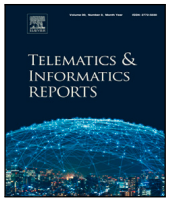
Telematics and Informatics Reports 14 (2024) 100129

Contents lists available at ScienceDirect

Telematics and Informatics Reports

journal homepage: www.elsevier.com/locate/teler



Enhancing security in Software-Defined Networks: An approach to efficient ARP spoofing attacks detection and mitigation

Vanlalruata Hnamte *, Jamal Hussain

Department of Mathematics and Computer Science, Mizoram University, Tanhril, Aizawl, 796004, Mizoram, India

ARTICLE INFO

Keywords:
Software-Defined Networking
ARP spoofing
Security
Anomaly detection
Network resilience

ABSTRACT

The proliferation of Software-Defined Networks (SDNs) has introduced unparalleled flexibility and efficiency to network management, but at the same time, it has introduced new challenges in securing network infrastructures. Among these challenges, Address Resolution Protocol (ARP) spoofing attacks remain a pervasive threat, compromising network integrity and data confidentiality. In this manuscript, we present an approach to ARP spoofing mitigation within SDNs, addressing the limitations of existing methodologies. Our proposed solution employs a multifaceted strategy that combines dynamic ARP cache management, real-time traffic analysis, and adaptive flow rule orchestration. Central to our approach is a dedicated device that continuously monitors the network topology and detects any deviations from established norms. Notably, our solution adapts seamlessly to networks of varying sizes, ensuring scalability and efficacy across diverse infrastructures. One of our key contributions is the integration of a deep learning-based Deep Neural Network (DNN) model to detect and mitigate ARP spoofing attacks. Leveraging a self-generated ARP spoofing dataset from SDN environments, our model demonstrates exceptional accuracy and adaptability, enhancing the network's capability to identify and counter such threats effectively. Our approach showcases exceptional reliability, achieving 100% accuracy rate in detection of ARP spoofing, which is crucial for sustaining network responsiveness.

1. Introduction

In recent years, the escalation of cyber attacks has become a pressing concern. Our modern lives are increasingly intertwined with technology and the internet, making us more vulnerable to malicious cyber activities. Cyber attacks encompass a wide range of intentional and malevolent efforts aimed at exploiting or disrupting computer systems, networks, and devices. These nefarious actions range from stealing sensitive information and personal data to sabotaging critical infrastructure and inflicting substantial financial losses. Cyber attackers employ a multitude of tactics, including malware dissemination, phishing schemes, social engineering ploys, and denial-of-service attacks, among others.

The scope of these threats is vast, targeting both individual devices and entire networks. The telecommunications sector, in particular, has witnessed remarkable advances, with computer networking serving as the linchpin of our extensive and interconnected communication ecosystem. However, the sharing of resources in such intricate networks inherently exposes them to vulnerabilities, ranging from resource misuse to degradation and, in severe cases, service denial. Conventional computer networks are no strangers to attacks like distributed denial of service (DDoS), packet spoofing, and link flooding [1]. In the pursuit of safeguarding network data, numerous challenges must be surmounted, including the ability to inspect voluminous packet streams without impeding traffic, detect malevolent intrusions promptly, and efficiently respond to emerging threats. Given the exponential growth in network scale, the demand for effective and scalable solutions in the realm of malware detection and response systems is more pressing than ever [2].

SDN emerges as a groundbreaking network architecture that is quickly becoming the standard for network function virtualization. By decoupling the data plane from the control plane, SDN enables the agile development and deployment of new network applications, streamlining global network configuration and simplifying network management [3]. The centralized control plane, governed by one or more controllers, enhances network performance and enables precise network monitoring [4]. SDN applications send requests to these controllers, which, in turn, relay instructions to networked devices. The prospect of revolutionizing the rigidity of conventional network models is what makes SDN so intriguing. It transforms traditional network paradigms into a more accessible, programmable, reliable, secure, and controllable framework. This approach also simplifies

* Corresponding author.
E-mail addresses: vanlalruata.hnamte@gmail.com (V. Hnamte), jamal.mzu@gmail.com (J. Hussain).

https://doi.org/10.1016/j.teler.2024.100129
Received 21 September 2023; Received in revised form 4 February 2024; Accepted 2 March 2024
Available online 6 March 2024

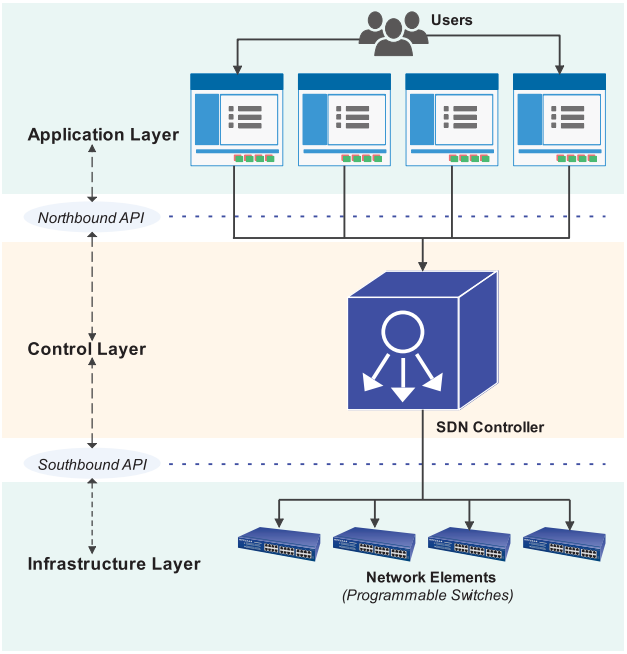


Fig. 1. SDN architecture.

intrusion detection, the process of identifying hacking or other malicious activities on a network. Although intrusion prevention adjusts network resources or configurations to counteract threats, it remains vulnerable to attacks.

The SDN architecture encompasses three foundational layers, delineated in Fig. 1. The lowest tier is the Infrastructure Plane, comprising essential network elements such as switches and routers. These components are responsible for the vital tasks of data forwarding and manipulation within the network infrastructure.

Above the Data Plane sits the Control Plane, which is both centralized and superimposed over the Data Plane. This layer is governed by a controller, a pivotal component that computes and disseminates intricate network policies to the interconnected devices.

The uppermost stratum is the Application Plane, wherein specific applications like load balancing, network management, and policy enforcement find their domain. These applications serve diverse functions vital to the network's operation and efficiency.

The Data Forwarding Plane, formed by SDN switches interconnected through wired or wireless networks, is the practical manifestation of the network's functionality. Each switch within this plane possesses a forwarding table, an essential repository of rules dictating the routing trajectory of data packets within the network.

At the Control Layer, overseen by the SDN Controller, meticulous scrutiny of the network transpires at the granular level of packet switching. Concurrently, the Application Layer interfaces with the Control Layer through a northbound API, ensuring seamless communication and integration of various network applications. This layered architecture, as illustrated in Fig. 1, forms the backbone of SDN, offering a systematic framework for network orchestration and management.

Although, the SDN architecture stands as a robust framework enhancing network security and streamlining network administration, it does not eradicate various forms of spoofing attacks. Among these, Distributed Denial of Service (DDoS) and Man-in-the-Middle (MitM) attacks persist as formidable threats, potentially compromising sensitive user data within the network. Notably, one of the most prevalent intrusions within Local Area Networks (LANs) is ARP spoofing.

ARP functions on Layer-2, serving to determine the physical (MAC) address corresponding to a specific IP address before initiating communication. ARP spoofing transpires when a malicious actor manipulates the ARP tables of other hosts within the network, assuming the disguise of a legitimate host. This subversion is orchestrated through the transmission of counterfeit ARP packets, deceiving other hosts into associating the attacker's MAC address with a genuine IP address. The request for an ARP may include ARP spoofing, and the request flow is depicted in Fig. 2.

The pervasive threat of ARP spoofing poses a significant risk to the security of computer networks, leading to potential eavesdropping, tampering, and disruption of network traffic. Detecting ARP attacks is challenging, and their consequences can be severe, including data theft and network vulnerabilities. In the realm of SDN, the risk expands to address spoofing and intrusion attempts targeting the SDN controller. This vulnerability, coupled with the frailties of ARP protocols, opens avenues for malicious actors to exploit network topology data, resulting in various security lapses. This study examines the impact of ARP attacks on SDN networks, exploring normal and attack scenarios to unveil potential security breaches and their ramifications.

To further advance the understanding and mitigation of ARP spoofing attacks in SDN environments, our study conducted the generation of an SDN-ARP Dataset directly from the SDN architecture. Leveraging this dataset, we employed cutting-edge techniques, including the training of a deep learning model. This model serves as a dynamic and efficient tool for the detection and mitigation of ARP spoofing attacks within the SDN framework. The utilization of our custom SDN-ARP Dataset ensures the relevance and realism of our experimental scenarios, providing valuable insights into the performance and adaptability of the proposed mitigation strategies.

In response to this security concern, a sophisticated mechanism has been meticulously crafted to scrutinize and validate the messages transmitted via the ARP. The principal aim of this mechanism is to discern and thwart potential ARP spoofing attacks effectively. Through the meticulous validation of ARP protocol messages, this mechanism adeptly identifies and obstructs the efforts of malicious entities attempting to disseminate

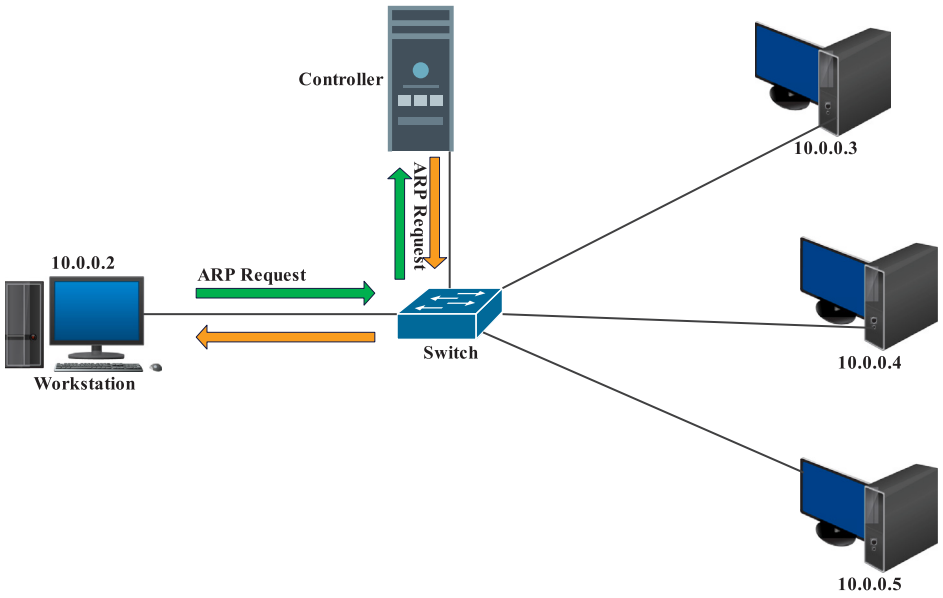


Fig. 2. ARP packet transfer.

counterfeit ARP packets. These deceptive packets, if successful, could jeopardize the integrity of the ARP tables residing within other hosts across the network.

Motivated by these challenges and the need for robust solutions, this paper delves into a comprehensive study of ARP spoofing and proposes innovative strategies for protection within the SDN environment. Our study presents the following key contributions:

1. **Innovative ARP Spoofing Scheme**: We propose an innovative and efficient deep learning based ARP spoofing detection and mitigation scheme that enhances the network's capability to detect and counter such attacks effectively.
2. **Improved Network Performance**: Our proposed system demonstrates improved network throughput and CPU utilization during and after ARP spoofing attacks, all while uncovering potentially hazardous situations through detailed packet analysis.
3. **Comprehensive Evaluation**: The performance of the proposed system is rigorously evaluated across various network sizes and traffic conditions, providing insight into its effectiveness and scalability.

The subsequent sections of this paper are organized as follows: Section 2 presents an extensive review of contemporary studies concerning ARP spoofing. In Section 3, the proposed methodology is comprehensively expounded, detailing the innovative approach undertaken to mitigate ARP spoofing incidents within SDN environments. The experimental setup, along with the results and subsequent discussions, is meticulously presented in Section 4. Finally, Section 5 encapsulates a thorough discussion, conclusions drawn from the study's findings, and outlines potential avenues for future research in this domain.

2. Recent studies

Recent studies have made substantial progress in countering the persistent threat of ARP spoofing attacks within SDNs. ARP spoofing, a malicious technique wherein attackers link their device's MAC address with a legitimate device's IP address, poses significant security challenges. In this discussion, we will scrutinize these studies, analyzing their methodologies, contributions, strengths, and limitations.

In their research, Girdler and Vassilakis [5] took a focused stance on countering ARP spoofing attacks within SDNs. Their primary technique was an Intrusion Detection and Prevention System (IDPS) harnessing SDN technology. This IDPS dynamically adapts SDN settings to identify and thwart suspicious network activities while blacklisting malicious MAC addresses. To personalize and evaluate the IDPS's efficacy, specialized software was designed, integrated with a dedicated library for user input validation. The study emphasizes SDN's advancements in attack detection, firewall and intrusion prevention, packet management, and reduced timeout settings. The authors conducted extensive experiments to validate their solution, demonstrating its rapid response to intrusion attempts. There is limited focus on specific attack scenarios and evaluation primarily based on simulations, potentially lacking real-world complexity.

The comprehensive study conducted by AbdelSalam et al. [6] aimed to combat both major types of ARP attacks in SDNs. The proposed solution extends the SDN controller's functionality by incorporating a dedicated ARP module. This module swiftly detects and mitigates attacks without overloading or causing Denial of Service (DoS) on the controller. The research meticulously examined the system's speed, reliability, and effectiveness across various attack scenarios. To facilitate communication between the controller and switches, the team leveraged OpenFlow, emulated through Mininet. Although highly promising, this study has one limitation: the proposed model was not tested across diverse network sizes. There may be potential challenge in real-time adaptation to rapidly evolving attack patterns.

Amin et al. [7] introduced an edge computing system designed to autonomously identify and counteract network intrusions, with a particular focus on ARP traffic. Utilizing a graph computation-based approach, this system precisely pinpoints attackers or intruders and revokes network access while permitting authorized users to continue. This bolsters the system's performance in areas such as attack detection, mitigation, and bandwidth optimization. There are still some limitations such as, the complexity of graph-based computation might impact real-time response. The study has limited discussion on false positive/negative rates in intrusion detection.

Khalid et al. [8] proposed a lightweight, reliable, and swift approach to thwart ARP spoofing through SDN features. They integrated a module into the SDN controller that scrutinizes each ARP packet within the network to combat spoofing attempts. This technique proved its resilience against ARP spoofing attacks in simulations conducted using Mininet. There is scalability concerns, as it is unexplored in their study, potentially may be affecting performance in larger networks.

Lin et al. [9] embarked on a journey to decipher the complexities of SYN flooding and ARP spoofing threats within SDNs. Their innovative solution revolves around a novel approach to combat SYN flooding by utilizing a minimal set of forwarding rules. Furthermore, they harnessed the power of Programming Protocol-independent Packet Processors (P4) to alleviate the controller's workload. While the proposed model exhibits great potential in thwarting these security threats, it is important to note that its effectiveness was not tested across a range of network sizes. Also, there may be potential challenges in large-scale implementation due to minimal rule sets.

Saritakumar et al. [10] introduced a groundbreaking algorithm that leverages SDN controllers to nullify the risk of ARP poisoning within LANs while preserving the standard ARP procedure. This ingenious technique bolsters network security by granting the controller the authority to determine whether to forward ARP packets based on IP-MAC address bindings and ARP response iteration counts. Their evaluation involved the use of Mininet as a network emulator and the Dsniff tool to simulate network attacks. By effectively countering ARP poisoning attacks, this algorithm enhances network security and exhibits practicality in real-world network environments. There may be potential complexity in managing ARP response iteration counts in dynamic networks.

Jitta Sai Meghana et al. [11] conducted a comprehensive survey that scrutinized existing solutions for detecting and mitigating ARP spoofing attacks in both traditional and SDN settings. Their meticulous evaluation led to a significant finding: SDN-based solutions outperform traditional approaches in identifying and neutralizing various ARP spoofing attacks. While this survey does not introduce novel concepts beyond existing literature, it serves as a valuable repository of insights into the current landscape of ARP spoofing threat mitigation. It underscores the urgency of addressing ARP spoofing risks in both conventional and SDN networks. The study lacked novel contributions beyond existing literature, and focused on comparative analysis, potentially missing in-depth exploration of specific solutions.

Aldabbas and Amin [12] proposed a pioneering mechanism designed to combat ARP spoofing. Their system operates through a dedicated machine that collaborates with the SDN controller to gather network topology information and ARP queries. The crux of this approach lies in redirecting ARP traffic to the dedicated machine, where specialized techniques analyze the data. Simulations unveiled promising results, showcasing significant improvements in network throughput and a remarkable 35% reduction in attack detection and mitigation time compared to existing methods.

Galal et al. [13] unveil a pioneering strategy for detecting and mitigating ARP poisoning, encapsulating a three-tiered module architecture. The first module acts as the gatekeeper, granting initial access while implementing rigorous security measures through MD5 hashing. The second module, a sentinel against internal ARP threats, fortifies network integrity. Simultaneously, the third module vigilantly tracks instances where a MAC address is associated with two IPs or an IP with two MACs, serving as an ARP watchdog. Their innovative framework incorporates a resilient database that adeptly stores ARP table information, adapting to the dynamic nature of network entries that often expire swiftly. To validate the prowess of their approach, extensive real-world network experiments were conducted, employing Ettercap to scrutinize their mechanism's functionality. The outcomes unequivocally affirm the effectiveness of their method, especially in identifying MAC-to-IP and IP-to-MAC anomalies. There is limited discussion on adaptability to rapidly changing network topologies, and potential challenges in maintaining accuracy with swiftly expiring ARP table entries.

In the context of SDN's dual planes – the control plane and data plane – Jamil et al. [14] proposed an ingenious auto-detection mechanism primed to unearth and safeguard SDN networks from ARP and DDoS attacks. Within this novel paradigm, they orchestrate two distinct algorithms, one dedicated to orchestrating flow rules and the other diligently sniffing out any nascent attacks. Augmenting the arsenal, a dedicated server stands sentinel, vigilantly monitoring the influx of potentially malevolent traffic. This auto-detection paradigm stands as a sentinel, galvanizing SDN network security by expeditiously recognizing and mitigating these attacks. The study has limited discussion on the system's adaptability to evolving DDoS attack patterns.

Xu et al. [15] offer a compelling duo of algorithms tailored for the detection of DDoS attacks within SDNs. Their first line of defense deploys the K-means++ algorithm, complemented by the Fast k Nearest Neighbour algorithm in the second method. The cornerstone of this approach is a modular detection system seamlessly integrated into the SDN controller. This vigilant controller periodically engages with switches to assess and identify network flows. Should an incoming flow bear the telltale signs of a DDoS attack, the controller promptly adjusts the flow table forwarding rules and dispatches notifications to the switch, orchestrating an agile response to the anomaly. There may be potential resource overhead due to periodic flow assessment, affecting network performance.

In a comprehensive study, Nguyen Huu Thanh et al. [16] delve deep into the ramifications of DDoS attacks on SDN architecture. Their meticulous evaluation included a benchmarking exercise involving prominent SDN controllers like POX, Ryu, and Floodlight. Stress tests were leveraged to scrutinize their influence on SDN switches and OpenFlow channels. Beyond this, their research unveiled new vulnerabilities and threats intrinsic to the SDN paradigm. It is worth noting that the study's scope was somewhat constrained, particularly in terms of CPU resource utilization exploration. There is limited discussion on mitigating strategies beyond identifying vulnerabilities.

Prasad et al. [17] introduce a groundbreaking two-pronged approach, seamlessly melding the powers of machine learning and device profiling to combat the menace of ARP spoofing-based Man-in-the-Middle (MitM) attacks. The machine learning facet serves as the vigilant guardian of network traffic, leveraging its analytical acumen to scrutinize for any anomalies that might betray the presence of a MitM attack. In real-world deployments, the device profiling module takes center stage, generating intricate device profiles that substantially elevate the accuracy of detection. This device profiling mechanism is facilitated by a client application responsible for the continuous monitoring of the ARP cache table, promptly identifying any compromise therein. Once an intrusion is detected, the profiler promptly dispatches a notification to a dedicated system, which swings into action by unmasking the intruder and promptly blacklisting their device from further network access. Crucially, this profiling system maintains a comprehensive DEV-PROFILE, serving as a vital reference for the machine learning module, thus elevating the accuracy of MitM attack detection. The symbiosis of machine learning and device profiling in this approach paints a promising picture for combating ARP spoofing-based MitM attacks. Rigorous experiments underscore the efficacy of this method, revealing its robustness in both detection and mitigation. There is limited discussion on the scalability of the device profiling mechanism in large-scale networks.

Table 1 presents a comparative analysis of recent studies on ARP Spoofing in SDNs, highlighting their respective strengths and limitations. Girdler and Vassilakis focus on an Intrusion Detection using SDN technology, showcasing dynamic adaptation of SDN settings and effective blacklisting of malicious MAC addresses.

Table 1
Comparison of Recent Studies on ARP Spoofing in SDNs.

Study	Approach	Strengths	Limitations	Relationship
Girdler and Vassilakis [5]	Intrusion Detection and Prevention System (IDPS) using SDN technology	Dynamic adaptation of SDN settings, Blacklisting malicious MAC addresses, Extensive experimental validation	Limited evaluation on diverse network sizes	Independent, Focused
AbdelSalam et al. [6]	Extension of SDN controller with dedicated ARP module	Swift attack detection and mitigation, Reliability across various attack scenarios	Not tested across diverse network sizes	Independent, Comprehensive
Amin et al. [7]	Graph computation-based approach for attack detection and bandwidth optimization	Precise attack identification, Autonomous network intrusion counteraction	Impact on real-time response	Independent, Innovative
Khalid et al. [8]	Integration of ARP packet scrutiny module into SDN controller	Lightweight, Reliable, Swift against ARP spoofing	Unexplored in larger networks	Independent, Effective
Lin et al. [9]	Novel approach utilizing minimal forwarding rules and Programming Protocol-independent Packet Processors (P4)	Effective SYN flooding combat, Reduced controller workload	Effectiveness not tested across diverse network sizes	Independent, Innovative
Saritakumar et al. [10]	Algorithm leveraging SDN controllers for ARP poisoning nullification	Enhanced network security, Practicality in real-world environments	Effectiveness not tested across diverse network sizes	Independent, Groundbreaking
Jitta Sai Meghana et al. [11]	Survey comparing ARP spoofing mitigation solutions in traditional and SDN settings	SDN-based solutions outperform traditional methods	Survey Only	Comparative, Informative
Aldabbas and Amin [12]	Mechanism redirecting ARP traffic to a dedicated machine for analysis	Improved network throughput, 35% reduction in attack detection time	Effectiveness not tested across diverse network sizes	Independent, Pioneering
Galal et al. [13]	Three-tiered module architecture with rigorous security measures	Effective in identifying MAC-to-IP and IP-to-MAC anomalies, Resilient database	No analysis on CPU usages	Independent, Pioneering
Jamil et al. [14]	Auto-detection mechanism employing flow rules orchestration algorithms	Swift recognition and mitigation of ARP and DDoS attacks	Limited DDoS attack patterns	Independent, Ingenious
Xu et al. [15]	K-means++ algorithm and Fast k Nearest Neighbour algorithm integrated into SDN controller	Agile response to DDoS attacks, Modular detection system	No analysis on CPU usages	Independent, Effective
Nguyen Huu Thanh et al. [16]	Benchmarking exercise with SDN controllers, stress tests	Unveiling vulnerabilities and threats intrinsic to SDN paradigm	No analysis on CPU usages	Independent, In-depth
Prasad et al. [17]	Machine learning and device profiling combination for ARP spoofing-based Man-in-the-Middle attacks	Elevated accuracy in detection and mitigation, Symbiotic approach	Limited discussion for large-scale networks.	Independent, Innovative

ARP spoofing attacks continue to loom as a substantial menace to network security, necessitating innovative solutions. SDN technology has emerged as a stalwart in this battle, enabling the development of dynamic intrusion detection and prevention systems. These systems adapt on the fly, continually adjusting their settings to detect and thwart any suspicious network activity while diligently maintaining a blacklist of pernicious MAC addresses. These recent studies underscore the continuous efforts to fortify SDNs against ARP spoofing attacks. Their innovative methodologies encompass adaptive IDPS, advanced ARP modules, graph computation-based detection, and lightweight, yet robust, SDN-integrated solutions. By addressing the security challenges posed by ARP attacks, these studies contribute significantly to enhancing the security and integrity of SDNs.

3. Methodology

In our methodology for detecting and mitigating ARP attacks within SDN networks, we present a comprehensive approach leveraging advanced techniques, including the use of a DNN model, to enhance network security. This method integrates various components to form a cohesive defense strategy against ARP spoofing attacks. The core methodology comprises four distinct elements:

- **Network Data Retrieval**: Initially, our system collects network traffic data from the entire SDN network. This data encompasses details about network devices and their interconnections. This comprehensive view of the network serves as a foundation for identifying any unusual or suspicious activities and potential attack vectors.
- **ARP Spoofing Detection Techniques**: Our proposed system employs cutting-edge techniques, including the utilization of a DNN model, to detect and mitigate ARP spoofing attacks effectively. The DNN model is trained to discern patterns indicative of ARP spoofing behavior, allowing for real-time tracking of MAC addresses associated with each network device. Additionally, our system cross-references ARP responses with a known cache of MAC address and IP pairings to identify discrepancies. Upon detecting an ARP spoofing attack, the system can take immediate action, such as isolating the offending device or alerting network administrators.

- **Comprehensive Traffic Analysis**: In response to detected threats, our system promptly alerts network administrators, enabling swift and informed decision-making. The system conducts comprehensive traffic analysis, distinguishing between normal network traffic and potentially malicious activities. Notably, network traffic data for both attack and normal scenarios are stored in CSV format, facilitating detailed analysis and aiding in future threat mitigation strategies.

3.1. ARP spoofing detection algorithm

We propose an ARP spoofing detection algorithm as depicted in algorithm 1 to identify and mitigate ARP attacks effectively in SDNs. This algorithm is designed to run on the SDN controller and switch nodes.

Algorithm 1 ARP Spoofing Detection Algorithm

```
Input: ARP Request Packet P
Output: Detected ARP Spoofing Attack
if Destination IP in P is in ARP Cache then
    if Corresponding MAC Address in ARP Cache is different from source MAC in P then
        Alert: Possible ARP Spoofing Attack
    end if
else
    Add Destination IP and MAC Address from P to ARP Cache
end if
if Source IP in P is in ARP Cache then
    if Corresponding MAC Address in ARP Cache is different from source MAC in P then
        Alert: Possible ARP Spoofing Attack
    end if
else
    Add Source IP and MAC Address from P to ARP Cache
end if
if Number of ARP Requests from Same IP Exceeds Threshold then
    Alert: Possible ARP Flooding Attack
end if
```

Let P be the ARP Request Packet, where $P = (Source\ IP, Source\ MAC, Destination\ IP, Destination\ MAC)$. Let ARP_Cache be the cache storing IP-MAC mappings, where $ARP\_Cache = \{(IP_1, MAC_1), (IP_2, MAC_2), \ldots\}$. Let $Threshold$ be the predefined threshold for the number of ARP requests from the same IP.

The ARP Spoofing Detection Algorithm 1 represents a sophisticated solution designed to fortify SDNs against the perils of ARP spoofing attacks. ARP spoofing, a nefarious tactic in which attackers forge associations between their device's MAC address and a legitimate device's IP address, poses a significant threat to network security. This algorithm, through its meticulous inspection of incoming ARP request packets P, engages in a proactive battle against these security breaches.

Algorithm 1 forms the basis for ARP attacks mitigation illustrated in Fig. 5, providing a foundational framework upon which the more detailed and specific actions are built. In Algorithm 1, the general logic and flow of how ARP packets are handled and evaluated are outlined. It sets the conditions under which ARP packets are processed and defines the actions to be taken based on the type of packet received (ARP, Broadcast, or other).

Fig. 5, on the other hand, expands on the actions specified in Algorithm 1, providing a more detailed breakdown of what happens when an ARP packet is received. It delves deeper into the evaluation process by specifying the steps involved in handling ARP packets, including increasing the count of ARP packets, comparing the count with a threshold, dropping the packet if necessary, and forwarding the packet to the Proposed System for evaluation. Fig. 5 also includes additional details such as evaluating ARP Request and Response packets separately, which are not explicitly mentioned in Algorithm 1.

In essence, Algorithm 1 provides the high-level structure and decision-making flow, while Fig. 5 refines this structure, specifying the exact steps and conditions for handling ARP packets. Fig. 5 is a more detailed and specific version of the handling process outlined in Algorithm 1.

At its core, the algorithm scrutinizes the source MAC and IP addresses within ARP request packets P by comparing them with entries residing in the ARP cache. Any inconsistencies or suspicious patterns detected during this comparison trigger an alert, serving as the first line of defense against potential ARP spoofing attacks. Additionally, the algorithm employs a vigilant approach by monitoring the frequency of ARP requests originating from the same IP address. This monitoring strategy is instrumental in detecting and mitigating potential ARP flooding attacks, where an overwhelming volume of ARP requests can disrupt network functionality.

Furthermore, our proposed system utilizes a DNN model, which is depicted in Fig. 3, to enhance ARP spoofing attack detection. The model consists of multiple layers of interconnected nodes, including input, hidden, and output layers, each performing specific functions crucial for accurate attack detection and mitigation. By training on self-generated ARP spoofing datasets from SDN environments, the DNN model learns to identify and mitigate ARP spoofing attacks effectively. This integration of DNN technology underscores our commitment to leveraging cutting-edge methods for bolstering network security in SDN environments.

The practical implementation of this innovative solution necessitates the deployment of a dedicated device endowed with the responsibility of gathering real-time network topology data. This data collection process encompasses information from both SDN controllers and switches, ensuring a comprehensive and up-to-date understanding of the network's configuration. The dedicated device operates in a continuous monitoring mode, staying alert to any alterations in the connections between switches and controllers. This real-time awareness of network changes is pivotal in responding promptly to dynamic network conditions.

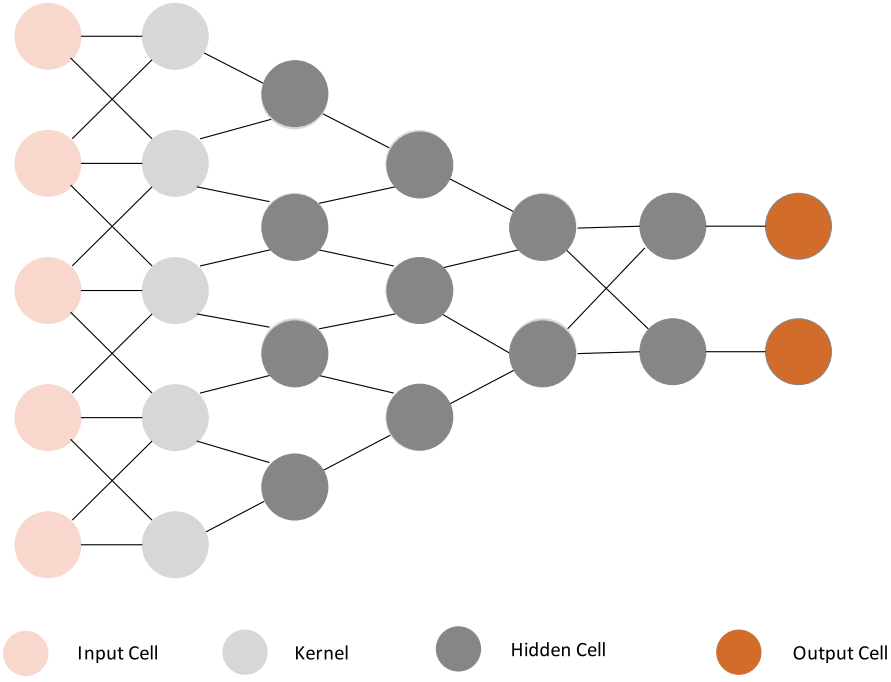


Fig. 3. Architecture of Deep Neural Network.

3.1.1. Dataset generation

```
from ryu.base import app_manager
from ryu.ofproto import ofproto_v1_3
from ryu.controller import ofp_event
from ryu.lib.packet import packet, ethernet, arp, ipv4, tcp, udp, ether_types
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import inet
import csv
import time

class ARP_IP_Handler(app_manager.RyuApp):
   OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

   def __init__(self, *args, **kwargs):
      super(ARP_IP_Handler, self).__init__(*args, **kwargs)
      self.benign_arp_csv = "benign_arp_traffic.csv"
      self.attack_arp_csv = "attack_arp_traffic.csv"
      self.benign_ip_csv = "benign_ip_traffic.csv"
      self.attack_ip_csv = "attack_ip_traffic.csv"

      cols = ["Timestamp", "Switch_ID", "In_Port", "Out_Port",
            "SrcMAC_eth", "DstMAC_eth", "SrcIP_ip", "DstIP_ip", "OpCode_ip",
            "PacketIn_Count", "Protocol", "SrcPort", "DstPort",
            "PktLoss", "RTT_avg", "TotalTime", "Label"]

      self.init_csv(self.benign_arp_csv, cols)
      self.init_csv(self.attack_arp_csv, cols)
      self.init_csv(self.benign_ip_csv, cols)
      self.init_csv(self.attack_ip_csv, cols)

      self.packet_in_count = 0
      self.rtt_sum = 0
      self.rtt_count = 0

   def init_csv(self, filename, header):
      with open(filename, 'w', newline='') as csv_file:
         csv_writer = csv.writer(csv_file)
         csv_writer.writerow(header)

   def update_csv(self, filename, row):
      with open(filename, 'a', newline='') as csv_file:
         csv_writer = csv.writer(csv_file)
         csv_writer.writerow(row)

   @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
   def switch_features_handler(self, ev):
      datapath = ev.msg.datapath
```

```
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            return

        if eth.ethertype == ether_types.ETH_TYPE_ARP:
            arp_pkt = pkt.get_protocol(arp.arp)
            self.handle_arp(datapath, eth, arp_pkt)

        elif eth.ethertype == ether_types.ETH_TYPE_IP:
            ip_pkt = pkt.get_protocol(ipv4.ipv4)
            self.handle_ip(datapath, eth, ip_pkt)

        self.packet_in_count += 1

    def handle_arp(self, datapath, eth, arp_pkt):
        timestamp = time.strftime("     row = [timestamp, datapath.id, arp_pkt.in_port, ofproto_v1_3.OFPPacketIn, eth.src, eth.dst,
            arp_pkt.src_ip, arp_pkt.dst_ip, arp_pkt.opcode, self.packet_in_count, "", "", "", "", "", ""]

        # Assume an attack if ARP Opcode is not 1 (ARP Request)
        if arp_pkt.opcode != 1:
            row[-1] = "attack"
            self.update_csv(self.attack_arp_csv, row)
        else:
            row[-1] = "normal"
            self.update_csv(self.benign_arp_csv, row)

    def handle_ip(self, datapath, eth, ip_pkt):
        timestamp = time.strftime("     row = [timestamp, datapath.id, ip_pkt.in_port, ofproto_v1_3.OFPPacketIn, eth.src, eth.dst,
            ip_pkt.src, ip_pkt.dst, "", self.packet_in_count, "", "", "", "", "", ""]

        if ip_pkt.proto == inet.IPPROTO_TCP:
            tcp_pkt = pkt.get_protocol(tcp.tcp)
            row[10] = "TCP"
            row[11] = tcp_pkt.src_port
            row[12] = tcp_pkt.dst_port
        elif ip_pkt.proto == inet.IPPROTO_UDP:
            udp_pkt = pkt.get_protocol(udp.udp)
            row[10] = "UDP"
            row[11] = udp_pkt.src_port
            row[12] = udp_pkt.dst_port

        # Check for alterations in source and destination MAC addresses and ports
        if eth.src != ip_pkt.src or eth.dst != ip_pkt.dst or
        row[4:13] != [eth.src, eth.dst, ip_pkt.src, ip_pkt.dst, row[10], row[11], row[12]]:
            row[-1] = "attack"
            self.update_csv(self.attack_ip_csv, row)
        else:
            row[-1] = "normal"
            self.update_csv(self.benign_ip_csv, row)

def launch():
    app_manager.RyuApp.init(['ryu.app.ofctl_rest', "ARP_IP_Handler"])

    # Connect to the Ryu controller at 10.0.0.1:6653
    ryu_app = app_manager.lookup_service_brick("ARP_IP_Handler")
    ryu_app.datapaths = {}
    ryu_app.CONF = {}
    ryu_app.CONF['address'] = '10.0.0.1'
    ryu_app.CONF['port'] = 6653
    ryu_app.start()
```

This application code 3.1.1 is designed to monitor network traffic, identify ARP and IP packets, for detecting potential attacks based on certain criteria, and store relevant information in CSV files. This generated dataset can then be used to train machine learning models or deep learning models for attack detection in SDN environments. It provides a foundation for building and evaluating models that can effectively address specific problems or tasks. Researchers and practitioners can ensure that the model learns from diverse and representative examples, enabling it to generalize well to unseen data.

DNNs have been integrated into the system to enhance the detection capabilities, leveraging their ability to learn intricate patterns in network traffic and identify anomalous behavior that may indicate potential security threats. This integration of DNN underscores our commitment to employing cutting-edge technologies in safeguarding SDN environments against evolving cyber threats.

3.1.2. Detection model code fragment

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.models import load_model

# Load CSV files into pandas DataFrames
benign_arp_df = pd.read_csv("benign_arp_traffic.csv")
attack_arp_df = pd.read_csv("attack_arp_traffic.csv")
...
...

# Concatenate benign and attack ARP datasets
arp_data = pd.concat([benign_arp_df, attack_arp_df, ...], ignore_index=True)

# Split features and labels
X = arp_data.drop(columns=["Source", ..., "Label"])
y = arp_data["Label"]

# Perform one-hot encoding
...
...

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 3: Define and Train the DNN Model
model = Sequential([
  Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
  Dense(32, activation='relu'),
  Dense(16, activation='relu'),
  Dense(8, activation='relu'),
  Dense(4, activation='relu'),
  Dense(2, activation='relu'),
  Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Define early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, callbacks=[early_stopping])

# Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test)

# Save the Model
model.save("arp_attack_detection_model.h5")
```

The provided code 3.1.2 is aimed at training a deep learning model, specifically a DNN, to detect ARP spoofing attacks.

3.1.3. Mitigation code fragment

```
from keras.models import load_model
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet, arp
from collections import defaultdict

class ARPFloodingMitigation(app_manager.RyuApp):
  OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

  def __init__(self, *args, **kwargs):
    super(ARPFloodingMitigation, self).__init__(*args, **kwargs)
    self.threshold = 50  # Define the threshold for ARP requests
    self.arp_request_count = defaultdict(int)  # Dictionary to store ARP request counts per IP
    self.model = load_model('arp_attack_detection_model.h5')  # Load the trained model

  @set_ev_cls(ofp_event.EventOFPPacketIn, main_handler.CONFIG_DISPATCHER)
  def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocol(ethernet.ethernet)

    if eth.ethertype == ether_types.ETH_TYPE_ARP:
      arp_pkt = pkt.get_protocol(arp.arp)
      self.handle_arp(datapath, arp_pkt)

  def handle_arp(self, datapath, arp_pkt):
    src_ip = arp_pkt.src_ip
    self.update_arp_request_count(src_ip)
    self.detect_arp_flooding(src_ip)

  def update_arp_request_count(self, src_ip):
    # Update ARP request count for the source IP
    self.arp_request_count[src_ip] += 1

  def detect_arp_flooding(self, src_ip):
    # Check if the source IP address exceeds the threshold
    if self.arp_request_count[src_ip] > self.threshold:
      # Alert or disconnect the specific host
      self.alert_or_disconnect(src_ip)

  def alert_or_disconnect(self, src_ip):
    print(f"Possible ARP flooding attack detected from {src_ip}")
    ...

  def predict_arp_attack(self, arp_packet):
    features = self.extract_features(arp_packet)
    preprocessed_features = self.preprocess_features(features)
    prediction = self.model.predict(preprocessed_features)
    return prediction

  def _update_arp_request_count(self):
    ..., pass

  def extract_features(self, arp_packet):
    # Extract relevant features from the ARP packet
    features = [
      arp_packet.opcode,
      arp_packet.src_ip,
      arp_packet.dst_ip,
```

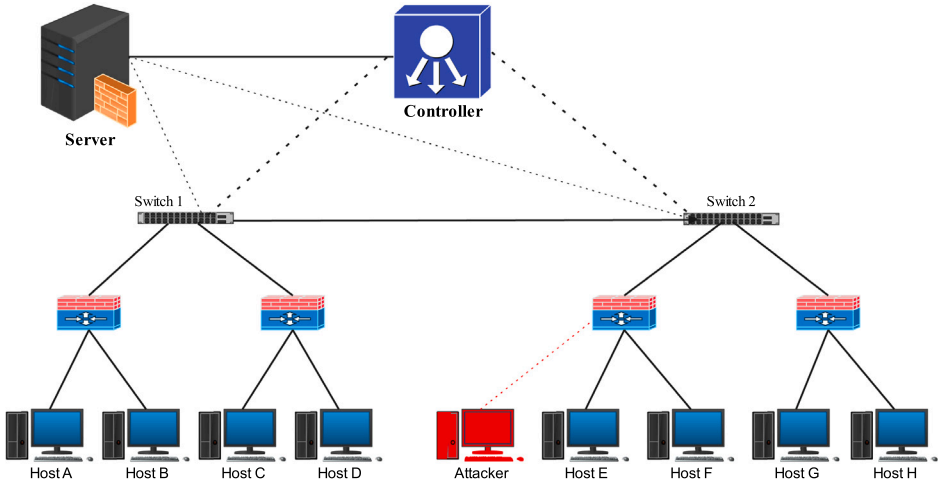


Fig. 4. Testbed for Proposed Architecture.

```
      arp_packet.src_mac,
      arp_packet.dst_mac,
      ...,
      ...
    ]
    return features

  def preprocess_features(self, features):
    preprocessed_features = pad_sequences(features, maxlen=max_length, padding='post')
    return preprocessed_features
```

This code 3.1.3 aims to mitigate ARP spoofing attacks by analyzing ARP packets in the network and taking appropriate actions based on the predictions made by the DNN model.

During the scrutiny process, the device implements a series of stringent checks to validate the authenticity of ARP packets. Firstly, it verifies whether ARP packets originate within the same network segment. Packets found to be routed outside of the designated network segment are promptly discarded, preventing potential cross-segment attacks. Secondly, the device cross-references the sender's IP address with the IP mapping database, ensuring coherence between IP addresses and their corresponding MAC addresses. Any disparities detected during this comparison lead to the rejection of the respective ARP packet, and notified the system administration about the activity.

A critical aspect of network security lies in the vigilance against impersonation attempts orchestrated by attackers. In this context, the dedicated machine remains ever watchful for ARP packets that could grant unauthorized access to various IP addresses within the network. By meticulously scrutinizing these packets, the algorithm effectively detects ongoing ARP spoofing attacks. Upon identification of such attacks, the algorithm triggers prompt and tailored responses, mitigating the threat and preserving the integrity of the network.

In essence, this comprehensive methodology fortifies SDN networks against ARP attacks and potential security vulnerabilities, offering a holistic and proactive approach to network security and threat mitigation. By combining real-time awareness, tailored flow control, and meticulous packet analysis, this algorithm stands as a formidable guardian, safeguarding SDNs from the insidious threats of ARP spoofing attacks.

The proposed testbed, as illustrated in Fig. 4, delineates a sophisticated interplay among essential components, including SDN devices, a dedicated machine, and a network topology data collector. This architectural framework stands as the linchpin of our solution, empowering network administrators with potent tools to combat ARP spoofing attacks, fortify network security, and prevent operational disruptions effectively. Central to our system's efficacy is its incorporation of a mechanism designed for monitoring ARP reply activities. This functionality endows the system with the ability to detect and identify malicious packets exhibiting similar behavioral patterns. To respond judiciously to such threats, we have established a predefined threshold, restricting the maximum number of permissible duplicate packets. This threshold serves as a crucial safeguard against ARP spoofing attacks, a tactic often employed by attackers who forge MAC address associations between targets and clients, inundating the controller with deceptive ARP reply packets, thus portraying the connection as legitimate.

Given the potential scale of attacks involving the transmission of thousands of packets per second and the frequent reception of numerous duplicate packets by switches, the established threshold plays a pivotal role in curtailing the impact of such onslaughts. Upon reaching this threshold, the controller promptly initiates packet discarding procedures, thereby enhancing the overall resilience of the network against ARP spoofing threats significantly.

In the experimentation phase of our study, we meticulously designed a diverse array of scenarios to rigorously assess the adaptability and efficacy of our proposed solution in countering ARP spoofing attacks across varied network conditions. Recognizing the paramount importance of comprehensively evaluating our system's performance under different traffic conditions and variable packet transmission capacities, our approach was meticulous and systematic.

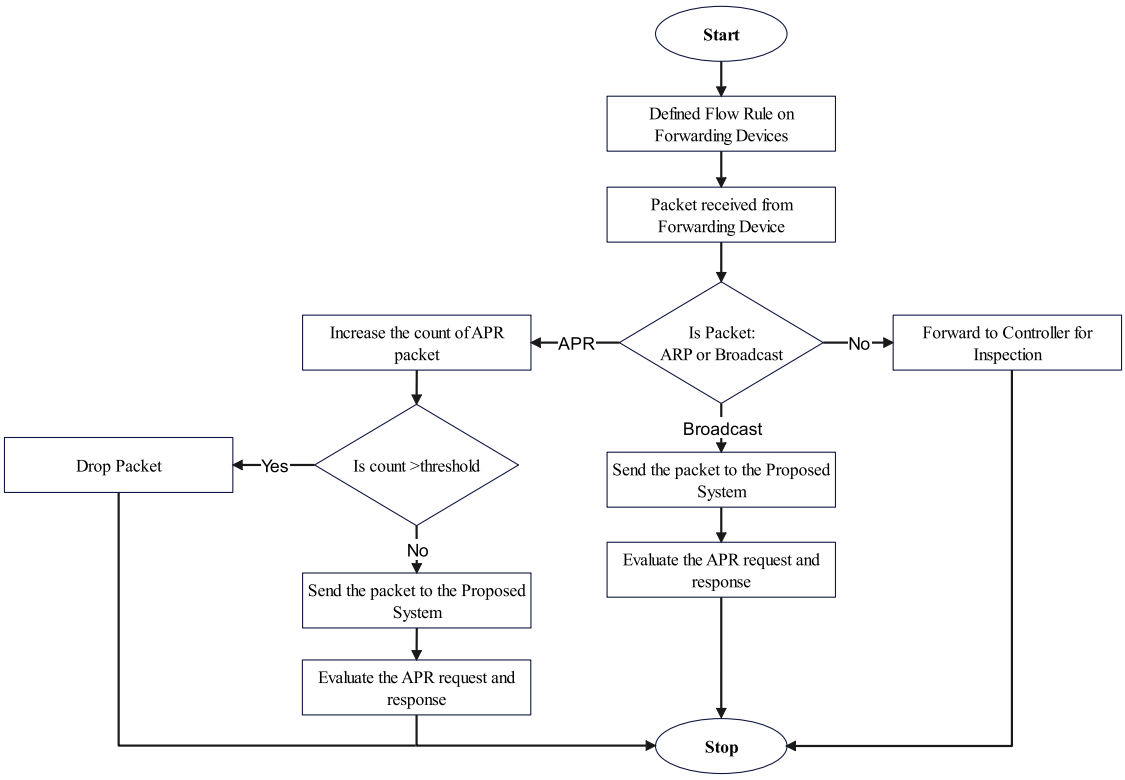


Fig. 5. Flow diagram of ARP attack mitigation.

4. Experimental setup and results discussion

Our experimental evaluation aimed to rigorously assess the performance and effectiveness of our proposed ARP spoofing detection and mitigation algorithm within a simulated environment. These experiments were conducted within a virtualized setup, utilizing robust hardware and software configurations to ensure a comprehensive analysis.

4.1. System setup

- **Hardware Configuration**: The experimentation platform was based on a high-performance 9th-generation Intel Core i9 machine equipped with 64 GB of RAM. This powerful hardware setup ensured the execution of simulations with efficiency and accuracy.
- **Operating Environment**: The host machine ran Ubuntu Linux, providing a stable and versatile environment for conducting experiments. We utilized Oracle VirtualBox as the hypervisor server to facilitate virtualization.

4.2. Simulation environment

- **Virtual Network Infrastructure**: To create a diverse range of SDN scenarios, we employed the Mininet simulation environment. This environment allowed us to model SDN switches, hosts, legacy switches, and controllers, replicating different network infrastructures.
- **Controller and Switches**: The experiments leveraged the RYU controller and OpenVSwitches to instantiate the SDN components. These choices provided a flexible and extensible platform for network emulation.
- **Network Topologies**: We configured network topologies by establishing virtual connectivity between devices, servers, and networks. The study objectives and proposed solutions were implemented within this virtualized framework, utilizing the links established between network entities.
- **Traffic Generation with Scapy**: The Scapy tool was integrated into our simulation environment to generate both benign and attack traffic. Scapy's capabilities were harnessed to craft and inject custom packets, allowing for the simulation of normal network communication and deliberate attack patterns. This integration enhanced the versatility and precision of our experimental setup.
- **Custom Code for Traffic Generation**: In addition to Scapy, we implemented custom code to generate specific attack and benign traffic patterns. This bespoke code provided us with fine-grained control over the characteristics of the simulated traffic, allowing us to tailor the experiments to different threat scenarios and network conditions.

4.3. An attack generating ARP traffic

```
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import OVSSwitch, RemoteController
```

```
from mininet.cli import CLI

class SimpleTopology(Topo):
  def build(self):
    switch = self.addSwitch('s1')
    h1 = self.addHost('h1')
    h2 = self.addHost('h2')
    h3 = self.addHost('h3')
    h4 = self.addHost('h4')

    self.addLink(h1, switch)
    self.addLink(h2, switch)
    self.addLink(h3, switch)
    self.addLink(h4, switch)

if __name__ == '__main__':
  topo = SimpleTopology()
  net = Mininet(topo=topo, switch=OVSSwitch, controller=RemoteController)
  net.start()
  # Static flow entry to forward packets between hosts
  for i in range(1, 5):
    net.get('s1').cmd('ovs-ofctl add-flow s1 in_port={},actions=output:{}'.format(i, 5 - i))

  # Generate ARP spoofing attack traffic from h1 to h2
  attack_script = '''
   from scapy.all import Ether, ARP, sendp
  pkt = Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(op=2, psrc="10.0.0.1", pdst="10.0.0.2", hwdst="00:00:00:00:00:00")
   sendp(pkt, iface="h1-eth0", count=5)
  '''
  net.get('h1').cmd('python -c "{}"'.format(attack_script))
  CLI(net)
  net.stop()
```

4.4. A normal ARP traffic

```
for i in range(1, 5):
  net.get('s1').cmd('ovs-ofctl add-flow s1 in_port={},actions=output:{}'.format(i, 5 - i))

# Generate benign ARP traffic between hosts
for src in range(1, 5):
  for dst in range(1, 5):
    if src != dst:
       net.get('h{}'.format(src)).cmd('arpping -I h{}-eth0 -c 3 {}'.format(src,
       net.get('h{}'.format(dst)).IP()))
```

The code 4.3 uses scapy to send ARP reply packets from host h1 to host h2 to simulate an ARP spoofing attack. The provided code sets up a simple network topology using Mininet, comprising a switch (s1) and four hosts (h1, h2, h3, h4) connected to the switch. Each host is connected to the switch via a link, establishing a basic network structure.

The key aspect of the code lies in the generation of ARP spoofing attack traffic from host h1 to h2. ARP spoofing is a type of attack where an attacker sends falsified ARP messages over a local area network. In this scenario, the attack script crafts ARP packets with falsified sender IP addresses (psrc) to deceive the target host into associating the attacker's MAC address with the IP address of the legitimate gateway or another host.

The attack traffic is generated using Scapy, a powerful packet manipulation tool, to construct and send ARP packets. The crafted ARP packets are sent from h1 to h2, simulating an ARP spoofing attack scenario.

Such simulations are crucial for network security research, testing intrusion detection systems, and evaluating network defense mechanisms. By creating controlled environments where various attack scenarios can be simulated, researchers and network administrators can better understand vulnerabilities and develop effective countermeasures to protect against real-world threats. Additionally, these simulations enable the testing and validation of security solutions without risking the integrity of production networks. Whereas the code 4.4 does not use scapy to generate ARP traffics, from each hosts to another hosts to simulate an ARP normal traffic.

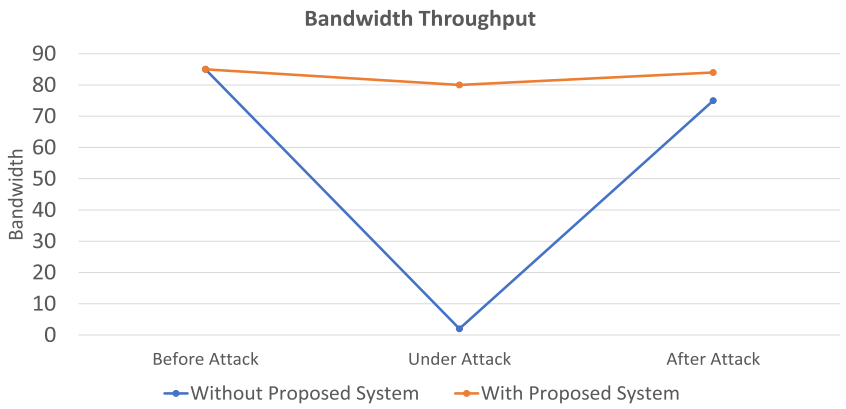


Fig. 6. Bandwidth throughput.

4.5. Results and discussions

Network performance is a critical aspect of any networking system, and an essential metric for evaluating it is throughput. Throughput essentially gauges the efficiency with which a network's resources are utilized. To comprehensively assess the impact of our proposed approach, we conducted a comparative analysis of network throughput. This evaluation involved measuring the network's throughput both before and after an attack, with uninterrupted communication between the host and controller maintained throughout these scenarios.

As illustrated in Figs. 6, 7, and 8, our system demonstrates a substantial enhancement in network performance. We quantified the SDN controller throughput considering various factors, including data packet transmission, TCP window size, and round-trip time. These factors were integrated into an equation that furnishes a precise estimation of the network's overall efficiency. This analysis was instrumental in gauging the tangible benefits our solution offers in terms of optimizing network performance.

Fig. 6 presents the bandwidth throughput in different states: before an ARP Spoofing attack (Before Attack), during the attack (Under Attack), and after the attack has been mitigated (After Attack). The data is presented for both scenarios: without the proposed system and with the proposed system. Analyzing these values provides insights into the efficiency of the proposed system in managing bandwidth throughput during and after ARP Spoofing attacks.

Without Proposed System:

- Before Attack: The network operates at a standard bandwidth throughput of 85, indicating the baseline performance under normal conditions.
- Under Attack: During the attack, the bandwidth throughput drastically drops to 2, indicating severe disruption caused by the ARP Spoofing attack. The network experiences significant congestion and reduced data transfer capabilities.
- After Attack: After the attack, the network's bandwidth throughput partially recovers to 75. While it improves from the lowest point during the attack, it does not fully restore to the pre-attack level, indicating lingering issues in network performance.

With Proposed System:

- Before Attack: Similar to the scenario without the proposed system, the network operates at a standard bandwidth throughput of 85, maintaining optimal performance.
- Under Attack: Remarkably, with the proposed system in place, the network's bandwidth throughput during the attack remains significantly higher at 80. This demonstrates the system's ability to mitigate the attack's impact on data transfer, ensuring more stable network performance even under malicious traffic conditions.
- After Attack: After mitigating the attack, the network's bandwidth throughput almost fully recovers to 84. This near-complete restoration signifies the effectiveness of the proposed system in efficiently handling the residual load and restoring the network to its optimal throughput.

Analysis:

- During Attack Mitigation: The proposed system manages to maintain a high bandwidth throughput (80) even during the attack, showcasing its efficiency in handling malicious traffic and preventing significant disruptions.
- Post-Attack Recovery: After mitigating the attack, the system almost fully restores the network's bandwidth throughput (84), indicating the successful resolution of the disruption caused by the ARP Spoofing attack.

Our proposed approach employs a multifaceted strategy to pinpoint counterfeit users within the network environment. This strategy is based on the deployment of port blocking policies in conjunction with packet-dropping thresholds, which depend on the number of prior attempts at unauthorized access.

Through the implementation of ARP spoofing detection mechanisms, our approach acts as a vigilant sentinel, promptly identifying potential intruders or spurious users. As a direct consequence, the network's overall load experiences a noticeable reduction. This reduction in load translates into a substantial increase in the bandwidth rates of the connected nodes, a phenomenon vividly depicted in Fig. 6.

Our algorithmic policy operates as a dynamic safeguard, efficiently reallocating network resources based on real-time user activity. This adaptability ensures that as the network's composition evolves, the bandwidth remains optimally distributed, allowing legitimate users to enjoy enhanced access while deterring and limiting the impact of any unauthorized incursions.

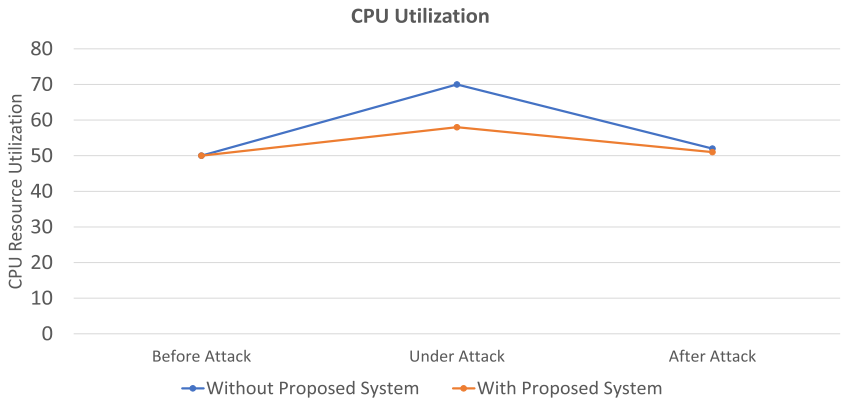


Fig. 7. CPU utilization.

We embarked on a meticulous journey to gauge the practical implications of our methodology on the intricate machinery of CPU utilization. Our aim was to discern how this critical metric evolves across the pre-attack, attack, and post-attack phases. The endeavor was not merely an exercise in observation; it was a testament to the resilience and efficacy of our proposed methodology.

But our evaluation did not end there; we delved deeper. We scrutinized performance not just during the attack but also during the crucial mitigation process. It was an essential facet of our analysis, ensuring that not only was the network shielded from harm but also that the mitigation process itself did not inflict undue strain on the system.

In essence, our findings affirmed the multifaceted value of our proposed methodology. It acted not only as a resilient fortress during attacks but also as a guardian of stability, preserving the network's integrity without overwhelming the CPU.

Fig. 7 illustrates the controller's CPU resource utilization across different states: before an ARP Spoofing attack, during an ongoing attack, and after the attack, both with and without the implementation of the Proposed System. CPU resource utilization is a crucial metric in evaluating the stability and efficiency of ARP Spoofing attack mitigation strategies.

In the initial state, before the attack occurs, the CPU resource utilization stands at a baseline level of 50, indicating the normal operational load on the system. When subjected to an ARP Spoofing attack (Under Attack), the CPU usage without the Proposed System increases significantly to 70, reflecting the additional processing demands imposed by the attack. However, with the implementation of the Proposed System, the CPU usage remains comparatively lower at 58 during the attack, showcasing the mitigation system's ability to manage the attack-induced load efficiently.

The stability in CPU resource utilization, coupled with the efficient management of increased loads during and after the attack, underscore the effectiveness of the Proposed System in mitigating ARP Spoofing attacks while maintaining stable and efficient resource utilization, enhancing the overall resilience of the network infrastructure. Fig. 8 presents the average CPU usage in different states: during an ARP Spoofing attack (Under Attack) and after the attack has been mitigated (After Mitigation). The data is presented for varying numbers of nodes in the network: 5 nodes, 15 nodes, and 30 nodes. Analyzing these values provides valuable insights into the system's efficiency in handling ARP Spoofing attacks across different network scales.

Under Attack:

- **5 Nodes**: During the attack on a smaller network (5 nodes), the system demonstrates efficient resource management, with an average CPU usage of 15. This indicates that the system copes well with the attack's demands on a small-scale network.
- **15 Nodes**: In a moderately sized network (15 nodes), the average CPU usage under attack increases to 52. While the system experiences a significant load, it still manages to operate, albeit with higher resource utilization.
- **30 Nodes**: In a larger network (30 nodes), the average CPU usage under attack rises to 80. This signifies a substantial increase in processing demands due to the larger network size and the corresponding increase in ARP Spoofing attack traffic.

After Mitigation:

- **5 Nodes**: After mitigation, the CPU usage drops significantly to 5 in the 5-node network, indicating the system's ability to efficiently handle the residual load post-mitigation.
- **15 Nodes**: In the 15-node network, the CPU usage after mitigation is 41. Although it is higher than the pre-attack state, it reflects the system's efforts to restore stability and manage the network's traffic effectively after mitigating the attack.
- **30 Nodes**: After mitigating the attack in the 30-node network, the CPU usage decreases to 73. While still relatively high, it represents a substantial improvement from the peak CPU usage experienced during the attack.

Analysis:

- The system demonstrates efficient resource management even during ARP Spoofing attacks, especially in smaller networks. However, as the network size increases, the system faces higher processing demands during attacks.
- After mitigation, the system works diligently to restore stability. While the CPU usage remains elevated, it significantly drops from the peak usage experienced during the attack, indicating successful recovery.
- The ability to handle large-scale networks and efficiently manage resource utilization, both during and after attacks, showcases the system's robustness and effectiveness in ARP Spoofing attack mitigation across various network sizes.

Detecting ARP spoofing attacks using DNN model represents a significant advancement as per our experiments. By leveraging the power of deep learning, particularly DNNs, it becomes possible to identify subtle patterns and anomalies in network traffic indicative of ARP spoofing attempts.

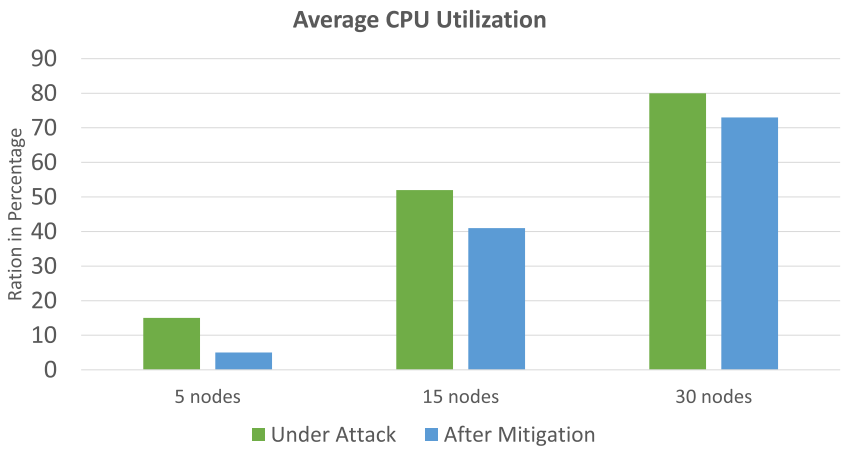


Fig. 8. Average CPU utilization.

One approach involves training the DNN model using publicly available datasets namely the IoT Network Intrusion MITM-ARP-Spoofing[1] Dataset. This dataset provides a diverse range of network traffic scenarios, allowing the DNN to learn from various ARP spoofing attack instances and their corresponding network contexts. In addition to publicly available datasets, self-generated datasets was utilized to proof the applicability and the effectiveness of DNN model for ARP spoofing detection. In Section 3.1.1, the methodology outlines the process of creating custom datasets tailored to the SDN environment. These datasets capture real-world network behaviors and attack patterns, offering insights into the intricacies of ARP spoofing attacks within the context of the targeted network infrastructure.

4.6. Proposed DNN model

The hyperparameters in the provided DNN model code are as follows:

- **Dropout Rate (0.1)**: Dropout is a regularization technique that randomly drops a fraction of input units during training to prevent overfitting. A dropout rate of 0.1 indicates that 10% of the input units are randomly dropped during each training epoch.
- **Learning Rate (0.0001)**: The learning rate determines the step size at which the model's weights are updated during training. A lower learning rate ensures smoother and more stable convergence but may require more training epochs.
- **Batch Size (32)**: Batch size refers to the number of training examples utilized in one iteration. A batch size of 32 means that 32 samples are processed before the model's weights are updated.
- **Epochs (50)**: An epoch represents one complete pass through the entire training dataset. Training the model for 50 epochs means that the dataset is traversed 50 times during training.

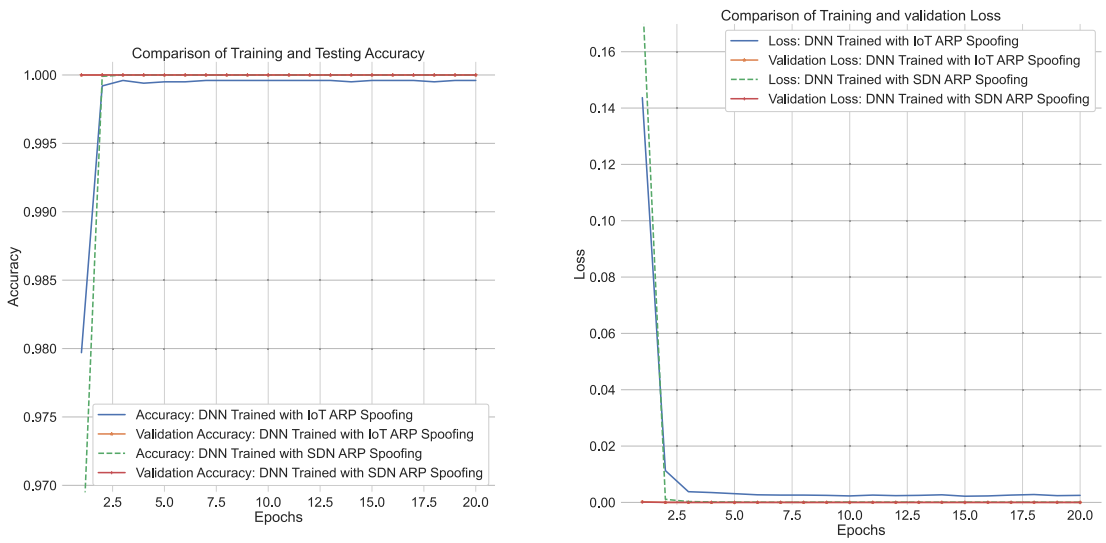
Fig. 9 depicts a visual representation of the DNN model's performance in detecting ARP spoofing attacks. Notably, the DNN exhibits commendable performance metrics, demonstrating its efficacy in identifying malicious ARP spoofing activities within network traffic. Impressively, training the DNN with a self-generated dataset sourced from SDN environments yielded superior performance compared to training with the publicly available IoT ARP Spoofing dataset.

During training, the DNN model trained on the self-generated dataset consistently outperformed its counterpart trained on the IoT ARP Spoofing dataset. This enhanced performance is evident in both the training and validation phases, where the model's loss rates remained exceptionally low. The robustness and accuracy of the DNN across these datasets provide compelling evidence of its suitability for ARP spoofing detection applications. Fig. 9 showcases the DNN's ability to discern between normal network behavior and suspicious ARP spoofing patterns. Through its sophisticated learning mechanisms, the model effectively captures subtle nuances in network traffic, enabling timely detection and mitigation of potential security threats. The DNN's consistent and convincing performance underscores its applicability as a valuable tool in safeguarding network infrastructures against ARP spoofing attacks.

Table 2 provides a comparative analysis between our proposed method and other studies in the context of ARP spoofing mitigation within SDNs. The table systematically assesses various crucial aspects, elucidating the distinctive features of our proposed system and its advantages over existing approaches.

- **Network Size**: Most existing studies focus on small to medium-sized networks, limiting their applicability to larger infrastructures. In contrast, our proposed method demonstrates adaptability by addressing ARP spoofing across networks of varied sizes, making it suitable for both small and large-scale deployments.
- **Reliability**: The reliability of ARP spoofing detection mechanisms is a critical factor. Our proposed system exhibits high reliability across diverse network sizes, ensuring consistent and accurate detection in different environments. While some previous studies achieve good reliability in small networks, their performance tends to diminish in larger setups.
- **CPU Utilization**: Efficient CPU utilization is essential for ensuring the seamless operation of network security solutions. Our method demonstrates good CPU utilization, indicating its ability to effectively detect and mitigate ARP spoofing attacks without imposing excessive computational overhead. Several previous studies achieve acceptable CPU utilization, but some do not specify their impact on CPU resources.

[1] https://doi.org/10.21227/q70p-q449



(a) Model Accuracy: Training and Validation

(b) Loss Rate: Training and Validation

Fig. 9. Training DNN model and validation.

Table 2
Comparison of proposed method and other studies.

Study	Network size	Reliability	CPU utilization	Network environment
Girdler and Vassilakis [5]	Small Network	Good	Not Specified	Simulated SDN
AbdelSalam et al. [6]	Small Network	Good	Acceptable	Simulated SDN
Amin et al. [7]	Small Network	Good	Not Specified	Simulated SDN
Khalid et al. [8]	Small Network	Good	Acceptable	Simulated SDN
Lin et al. [9]	Small Network	Average	Not Specified	Simulated SDN
Saritakumar et al. [10]	Medium Network	Average	Acceptable	Simulated SDN
Aldabbas and Amin [12]	Small Network	Good	Acceptable	Simulated SDN
Galal et al. [13]	Tiny Network	Limited	Not Specified	Real-world Network
Jamil et al. [14]	Small Network	Good	Acceptable	Simulated SDN
Xu et al. [15]	Tiny Network	Good	Not Specified	Simulated SDN
Our Proposed Method	Small Network	High	Good	Simulated SDN

- **Network Environment**: The distinction between simulated SDN environments and real-world networks is pivotal. Our proposed method demonstrates its prowess in simulated SDN settings, ensuring its viability for practical implementation and testing. Notably, Galal et al.'s approach operates in a real-world network, showcasing the potential for real-world applicability.

In SDN environment, data plane devices can implement Access Control Lists (ACLs) to control which traffic is allowed to reach the dedicated machine for ARP processing. By carefully defining and enforcing access policies, the devices can mitigate the risk of unauthorized access or malicious manipulation of ARP traffic. Additionally, implementing encryption and authentication mechanisms for communication between data plane devices and the dedicated machine can ensure the integrity and confidentiality of ARP traffic. Techniques such as Internet Protocol Security (IPsec) can be employed to secure ARP messages and prevent tampering or eavesdropping.

Deploying IDS solutions on data plane devices enables real-time monitoring of network traffic for signs of suspicious or anomalous behavior. IDS can detect and mitigate ARP spoofing attacks by analyzing traffic patterns, detecting ARP anomalies, and triggering appropriate response actions. On the otherhand, segmenting the network and implementing Virtual Local Area Networks (VLANs) or other network segmentation techniques can isolate ARP traffic and restrict access to critical resources, minimizing the impact of potential attacks on the dedicated machine. Most importantly, regular monitoring and auditing network traffic, configurations, and access controls help identify potential security vulnerabilities or policy violations. By maintaining visibility into network activity, administrators can promptly detect and respond to security incidents or unauthorized access attempts.

One crucial parameter we considered during testing was network congestion, a pivotal factor affecting network performance. We measured congestion by analyzing the percentage of the network's available bandwidth in use at any given time. This metric provided valuable insights into the load imposed on the network infrastructure.

Table 3 provides a comprehensive short comparison of ARP Spoofing Detection systems, highlighting the efficacy of our proposed system in detecting and mitigating ARP spoofing attacks. Our study emerges as a frontrunner in terms of detection accuracy, owing to several key factors, notably the utilization of self-generated datasets tailored to SDN environments.

The striking accuracy of 100% achieved by our system outshines other studies listed in the table. Leveraging a self-generated dataset derived from SDN environments, our model exhibits unparalleled performance, setting a new standard for ARP spoofing detection. This superior accuracy underscores the robustness and adaptability of our approach to real-world network scenarios.

Table 3
Comparison of ARP Spoofing Detection with proposed system.

Study	Dataset	Model	Accuracy	Loss	Recall	Precision score	F1-Measure	ROC-AUC	Detection time
[5]	*	*	*	*	*	*	*	*	2.20 (s)
[8]	*	*	*	*	*	*	*	*	0.006 (s)
[7]	*	*	*	*	*	*	*	*	1 (s)
[15]	*	K-FKNN	*	*	*	*	97.66%	*	4.39 (s)
[18]	Self Generated	CNN-LSTM	99.73%	0.017	95.6%	97.8%	96.2%	*	1 (s)
[19]	IoT network intrusion	ARP-PROBE	99.98%	0.0009	99.98%	99.98%	99.98%	*	0.0607 ms
This Study	IoT MITM ARP Spoofing	DNN	100%	0.0000	100%	100%	100%	100%	0.0064 (s)
	Self Generated		100%	0.0000	100%	100%	100%	100%	0.0025 (s)

Moreover, the negligible loss rates and swift detection times further validate the effectiveness of our methodology. By minimizing false positives and false negatives, our system ensures prompt and accurate detection of ARP spoofing attacks, thereby enhancing network security without compromising performance.

The choice of a DNN model for detection proves to be a strategic one, offering a powerful framework capable of learning complex patterns inherent in ARP spoofing attacks. Additionally, the model's training and validation using the IoT MITM ARP Spoofing dataset demonstrate its applicability to real-world scenarios, further solidifying its relevance and reliability.

5. Conclusion

The rising incidence of ARP spoofing attacks within SDNs poses a significant security concern, capable of disrupting network operations, compromising data integrity, and eroding system trust. In response, we have introduced an innovative approach harnessing SDN controller programmability to detect and mitigate ARP spoofing incidents in real-time effectively.

Our method integrates sophisticated mechanisms to identify suspicious ARP traffic patterns promptly, facilitating swift response actions. Moreover, our response mechanism tactfully isolates affected network segments, halting the spread of malicious ARP packets while minimizing disruptions to legitimate network traffic. Through rigorous experimentation in a controlled SDN environment, we have demonstrated the effectiveness of our technique in mitigating ARP spoofing attacks. By bolstering network security without sacrificing availability or performance, our solution contributes significantly to fortifying SDN infrastructures and fostering their continued adoption in modern network environments.

Incorporating deep learning techniques for real-time anomaly detection, our study demonstrates the superiority of employing DNN models in detecting ARP spoofing attacks. Tested with a self-generated dataset from SDN environments, leveraging Scapy tools and custom code to simulate attacks, our model showcases its adaptability and effectiveness. Furthermore, its training and validation using the IoT MITM ARP Spoofing dataset underscore its applicability to real-world scenarios.

As the landscape of network security evolves, proactive measures like our proposed technique become increasingly vital in safeguarding critical network assets and ensuring SDN resilience. We envision this research inspiring further advancements in network security strategies, fostering more secure and reliable SDN environments for organizations and individuals. Undoubtedly, exploring the integration of deep learning techniques for real-time anomaly detection represents a promising avenue for future research in network security.

CRediT authorship contribution statement

Vanlalruata Hnamte: Conceptualization, Data curation, Formal analysis, Methodology, Validation, Visualization, Writing – original draft. **Jamal Hussain:** Supervision, Visualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] H. Bour, M. Abolhasan, S. Jafarizadeh, J. Lipman, I. Makhdoom, A multi-layered intrusion detection system for software defined networking, Comput. Electr. Eng. 101 (2022) 108042, http://dx.doi.org/10.1016/j.compeleceng.2022.108042.
[2] V. Hnamte, A.A. Najar, H. Nhung-Nguyen, J. Hussain, M.N. Sugali, Ddos attack detection and mitigation using deep neural network in SDN environment, Comput. Secur. 138 (2024) 103661, http://dx.doi.org/10.1016/j.cose.2023.103661.
[3] S. Buzura, M. Lehene, B. Iancu, V. Dadarlat, An extendable software architecture for mitigating ARP spoofing-based attacks in SDN data plane layer, Electronics 11 (13) (2022) http://dx.doi.org/10.3390/electronics11131965.
[4] R.C. Meena, S. Bhatia, R.H. Jhaveri, L. Cheng, A. Kumar, A. Mashat, HyPASS: Design of hybrid-SDN prevention of attacks of source spoofing with host discovery and address validation, Phys. Commun. 55 (2022) 101902, http://dx.doi.org/10.1016/j.phycom.2022.101902.
[5] T. Girdler, V.G. Vassilakis, Implementing an intrusion detection and prevention system using software-defined networking: Defending against ARP spoofing attacks and blacklisted MAC addresses, Comput. Electr. Eng. 90 (2021) 106990, http://dx.doi.org/10.1016/j.compeleceng.2021.106990.
[6] A.M. AbdelSalam, A.B. El-Sisi, K. V. Reddy, Mitigating ARP spoofing attacks in software-defined networks, in: 2015 25th International Conference on Computer Theory and Applications, ICCTA, 2015, pp. 126–131, http://dx.doi.org/10.1109/ICCTA37466.2015.9513433.

[7] R. Amin, M. Hussain, M. Alhameed, S.M. Raza, F. Jeribi, A. Tahir, Edge-computing with graph computation: A novel mechanism to handle network intrusion and address spoofing in SDN, Comput. Mater. Continua 65 (3) (2020) 1869–1890, http://dx.doi.org/10.32604/cmc.2020.011758.
[8] H.Y.I. Khalid, P.M. Ismael, A. Baheej Al-Khalil, Efficient mechanism for securing software defined network against ARP spoofing attack, J. Duhok Univ. 22 (2019) 124–131, http://dx.doi.org/10.26682/sjuod.2019.22.1.14.
[9] T.-Y. Lin, J.-P. Wu, P.-H. Hung, C.-H. Shao, Y.-T. Wang, Y.-Z. Cai, M.-H. Tsai, Mitigating syn flooding attack and ARP spoofing in SDN data plane, in: 2020 21st Asia-Pacific Network Operations and Management Symposium, APNOMS, 2020, pp. 114–119, http://dx.doi.org/10.23919/APNOMS50412.2020.9236951.
[10] S. N., A. K.V., A. S., Detection and mitigation of ARP poisoning attack in software defined network, in: Proceedings of the First International Conference on Combinatorial and Optimization, ICCAP 2021, December 7-8 2021, Chennai, India, EAI, 2021, pp. 1–9, http://dx.doi.org/10.4108/eai.7-12-2021.2314502.
[11] J.S. Meghana, T. Subashri, K. Vimal, A survey on ARP cache poisoning and techniques for detection and mitigation, in: 2017 Fourth International Conference on Signal Processing, Communication and Networking, ICSCN, 2017, pp. 1–6, http://dx.doi.org/10.1109/ICSCN.2017.8085417.
[12] H. Aldabbas, R. Amin, A novel mechanism to handle address spoofing attacks in SDN based IoT, Cluster Comput. 24 (4) (2021) 3011–3026, http://dx.doi.org/10.1007/s10586-021-03309-0.
[13] A.A. Galal, A.Z. Ghalwash, M. Nasr, A new approach for detecting and mitigating address resolution protocol (ARP) poisoning, Int. J. Adv. Comput. Sci. Appl. 13 (6) (2022) http://dx.doi.org/10.14569/IJACSA.2022.0130647.
[14] F. Jamil, H. Jamil, A. Ali, Spoofing attack mitigation in address resolution protocol (ARP) and DDoS in software-defined networking, J. Inf. Secur. Cybercrimes Res. 5 (1) (2022) 35–46, http://dx.doi.org/10.26735/VBVS3993.
[15] Y. Xu, H. Sun, F. Xiang, Z. Sun, Efficient ddos detection based on K-FKNN in software defined networks, IEEE Access 7 (2019) 160536–160545, http://dx.doi.org/10.1109/ACCESS.2019.2950945.
[16] N.H. Thanh, N.N. Tuan, D.A. Khoa, L.C. Tuan, N.T. Kien, N.X. Dung, N.Q. Thu, F. Wamser, On profiling, benchmarking and behavioral analysis of SDN architecture under DDoS attacks, J. Netw. Syst. Manage. 31 (2) (2023) 1–32, http://dx.doi.org/10.1007/s10922-023-09732-5.
[17] A. Prasad, S. Chandra, Defending ARP spoofing-based mitm attack using machine learning and device profiling, in: 2022 International Conference on Computing, Communication, and Intelligent Systems, ICCCIS, 2022, pp. 978–982, http://dx.doi.org/10.1109/ICCCIS56430.2022.10037723.
[18] N. Ahuja, G. Singal, D. Mukhopadhyay, A. Nehra, Ascertain the efficient machine learning approach to detect different ARP attacks, Comput. Electr. Eng. 99 (2022) 107757, http://dx.doi.org/10.1016/j.compeleceng.2022.107757.
[19] M.M. Alani, A.I. Awad, E. Barka, Arp-probe: An ARP spoofing detector for Internet of Things networks using explainable deep learning, Internet Things 23 (2023) 100861, http://dx.doi.org/10.1016/j.iot.2023.100861.