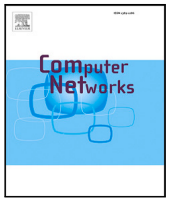
Computer Networks 234 (2023) 109935

Contents lists available at ScienceDirect

Computer Networks

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



Defending SDN against packet injection attacks using deep learning

Anh Tuan Phu [a], Bo Li [a], Faheem Ullah [a], Tanvir Ul Huque [b], Ranesh Naha [a,c,*], Muhammad Ali Babar [d], Hung Nguyen [a]

[a] The University of Adelaide, Australia
[b] CyberCX, Australia
[c] Centre for Smart Analytics, Federation University Australia, Australia
[d] The Centre for Research on Engineering Software Technologies, The University of Adelaide, Australia

ARTICLE INFO

Keywords:
Neural network
Software-defined networking
Network security
Packet injection attack
Attack detection

ABSTRACT

The (logically) centralized architecture of software-defined networks makes them an easy target for packet injection attacks. In these attacks, the attacker injects malicious packets into the SDN network to affect the services and performance of the SDN controller and overflows the capacity of the SDN switches. Such attacks have been shown to ultimately stop the network functioning in real-time, leading to network breakdowns. There have been significant works on detecting and defending against similar DoS attacks in non-SDN networks, but detection and protection techniques for SDN against packet injection attacks are still in their infancy. Furthermore, many of the proposed solutions have been shown to be easily bypassed by simple modifications to the attacking packets or by altering the attacking profile. In this paper, we develop novel Graph Convolutional Neural Network models and algorithms for grouping network nodes/users into security classes by learning from network data. We start with two simple classes — nodes that engage in suspicious packet injection attacks and nodes that are not. From these classes, we then partition the network into separate segments with different security policies using distributed Ryu controllers in an SDN network. We show in experiments on an emulated SDN that our detection solution outperforms alternative approaches with above 99% detection accuracy for various types (both old and new) of injection attacks. More importantly, our mitigation solution maintains continuous functions of non-compromised nodes while isolating compromised/suspicious nodes in real-time. All code and data are publicly available for the reproducibility of our results.

1. Introduction

Software-defined networking (SDN) provides a more agile, efficient, and secure way to manage and operate modern networks, and is increasingly being adopted by organizations of all sizes and industries. It is considered a potential solution to problems that are inherent in traditional network architecture. These problems include but are not limited to network complexity, network scalability, network agility, network security, and network visibility [1]. SDN is gaining more popularity in managing virtualized networks, such as cloud infrastructures. SDN, for example, is being employed in data centers and workload-optimized systems [2]. The separation of the control plane and data plane provides network administrators with the ability to manage the network using a centralized approach, accelerating the provisioning of both physical and virtual network devices.

However, the networking architecture of SDN makes it an easy target for packet injection attacks. In SDNs, a logically centralized controller collects the devices information (e.g., host location, switch information) and constructs a global topology view. Based on the topology view, the controller then deploys forwarding policies by installing flow rules on each networking device (e.g., switches). When deployed in an enterprise network, any data packet flow coming from the *Outside network*, *i.e.*, other networks, passes through the gateway switch to get access to the network. Similarly, the data packet flow, generated in the enterprise network and destined to *Outside network*, goes through the gateway switch of the enterprise network. Thus the gateway switch acts as the only access point for the data packet flows to come in/go out to/from the network which works reactively, whereas the core switches work proactively in the software-defined enterprise network [3].

An OpenFlow switch working in the reactive mode triggers a Packet-In message for any unmatched data packet flow when it encounters and, then sends the message to the controller. After getting

* Corresponding author at: The University of Adelaide, Australia.
E-mail addresses: anhtuan.phu@student.adelaide.edu.au (A.T. Phu), b.li@adelaide.edu.au (B. Li), faheem.ullah@adelaide.edu.au (F. Ullah), tanvir.huque@yahoo.com (T. Ul Huque), n.naha@federation.edu.au (R. Naha), ali.babar@adelaide.edu.au (M.A. Babar), hung.nguyen@adelaide.edu.au (H. Nguyen).

https://doi.org/10.1016/j.comnet.2023.109935
Received 19 January 2023; Received in revised form 12 July 2023; Accepted 14 July 2023
Available online 20 July 2023

Packet-In message, the controller installs rules at the SDN switch to manage that flow [4]. The controller either allows the data packet flow to pass through the core network or simply drops the data packet flow at the gateway switch. Note that, with the increasing number of the new incoming data packet flows at the gateway switch, the number of Packet-In messages managed by the controller and the number of rules managing the incoming data packet flows installed at the gateway switch increases.

This type of deployment opens the door for a new type of attack called *Packet Injection Attacks* [5]. In this attack, the attacker can affect the services and performance of the SDN controller and can overflow the capacity of the SDN switch significantly by injecting malicious packets into the SDN network. Because the SDN controller has no mechanism to verify the legitimacy of the Packet-In message, whether it comes from matched or unmatched packets, adversarial users can exploit this principle of SDN by sending an excessive number of falsely crafted packets from multiple end hosts to flood the network [6]. Doing so ultimately stops the network from functioning in real time, leading to network breakdowns. Thus, packet injection attacks are a primary threat to software-defined enterprise networks, in which continuous connectivity and real-time network functioning are two essential requirements.

It is also important to note that while it is true that in some cases, unknown packets can be handled by gateway or default routers in traditional networks, this approach may not be feasible in all SDN deployments due to the different traffic patterns and routing mechanisms. In particular, SDN controllers rely on receiving packet_in messages for unknown flows or packets to dynamically update their forwarding rules and maintain network-wide consistency. As a result, an attacker who can inject such messages can cause the controller to overload or misbehave, leading to a range of security and performance issues.

Detection and defense against these types of attacks are hard. The recent solution techniques against *Packet injection* attack, discussed in Section 6, can stop malicious data packet flows from entering the network up to a certain level successfully, but these techniques cannot prevent an SDN switch from sending the malicious Packet-In messages to the controller. Furthermore, more recent low-profile attacks have shown to be able to bypass most of the current detection methods. To address these two open issues, we propose in this paper to use a separate detection method that does not involve the controller. Our solutions continuously monitor traffic in the network and use deep learning models to differentiate between benign and attack traffic. Once an attack is detected and the malicious nodes are identified, the controller installs a set of blocking rules at the switches that allow normal traffic but stop malicious traffic without triggering Packet-In events. The mitigation algorithm scales linearly with the number of switches, the number of hosts and the number of attacking nodes.

Our main contributions in this paper are

1. A novel deep learning model that uses graph convolutional neural networks to detect and identify various types of packet injection attacks in SDN with high accuracy. Our solution is the first that can detect and distinguish in real time between different types of attacks, such as slow/fast DoS in SDN.
2. A scalable SDN-based mitigation solution that helps defend networks against various types of packet injection attacks without disrupting normal traffic.
3. SDN emulation setup with Mininet that demonstrates the effectiveness of our solutions and generate for the first time a publicly (emulated) available dataset of packet injection attacks.

Our code and data are publicly available at:
https://github.com/nahaUoA/SDNPacketInjectionAttack.

The rest of this paper is organized as follows. We discuss the background information about *Packet Injection Attacks* and their impacts in Section 2. We then describe the models, hypotheses, and techniques used in this work to address limitations in current state-of-the-art packet injection defense in SDNs in Section 3. Section 4 explains the two-layer detection using Graph Convolutional Network (GCN) with the proposed attack detection and identification model. We discuss the performance and results of our techniques in Section 5. Related work is discussed in Section 6. Finally, we conclude the study in Section 7.

2. Background and motivation

In this section, we describe the motivation for our work by illustrating the effects of packet injection attacks on both the controller and SDN switches. We also discuss the different variants of packet injection attacks.

2.1. Packet injection attacks

By default, an SDN switch in reactive mode generates and sends a packet-in message to the controller when it receives an unknown data packet flow [4]. The controller then installs rules at the switch to manage the flow after receiving the packet-in message. The controller either allows a data packet to flow to the network or drops the flow at the switch.

The number of Packet-In messages increases proportionally to the number of incoming unknown data packet flows at the switch. The increasing number of packet-in messages places a heavy computational burden on the controller. Attackers can take advantage of packet-in messages to launch packet injection attacks on SDN networks. Here, the attacker sends an enormous number of malicious data packet flows to the switch, which slows down or breaks down the network.

Several recent studies (e.g., [3,5,7,8]) have demonstrated that packet injection attacks are a real and significant threat to SDN networks, especially when they target the control plane. Service Providers (SPs) have taken measures to mitigate them. For example, some SPs use hardware-based defenses, such as rate limiting, filtering, and deep packet inspection, to prevent malicious packets from reaching the SDN controller. Others deploy software-based solutions, such as intrusion detection systems, firewalls, and anomaly detection algorithms, to monitor and detect suspicious network activities and isolate compromised nodes. In general, the choice of defense mechanism depends on the SP's risk tolerance, budget, and expertise, as well as the specific characteristics of the network and the types of attacks it faces. However, it is worth noting that despite these defenses, packet injection attacks remain a challenging problem to address, as attackers can easily adapt their tactics to evade detection and exploit network vulnerabilities. Moreover, as SDN networks continue to evolve and scale, new attack vectors and vulnerabilities may emerge, requiring constant vigilance and innovation.

2.2. Threat models of packet injection attacks

Deng et al. [5] proposed an initial threat model for packet injection attacks in their seminal work. This threat model defines a packet injection attack as a DoS-type attack in which the attacker sends numerous malicious packets to an SDN switch ceaselessly. The majority of packet injection attack detection techniques discussed in Section 6 use this threat model to validate their proposed solutions. These techniques identify a network attack as a packet-injection attack when a malicious packet arriving at a switch exceeds a certain threshold for a defined period.

The threat model proposed by Deng et al. [5] cannot detect packet injection attacks when the malicious incoming rate to a switch is below a threshold or fluctuates frequently. In this study, we addressed the shortcomings of Deng's threat model by developing novel threat models that cover a wide variety of packet rates and injection patterns. More specifically, we propose two variants of packet injection attacks: low-rate packet injection attacks and discontinuous packet injection attacks. We explain these two new modes of attack in further detail below. It is also possible to have a hybrid-type packet injection attack that combines two or more variants of the packet injection attacks.

2.2.1. Low-rate packet injection attack

We define a low-rate packet injection attack as a variant of the packet injection attack in which the attacker's packet sending rate is much lower than the threshold value. For example, Khorsandroo et al. [9] showed that an attacker sending malicious packets at a rate of almost 1% of the total throughput can exhaust both the switches and the controller significantly. In these cases, the attacker's malicious packet sending rate to a switch does not cross the threshold value, rendering solutions based on detecting high packet rates ineffective. By launching this attack on multiple switches of a network simultaneously, the attacker can increase the computational burden on the controller to the same level as in high-rate packet-injection attacks.

2.2.2. Discontinuous packet injection attack

Another shortcoming of the current detection techniques is that they can only detect malicious high-rate attacks over a certain period of time. They fail when the rate of incoming malicious packets fluctuates around the threshold value within a fixed detection time window, even if the overall attack volume is very high. Attackers can easily bypass these detection techniques by following irregular sending patterns. We define this type of attack as a *discontinuous packet injection* attack in which the attacker's malicious packet sending rate frequently varies over time.

2.3. Impact of packet injection attack on the network

In packet injection attacks, an attacker intentionally sends a large number of forged data packet flows to the switch to overflow both the switch and controller capacity. The devastating effects of this attack on the network are discussed below.

2.3.1. Computational burden to the controller

By sending an enormous number of Packet-In messages to the controller, an attacker can keep the controller busy managing the malicious Packet-In messages. The increasing number of Packet-In messages proportionally increases the controller's CPU utilization. This results in a scarcity of resources for the SDN controller to manage the network's regular functionalities. That eventually stops the controller from functioning in real-time, leading to an enterprise network breakdown.

For example, Deng et al. [5] showed in their experiment that when the controller received a Packet-In message at a rate of 1600 packets/s, it stopped accepting any new Packet-In messages of the network. In our experiment, we used the same network topology as Deng et al. [5] and observed that the controller stopped functioning in real-time when the incoming packet-in message rate at the controller exceeded 600 packets/s.

2.3.2. Rule-space overflow of the SDN switch

The majority of techniques discussed in Section 6 mitigate packet injection attacks by installing rules at the switch in which each rule is commonly used to manage a request, such as blocking a specific data packet flow. This type of rule functions mainly by dropping malicious data packet flows at the switch. The number of rules installed at a switch is proportional to the number of malicious data-packet flows encountered by the switch. However, the rule space itself can be a target for packet injection attacks.

A typical SDN switch can hold approximately 55 000 rules at a time [10], and it can send or receive approximately 18 000 requests/s at its busiest hour [11]. A switch cannot hold a new rule to manage a new request if its rule space is occupied. It was shown in [5] that an attacker can easily launch a packet injection attack by sending malicious data packet flows at a rate of 60 000 requests/s to the network. In this case, the switch's rule space overflows as its rule space exceeds the capacity, that is, 55 000 rules. Thus, it is essential to limit the number of installed rules to manage packet injection attacks in the rule space of the switches.

3. Proposed technique

3.1. Design hypotheses

We assume that

1 A database server stores the network information as a *3 tuple* instance, *that is* , *<MAC addresses, IP addresses, Switch ID:port no>* in a list called $Network\_List$. The controller obtains full access to the database server by using a simple password-based authentication system.
2 The switches work in reactive mode and any data packet flow coming to the switch contains its source and destination address as *3 tuple* instance, *i.e.*, *<MAC addresses, IP addresses, Switch ID:port no>* in the network, similar to the PacketChecker technique [5].

3.2. Defense strategy

Based on flow-based traffic, we identify benign users and attackers using a trained classifier with a Graph Convolutional Network (GCN). Section 3.3 provides a detailed description of the attackers' detection technique. The technique models the activity flows of devices using a GCN and applies deep learning methods to detect nodes with abnormal behaviors. After detecting the attackers, we classify the attacks into attack types such as DDoS and PortScan attacks — two popular packet injection attacks in existing datasets. Note that our solution is capable of identifying other types of attacks in addition to these two types. Once we identify and classify the attacks, we mitigate them by adding new blocking rules to the switch. We also maintain a list to observe the network activities. Following the blocking rule, the switch discards incoming packets from a malicious device. The observation list analyzes traffic flow to identify possible attacks.

3.3. Attack detection technique

3.3.1. Attack detection

We use flow-based traffic data to characterize the activities of both benign users and attackers. By analyzing the traffic generated from each source, we were able to extract the characteristics of the sources and train our classifier to identify attack sources. Fig. 1 shows the general pipeline for our packet injection attack detection and classification.

We first consider all traffic that is not "benign" in the dataset to be "attacks" attack traffic. In this way, we simplify the problem into binary classification, in which the two classes are "benign" and "attack". Similar to [12], we group packets into basic flows, where all flows with the same SrcIP:SrcPort and DstIP:DstPort are grouped together, and the mean of their attributes is computed to represent the overall characteristics of the flow. We further group basic flows into activity flows, where newly generated basic flows are grouped together based on the source IP and source port. Again, the mean is used to aggregate the features of the flows. We attempted to use protocols and count the number of destination ports as features, but eventually discarded them because they did not yield a positive result. For nodes that are not associated with any output flows, we perform statistical analysis on the low profile benign sources — gathering the distribution of their features, populating the features of those nodes by sampling from those distributions, and labeling them as "benign" nodes.

By performing a statistical analysis on the traffic flows transmitted by a source node, we can characterize the time-based features of each node. Attacker nodes tend to have traits that are different from regular sources because of the irregular activities they carry out in attempts to hijack the network. Finally, we obtain a new representation of data that contains information on the characteristics of each source, allowing our classifier to learn from this information and perform the predictions.

Using this new representation, we apply a particular variation of GCN, called HyperGCN [13] to identify hosts with malicious packet

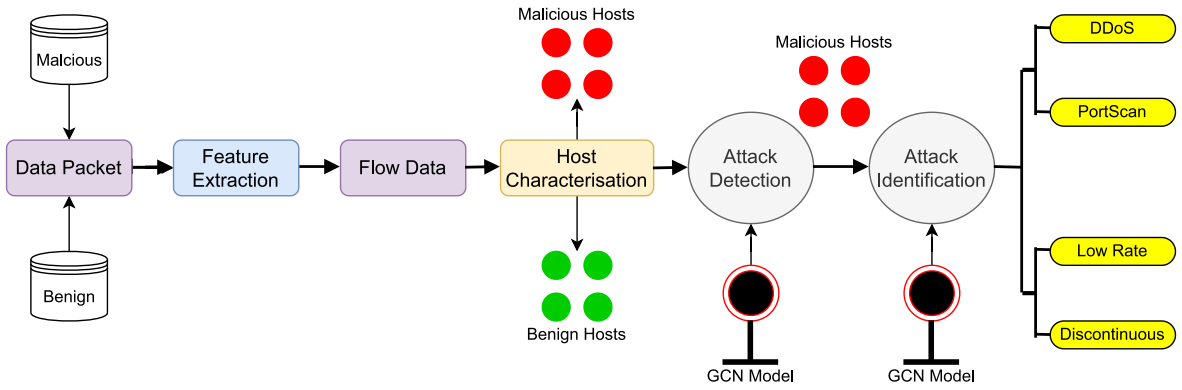


Fig. 1. General pipeline of the attack detection and classification architecture.

injection attack patterns. A hyperGCN is a deep learning model that operates on non-Euclidean structures and allows multiple connections between two nodes. HyperGCN [13], which is a variation of GCN that operates on hypergraphs, is capable of learning structural information and localized information between data samples represented as non-uniform structures, such as graphs, and is suitable for packet injection attacks with multiple flows between switches. Here, we use the HyperGCN to solve the problem of network intrusion detection and classification. This problem can be considered graph-based semi-supervised learning, where the graph represents a network with users as nodes and connections between users as edges. Each node was associated with a unique feature vector. Moreover, in addition to the node features, the GCN also considers connectivity in the network. This utilizes the power of the graph structure, thus providing useful information for the learning process when attackers effectively hide their traits and generate traffic that is indistinguishable from normal traffic. By incorporating the graph structure into the learning process, it is expected that the model can handle scenarios in which the links between nodes contain information not present in the data.

As shown in Fig. 3, the detection module uses traffic flows as the input. These flows are collected by mirroring the traffic in the network. The detection module produces a list of suspicious packet injection nodes.

3.3.2. Attack identification

Once we detect malicious hosts, we want to identify the type of malicious activities in which the malicious host participated.

We extend the GCN algorithms to solve multiclass classification tasks by using them to classify the characterized hosts. A similar approach for representing the data was adopted in the Attack Detection stage. In this stage, the algorithms operate only on malicious samples of the dataset, facilitate the classification process, and potentially improve the classification performance.

Using the same graph structure as in the *Attack Detection* model, we classify attack vectors on a set of malicious hosts. In this *Identification* module, semi-supervised learning with a GCN was modified to handle multiclass classification. Benign nodes that are known from the *Detection* module are deactivated at this stage of the pipeline. Only malicious nodes were used to identify attack variants. Labels based on attack classes, whether DDoS/PortScan for CICIDS data or low-rate/discontinuous for SDN applications are assigned to the training portion of the graph model. The GCN algorithms effectively learn from the provided data and propagate the information to identify the attack classes of the other hosts in the network.

The output of the *Identification* module is a list of suspicious nodes with DDoS or PortScan classification labels. This list is then passed to the controller to implement mitigation solutions against malicious nodes.

3.4. Attack mitigation technique

The last step in our solution is the mitigation technique, which can effectively fend off various types of packet injection attacks on SDN without overloading the controller. The controller runs Algorithm 1 repeatedly to mitigate packet injection attacks at the switch level.

Algorithm 1 Attack mitigation algorithm

Input: (1) $\mathbb{S}$ (2) $\mathbb{E}$ (3) $\mathbb{N}$
(4) $Packet\_In < Pkt\_In_{src}, Pkt\_In_{dest} >$

Output: (1) $Block\_List$

Initialize $k = 1$, where $k \in \mathbb{R}$
for all $\mathbb{S}$ such that $k \leq K$ **do**
  Initialize $i = 1$, where $i \in \mathbb{R}$
  for all $\mathbb{E}$ such that $i \leq I$ **do**
    if $Packet\_In \in \mathsf{E}$ **then**
      Install *Pkt-blocking* rule at switch S_i
      Update $Block\_List \longleftarrow Pkt\_In_{src}$
      Break
    else
      Initialize $j = 1$, where $j \in \mathbb{R}$
      for all $\mathbb{N}$ such that $j \leq L$ **do**
        if $Packet\_In \in \mathsf{N}$ and $|Pkt\_In_{dest} \cap \mathsf{N}_j| \neq 3$ **then**
          Update $Observing\_List \longleftarrow Packet\_In$
          Break
        else
          Continue
        end if
      end for
    end if
  end for
end for

At an abstract level, the algorithm follows a number of steps. First, the algorithm initializes variables and sets the necessary data structure. It then iterates over the switches in the network, and within each switch, iterates over entries related to the observation of suspicious hosts. For each Packet-In message received, it checks for matches with entries. If a match is found, a packet-blocking rule is installed at the corresponding switch and $Block\_List$ is updated. If no exact match is found, it checks for partial mismatches with entries related to the network. In the case of partial mismatch, the source address is added to $Observing\_List$. If there is an exact match with an entry into the network, a rule allowing the data packet to flow into the network is installed. This algorithm is recurrently executed by the controller to effectively mitigate packet injection attacks in the SDN network, leveraging the identification of malicious packets through field mismatches.

Table 1
Input parameters of Algorithms 1.

Parameter	Representation
$\mathbb{S}$ = The set of all switches $\mathbb{E}$ = The set of all entries of $Observing\_List$ $\mathbb{N}$ = The set of all entries of $Network\_List$	$\lvert\mathbb{S}_{\mathbb{E}}\rvert = K$ and $\mathbb{S}_{\mathbb{E}} = [S_1, S_2, ..S_K]$ $\lvert\mathbb{E}\rvert = I$ and $\mathbb{E} = [E_1, E_2, ..E_I]$ $\lvert\mathbb{N}\rvert = L$ and $\mathbb{N} = [N_1, N_2, ..N_L]$
$Packet\_In_{src} := \langle src\_IP, src\_MAC, src\_SwitchID : Port\_ID\rangle$	⟨Src IP address, Src MAC address, Src switch-port number⟩
$Packet\_In_{dest} := \langle dest\_IP, dest\_MAC, dest\_SwitchID : Port\_ID\rangle$	⟨Dest IP address, Dest MAC address , Dest switch-port number⟩

The inputs to Algorithm 1 are listed in Table 1. Algorithm 1 takes inputs from $Observing\_List$ and $Network\_List$ lists and keeps updating $Block\_List$ list recurrently. Note that $Network\_List$ contains the information of all hosts (benign and suspicious hosts) connected to the network, $Observing\_List$ contains the information of suspicious hosts, and $Block\_List$ stores the information of the attackers. These lists keep the host (benign, suspicious, and attacker) information as 3−tuple instances, *i. e.*, <IP addresses, MAC addresses, and Switch ID:port nunmer>.

In an SDN network, when a switch encounters an unknown data Packet _In flow, it sends $Packet\_In$ messages to the SDN controller. $Packet\_In$ messages contain the source address, $Packet\_In_{src}$, and destination address, $Packet\_In_{src}$, of that flow. The controller matches the source address of the received $Packet\_In$ message, $Packet\_In_{dest}$, with the entries in $Observing\_List$. The controller installs the *Pkt-blocking* rule at the switch to drop the data Packet _In flow immediately (steps 2 to 8 in Algorithm 1) when it obtains an exact match between $Packet\_In_{src}$ and an entry of $Observing\_List$.

If the controller obtains a mismatch between $Packet\_In_{src}$ and an entry of $Observing\_List$, it further matches the destination address of the received $Packet\_In$ message, $Packet\_In_{dest}$, with the entries of $Network\_List$. It adds the source address of the $Packet\_In$ message, $Packet\_In_{src}$, to the $Observing\_List$ (steps 10 to 14 in Algorithm 1) if it obtains a partial mismatch between $Packet\_In_{dest}$ and an entry of $Network\_List$. Here, the logic is that malicious packets sent by packet injection attackers generally cannot have all fields correctly set (hypothesis) because every single packet must be classified by the switch as "unknown" to be forwarded to the controller. On the other hand, the controller installs a rule at the switch, allowing the data Packet _In flow into the network (step 16 in Algorithm 1) when it obtains the exact match between $Packet\_In_{dest}$ and an entry of $Network\_List$.

3.5. Complexity analysis

Algorithm 1 operates in a nested loop structure, which iterates over the sets of switches, entries, and network entries. Firstly, the initialization step sets up variables and data structures, which is a constant-time operation, denoted as $O(1)$.

The algorithm then enters the first loop, iterating over the set of switches in the network. Assuming there are K switches, this loop has $O(K)$ complexity. Inside this loop, the algorithm enters another loop, iterating over the set of entries related to observing suspicious hosts. If there are I entries per switch, this inner loop has a $O(I)$ complexity.

Furthermore, if there is an exact match between the received Packet-In message and an entry in the network, the algorithm installs a rule allowing the data packet to flow into the network. This operation has a $O(L)$ complexity since it may need to compare the $Packet - In$ message with each network entry.

Considering the nested loops, the total complexity of the algorithm is the product of the complexities of each step. Therefore, the overall complexity can be represented as $O(K * I * L)$, where K represents the number of switches, I represents the number of entries, and L represents the number of network entries in the SDN network. Hence, the run time of Algorithm 1 is $O(KIL)$ — linear with the numbers of switches, hosts, and attackers.

The complexity analysis shows that Algorithm 1 has a polynomial complexity in terms of the number of switches, entries, and network entries involved. As the size of the network increases, the execution time of the algorithm scales. Understanding the algorithm's complexity helps assess its efficiency and scalability in handling larger SDN networks.

4. A two-layer detection model using GCN

We propose a two-layer detection model or a dual model architecture to detect whether the incoming source of the packets is an attacker and identify the type of attack. Furthermore, we attempt to provide mitigation techniques depending on the type of attack and implement them in the SDN controller.

4.1. Graph convolutional network

As described in [14], Graph Convolutional Network (GCN) is a graph-based neural network model, which has the layer-wise propagation rule as presented below

$$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{1}$$

The model f(X, A) is flexible and provides sufficient information propagation across the graph, and is useful for the problem of semi-supervised node classification. In this class of problem, adjacency matrix A and data X of the graph structure are used for training the model. We used a two-layer forward model in this network intrusion detection, which is similar to the forward model in [13]:

$$Z = f(X, A) = softmax(\hat{A}ReLU(\hat{A}XW^{(0)})W^{(1)}) \tag{2}$$

with

$$\hat{A} = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}} \tag{3}$$

The weights were trained using gradient descent. The full dataset was used for every training iteration because the training process propagated through the entire graph in each part of the training. We also included dropout for stochasticity, and weight decay was used to prevent overfitting.

HyperGCN [13] is a new method that addresses the problem of training GCN using hypergraphs. This method approximates the hypergraph by a graph, where each hyperedge is represented by a subgraph with connections to that hyperedge. The connections involve the edge between the maximally disparate nodes and the edges of these two nodes with all the other nodes in the hyperedge. This operator is called the hypergraph Laplacian. After applying the Laplacian hypergraph to all existing hyperedges, an approximation graph is constructed, and the GCN algorithm is run on the resulting graph.

We applied HyperGCN to solve the problem of network intrusion detection and classification. This problem can be considered graph-based semi-supervised learning, where the graph represents a network, with users as nodes and connections between users as edges, as shown in Fig. 2. Each node is associated with its unique feature vector. By performing statistical analysis on the traffic flows that were transmitted by a source node, we can characterize the time-based features of each node. Attacker nodes tend to have traits that are different from regular sources because of the irregular activities it carries out in attempts to

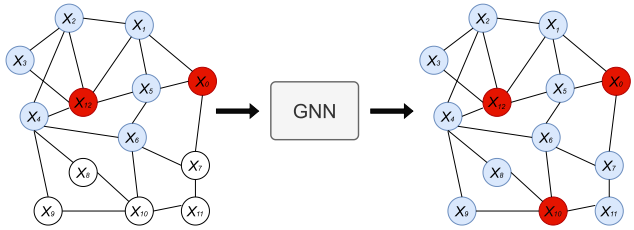


Fig. 2. Semi-supervised node classification for intrusion detection with Graph Convolutional Networks (GNN), blue nodes denote benign users, red nodes denote attackers and white nodes denote unlabeled users. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

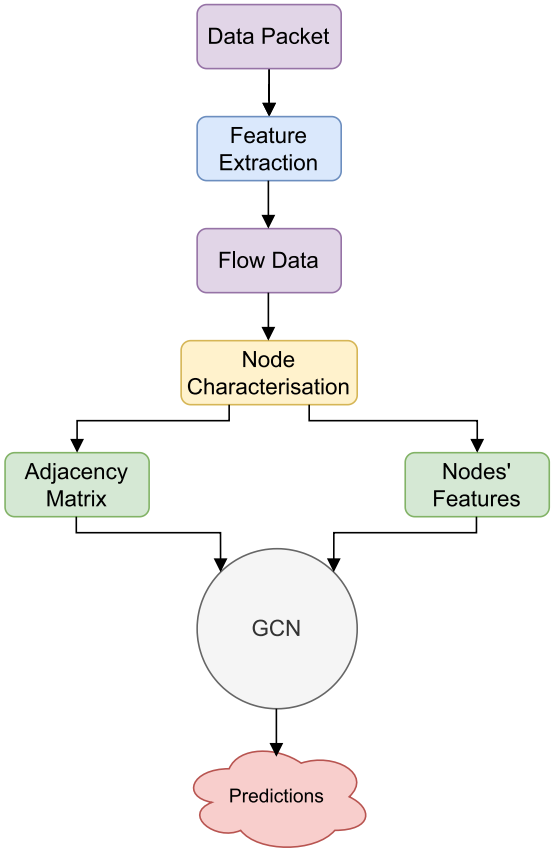


Fig. 3. General pipeline of intrusion detection model.

hijack the network. Moreover, apart from the node features, the HGCN also considers connectivity in the network. This utilizes the power of the graph structure, thus providing useful information for the learning process when attackers effectively hide their traits and generate traffic that is indistinguishable from normal traffic. By incorporating the graph structure into the learning process, it is expected that the model can be extremely powerful in scenarios where the links between nodes contain information not present in the data. To the best of our knowledge, our work is the first attempt to apply GCN to network intrusion detection.

4.2. Proposed detection model

4.2.1. Layer 1: Attack detection

We used flow-based traffic data (CIC-IDS2017) to characterize the activities of both benign users and attackers. The details of this dataset are presented in Section 6. By examining the traffic generated from each source and using some data processing techniques, we were able to extract the characteristics of the sources and train our classifier to identify the attacking sources.

We first consider all traffic that is not "benign" in the dataset to be "attacks" attack traffic. In this manner, we simplified the problem into binary classification, in which the two classes are "benign" and "attacker". For the GCN method, the graph structure must be extracted from the traffic data. We did this by representing each IP:PORT sends out or receives flow as a node in the graph, and the edge between nodes exists if there are flows between them. Based on the technique presented in [12] techniques, the feature vector of the nodes are obtained using the following steps, or as illustrated in Fig. 4:

Step 1: Group the packets into *basic flows* — all flows with the same SrcIP:SrcPort and DstIP:DstPort are considered in the same group.

Step 2: Take the average of attributes, to obtain an overall representation of each *basic flow* object.

Step 3: Further group the *basic flows* into *activity flows*, by considering all the objects with the same SrcIP:SrcPort in the same group.

Step 4: Use mean aggregation to obtain a feature vector for each *activity flow* object.

Step 5: The aggregated *activity flow* is a node in a graph.

Regarding nodes that are not associated with any output flows, we perform statistical analysis on the low profile (few edges) benign sources — gathering the distribution of their features, populating the features of those nodes by sampling from those distributions, and labeling them as "benign" nodes. Finally, we obtained a new dataset that contained information on the characteristics of each source, allowing our classifier to learn from this information and perform the predictions.

4.2.2. Layer 2: Attack identification

We used the CICIDS2017 dataset for experiments related to attack identification. This dataset contains the most up-to-date types of attacks that hackers attempt to use for network intrusions. However, as mentioned earlier, we only consider two classes of attacks, namely, Port Scan and DDoS attacks, because of the number of samples that are present in the dataset. Benign traffic and other kinds of attacks except Portscan and DDoS were removed from the dataset to keep the aggregation clean and maintain the traits of each type of attack. Furthermore, we also generated a new dataset that contains different types (e.g., slowDDoS, fastDDoS, and discontinuousDDoS) of DDoS attacks that are common in SDN, and perform classification using the same algorithms on this newly generated dataset.

5. Evaluation

In this section, we first present the datasets and emulation details. We then report our evaluation results.

5.1. Data sets

We used CICIDS2017 [15] dataset for the evaluation of our proposed solution. We selected the CICIDS2017 dataset because it contains benign and up-to-date attack traffic, which was captured by the Canadian Institute for Cybersecurity (CIC) in a format that resembles real-world data (PCAP).

CICIDS2017 dataset contains 10 types of attacks - BruteForce, SFTP, SSH, DoS, Heartbleed, slowloris Slowhttptest, Hulk, Infiltration, DDoS and Portscan. We focused mainly on DDoS and PortScan attackers, discarding other types of attacks, and performed classification between these two types of attacks to evaluate our algorithms. The reason these

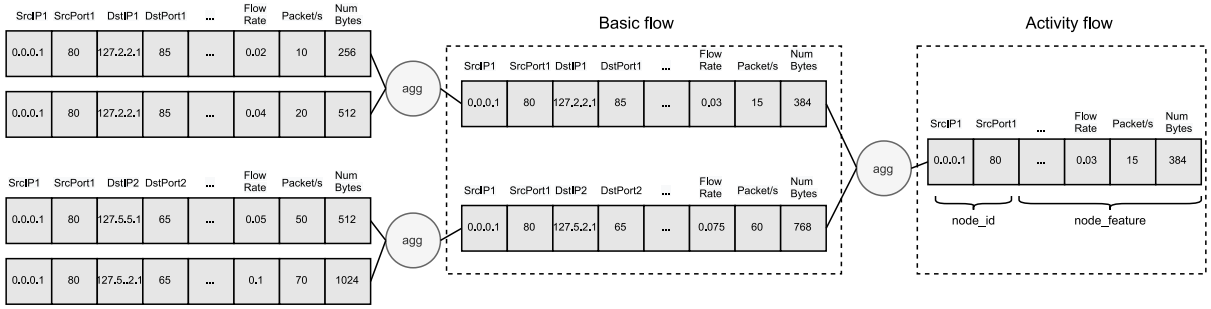


Fig. 4. Node Characterization process for extracting nodal features.

two types of attacks are chosen is twofold — (i) the majority of the samples 52% in the dataset belong to these two classes and (ii) these two are the most commonly reported attacks on SDN [16,17]. Specifically, there were 2,103,072 samples of benign traffic and 554,375 samples of attacking traffic. Among the attacking traffic samples, there were 128,025 instances of DDoS traffic and 158,804 instances of Portscan traffic, which accounted for 52% of the total instances in the dataset.

DDoS: Distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.

PortScan: A port scan is an attack that sends packets to a range of server port addresses on a host, aiming to explore the host and discover the vulnerability for potential attacks.

To perform the classification, we first performed traffic analysis to extract the activity patterns of each host, such as the flow duration and flow packets/s. We then used ML algorithms to classify the benign and malicious sources. We further utilize our algorithms to classify different types of attackers, forming a two-layer model that can distinguish between benign users and attackers, as well as the type of attackers in the network with high confidence.

5.2. Mininet emulation and data

5.2.1. Emulation setup

Ubuntu 20.04 LTS was selected as the operating system for the simulation environment. That includes mininet V2.2.2, Open vSwitch (OVS) V2.13.0 and Python 3.8, which are the necessary software to build the emulation environment. In addition, xterm should be installed to simulate hosts and install it with the following command.

```
1  sudo apt-get update -y
2  sudo apt-get install -y xterm
```

Furthermore, Ryu which is a controller software supporting python also needs to be installed. Use the following command to install, please pay attention to the version of python.

```
1  sudo apt-get update
2  sudo apt-get install git -y
3  git clone git://github.com/osrg/ryu.git
4  cd ryu/
5  sudo python ./setup.py install
6  Finally, CouchDB is installed through subordinate
       commands.
7  curl -L https://couchdb.apache.org/repo/bintray-pubkey
       .asc | sudo apt-key add-
8  echo "deb https://apache.bintray.com/couchdb-deb focal
       main" | sudo tee -a /etc/apt/sources.list
```

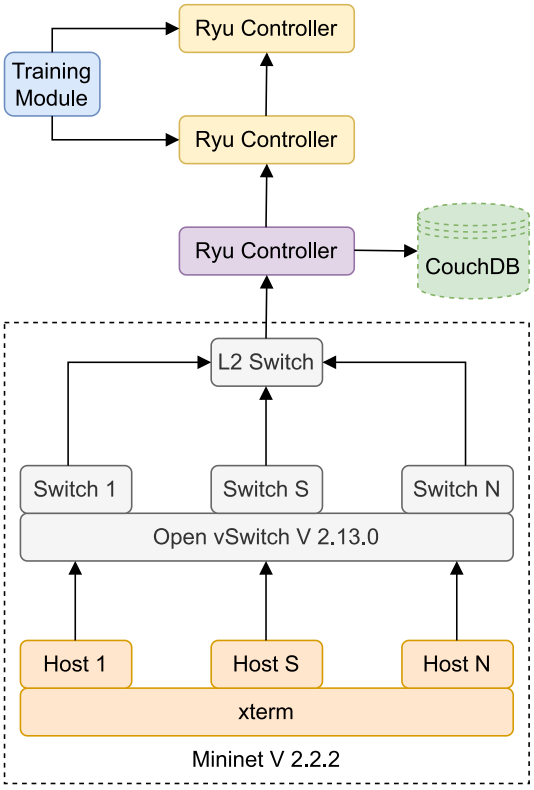


Fig. 5. Emulation environment architecture.

```
9  sudo apt update
10 sudo apt install couchdb
```

After the installation, the mininet emulation architecture is created as shown in Fig. 5.

5.2.2. Dataset generation

We performed our analyses on two datasets: CICIDS2017 and our own DDoS variant dataset. CICIDS2017 is a publicly available dataset that contains samples of benign network traffic and different classes of cyberattacks, including DDoS. We generated a DDoS variant dataset by using an emulated SDN environment. By varying the various injection rates and methods, we produced several variants of DDoS attacks in the DDoS Variants dataset, including slow/fast, discontinuous, and hybrid attacks.

To generate the DDoS Variants dataset, an SDN environment was established using Mininet [18] with OpenFlow switches and a Ryu controller. Fig. 6 shows a typical network topology of two hosts using

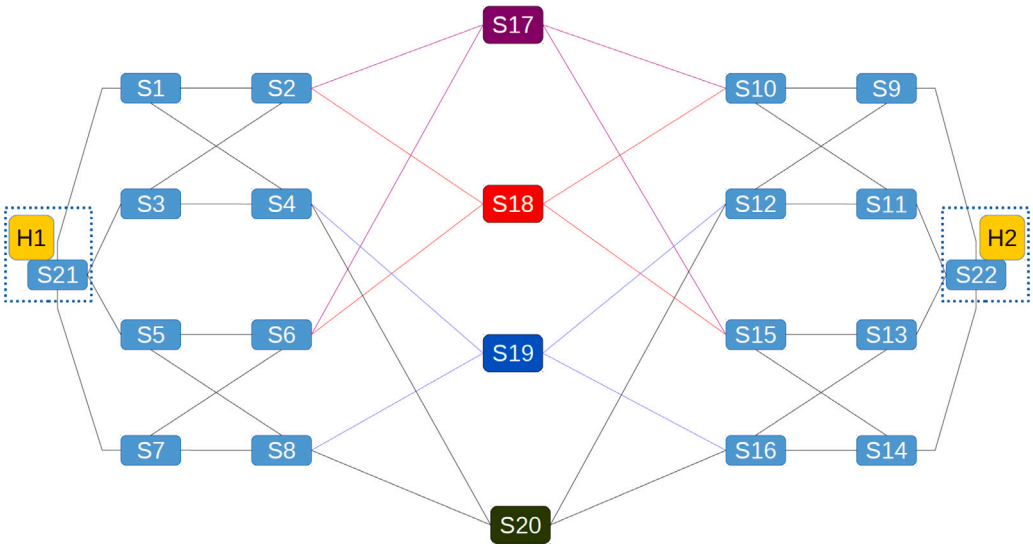


Fig. 6. Network topology.

Table 2
Properties of DDoS variants.

Variant	Rate (packet/s)	Sleep timeout (s)
slowDDoS	5–100	N/A
fastDDoS	1000–20 000	N/A
slowDcDDoS	5–50	3-7
fastDcDDoS	1000–20 000	3-10

Table 3
HyperGCN parameters configuration.

Params	Values
`Rate`	`0.15`
`Layers`	`2`
`Activations`	`128`
`Decay`	`0.0005`
`Dropout`	`0.5`

22 switches that were used in our evaluation. Other topologies can be used without affecting the algorithm results.

For packet injection attack generation, we used *nping* tools to simulate attack traffic between hosts in the emulated network. Each host was assigned an IP address and carried out attacks from different port ranges. For the purpose of this work, we simulated four variants of DDoS attacks: fastDDoS, slowDDoS, fastDiscontinuousDDoS, and slowDiscontinuousDDoS. Table 2 presents the properties of each DDoS variant. 'Rate' parameter is adjusted to vary the rate of DDoS attacks and the 'sleep timeout' parameter is introduced to vary the continuity property.

In the emulation, we captured the data at the packet level. We then used the CIC-Flowmeter software to extract the statistical features of the traffic flow in real-time. A traffic flow can be defined as packets within an arbitrary period that share the same source/destination IP, source/destination port, and transmission protocol. Some of the extracted features include Flow Duration, Number of Packets, and Number of bytes.

5.2.3. Data from emulation

After analyzing the traffic data, we obtained a new dataset that contained the time-based characteristic data of each flow source. We stored this in csv format to classify it with other ML techniques. For the HyperGCN algorithm, we used traffic data to extract the graph. In this graph, each IP:PORT combination that sends out a flow to another IP:PORT is considered as a node, and an edge between two nodes is constructed if there is a flow between them. Each node is associated with features based on its flow pattern, and based on these features, the HyperGCN, as well as other ML techniques, can identify whether they are attackers in the network. If they are, they further classify the types of attacks they involve.

We used the same approach as in the attack detection problem to characterize the flow pattern of each IP:PORT node. However, some sources involve both port scans and DDoS attacks. We addressed this problem by eliminating the DDoS traffic and maintaining the scan. Hence, making the pattern for DDoS and Port Scan more distinct is closely aligned to the real-world situation, where a source can only send out one attack type at a time. We chose to eliminate DDoS traffic instead of port scans because of the imbalance of the dataset. By maintaining Portscan traffic and discarding DDoS traffic from the sources that send out both, we were able to maintain a more balanced ratio between DDoS and Port Scan traffic in the dataset. For nodes that had no output flow, we assigned them as "Noflow benign", filling their feature vectors with zeros.

5.3. Results

We present results with respect to three parts — attack detection, attack identification, and attack mitigation.

5.3.1. Attack detection

The HyperGCN model is implemented using PyTorch. In our implementation, we applied the neural network parameters as in Table 3 to avoid overfitting. The configuration was obtained via the trial-and-error hyper-parameters tuning method.

We compare our HyperGCN-based detection solution against other common machine learning frameworks such as

1. Random Forest (RF) [19]: RF is a supervised machine learning algorithm that is based on decision trees. It can be considered as an ensemble of decision trees that operate on various subsets of the dataset, and the final output is based on the prediction that occurs the most among all decision trees. The RF algorithm was implemented with the number of decision trees set to 100, maximum depth set to 3, and random state set to 32. In addition, with the criterion set to 'entropy,' 80% of the datasets were randomly allocated for training, and 20% for testing.

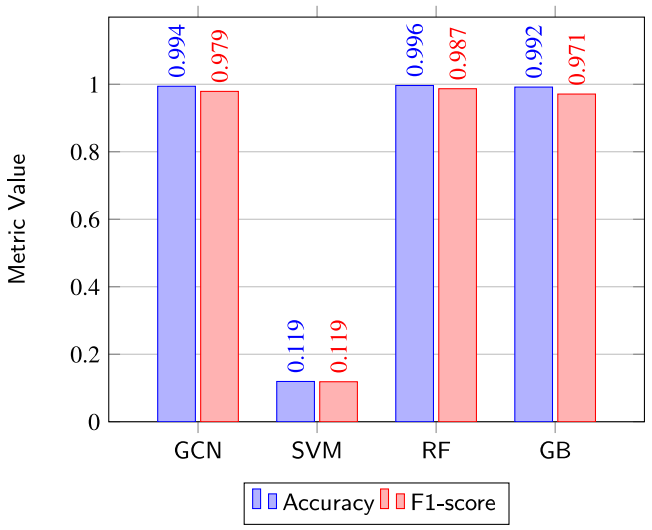


Fig. 7. Attack detection with CICIDS dataset.

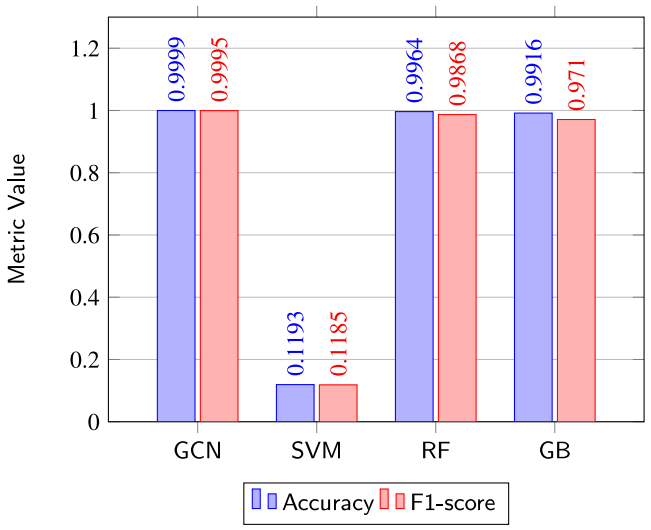


Fig. 8. Attack identification with CICIDS dataset with fixed sample.

2. Support Vector Machines (SVM) [20]: SVM is another classic machine learning algorithm that is tailored for feature-rich data. The algorithm constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space that separates the classes of samples. For the SVM algorithm implementation, we performed training via SVC, where the kernel parameter was set to 'rbf,' C was set to 1, gamma was set to 'auto,' and the maximum number of iterations was set to 8000. Additionally, the dataset was randomly allocated to 80% for training and 20% for testing.
3. Gradient Boost (GB) [21]: Gradient boosting is a form of boosting that is used in machine learning. It is based on the assumption that when the best feasible feature model is merged with the previous ones, the overall prediction error is minimized. To decrease this error, it is important to establish the intended outcomes for the next model. The GB algorithm is similar to the RF algorithm in that it is a decision-tree-based machine-learning algorithm, but the key parameters are different. The n_estimators and learning_rate parameters were set to the default values of 100 and 1. The maximum depth parameter was set to 3. The random allocation of datasets is the same as that for RF and SVM, with 80% being used for training and 20% for testing.

The CICIDS dataset includes a combination of attacks and benignity. In the dataset, we have to deal with the situation of imbalanced data, which is common in cybersecurity, where the number of benign users accounted for most of the samples (387 365 samples), while the number of attackers is much fewer (48 396 samples). To avoid overfitting, we selected 20k samples randomly from the benign user class, and 20k samples randomly from the attacker class to construct our training set, and then used the remaining samples for our testing set. Fig. 7 shows the attack detection results using GCN, SVM, RF and GB machine learning algorithms. SVM underperformed in terms of accuracy and the F1-score. However, all three other algorithms (e.g. GCN, RF, and GB) attained similar performances for accuracy and F1-score. Furthermore, RF outperformed all the other algorithms, and its detection accuracy was close to 100

We also note that even though GCN provides the best overall performance in terms of both accuracy and F-1 scores, other ML approaches, such as RF and GB, have very similar performance. However, GCN could generalize better than the other approaches, as demonstrated later.

5.3.2. Attack identification

We further break down the attacker class and classify the type of attacks they are performing, focusing on DDoS and PortScan. This is the second layer of our intrusion-detection solution. There is also a class imbalance with these classes, with 34 254 recorded DDoS attackers and 14 142 cases of portscan; the remaining 999 nodes were marked as "benign noflow". The GCN model was able to handle this imbalance with a train/test split of 5 percent training and 95 percent testing on all the samples. We also changed the number of hidden layers to three to improve the performance of the GCN. The overall accuracy and F1-score of the GCN were 0.9998508 and 0.9995359, respectively.

Comparisons with other approaches: Again here, we compare GCN against the performance of other machine learning algorithms, namely Random Forest, SVM and BoostingTree. In this evaluation study, the same datasets were divided such that 20k attack samples and 20k benign users were randomly selected, and the rest of the data were used for prediction. We refer to this data division process as fixed sampling. In addition, based on the same ratio, the dataset division is 80 percent of the data for training and 20 percent of the data for prediction. F1-score and accuracy are very important attributes for researching the algorithm's performance. For an imbalanced class data distribution, the F1-score is better for evaluating the model because it calculates the harmonic mean of precision and recall.

Fig. 8 shows the results for a fixed sample for different machine learning algorithms. GCN outperformed the SVM, RF, and GB in terms of accuracy and F1-score. However, SVM showed worse performance, similar to attack detection. Although the attack identification accuracies of GCN, RF, and GB are very close to 100%, if we consider the F1-score, GCN performs better than all the other algorithms. Because it deals with an imbalanced dataset, a better F1-score proves that the GCN is the best for attack identification.

The experimental results of the 80–20 training and testing datasets are shown in Fig. 9. According to the figure, SVM showed better performance than the fixed dataset sample. However, the 0.52 F1-score of the SVM makes it unsuitable for attack identification. In contrast, GCN, RF, and GB showed the same performance for both accuracy and F1-score. Therefore, we can conclude that fixed sampling is the best for testing the performance, and the F-score metric is the best for evaluating attack identification algorithms.

We tested all algorithms with varying numbers of training samples for further evaluation. Fig. 10 shows the test results for F-score with the varying number of training samples. We varied the test sample size to 50, 250, 500, 750 and 1000 samples for each experiment. The

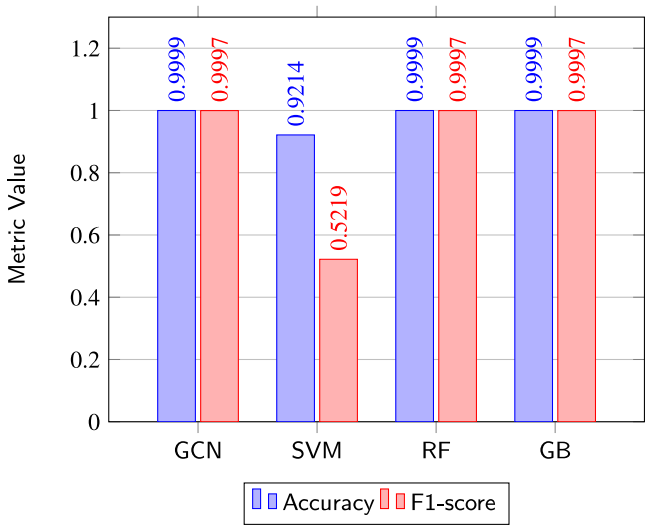


Fig. 9. Attack identification with CICIDS dataset with 80–20 training and testing.

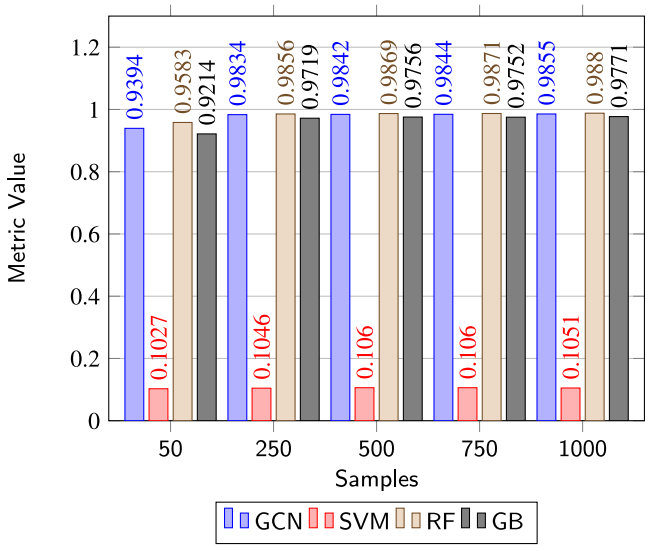


Fig. 10. Performance when varying number of training samples.

results show that the GCN, RF, and BB performances are similar and improve with an increasing number of training samples. However, the performance of SVM is not satisfactory. Testing with 50 samples did not yield consistent results compared with 250, 500, 750, and 1000 test samples. From these results, we can say that we need at least 250 test samples to have a consistent F-score. Furthermore, we can conclude that the GCN and RF algorithms exhibit outstanding performance. In addition, although the performance of the RF algorithm is the best among the algorithms used for comparison, the performances of GCN and RF are very similar, and the performance difference is only 0.3%. Hence, we can conclude that the GCN or RF can be used for attack identification. Because GCN is computationally expensive compared with RF, RF can be used if we consider the computation. However, the GCN is significant if we do not want to worry much about feature engineering, and it performs better with large datasets.

Following the experimental results from the CICIDS dataset, we utilized the GCN and measured the F-score for the DDoS variant dataset. Fig. 11 shows F1-score results for SlowDDoS, FastDDoS, SlowDcDDoS and FastDcDDoS variant dataset. For all cases, the GCN showed better performance, as expected. As shown in Fig. 11, the F-score is

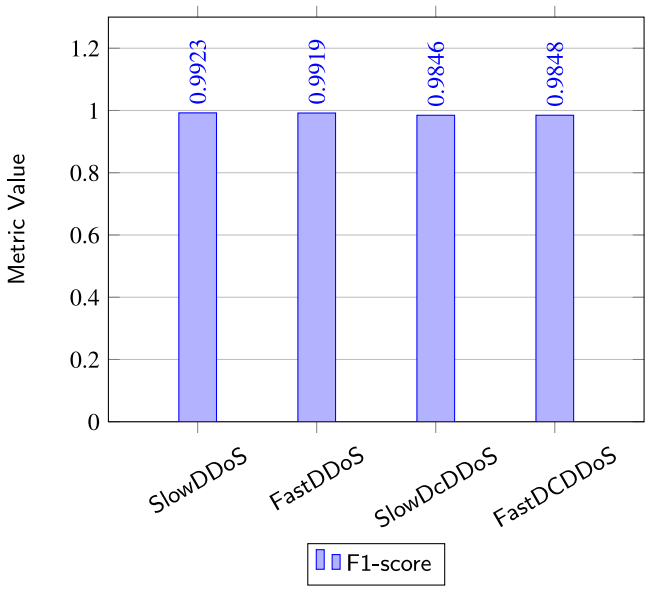


Fig. 11. Attack identification with DDoS variant dataset.

between 0.98 to 0.99, which is excellent for an imbalanced dataset. Furthermore, the overall attack identification accuracy was 0.99.

5.4. Attack mitigation

In the proposed defense mechanism, identifying an attack is important because we can quickly implement the rules in the switch through the controller once the attack is identified. However, the process should not consume excessive resources for the controller to collapse. Therefore, we need to prove that the CPU and memory utilization is lower while the attack mitigation technique is running on the controller. We tested the mitigation technique in four different scenarios: (i) observe CPU and memory utilization every 0.5 s while the system is in idle mode, (ii) observe CPU and memory utilization every 0.5 s while the system employs the mitigation technique; (iii) observe CPU and memory utilization every 5 s while the system is in idle mode; and (iv) observe CPU and memory utilization every 5 s while the system employs the mitigation technique. We initiated the DDoS attack with a packet sending rate of 100–500 s. We logged the utilization every 0.5 and 5 s to understand the resource utilization behavior of the controller. The experimental results for these four scenarios are shown in Figs. 12, 13, 14, and 15.

Fig. 12 shows the utilization of CPU and memory in idle mode with 0.5 s utilization capturing interval. According to the figure, the maximum CPU utilization was 13.3%, and the minimum value for CPU utilization was 0. The mean value and standard deviation are 4.93 and 3.19, respectively, suggesting a significant variation in CPU utilization over time. However, CPU utilization did not exceed 13.3% in the idle mode. After applying the mitigation technique in the same scenario with a DDoS attack, the CPU utilization increased to a maximum of 35.3% (see Fig. 13). However, the mean value and standard deviation are 5 and 4.72, respectively, which suggests that CPU utilization variation is high. However, reaching 35.3% CPU utilization is not that frequent. We did not find any difference in memory utilization, which remained the same over time. It is clear that CPU and memory utilization do not differ significantly from our proposed mitigation technique. During mitigation, if resource utilization reaches a peak, then the method cannot be used in the production environment. However, our experimental results prove that the proposed mitigation technique performs better with lower resource utilization.

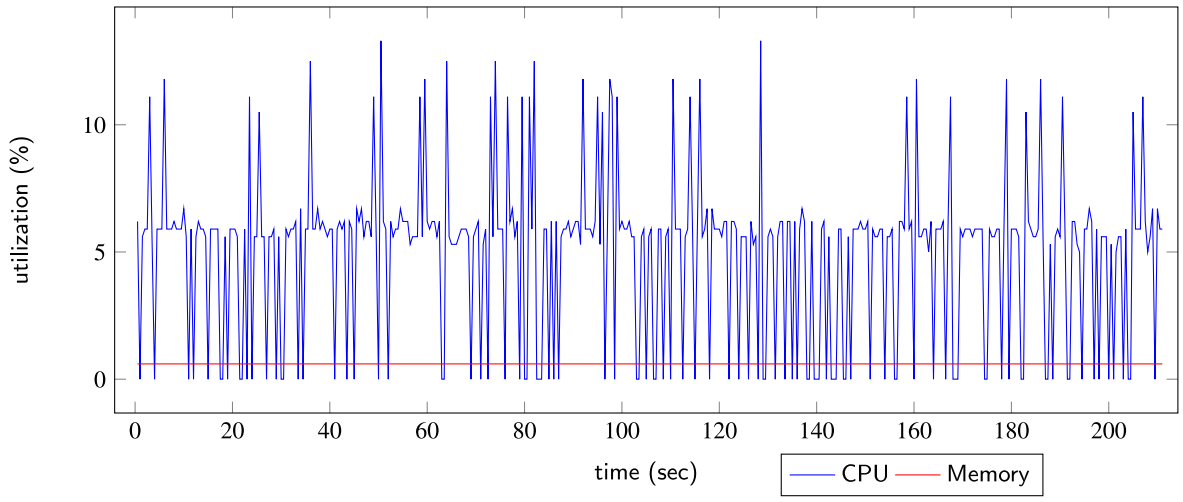


Fig. 12. Idle mode CPU and memory utilization in every 0.5 s.

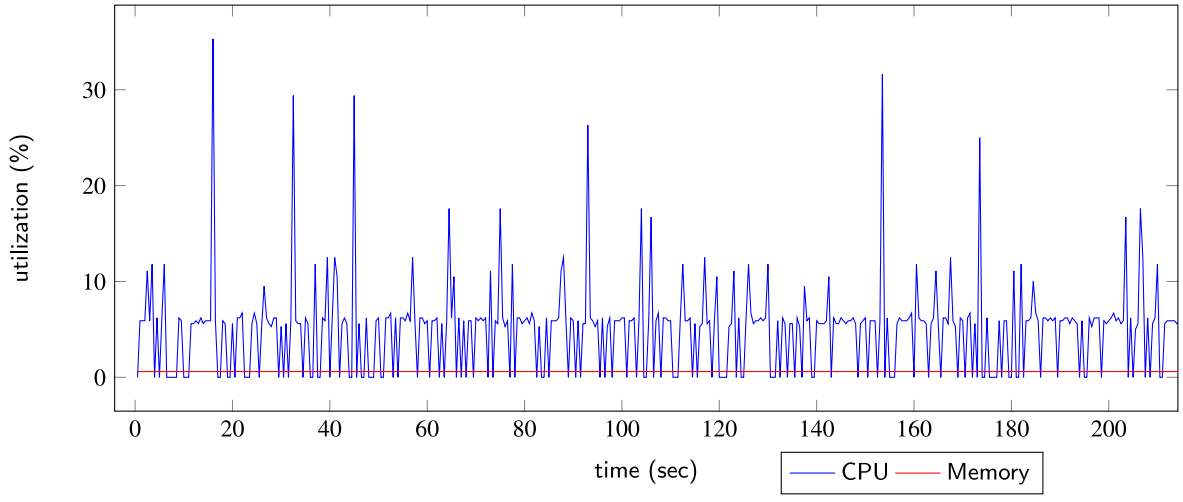


Fig. 13. CPU and memory utilization in every 0.5 s with mitigation technique.

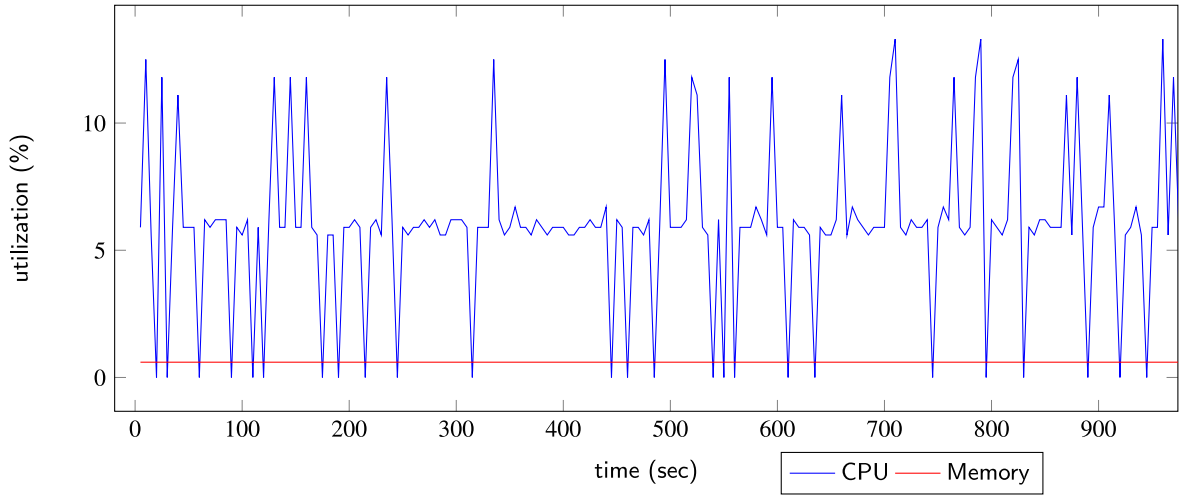


Fig. 14. Idle mode CPU and memory utilization in every 5 s.

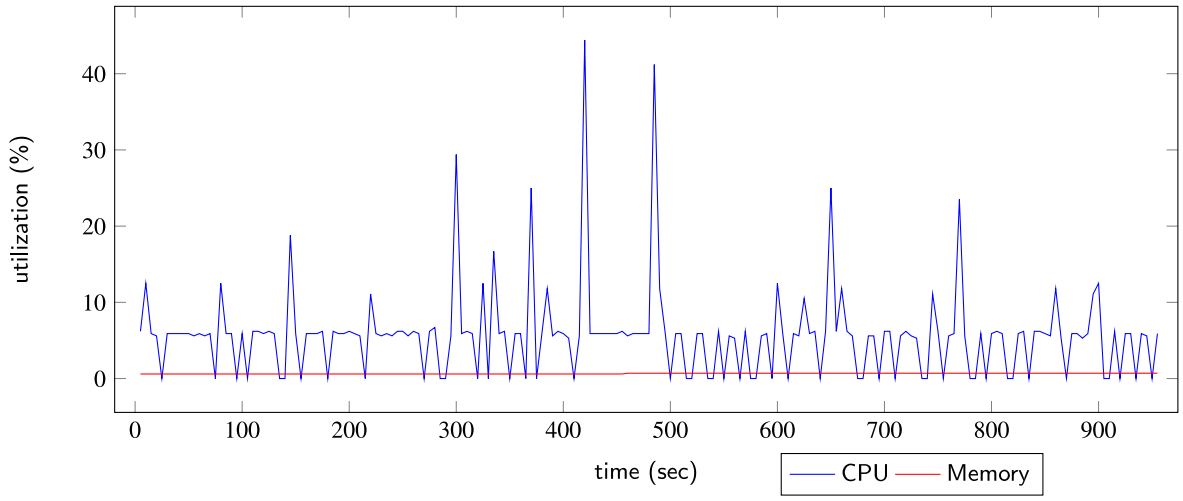


Fig. 15. CPU and memory utilization in every 5 s with mitigation technique.

We experimented with a 5-s utilization capturing interval for a longer time than the first scenario with the same packet sending rate. Fig. 14 shows the utilization of CPU and memory in idle mode with 5-s utilization capturing interval. According to the figure, the maximum CPU utilization was 13.3%, and the minimum value for CPU utilization was 0, which was exactly the same as in the first scenario. The mean value and standard deviation are 5.96 and 3.02, respectively, suggesting a similar CPU utilization with scenario (i). After applying the mitigation technique with a 5-s utilization capturing interval with a DDoS attack, the maximum CPU utilization reached 44.4%. However, the mean value and standard deviation are 5.72 and 5.97, which suggests a slight increase in CPU utilization. However, memory utilization also increased slightly, reached to 0.7 from 0.6 for around 50% cases. Overall, from our experimental results, we can conclude that the proposed mitigation technique performs better with reasonable resource utilization.

5.5. Implementation in a real scenario

When implementing the proposed scheme in a real scenario, several steps are involved to ensure the successful deployment of attack detection, identification, and mitigation capabilities. The first step is to design the system architecture, where the hardware and software requirements, network infrastructure, and additional resources are determined. This design phase includes components specifically dedicated to attack detection, attack identification, and attack mitigation, ensuring that the system is well-equipped to handle various security challenges.

Data collection and preprocessing form the second crucial step. Mechanisms for collecting relevant network data, such as traffic data, $Packet - In$ messages, and network information, are established. These data were then preprocessed by handling missing values, removing noise, and extracting meaningful features for effective attack detection and identification. Proper preprocessing techniques ensure that the collected data is prepared and refined for subsequent analysis.

The subsequent steps focus on attack detection, identification, and mitigation. Attack detection involves the development of algorithms or models that utilize approaches such as GCN. The model is trained using labeled data and optimized by fine-tuning the hyperparameters for the accurate detection of suspicious activities or attacks. Attack identification mechanisms were implemented to analyze the detected activities and identify the specific type of attack occurring in the network. Utilizing the captured data, features, and patterns, different types of attacks were classified and identified. Finally, attack mitigation techniques are developed to effectively fend off the detected attacks. This involves the implementation of algorithms or rules to block malicious traffic and installing blocking rules at switches.

Integrating the scheme with the existing network infrastructure is another critical step. This is accomplished by deploying the implemented components on suitable hardware or utilizing existing network devices with appropriate software configurations. Ensuring proper connectivity, compatibility, and scalability is essential for handling network traffic and satisfying the demands of real-world scenarios. Additionally, real-time monitoring and response mechanisms are set up to continuously analyze network traffic, trigger alerts or notifications when suspicious activities or attacks are detected, and define response protocols for effective mitigation and response.

5.6. Discussion

The description of implementing the proposed method in a real scenario showcases its effectiveness in providing robust attack detection, identification, and mitigation capabilities. The implementation steps outlined above serve as a comprehensive guide for successfully deploying the scheme. The GCN model optimized through fine-tuning showed superior performance compared with other traditional models. Additionally, the scheme's attack mitigation techniques effectively blocked malicious traffic and took appropriate actions to mitigate the impact of attacks on the network infrastructure, showcasing its practical utility in real-world scenarios.

The real-time monitoring and response mechanisms of the implemented scheme enabled prompt detection and response to suspicious activities or attacks. Through continuous analysis of network traffic, the scheme triggered alerts and notifications, allowing network administrators and security personnel to swiftly take necessary actions. This proactive approach minimizes the potential damage caused by attacks and ensures the integrity and security of the network. Overall, the results obtained from implementing the proposed scheme in a real scenario validate its effectiveness, accuracy, and practical viability in providing robust attack detection, identification, and mitigation capabilities for secure network environments.

6. Related works

We cover in this section related research in both SDN packet injection attacks, mitigation techniques and machine learning applications in network security.

6.1. Packet injection attacks on SDN

Packet injection attacks on SDN networks were first discussed in the seminal work of Deng et al. [5]. Their proposed technique, PacketChecker, can detect and mitigate packet injection attacks at switches. However, this technique increases the computational burden on the controller and switches; thus, the controller may eventually break down. This also increases the rule-space overhead of the switch during an attack.

Alshra'a et al. [3] proposed a packet injection attack mitigation technique to maintain real-time network functioning. In their approach, a hardware-software device, INSPECTOR, separates the controller from managing Packet-In messages of malicious data packet flows. This technique reduces the computational burden on the controller and the rule-space overhead of the SDN switches noticeably. The primary drawback of their approach is that it does not match the conformity of the SDN architecture.

Both PacketChecker and INSPECTOR techniques require the controller's involvement in tackling packet-injection attacks. Scott-Hayward et al. [7] propose a technique to mitigate packet injection attacks without the controller's intervention. This technique uses a stateful data plane approach to stop packet-injection attacks at the switch level within a limited scope. This switch-level technique reduces the controller's computational burden and rule-space overhead of switches compared with PacketChecker and INSPECTOR. However, this technique cannot detect an attack when it is launched using an unregistered host address.

The above techniques focus only on high-rate packet injection attacks, *that is* , malicious packets coming in at such a high rate that they remove the controller. However, low-rate packet injection attacks, *e. g.* 10 malicious packets/s, may also pose a real threat. Khorsandroo et al. [9] discussed the effect of low-rate packet injection attacks on SDN networks, but did not offer any solution to mitigate this type of attack. They showed that an attacker generating offensive traffic at a rate of almost 1% of the total throughput in off-peak hours can exhaust almost 25% of rule-space overheads of switches and decrease throughput by nearly 40%.

Zhan et al. [8] proposed a probabilistic technique for detecting and mitigating packet injection attacks on an SDN. They used a machine learning technique to detect packet injection attacks that offered almost 91% attack detection accuracy. Our interest in this study is focused on designing a deterministic technique to detect and mitigate packet-injection attacks. More recently, ul Huque and Hartog [22] presented a workload offloading approach to protect SDN from packet-injection attacks. According to this approach, when malicious traffic inflow crosses a certain threshold, the edge controller controls the core controller. Thus, the core controller remains available even when the gateway switch is attacked. Unlike [22], our work uses deep learning to detect and mitigate attacks.

In comparison to our paper, which focuses on defending against packet injection attacks using deep learning, Sudar et al. [23] explore the vulnerability of SDN to different types of attacks. While our paper aims to develop novel solutions for defending against packet injection attacks, the work of Sudar et al. demonstrates the need for investigating the resilience of SDN against security attacks and suggests the importance of hardening the technology to provide a secure network environment.

In [24], the authors propose a defense framework called DoSGuard to detect and mitigate denial-of-service (DoS) attacks in SDN networks. The framework was implemented as an extension module for the SDN controllers. This study shows that DoSGuard is effective in mitigating DoS attacks against SDN networks with limited overhead. In comparison to our paper, while both papers focus on SDN security, DoSGuard specifically targets the mitigation of DoS attacks, while our research investigates the problem of packet_in flooding in SDN networks and proposes a solution to mitigate this type of attack.

Keerthan et al. [25] propose PDACS module that protects the SDN controller when attacked by remote malicious users, specifically against packet injection and denial of service attacks. The authors tested the real-time implementation of the module using a Zodiac FX switch and various hosts including attackers, controllers, and preventers. Similar to our study, this work aims to address security concerns in SDN networks; however, it differs from ours in its approach and methodology. Additionally, our paper includes a thorough experimental evaluation of the proposed solution, while this work does not provide evaluation details.

To detect packet injection attacks on SDN, Jishuai et al. [26] added a new component (PIEDefender) to an SDN controller. This add-on is protocol-independent and does not require any additional hardware. This add-on monitors the verification of *Packet-In* messages. The authors evaluated this add-on by using a Floodlight controller. It was found that this add-on enabled the controller to detect packet injection attacks with 97.8% precision. Unlike [26], our approach does not add any component to the controller but incorporates deep learning to detect packet injection attacks.

6.2. Deep learning methods for protecting SDN

Several studies (e.g., [27–32]) have used Deep Learning (DL) methods to detect and prevent various types of attacks on SDN. Niyaz et al. [27] proposed a DL-based technique for detecting packet injection attacks in an SDN environment. The stacked autoencoder-based DL technique was deployed on an SDN controller to extract useful features from the network traffic. The approach was evaluated using a testbed consisting of an SDN controller (POX), Open vSwitch, 12 network devices, and various attacks launched using hpring3. The results showed that the DL-based approach could classify benign and malicious traffic with 99.82% accuracy and different types of attacks with 95.65% accuracy.

Tang et a. [28] also proposed a Deep Neural Network (DNN) model for detecting attacks in an SDN environment. The proposed technique relies on a very small number of features (only six) to train a DNN model. The DNN model was deployed on the SDN controller, which received information from the entire network from the OpenFlow switches for analysis to detect attacks. The authors evaluated the model using the NSL-KDD dataset. The results revealed that the model can detect attacks in an SDN environment with an accuracy of 75.75

The authors of [29] integrated several DL algorithms (i.e., RNN, CNN, and long short-term memory) to build an attack detection system for detecting attacks on an OpenFlow-based SDN. The ISCX dataset was used for training the DL models, and the test was conducted via real-time packet injection attacks. The attacks were detected with an accuracy level as high as 98

To enhance the reliability of SDN, Garg et al. [30] proposed a DL-based anomaly detection technique for detecting suspicious data flows in the social media domain. This hybrid technique integrates a Boltzmann machine with an SVM for anomaly detection and leverages an end-to-end data delivery approach to satisfy QoS requirements. The proposed approach was evaluated using both a benchmark dataset (KDD) and real-time data collected from the TIET Institute in India. The experimental results revealed an anomaly detection rate of up to 99

The Motivated by the fact that infected nodes can be isolated via SDN, Maeda et al. [31] proposed MLP's DL-based botnet detection technique, which runs on the SDN controller. Once a botnet is detected, the infected areas (e.g., nodes) of the network are isolated to ensure that the infection does not spread to the rest of the network. To evaluate this approach, the authors used the Ryu SDN framework as the SDN controller and Open vSwitch as the OpenFlow switch. CTU-12 and ISOT datasets were used to assess the accuracy of the DL technique. It was found that this approach could accurately detect 99.2% of botnet attacks.

Mehdi et al. [32] demonstrated the implementation of four DL algorithms in NOX SDN controller for anomaly detection. The DL algorithms were evaluated using data collected in three scenarios: the ISP router, the switch of a lab, and the home network router. The paper showed that the anomaly detection technique could accurately detect anomalies in home and lab scenarios; however, it failed in the ISP router case. It was also found that the detection technique not only has decent accuracy but also does not significantly slow down the flow of packets.

The proposed solution in [33] uses deep learning and real-time evaluation of trustworthiness measures to prevent misbehaving nodes from causing disruptions in the network. In contrast, our proposed solution uses a threshold bit rate at every port of the switch to detect and prevent DoS and packet-injection attacks. Wani et al. [34] propose an IDS for IoT devices that use a deep learning classifier to detect anomalies. The proposed IDS is based on software-defined networking (SDN), which allows centralized control of the network and decouples the control and data planes. Whereas our study focuses on defending SDN against packet injection attacks using deep learning, Lee et al. [35] focused on preventing secure shell (SSH) brute-force attacks and distributed denial-of-service (DDoS) attacks in SDN using deep learning. Additionally, our study uses Graph Convolutional Neural Network models and algorithms for grouping network nodes/users into security classes. [35] uses four different deep learning models to identify anomalous and malicious packets based on packet length. In [36], the authors focused on detecting and defending against DDoS attacks in SDN using an adversarial deep-learning approach. Similar to our paper, this paper uses deep learning techniques for security in SDN, however, this paper addresses different types of attacks and uses different deep learning models.

The majority of DL-based methods for protecting SDN rely solely on network features without considering the relational architecture of the network itself. There are many applications where data can be represented in the form of graphs [37]. Computer networks can be naturally represented by a graph structure, where nodes can be network objects (i.e., hosts, switches, servers), and the edges can be the connections between those objects (i.e., whether a host talks to a server). Our work successfully captured these relational features between network objects by constructing a graph from network flow vectors and training the GCN model using this graph. Furthermore, this work leverages the network-flow features for embedding into the nodal feature vector, which will be learned by the GCN model in addition to the relational features. In addition to the GCN model, our work differs from previous studies in terms of goals (detection, identification, and mitigation), architecture, datasets, and evaluation outcomes.

7. Conclusion

Detecting and preventing packet injection attacks on SDN in real-time is challenging. Therefore, in this study, we developed Deep Learning (DL)-and Machine Learning (ML)-based approaches to secure SDN against packet injection attacks. Our approach first uses DL/ML to detect attacks in real-time. It then identifies the type of attack, such as a slow or fast packet injection attack. Finally, our approach mitigates the attack by updating the network rules in real-time. We evaluate our approach using an SDN emulation setup. In the evaluation, we used two datasets: CICIDS2017 and our own generated dataset, which is the first dataset for packet injection attacks. Our evaluation revealed that (i) the Graph Convolutional Network (GCN) and Random Forest (RF) attained the highest accuracy for attack detection and identification, (ii) the accuracy of GCN, RF, and Gradient Boosting (GB) improves with an increase in the number of training samples, (iii) all tested algorithms can accurately detect the various types of packet injection attacks (mean accuracy of approximately 0.99), and (iv) our approach consumes minimal resources (CPU and memory) for mitigating packet injection attacks. Our code and data are publicly available at: https://github.com/nahaUoA/SDNPacketInjectionAttack

In the future, we plan to investigate the impact of increasing SDN packet injection attacks in a distributed manner (attack generation from at least 10 to 100 hosts) on the effectiveness of our approach. We also plan to observe the impact on resource utilization of SDN switches along with the controller as we increase the number of SDN packet injection attacks. Furthermore, we plan to evaluate the proposed approach in a real test bed environment. Another fruitful direction is to investigate the impact of adding or removing more rules for switches and controllers.

CRediT authorship contribution statement

Anh Tuan Phu: Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Bo Li:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Faheem Ullah:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Tanvir Ul Huque:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Ranesh Naha:** Writing – original draft, Writing – review & editing, Data curation, Visualization, Supervision. **Muhammad Ali Babar:** Writing – review & editing, Conceptualization, Supervision, Project administration. **Hung Nguyen:** Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing, Supervision, Project administration.

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.

Acknowledgment

Tanvir ul Huque acknowledges the support of the Commonwealth of Australia and Cybersecurity Research Centre Limited, Australia.

References

[1] W. Xia, Y. Wen, C.H. Foh, D. Niyato, H. Xie, A survey on software-defined networking, IEEE Commun. Surv. Tutor. 17 (1) (2014) 27–51.
[2] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, et al., Carving research slices out of your production networks with OpenFlow, ACM SIGCOMM Comput. Commun. Rev. 40 (1) (2010) 129–130.
[3] A.S. Alshra'a, J. Seitz, Using INSPECTOR device to stop packet injection attack in SDN, IEEE Commun. Lett. 23 (7) (2019) 1174–1177, http://dx.doi.org/10.1109/LCOMM.2019.2896928.
[4] Sciencedirect, Openflow Switch, 2017, URL: https://www.sciencedirect.com/topics/computer-science/openflow-switch.
[5] S. Deng, X. Gao, Z. Lu, X. Gao, Packet injection attack and its defense in software-defined networks, IEEE Trans. Inf. Forensics Secur. 13 (3) (2018) 695–705, http://dx.doi.org/10.1109/TIFS.2017.2765506.
[6] S. Deng, X. Gao, Z. Lu, X. Gao, Packet injection attack and its defense in software-defined networks, IEEE Trans. Inf. Forensics Secur. 13 (3) (2017) 695–705.
[7] S. Scott-Hayward, T. Arumugam, OFMTL-SEC: State-based security for software defined networks, in: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2018, pp. 1–7, http://dx.doi.org/10.1109/NFV-SDN.2018.8725686.
[8] X. Zhan, M. Chen, S. Yu, Y. Zhang, Adaptive detection method for Packet-In message injection attack in SDN, in: S. Wen, A. Zomaya, L.T. Yang (Eds.), Algorithms and Architectures for Parallel Processing, Springer International Publishing, 2020, pp. 482–495.

[9] S. Khorsandroo, A.S. Tosun, White box analysis at the service of low rate saturation attacks on virtual SDN data plane, in: 2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium), 2019, pp. 100–107, http://dx.doi.org/10.1109/LCNSymposium47956.2019.9000660.
[10] Z. Teo, K. Birman, R.V. Renesse, Experience with 3 SDN controllers in an enterprise setting, in: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), 2016, pp. 97–104, http://dx.doi.org/10.1109/DSN-W.2016.20.
[11] S.C. Madanapalli, M. Lyu, H. Kumar, H.H. Gharakheili, V. Sivaraman, Real-time detection, isolation and monitoring of elephant flows using commodity SDN system, in: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–5, http://dx.doi.org/10.1109/NOMS.2018.8406200.
[12] S. Almotairi, A. Clark, G. Mohay, J. Zimmermann, Characterization of attackers' activities in honeypot traffic using principal component analysis, in: 2008 IFIP International Conference on Network and Parallel Computing, IEEE, 2008, pp. 147–154.
[13] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, P. Talukdar, HyperGCN: A new method of training graph convolutional networks on hypergraphs, 2018, arXiv preprint arXiv:1809.02589.
[14] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016, arXiv preprint arXiv:1609.02907.
[15] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, in: ICISSp, 2018, pp. 108–116.
[16] Y. Cui, L. Yan, S. Li, H. Xing, W. Pan, J. Zhu, X. Zheng, SD-anti-DDoS: Fast and efficient DDoS defense in software-defined networks, J. Netw. Comput. Appl. 68 (2016) 65–79.
[17] J. Boite, P.-A. Nardin, F. Rebecchi, M. Bouet, V. Conan, Statesec: Stateful monitoring for DDoS protection in software defined networks, in: 2017 IEEE Conference on Network Softwarization (NetSoft), IEEE, 2017, pp. 1–9.
[18] Open Networking Foundation, OpenFlow protocol technical specifications, 2015, URL: https://www.opennetworking.org/software-defined-standards/specifications/. accessed 2017-03.
[19] M. Aamir, S.S.H. Rizvi, M.A. Hashmani, M. Zubair, J. Ahmed, Machine learning classification of port scanning and ddos attacks: A comparative analysis, Mehran Univ. Res. J. Eng. Technol. 40 (1) (2021) 215–229, http://dx.doi.org/10.22581/muet1982.2101.19, URL: https://publications.muet.edu.pk/index.php/muetrj/article/view/1999.
[20] I. Steinwart, A. Christmann, Support Vector Machines, Springer Science & Business Media, 2008.
[21] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, Lightgbm: A highly efficient gradient boosting decision tree, Adv. Neural Inf. Process. Syst. 30 (2017).
[22] T. ul Huque, F. den Hartog, Protecting software-defined enterprise networks from packet injection attacks, in: 2021 IEEE 46th Conference on Local Computer Networks, LCN, IEEE, 2021, pp. 287–292.
[23] K.M. Sudar, P. Deepalakshmi, A. Singh, P.N. Srinivasu, TFAD: TCP flooding attack detection in software-defined networking using proxy-based and machine learning-based mechanisms, Cluster Comput. 26 (2) (2023) 1461–1477.
[24] J. Li, T. Tu, Y. Li, S. Qin, Y. Shi, Q. Wen, Dosguard: Mitigating denial-of-service attacks in software-defined networks, Sensors 22 (3) (2022) 1061.
[25] T. Keerthan Kumar, M. Srikanth, V. Sharma, J. Anand Babu, Performance evaluation of packet injection and DOS attack controller software (PDACS) module, in: Inventive Communication and Computational Technologies: Proceedings of ICICCT 2021, Springer, 2022, pp. 793–809.
[26] J. Li, S. Qin, T. Tu, H. Zhang, Y. Li, Packet injection exploiting attack and mitigation in software-defined networks, Appl. Sci. 12 (3) (2022) 1103.
[27] Q. Niyaz, W. Sun, A.Y. Javaid, A deep learning based ddos detection system in software-defined networking (SDN), 2016, arXiv preprint arXiv:1611.07400.
[28] T.A. Tang, L. Mhamdi, D. McLernon, S.A.R. Zaidi, M. Ghogho, Deep learning approach for network intrusion detection in software defined networking, in: 2016 International Conference on Wireless Networks and Mobile Communications, WINCOM, IEEE, 2016, pp. 258–263.
[29] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li, L. Gong, Detection and defense of DDoS attack–based on deep learning in OpenFlow-based SDN, Int. J. Commun. Syst. 31 (5) (2018) e3497.
[30] S. Garg, K. Kaur, N. Kumar, J.J. Rodrigues, Hybrid deep-learning-based anomaly detection scheme for suspicious flow detection in SDN: A social multimedia perspective, IEEE Trans. Multimed. 21 (3) (2019) 566–578.
[31] S. Maeda, A. Kanai, S. Tanimoto, T. Hatashima, K. Ohkubo, A botnet detection method on SDN using deep learning, in: 2019 IEEE International Conference on Consumer Electronics, ICCE, IEEE, 2019, pp. 1–6.
[32] S.A. Mehdi, J. Khalid, S.A. Khayam, Revisiting traffic anomaly detection using software defined networking, in: International Workshop on Recent Advances in Intrusion Detection, Springer, 2011, pp. 161–180.
[33] A. El Kamel, H. Eltaief, H. Youssef, On-the-fly (D) DoS attack mitigation in SDN using deep neural network-based rate limiting, Comput. Commun. 182 (2022) 153–169.
[34] A. Wani, R. Khaliq, SDN-based intrusion detection system for IoT using deep learning classifier (idsIoT-SDL), CAAI Trans. Intell. Technol. 6 (3) (2021) 281–290.
[35] T.-H. Lee, L.-H. Chang, C.-W. Syu, Deep learning enabled intrusion detection and prevention system over SDN networks, in: 2020 IEEE International Conference on Communications Workshops (ICC Workshops), IEEE, 2020, pp. 1–6.
[36] M.P. Novaes, L.F. Carvalho, J. Lloret, M.L. Proença Jr., Adversarial deep learning approach detection and defense against DDoS attacks in SDN environments, Future Gener. Comput. Syst. 125 (2021) 156–167.
[37] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. 32 (1) (2020) 4–24.

Anh Tuan Phu worked as a Research Assistant at the University of Adelaide Australia. He received his Bachelor's degree in Electrical and Electronics Engineering from The University of Adelaide, Australia, in 2021. He was a Research Intern at CREST - Centre for Research on Engineering Software Technologies, which is an interdisciplinary research centre at the University of Adelaide. His current work primarily focuses on IoT systems, software-defined radio and satellite communications.

Bo Li received the master's degree in Cybersecurity from The University of Adelaide, Australia, in 2022. He has been a Teaching Assistant of Cybersecurity Fundamental lecture at the University of Adelaide, in 2022. He worked as a Research Intern with CREST, Centre for Research on Engineering Software Technology, which is an interdisciplinary research centre at the University of Adelaide. He is working on data analysis and data integration in a nonprofit organisation AnglicareSA.

Faheem Ullah is a Lecturer with the School of Computer Science, The University of Adelaide, Australia. He is also the program director for the Master of Cyber Security program. He is a member of CREST — Centre for Research on Engineering Software Technologies, which is an interdisciplinary research centre at the University of Adelaide. He completed his Ph.D., focused on the intersection of big data and cyber security, from the University of Adelaide, Australia. Since then he has been actively involved in teaching undergrad and master courses in the area of cyber security and software engineering. He has supervised/co-supervised more than 40 undergrad and master students. His current research primarily focuses on cyber security, big data analytics, and software engineering.

Md. Tanvir ul Huque received the master's degree in electrical engineering from the University of Sydney, Canberra, Australia, in 2014, and the Ph.D. degree in electrical engineering from the University of New South Wales (UNSW), Sydney, Australia, in 2019.,He is a Research Fellow in cybersecurity with Queensland University of Technology, Brisbane, Australia. He was with the UNSW, Canberra, Data61-CSIRO, Sydney, and the University of Helsinki, Helsinki, Finland. He is actively working on projects of critical infrastructure security.

Ranesh Naha is a research fellow with the Centre for Smart Analytics, Federation University Australia. Prior to this he worked as a grant-funded researcher at the University of Adelaide. He received his Ph.D. degree from the University of Tasmania, Australia. He has awarded a Tasmania Graduate Research Scholarship (TGRS) to support Ph.D. studies. He received his M.Sc. degree from Universiti Putra Malaysia, in 2015. He has awarded a prestigious Commonwealth Scholarship provided by the Ministry of Higher Education, Malaysia. His research interests include software-defined networking (SDN), distributed computing (Fog/Edge/Cloud), cybersecurity, Blockchain, Internet of Things (IoT), AI and ML. He authored more than 30 peer-reviewed scientific research articles.

M. Ali Babar is a Professor in the School of Computer Science, University of Adelaide, Australia. He leads a theme on architecture and platform for security as service in Cyber Security Cooperative Research Centre (CSCRC), a large initiative funded by the Australian government, industry, and research institutes. Professor Babar is the technical project lead of one of the largest projects on "Software Security" in ANZEC region funded by the CSCRC. SOCRATES brings more than 75 researchers and practitioners from 6 research providers and 4 industry partners for developing and evaluating novel knowledge and AI- based platforms, methods, and tools for software security. After joining the University of Adelaide, Prof Babar established an interdisciplinary research centre called CREST, Centre for Research on Engineering Software Technologies, where he directs the research, development and education activities of more than 25 researchers and engineers in the areas of Software Systems Engineering, Security and Privacy, and Social Computing. Professor Babar's research team draws a significant amount of funding and in-kind resources from governmental and industrial organisations. Professor Babar has authored/co-authored more than 290 peer-reviewed research papers at premier Software journals and conferences. Professor Babar obtained a Ph.D. in Computer Science and Engineering from the school of computer science and engineering of University of New South Wales, Australia. He also holds a M.Sc. degree in Computing Sciences from University of Technology, Sydney, Australia.

Hung Nguyen is an associate professor in the School of Computer and Mathematical sciences and the leader of the Information warfare and advanced cyber theme within the Defence trailblazer, the University of Adelaide. He leads a research group on Cyber-AI, applying new advances in AI to solve problems in network security. His research focuses on developing autonomous and provable cyber defensive solutions. He has published more than 80 peer-reviewed publications on these topics.