will have their ability to access other hosts restricted. Botnets that rely on loaders often do not have loaders that can communicate with the botmaster. However, they used a centralized decoy manager, which is inefficient for handling high volumes of traffic and is vulnerable to becoming a central point of failure.

Table 1. Summary of the previous studies, contribution and limitation

Study	Contribution	Limitation
Du & Wang [14]	Provides systems protections from Distributed Denial of Services (DDoS) attacks	Lack multi-vector detection combining signature, threshold, and correlation-based attacks
Pillutla & Arjunan [19]	A Fuzzy self-organizing maps-based DDOS mitigation (FSOMDM) technique that is ideally and suitably	It does not handle scanning, botnet behavior, ICMP floods, signature-based attacks
Hassan et al [20]	An entropy and machine learning based approach for DDoS attacks detection in software defined networks	Missing early-stage reconnaissance detection and provide no real-time mitigation
Ichise et al [22]	Detection and blocking of anomaly DNS Traffic by analyzing achieved NS record history	Does not incorporate deception, multi-protocol inspection, signature matching, or coordinated attack analysis
Achleitner et al [23]	SDN-based Reconnaissance Deception System (RDS) that generates virtual network topologies to mislead insider scanners, delay host discovery, and identify scanning sources through SDN flow-statistics analysis.	It does not perform real-time multi-vector botnet detection
Ja'fari et al [24]	An intelligent botnet blocking approach in software defined networks using honeypots	Does not incorporate deception, multi-protocol inspection, signature matching, or coordinated attack analysis

Existing SDN honeypot approaches lack multi-vector detection combining signature, threshold, and correlation-based methods. As a result, current solutions struggle to keep pace with increasingly complex attacks while placing heavy pressure on administrators. This paper addresses that gap by proposing a proactive automation-based botnet blocking system that unifies honeypots, SDN, and automated detection to strengthen network resilience. By minimizing manual intervention and reducing reliance on scarce cybersecurity expertise, the system streamlines threat detection, investigation, mitigation, and prevention, enabling faster and more effective responses to emerging attacks.

2. METHODS

2.1. Network Topology and Roles

We constructed a tree topology (depth=2, fanout=4) that yields exactly five switches: one root and four child switches, connected in Mininet as in Figure 1. Four special hosts were attached to randomly chosen switches: a honeypot and three decoys. The honeypot (IP 10.0.0.200/8) ran services on Telnet (TCP/23), SMB (TCP/445), and DNS (UDP/53). The decoys provided benign services: decoy1 (IP 10.0.0.201) ran an HTTP listener on TCP/8080; decoy2 (IP 10.0.0.202) ran an SSH listener on TCP/2222; decoy3 (IP 10.0.0.203) ran an SNMP listener on UDP/161. All other hosts (h1–h50, IPs 10.0.0.1–10.0.0.50/8) were normal clients.

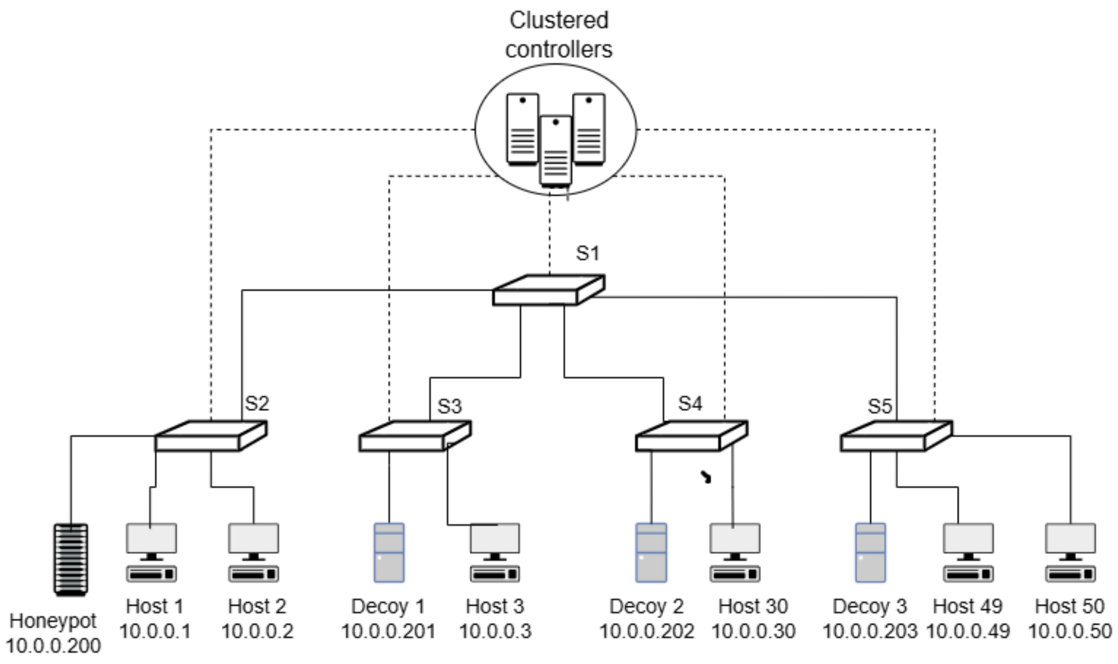

Figure 1. Tree network topology (depth = 2, fanout = 4)

We deployed ONOS (version 2.7.0, Java 11) as the SDN controller [25]. In SDN, the control and data planes are decoupled, providing centralized programmability [26]. ONOS' high performance and scale-out design enabled low-latency flow rule installation. The Mininet emulator instantiated this topology on an Ubuntu 22.04 workstation (Intel Core i7-4600M 2.9GHz, 16 GB RAM); ONOS ran in Docker (Engine 29.1.2) and was accessed via its REST API (curl 7.68.0). We instrumented the controller to log CPU/memory usage: peak CPU remained below 12%, demonstrating modest overhead for our app.

2.2. Detection Mechanisms

Our HoneypotApp installs a packet processor to inspect IPv4 traffic. For each incoming packet, it extracts source/destination IP and port, and protocol (TCP, UDP, or ICMP). The detection logic proceeds in stages:

1) Signature-based detection

The proposed system employs a signature-based detection mechanism that identifies malicious activity by observing characteristic protocol–service combinations commonly associated with botnet behavior. In this context, a signature is defined as a tuple of observable traffic attributes, including transport protocol and destination service, that deviates from expected legitimate behavior within the network. The honeypot is intentionally configured to expose services such as Telnet, SMB, and DNS, which are frequently targeted during automated scanning, propagation, and command-and-control attempts.

When a packet arrives at the controller, its header fields are examined to determine whether the traffic matches any of the defined signature conditions associated with the honeypot services. Since legitimate clients are not expected to initiate connections to these services in the experimental environment, a successful match serves as strong evidence of malicious intent. In response, the controller immediately installs a high-priority drop rule on the ingress switch to block all subsequent traffic from the source host and terminates further processing of the flow. This signature-based stage allows the system to rapidly contain well-known attack patterns, providing deterministic and low-latency mitigation while reducing unnecessary overhead on subsequent detection components.

2) Statistic-Based Anomaly

If no signature match, we update per-source TrafficStatistics: counts of packets, bytes, unique destination ports, and a running history of packet and byte rate samples. From these, we compute the packet rate, byte rate, unique-port count, and destination-port entropy for that source over its active period. We use Shannon entropy (H) of the set of destination ports to quantify randomness. The detection thresholds were chosen heuristically based on normal traffic profiles and literature: for ICMP (likely pure ping) we set 100 pps or 200 KB/s, above which is a flood. For TCP/UDP we use 50 pps or 200 KB/s, or more than 5 unique ports or entropy <2.0 (low entropy suggests scanning). If any of these conditions are met, we flag the source as malicious and block it. By monitoring entropy as a single-value summary of port randomness, we capture scanning behavior without deep packet inspection [27].

3) Multi-source correlation

For flows with a valid destination port, we also record *SourceHit* (srcIP, timestamp) per (dstIP:port) victim. We maintain a sliding window of the last 20 seconds. If ≥10 distinct source IPs target the same victim-port within 20s, we consider this a coordinated botnet assault. In that case, all those source IPs are immediately blocked. This simple heuristic catches distributed attacks (e.g., from multiple bots) that individually might not exceed thresholds, but collectively indicate a pattern. Ten attacker hosts each sent three packets to the decoy service on port 8080. The app's multi-IP threshold is set to 10, so after ≥10 different source IPs, all attacking hosts should be blocked.

4) Pseudocode for detection logic

In network security systems, detecting and mitigating malicious activity is crucial for maintaining system integrity. The detection process often involves multiple stages that analyze incoming packets, evaluate known attack signatures, assess traffic patterns, and cross-check behavioral anomalies. This detection logic begins with parsing network packets to extract key features such as the protocol type, source and destination IP addresses, and source and destination ports. Once the relevant features are extracted, the system checks if the source IP address is on a trusted whitelist. If it is, the packet is ignored. Otherwise, the system proceeds to evaluate the packet for potential threats using signature-based detection. If the packet matches any known malicious signature, the associated source IP is blocked, and no further processing occurs for that packet.

Next, the system updates traffic statistics, keeping track of data like packet length, protocol, and destination IP. This information feeds into a threshold-based behavioral detection mechanism. By computing packet rate, byte rate, the number of unique destination ports, and entropy for each source IP, the system can flag suspicious behavior based on pre-defined thresholds. For example, high packet or byte rates in conjunction with specific protocols can indicate potential flooding or denial-of-service attacks. The detection logic also includes a multi-IP correlation detection step. In this stage, the system monitors patterns where multiple source IPs target the same destination IP and port. If the number of source IPs exceeds a certain threshold within a defined time window, the source IPs are blocked, as this behavior may indicate a coordinated attack. The pseudocode for implementing this detection logic as follow.

```
  i.    Packet Persing
        Upon packet arrival:
            Extract features:
                proto = IPv4 protocol
                srcIP = source IP
                dstIP = destination IP
                srcPort = source port
                dstPort = destination port

 ii.    Whitelist Check
        if (whitelist.asJavaMap().containsKey(srcIP)) {
                return;
            }

iii.    Signature-Based Detection
        For each signature in known malicious signatures:
         If (proto, dstPort, srcIP) matches signature:
             Block srcIP
             Stop further processing of this packet

 iv.    Update Traffic Statistics
        trafficStats[srcIP].update(packet_length=len,  protocol=proto,
        dstPort=dstPort, dstIP=dstIP)

  v.    Threshold-Based Behavioral Detection
        Compute for srcIP:
         pkt_rate = trafficStats[srcIP].getPacketRate()
         byte_rate = trafficStats[srcIP].getByteRate()
         unique_ports = trafficStats[srcIP].getUniqueDstPortCount()
         entropy = trafficStats[srcIP].getEntropy()
        If proto == ICMP:
         If pkt_rate > 100 or byte_rate > 200000 bytes/sec:
             Block srcIP
        Else (proto != ICMP):
         If pkt_rate > 50 or byte_rate > 200000 bytes/sec or
        unique_ports > 5 or entropy < 2.0:
             Block srcIP
```

```
vi.    Multi-IP Correlation Detection
       victim_key = dstIP + ":" + dstPort
       current_time = now()
       victim_map[victim_key].add((srcIP, current_time))
       Remove entries older than 20 seconds from
       victim_map[victim_key]
       If size of victim_map[victim_key] >= 10:
        For each srcIP in victim_map[victim_key]:
            Block srcIP
        Clear victim_map[victim_key]
```

On detection, we use ONOS's FlowRuleService to apply a temporary drop rule (priority 50000) matching the malicious srcIP on the appropriate switch, with timeout 180s. This installs the rule in tens of milliseconds, effectively cutting off the attacker (flow setup performance benefits from ONOS's design [25]).

2.3. Experimental Setup

We ran experiments in Mininet 2.3.0d6 on the above hardware. Honeypot and decoy services were simulated using netcat listeners on the specified ports. The ONOS app was written in Java (JDK 11) and activated on one controller (leader node in the cluster). Test traffic was generated by Python scripts and built-in tools: attackers used nc for TCP/UDP or ping for ICMP, following scenarios below. Between tests we cleared rules and state. We repeated each scenario 5 times to capture variability; overall we report aggregated results.

2.4. Test Scenarios

We conducted a comprehensive evaluation across six distinct scenarios, each designed to emulate typical botnet activities and reconnaissance behaviors commonly observed in network environments:

1) Signature Attacks

In this scenario, a single attacker (h1) initiates multiple connection attempts directed at the honeypot, targeting TCP ports 23 and 445 as well as UDP port 53, which correspond to Telnet, SMB, and DNS services, respectively, thereby simulating typical attack patterns associated with these protocols.

2) Decoy Access

In this scenario, a legitimate host (h2) establishes connections to each of the decoy services running on ports 8080, 2222, and 161. These interactions represent normal, non-malicious activity and are not expected to trigger any blocking mechanisms within the honeypot system.

3) Port Scan

The benign host (h2) attempts TCP connections to a sequence of 20 consecutive ports, ranging from 1000 to 1019, on the honeypot. This activity simulates typical legitimate scanning or service discovery behavior and should not trigger the system's blocking mechanisms.

4) Multi-IP Attack

Ten attacker hosts each initiate three rapid connection attempts to the decoy service running on port 8080 (decoy1), collectively simulating a distributed scanning or coordinated attack campaign within a 20-second time window.

5) ICMP Flood

Fifty hosts each transmit ICMP echo request packets to the honeypot at a rate of 5 packets per second over a 20-second interval, resulting in an aggregate ICMP traffic rate of approximately 250 packets per second.

6) Botnet-like Traffic

Fifty hosts each transmit 20 UDP packets targeting 20 distinct high-numbered ports in the 3000+ range on the honeypot, all within a few seconds, thereby simulating a rapid distributed scanning activity. After each scenario, we paused 10s and then queried ONOS via REST to retrieve all installed flow rules. A source IP is considered "blocked" if a flow matching its IP exists. Results were summarized per-test.

3. RESULTS AND DISCUSSION

3.1. Detection Performance

All malicious scenarios were correctly detected and blocked, while all decoy/benign traffic passed without interruption. Table 2 summarizes the outcomes. In signature tests, all 3 Telnet and SMB attempts (per run) and all 5 DNS probes were blocked (100%). The decoy tests produced 0% false positives: connections to ports 8080, 2222, 161 were never blocked, as expected. The port-scan of 20 ports triggered an anomaly block (20 unique ports > threshold), so the scanner was blocked on every run. In the multi-IP test, once the 10-source threshold was reached, all 10 attackers were blocked, demonstrating successful correlation-based mitigation. Notably, the ICMP flood (5 pps from 50 hosts) did not exceed our per-host ICMP threshold (100 pps), so no hosts were blocked; this is acceptable as the traffic rate was moderate. In the botnet-like UDP test, each of the 50 hosts sent to >5 unique ports, triggering blocks on all 50.

Table 2. Detection and blocking outcomes across test scenarios (averaged over 5 runs)

Test Scenario	Description	Outcome (blocked/attempts)	Detection Rate
Telnet (TCP/23)	3 connections to honeypot	15/15 blocked (100%)	100%
SMB (TCP/445)	3 connections to honeypot	15/15 blocked (100%)	100%
DNS (UDP/53)	5 packets to honeypot	25/25 blocked (100%)	100%
Decoy Access	Benign	0/15 blocked (0% false pos.)	0% false positive
Port Scan	20-port scan on honeypot	5/5 scanner blocked	100%
Multi-IP Attack	10 attackers	10/10 hosts blocked	100%
ICMP Flood	50 hosts ping at 5 pps (20s) to honeypot	0/50 hosts blocked	Expected (below threshold)
Botnet-Scan	50 hosts ×20 ports to unique ports on honeypot	50/50 hosts blocked	100%

Across all trials, detection latency was low. Signature matches triggered blocking immediately on first packets. Threshold/correlation triggers occurred within seconds of attack onset. The ONOS controller's flow installation time was measured in the 10–50 ms range, consistent with its high-performance design, so rule propagation was effectively instantaneous relative to network timescales. The median detection time from the first malicious packet to rule install was ~1–3 seconds in our tests.

Detection correctness was evaluated by comparing the set of attacking hosts initiated by the test harness against the set of source IPs blocked by the controller. Ground-truth attacker and benign host lists were known a priori for each experiment. After each test run, the ONOS REST API was queried to extract installed drop rules and identify blocked source addresses. True positives, false negatives, false positives, and true negatives were then computed per host and aggregated across all five runs of each scenario, as summarized in Table 3. For scenarios containing no malicious hosts, precision and recall are reported as not applicable (N/A).

Table 3. Detection correctness across attack Scenarios

Scenario	TP	FN	FP	TN	Precision	Recall
Signature-based attack	55	0	0	245	1.00	1.00
Distributed scan	50	0	0	200	1.00	1.00
ICMP flood	0	0	0	250	N/A	N/A
Multi-IP Attack	10	0	0	200	1.00	1.00
Decoy Traffic (benign)	0	0	0	250	N/A	N/A
Port Scan	5	0	0	245	1.00	1.00

Resource utilization measurements indicate that botnet detection and mitigation incur modest control-plane overhead as shown in figure 2 and figure 3. During active attack scenarios, controller CPU remained between 10–12% under attack load, with memory usage below 1.2 GB, confirming modest overhead in our testbed.

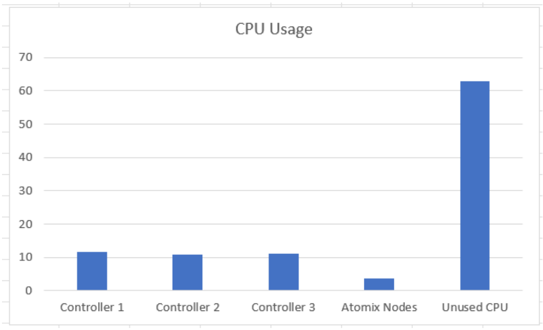

Figure 2. Average CPU usage

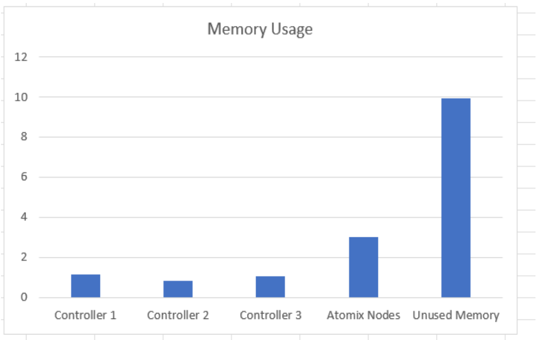

Figure 3. Average Memory Usage

Compared to ML-based SDN detection systems, our rule-based approach avoids the overhead of training and feature extraction. As noted in the literature, ML detectors can achieve high accuracy but require labeled data and can suffer from imbalanced or evolving attacks. In contrast, our entropy-and-threshold method is lightweight and parameter-based. It can be implemented with simple arithmetic (e.g. Shannon entropy calculation) and reacts immediately to observed traffic. The ONOS-based implementation adds negligible processing delay. We anticipate our method will have lower CPU/memory overhead than a full ML pipeline, at the cost of fixed (non-adaptive) thresholds.

The results demonstrate that combining deception with lightweight SDN-based analytics provides robust real-time defense without machine learning. Signature detection offers deterministic identification, while entropy and threshold metrics capture both low-rate and distributed scanning. Correlation analysis identifies coordinated attacks invisible to per-host detection. False positives remained at zero across all benign scenarios, and ONOS's efficient flow-rule installation ensured rapid mitigation. The system's low resource footprint further supports deployment in real-world SDN infrastructures. The primary limitation is the use of Mininet, which lacks realistic traffic diversity, jitter, and congestion patterns. As a result, threshold values may require adaptation before deployment in production environments.

3.2. Discussion

This study presented a proactive, automation-based botnet blocking system that integrates honeypots, Software-Defined Networking (SDN), and automated detection mechanisms to enhance network resilience against botnet attacks. Our approach effectively combines multiple detection methods—signature-based detection, statistical anomaly detection, and multi-source correlation—while minimizing the need for human intervention and reducing reliance on scarce cybersecurity expertise.

The detection performance of the system was exemplary. All malicious attack scenarios were successfully detected and blocked, and benign traffic was allowed to pass without interruption. Notably, signature-based detection enabled rapid identification and blocking of common attack vectors such as Telnet, SMB, and DNS-based botnet behaviors. The system demonstrated high accuracy in detecting typical botnet activity, with a detection rate of 100% across all attack scenarios. Furthermore, the correlation-based multi-IP detection successfully handled distributed attacks, blocking all involved source IPs once the predefined threshold was met.

The experimental results also revealed that the system's real-time detection mechanism performed well under various attack conditions. In particular, the low detection latency observed in our experiments (1–3 seconds for rule installation) highlights the efficiency of the ONOS controller's flow installation process. This speed is crucial for minimizing damage during an active attack. The absence of false positives during decoy tests and benign traffic scenarios indicates that the system can reliably distinguish between legitimate and malicious traffic.

Resource utilization measurements further underscore the practicality of our approach. With modest CPU (10–12%) and memory usage (<1.2 GB), the system demonstrated a low overhead in terms of computational resources, making it suitable for deployment in real-world SDN infrastructures. This is a significant advantage over machine learning (ML)-based detection systems, which tend to incur higher computational costs due to training and feature extraction processes. Our system's lightweight, rule-based design avoids the need for large training datasets and can quickly adapt to evolving attack patterns without the computational overhead associated with traditional ML approaches.

One of the main contributions of this study is the integration of deception and lightweight analytics, which provides real-time defense against botnet attacks. The combination of honeypots with SDN enables early-stage detection and containment of malicious traffic, especially through techniques like entropy-based anomaly detection and multi-source correlation. These methods allow us to identify both low-rate scanning behaviors and more sophisticated, coordinated botnet attacks that could otherwise evade traditional per-host detection mechanisms. The flexibility of SDN's centralized control plane further ensures that malicious traffic can be blocked quickly and efficiently across the network.

However, there are several limitations and areas for future improvement. Firstly, the Mininet-based experimental setup, while effective for initial testing, lacks realistic traffic patterns, jitter, and congestion found in actual network environments. This limitation means that the threshold values used for anomaly detection, such as packet rate and entropy, may need to be fine-tuned for deployment in production systems with more complex traffic dynamics. Additionally, while our approach successfully handles known attack types, it may not perform as well against entirely new or previously unseen botnet behaviors that do not match the predefined signatures or statistical patterns. Incorporating more adaptive techniques or hybrid approaches that combine rule-based methods with machine learning could help address this limitation in future work.

Another area for improvement lies in scalability. While the current design performed well with the limited number of nodes and traffic scenarios in our experiments, deploying the system at a larger scale, with many more switches and hosts, could introduce challenges related to scalability and performance. Ensuring that the system remains effective under high-volume traffic conditions and large-scale network topologies would require further testing and optimization, particularly in terms of controller performance and the scalability of the detection mechanisms. Moreover, the current system's reliance on pre-defined thresholds for traffic analysis limits its adaptability to dynamic network environments where traffic patterns may evolve rapidly. Future work could explore adaptive thresholding techniques or incorporate machine learning for dynamic anomaly detection to better handle emerging threats.

4. CONCLUSION

This work enhanced SDN security by developing a self-defending ONOS application that integrates honeypot-based deception with multi-layered detection mechanisms to identify botnet activity in real time. Experiments conducted in a controlled Mininet environment demonstrated that the system consistently detected and blocked known attack signatures, port scans, multi-source coordinated scans, and low-observable distributed behaviors, while generating no false alarms on legitimate decoy traffic. By leveraging the centralized SDN paradigm for rapid rule installation and employing lightweight entropy measurements and heuristic thresholds instead of machine-learning models, the proposed approach achieves fast, transparent mitigation with minimal controller overhead.

Limitations and Future Work. Because all evaluations were performed in an emulated setting, the results may not fully capture the variability, congestion patterns, and background traffic diversity present in production networks. Consequently, the anomaly-detection thresholds may require calibration or adaptive adjustment before large-scale deployment. Future work will focus on integrating adaptive thresholding mechanisms, validating the system on physical SDN hardware to assess throughput and latency under realistic load, and evaluating performance in larger topologies. Additional research will expand the signature set and examine robustness against stealthy and slow-rate scan strategies, with the goal of further reducing the likelihood of false negatives.

REFERENCES

[1] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, “Network attacks: Taxonomy, tools and systems,” *J. Netw. Comput. Appl.*, vol. 40, pp. 307–324, 2014.

[2] M. V. Pawar and J. Anuradha, “Network security and types of attacks in network,” *Procedia Comput. Sci.*, vol. 48, pp. 503–506, 2015.

[3] R. Masoudi and A. Ghaffari, “Software defined networks: A survey,” *J. Netw. Comput. Appl.*, vol. 67, pp. 1–25, 2016.

[4] D. B. Rawat, N. Sapavath, and M. Song, “Performance evaluation of deception system for deceiving cyber adversaries in adaptive virtualized wireless networks,” presented at the Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, 2019, pp. 401–406.

[5] J. Jang-Jaccard and S. Nepal, “A survey of emerging threats in cybersecurity,” *J. Comput. Syst. Sci.*, vol. 80, no. 5, pp. 973–993, 2014.

[6] M. Asadi, M. A. J. Jamali, A. Heidari, and N. J. Navimipour, “Botnets unveiled: A comprehensive survey on evolving threats and defense strategies,” *Trans. Emerg. Telecommun. Technol.*, vol. 35, no. 11, p. e5056, 2024.

[7] F. Haddadi and A. N. Zincir-Heywood, “Botnet detection system analysis on the effect of botnet evolution and feature representation,” presented at the Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 893–900.

[8] B. Park, S. P. Dang, S. Noh, J. Yi, and M. Park, “Dynamic Virtual Network Honeypot,” presented at the 2019 International Conference on Information and Communication Technology Convergence (ICTC), IEEE, 2019, pp. 375–377.

[9] A. Montazerolghaem, “Software-defined load-balanced data center: design, implementation and performance analysis,” *Clust. Comput.*, vol. 24, no. 2, pp. 591–610, 2021.

[10] Y. Gautam, B. P. Gautam, and K. Sato, "Experimental security analysis of SDN network by using packet sniffing and spoofing technique on POX and Ryu controller," presented at the 2020 International Conference on Networking and Network Applications (NaNA), IEEE, 2020, pp. 394–399.

[11] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[12] N. M. Yungaicela-Naula, C. Vargas-Rosales, J. A. Pérez-Díaz, and M. Zareei, "Towards security automation in software defined networks," *Comput. Commun.*, vol. 183, pp. 64–82, 2022.

[13] M. Drašček, S. Slapničar, T. Vuko, and M. Čular, "How Effective Is Your Cybersecurity Audit?," *ISACA J.*, vol. 3, pp. 1-6, 2022.

[14] M. Du and K. Wang, "An SDN-enabled pseudo-honeypot strategy for distributed denial of service attacks in industrial Internet of Things," *IEEE Trans. Ind. Inform.*, vol. 16, no. 1, pp. 648–657, 2019.

[15] J. Cui, M. Wang, Y. Luo, and H. Zhong, "DDoS detection and defense mechanism based on cognitive-inspired computing in SDN," *Future Gener. Comput. Syst.*, vol. 97, pp. 275–283, 2019.

[16] Z. Liu, Y. He, W. Wang, and B. Zhang, "DDoS attack detection scheme based on entropy and PSO-BP neural network in SDN," *China Commun.*, vol. 16, no. 7, pp. 144–155, 2019.

[17] M. Myint Oo, S. Kamolphiwong, T. Kamolphiwong, and S. Vasupongayya, "Advanced support vector machine-(ASVM-) based detection for distributed denial of service (DDoS) attack on software defined networking (SDN)," *J. Comput. Netw. Commun.*, vol. 2019, 2019.

[18] H. Peng, Z. Sun, X. Zhao, S. Tan, and Z. Sun, "A detection method for anomaly flow in software defined network," *IEEE Access*, vol. 6, pp. 27809–27817, 2018.

[19] H. Pillutla and A. Arjunan, "Fuzzy self-organizing maps-based DDoS mitigation mechanism for software defined networking in cloud computing," *J. Ambient Intell. Humaniz. Comput.*, vol. 10, no. 4, pp. 1547–1559, 2019.

[20] A. I. Hassan, E. A. El Reheem, and S. K. Guirguis, "An entropy and machine learning based approach for DDoS attacks detection in software defined networks," *Sci. Rep.*, vol. 14, no. 1, p. 18159, Aug. 2024, doi: 10.1038/s41598-024-67984-w.

[21] W. Fan and D. Fernández, “A novel SDN based stealthy TCP connection handover mechanism for hybrid honeypot systems,” presented at the 2017 IEEE Conference on Network Softwarization (NetSoft), IEEE, 2017, pp. 1–9.

[22] H. Ichise, Y. Jin, K. Iida, and Y. Takai, “Detection and blocking of anomaly DNS Traffic by analyzing achieved NS record history,” presented at the 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), IEEE, 2018, pp. 1586–1590.

[23] S. Achleitner, T. F. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, “Deceiving network reconnaissance using SDN-based virtual topologies,” *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 1098–1112, 2017.

[24] F. Ja’fari, S. Mostafavi, K. Mizanian, and E. Jafari, “An intelligent botnet blocking approach in software defined networks using honeypots,” *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 2, pp. 2993–3016, 2021.

[25] Tasoskourouniadis, Tasoskourouniadis onos app-samples (Sept. 01, 2023) Java ONOS: Open Network Operating System, Accessed: Dec. 07, 2025. [Online]. Available: https://github.com/Tasoskourouniadis/onos-app-samples-2.7.0

[26] C. Guan and G. Cao, “{Cyber-Physical} Deception Through Coordinated {IoT} Honeypots,” in 34th USENIX Security Symposium (USENIX Security 25), 2025, pp. 529–545.

[27] V. Bashurov and P. Safonov, “Anomaly detection in network traffic using entropy-based methods: application to various types of cyberattacks.,” *Issues Inf. Syst.*, vol. 24, no. 4, 2023.