electronics

MDPI

Article

Enhancing Software-Defined Networks with Intelligent Controllers to Improve First Packet Processing Period

Ramesh Chand Meena [1], Surbhi Bhatia [2,*], Rutvij H. Jhaveri [3,*], Piyush Kumar Shukla [4], Ankit Kumar [5], Neeraj Varshney [5] and Areej A. Malibari [6]

[1] Software Technology Parks of India, Ministry Electronics and Information Technology, Government of India, Jaipur 302222, India
[2] Department of Information Systems, College of Computer Science and Information Technology, King Faisal University, Al Hasa 31982, Saudi Arabia
[3] Department of Computer Science and Engineering, School of Technology, Pandit Deendayal Energy University, Gandhinagar 382007, India
[4] Department of Computer Science & Engineering, University Institute of Technology, Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal 462033, India
[5] Department of Computer Engineering & Application, GLA University Mathura, Chaumuhan 281406, India
[6] Department of Industrial and Systems Engineering, College of Engineering, Princess Nourah Bint, Riyadh 11671, Saudi Arabia
* Correspondence: sbhatia@kfu.edu.sa (S.B.); rutvij.jhaveri@sot.pdpu.ac.in (R.H.J.)

Abstract: Software-Defined Networking (SDN) has a detailed central model that separates the data plane from the control plane. The SDN controller is in charge of monitoring network security and controlling data flow. OpenFlow-enabled routers and switches work as packet-forwarding devices in the network system. At first, OpenFlow forwarding devices like routers and switches do not know how to handle the data packets transmitted by the host. This is because they do not have any security controls, policies, or information. These packets are sent to their destination. In this situation, the OpenFlow forwarding device sends the first data packet of a host to the SDN controller, which checks the control packets for the data packet and creates flow entries in the switch flow table to act on the following categories of data packets coming from the host. These activities at the SDN controller and switch levels are time-intensive, and the first data packet from the host always takes a longer time to reach its destination. In this article, we suggest an SDN controller with instant flow entries (SDN-CIFE) to reduce the amount of time it takes for the host to transmit its first data packet. Before traffic comes from the host, our method adds the necessary flow entries to the flow table of the OpenFlow switch. The technique was made in Python and tested on a Mininet network emulator using the RYU controller. The results of the experiment show that the time it takes to process the first data packet is reduced by more than 83%.

Keywords: controller; instant; flow; entry; SDN-CIFE; first packet; processing; period

Citation: Meena, R.C.; Bhatia, S.; Jhaveri, R.H.; Shukla, P.K.; Kumar, A.; Varshney, N.; Malibari, A.A. Enhancing Software-Defined Networks with Intelligent Controllers to Improve First Packet Processing Period. *Electronics* **2023**, *12*, 600. https://doi.org/10.3390/electronics12030600

Academic Editor: Lionel Nkenyereye

Received: 21 December 2022
Revised: 13 January 2023
Accepted: 18 January 2023
Published: 25 January 2023

1. Introduction

The flow entries in the SDN switch flow table are used to handle incoming host packets in an SDN network, and many fields of the packet header are used to form flow entries. SDN switches also keep statistics on network traffic. The benefits of SDN technology, such as programmability, simplicity, and elasticity to network managers, have led to its increased use in various enterprise networks and data center solutions. With traditional systems, incoming host traffic was forwarded [1].

Initially, OpenFlow forwarding devices do not have any security or information controls or policies to handle data packets created by the host and sent to its target. In this case, the forwarding device forwards a host's first packet to the controller for checking, generating control packets for a data packet, and creating flow entries in the switch flow

table to act on the subsequent data packets generated by the host. These tasks at the SDN controller and switch levels take time. The first data packet sent by a host always takes longer to reach its destination.

Another issue is using all network resources when there is a high volume of traffic on the network. These packets necessitate the generation and transmission of control packets between the SDN controller and the switch. This situation may cause additional delays in generating control packets and the delivery of data packets to their destinations. SDN switches require flow rules/entries in their flow tables to forward data packets to their destinations; otherwise, they will need the assistance of the SDN controller to generate control packets. The controller may create control packets to insert match entries with actions into the flow table of the SDN switch, or it may forward the data packet to the outport or both. Some studies have proposed methods for reducing the overload of forwarding data packets to the controller and generating control packets.

The authors of [2] stated that many control packets create overhead for the controller and that reducing the number of control messages reduces the controller's workload. If a network has a high volume of packets, it may require more processing resources and drop more packets. The authors of [3] also proposed a scheme for classifying data packets as important or unimportant, recommending that unimportant data packets be dropped to reduce the load in OpenFlow switches and controllers. Switch Reduce [4] claimed that the number of match entries in the first SDN switch should be less than the number of packet actions for the packet load and that flow entries should be installed only in the first hop SDN switch. Nonetheless, this approach necessitates the configuration of wild card flow entries in all in-between hop SDN switches, including the last hop SDN switch.

1.1. Major Contribution

1. We have identified that no single approach provides an effective solution for setting up flow entries before transmitting actual data packets from the host. This could not be accomplished without detecting the host and its details, such as the MAC IP address, and connected switch port.
2. Most approaches are unaware of these details until the host begins transmitting packets.
3. This paper describes the design of an SDN Controller with Instant Flow Entries (SDN-CIFE) that detects the host instantly, creates the necessary flow entries in the forwarding device flow table, and minimizes the arrival time of the host's first data packet. The system was written in Python and used to simulate an RYU controller on a Mininet network emulator.
4. Before the SDN Controller proceeds further, SDN-CIEF generate actual traffic in the network. They maintain the Host Binding Table and set up the required flows in the networks.
5. The proposed method can identify the address of a malicious host from which a packet originated. It also provides real-time protection against attacks using address spoofing.

1.2. Paper Organization

The remainder of the paper is arranged as follows. In a nutshell, it explains the history of SDN. Section 2 focuses on SDN topology detection. Section 3 goes over related works. The proposed SDN-CIFE is described in Section 4. SDN-CIFE implementation & investigates SDN performance are described in Section 5. Section 6 Finally, brings the work to a close.

2. Related Works

In Switch Agent [5] the authors offer a system in which the controller produces and forwards TDP-Request multicast messages. The switches shift to either the Active or Father node from the Standby node after receiving such messages. Every node asserts neighbor detail, and only the father node sends the details to the controller asynchronously. In [6], the authors offer a solution for running Link Layer Discovery Protocol (LLDP) at the switch level to discover links. The controller periodically asks to collect the topology details.

SDN technology is low-cost, dynamic, adaptable, and manageable, making it ideal for active applications with high bandwidth requirements. SDN decouples network forwarding and control actions, and it directly configures network control and applications through programming while the underlying infrastructure abstracts network services. The OpenFlow protocol is the central and essential component of an SDN network. The Open Network Foundation promotes it to provide the southbound interface between controllers and SDN switches.

Handshake messages are sent to the controller and switches to establish an OpenFlow connection. An encrypted TCP connection is established between the switches and the controller to exchange configuration information. The controller sends an OFPT FEATURE REQUEST message to the OpenFlow protocol enabled switch, and the switch responds with an OFPT FEATURE REPLY message to establish an OpenFlow connection. The switch forwards its unique identifiers, such as the MAC addresses of active switch ports and the switch's data path id [7]. During handshaking, the switch is discovered, but this process does not reveal interconnections in the forwarding devices. Details about the network environment are required to perform various network management and control tasks. The detection and configuration of appropriate paths are critical tasks for allowing a switch to forward network traffic [8].

Group tables and flow entry tables are available on OpenFlow protocol enabled switches. Through OFPT FLOW MOD messages, the controller makes necessary changes to flow tables such as additions, deletions, and modifications. Flow entry includes the packet header field's structure, counters, and actions. When a packet arrives, its header fields are compared to the field values of flow entries, and when a packet matches any flow entry, the relevant counter sets are increased, and related actions are performed [9]. When no flow entry reaches the packet header field, a table missing message is generated, containing instructions to switch a packet to forward to the controller, forward to another table, or drop it. When a packet is forwarded to the controller, the switch sends an OFPT PACKET IN event to the controller. The controller inspects the arriving packet, generates the necessary control packets, and returns the original packet to the switch along with the required actions. The controller creates the required flow entries in the OpenFlow switch flow table by sending OFPT FLOW MOD messages to deal with such potential packets.

Detection of Topology in SDN

To publicize their neighbors and properties in a wired LAN, forwarding devices use a single-hop Link Layer Discovery Protocol (LLDP). Frame LLDP messages with other types of field values, such as the hex 88cc value for the multicast MAC address of the bridge, e.g., 01:80:c2:00:00:0E, are sent at a predefined time from their active port forwarding devices and do not generate LLDP packets in an SDN network.

The procedure described in [10] was used to detect the network topology. This discovery procedure detects only connected switches and their interconnections. It does not reveal any information about the connected host. OpenFlow controllers have a fundamental host recognition process based on the table miss flow entries of SDN forwarding devices. The process tells the SDN device to send a packet to the controller. The host initiates the generation of Address Resolution Protocol (ARP) or Internet Protocol (IP) traffic which is then delivered to the SDN forwarding device. It does not have traffic flow rules. In this

case, the first packet from the connected host is sent to the controller. The controller extracts host information from the first packet and completes the host discovery process.

We can use LLDP [11] to find a connected host, but the protocol must be implemented at the host level. Because hosts are monitored and managed by different entities, LLDP implementation at the host level is complicated.

The host discovery process depends on the DHCP and ARP packet details described in [12], which are used via the PacketIn event in the L2 protocol. According to the study, the OpenFlow Discovery Protocol is limited to learning about the controller's forwarding devices in the network using the LLDP frame format. The authors did not propose any host discovery techniques before generating connected host traffic.

In [13], Network Mapper (NMAP) utility-inspired researchers propose a host discovery approach that sends ARP-Request messages from the controller side to determine the host details as soon as an OpenFlow switch-to-controller connection is established. The authors used OFPT PACKET CONTROL messages to generate ARP-request messages to switch the port with a random destination IP and broadcast MAC. The OpenFlow switch broadcasts ARP-request messages. The connected live host with an IP address will respond to the switch with an ARP reply message during this process. The OpenFlow switch will forward ARP-reply messages to the controller to extract the necessary host information from the messages. The controller will now have complete network environment details, such as the host IP and MAC, switches and switch ports, and interconnectivity between them thanks to this host discovery module and LLDP. This method worked properly when an OpenFlow switch was connected to an operational controller. The approach did not perform well when the switch was started before the controller or when the switch was detached from the controller; as such, some changes were made to the status of the switch port. The same switch was connected to the controller again. The approach used the OFPT PORT STATUS message, and it worked with changes such as deletions, additions, and status changes in switch port status.

Researchers in [14] proposed delivering the first packet via the subdomain cluster model by selecting the high priority controller based on the load performance index. For data packet forwarding, the approach chooses a relatively less busy controller. This approach necessitates the use of more than one controller, and controller load records must be kept. If all controllers have the same processing load, it may take longer to process the data packets, and the approach must find a low load controller. The increased number of controllers necessitates more processing power, increasing the network cost and complexity.

3. Proposed SDN-CIFE

In this section, we discuss the proposed approach.

3.1. Objectives of SDN-CIFE

- Detection of connected host information, such as the MAC address and OpenFlow switch ID and switch port ID, by handshaking between the OpenFlow switch and controller. Modifications occur on the switch port/host.
- We have managed the HostLink table at the Controller level to store the host MAC, SDN switch ID and Port ID.
- We have loaded the essential flow entries into the OpenFlow switch flow table before actual data packet generation by a host.

3.2. SDN-CIFE Architecture

The network elements in our SDN-CIFE structure include OpenFlow switches, connected hosts, and a controller. Hosts communicate with the OpenFlow switch via a switch port. OpenFlow switches are connected to a controller and have interconnections. Figure 1 depicts the architectural outline of SDN-CIFE. The Open Southbound API connects the

RYU Controller to the OpenFlow switches. THE PROPOSED APPROACH implements the SDN-CIFE function on top of the RYU Controller. The details of various SDN-CIFE [15,16] modules with specific functions are described in the following subsection.

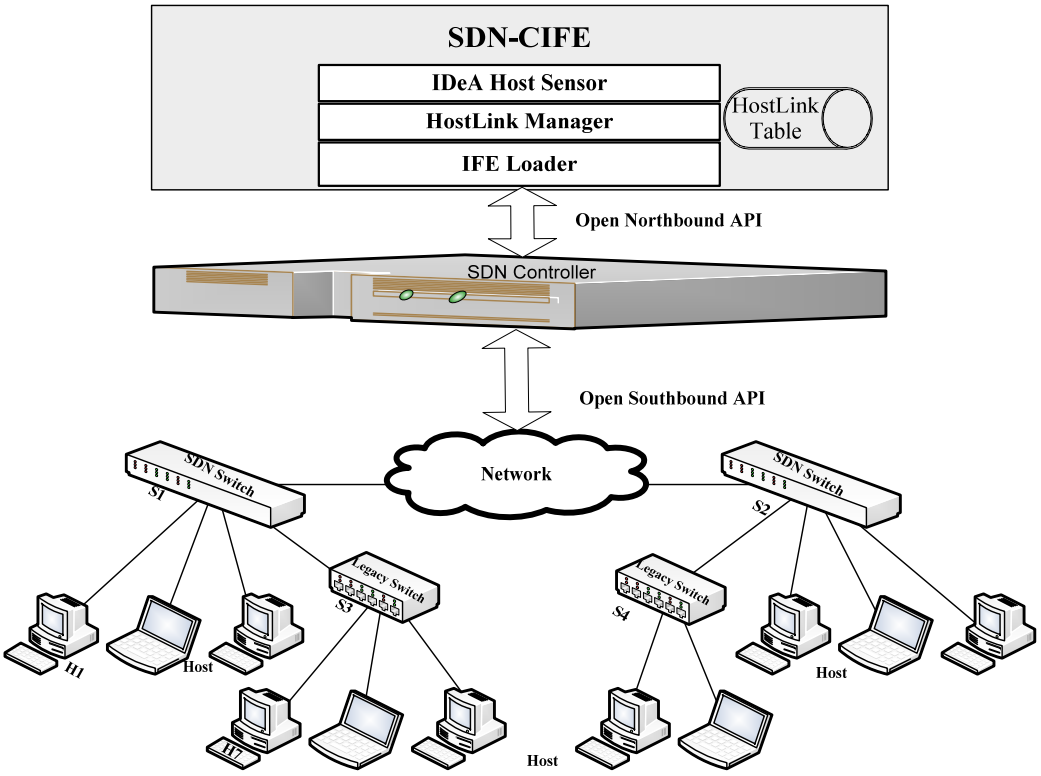

Figure 1. Architecture of SDN-CIFE.

3.3. Modules of SDN-CIFE

The controller requires host information to create filtering rules, generate control packets, and apply security policies. The OpenFlow protocol can detect a partial network topology, which is limited to detecting forwarding devices, their interconnectivity, and the presence of a controller. Topology detection does not include revealing the existence of hosts before the generation of network traffic.

3.3.1. IDeA Host Sensor

Filtering and forwarding rules are initially unavailable for the first host data packet generated in packet forwarding devices. Due to a flow miss event, the host packet is sent by forwarding the device to the controller. At the controller level, the incoming packet is examined. To enable the device to work with incoming traffic from the host, host details are extracted from the packet and policies and rules for the packet are defined.

Rules are set up in the forwarding device flow table. The initial processes take time and must be completed at the controller level, and the forwarding device must wait until the preceding functions are completed. Because of the waiting period, the host's first packet takes longer to reach its destination. Instant Host Detection [17] was proposed in Software Defined Network technology to identify connected hosts during handshaking between the SDN switch and controller. IDH-SDN [18] generates ARP-Request packets and sends them to the network. The SDN controller examines the ARP-Reply packet generated by the connected host. The connected host details are extracted from the ARP reply packet and saved in a table. This section proposes an Instant Detection Approach (IDeA) for the host in an RYU SDN Controller which identifies connected hosts when a

switch/switch port/host is introduced into the network. Figure 2 depicts the IDeA, algorithm, and flow data [19].

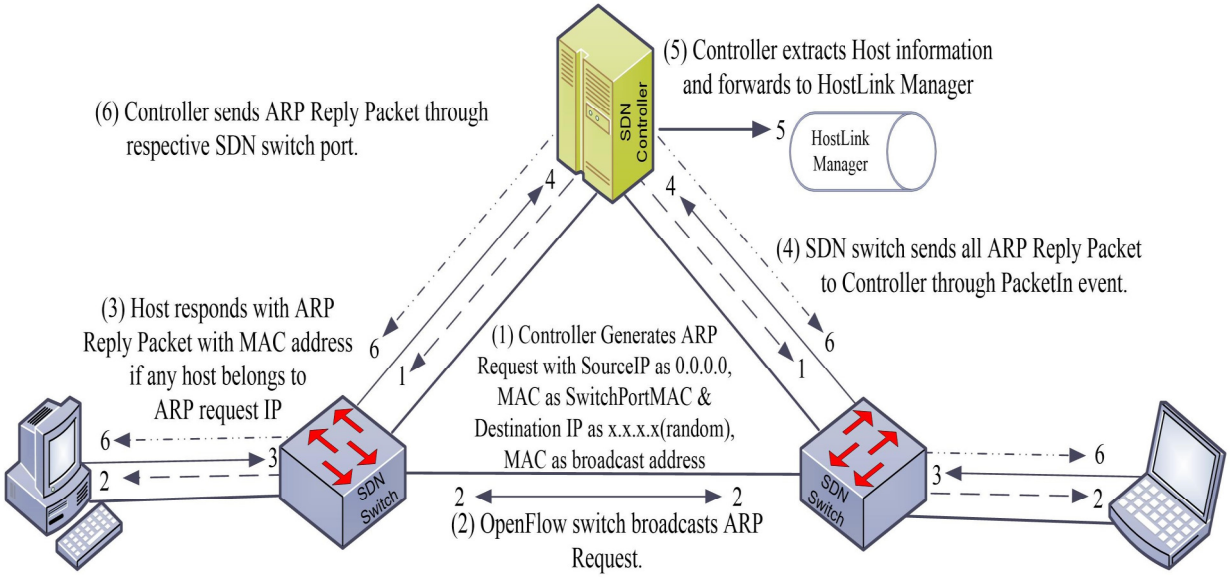

Figure 2. Algorithm and Flow Diagram for IdeA.

3.3.2. HostLink Manager

After extracting host information from an ARP reply packet, IDeA sends it to the HostLink Manager to create the necessary entry in the HostLink Table [20]. The HostLink Manager verifies the host identity from the HostLink Table and performs the following verification tasks. If there is no host information in the table, it adds host information. In [21], a table was proposed to record the connected host MAC and IP details, similar to the HostLink Table. Host identity information is stored in the host MAC, SDN Switch Port ID, and SDN Switch ID. Figure 3 depicts the format of the HostLink Table.

Host MAC	SDN Switch Port ID	SDN Switch ID

Figure 3. Host link format.

After recording the required host information into HostLink Tables, the module passes the HostLink Table to the IFE Loader [22] for further action.

3.3.3. IFE Loader

The Instant Flow Entry (IFE) Loader is proposed to define and supervise flow entries in Openflow switch flow tables whenever the IDeA Host Sensor detects a new host. Host details are entered into the HostTable by the HostLink Manager. After the host has been seen, the HostLink Manager passes the HostLink Table to the Instant Flow Entry (IFE) Loader. The IFE Loader creates flow entries in the device flow table based on the host information in the HostLink Table in order to send packets directly to their destination address. These required flow entries are generated during handshaking between an OpenFlow switch and a controller or whenever a change is detected at a switch port.

After completing the SDN/OpenFlow switch handshake with the controller or changing the switch port, the host begins transmitting data packets [23]. Our module loads all required flow entries into the SDN switch flow table for handshaking or changes detected

at the SDN switch port. As a result, the host's packets are then forwarded to their destination address by matching with the flow entries of the respective SDN switch. There is no flow miss event, and no packets are sent to the controller. We can say that no flow miss events occur because all required flow matches are already present in the flow table of the SDN switch, and all actual traffic from the connected hosts is forwarded directly to the destination.

In this case, the controller does not receive the first packet from the host for inspection, decision-making, creating control packets for the data packet, or installing flow entries and actions into the SDN switch flow table for subsequent data packets received from a host. By instantly detecting connected hosts, storing host link details in the Host Link Table, and making necessary flow entries proactively during SDN switch handshaking [24], we have reduced the generation of control packets and decreased the forwarding time of the host's first data packet.

4. SDN-CIFE Implementation

We set up flow entries well before the host starts generating traffic to reduce the first packet processing time and the generation of control packets between the controller and the SDN switch. This process must be completed when the handshaking between the forwarding device and the controller occurs and when the status of the switch ports changes. At the controller level, our proposed IDeA generates ARP-Request packets with source IP = 0.0.0.0, SrcMAC = SwitchPortMAC, DstMAC = broadcast, and DstIP = random IP address. It is used in a controller for SDN switch handshaking and notification of switch port status changes. Algorithms 1 and 2 concern the HostLink Table Manager and IFE Loader, respectively.

Algorithm 1 HostLink Manager

```
Level: PacketIn event // implementation level
Input: env PacketIn massage // input value for event
Output: HostDetails for HostLink Table //
Extract DstMAC, SDNSwitchID and SDNSwitchPortID, PacketType from evn // extraction of host information from ARP-request
If SrcIP is "0.0.0.0" and DstMAC is broadcast goto 9 // incase ARP-Request messages
If DstIP is "0.0.0.0" than // ARP-reply
    If SrcMAC not exists in HostTable and DstIP is "0.0.0.0" than
        Forward even and HostLink Table to IFE Loader
        Add SrcMAC, SDNSwichID, SDNSwitchPortID into HostLink Table // save host details into HostLink Table
    End if
End if
Forward packet to outport
Return
```

Now, we use the IEF Loader Algorithm for SDN-CIFE Implementation.

Algorithm 2 IFE Loader

```
Implementation as: IFE Loader
Input: env PacketIn and HostLink Table from HostLink Manager
Output: Flow Entries for SDN Switch
Extract DstMAC, SwitchID and SourcePortID, PacketType from evn
For x in HostLink Table (Switch ID) do
            // load flow entries for forward traffic
    FlowEntry = SourceMAC = SrcMAC,
        In_port = SwitchPortID
        DstMAC = (HostLink Table(x).DstMAC)
        Outport = (HostLink Table(x).SDNSwitchPortID)
    Set FlowEntry // install flow entry
            // load flow entries for reverse traffic
     FlowEnrty = SourceMAC = (HostLink Table(x).DstMAC),
        In_port = (HostLink Table(x).SDNSwitchPortID)
        DstMAC = SrcMAC
        Outport = SwitchPortID
    Set FlowEntry // install flow entry
            // load ARP flooding packets
    FlowEnrty = SourceMAC= SrcMAC, //load ARP flooding packets
        In_port = SwitchPortID
        DstMAC = "ff:ff:ff:ff:ff:ff" //broadcast address
        Outport = Flood
    Set FlowEntry // install flow entry
End for
Return
```

5. SDN-CIFE Performance

This section of the paper compares the performance of our technique using bandwidth tests with and without SDN-CIFE, the number of flow entries generated, and First Packet Processing Time [25]. The goals of this research are to detect connected hosts, extract host details such as SwitchID, SwitchPortID, and hostMAC, manage the HostLink table at the controller, load the required flow entries into the forwarding device during OpenFlow switch handshaking with the controller, and detect any changes reported on the switch port before the actual host packet transmission. These goals also include reducing the process time for the generation of control packets and establishing flow entries in an OpenFlow switch to transmit a host's first packet. The proposed approach loads the required flow entry into the OpenFlow forwarding device well before the host's first packet is sent. The first and subsequent packets are forwarded directly by forwarding the device to their target address by matching the flow entries in the device flow table. It also reduces controller overhead by reducing control packet generation requirements and increasing network packet transmission efficiency.

5.1. Simulation Environment

The proposed SDN-CIFE system was tested. The experimental environment comprised an Intel processor i5-6200U@2.30–2.40Ghz with 4GB RAM and 500GB SSD running OS Ubuntu 18.04 LTS. RYU-4.28 SDN controller and Python version 2.7.12 were installed and configured for different SDN network scenarios. The Python programs required during the experiment were written using the Gedit text editor. We also used Wireshark-2.2.6 to capture and inspect the network packets forwarded among hosts and switches.

5.2. Network Scenarios

The system was implemented and tested using different SDN network scenarios. The first network topology was the Single SDN Switch Scenario (4S) [26], with one SDN Controller, one SDN Switch, and four hosts. The second scenario, i.e., the Multiple SDN Switch Scenario (M3S), comprised one SDN Controller, three SDN switches, and seven hosts. The third scenario, i.e., the Hybrid SDN Switch Scenario (H3S), comprised one SDN Controller, two SDN Switches, two Non-SDN (legacy) [27] switches and eleven hosts. Figures 4–6 show diagrams of the three network scenarios:

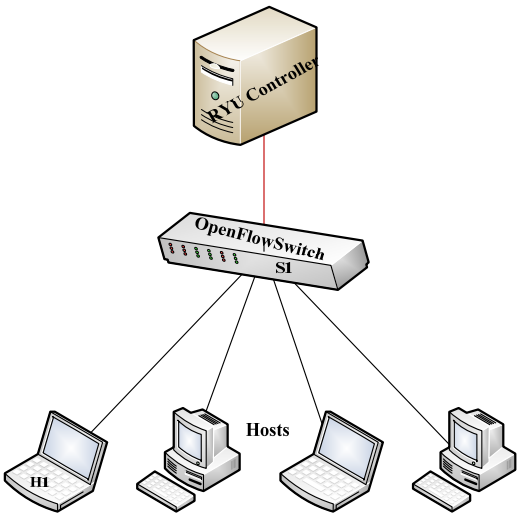

Figure 4. Single SDN Switch Scenario (4S).

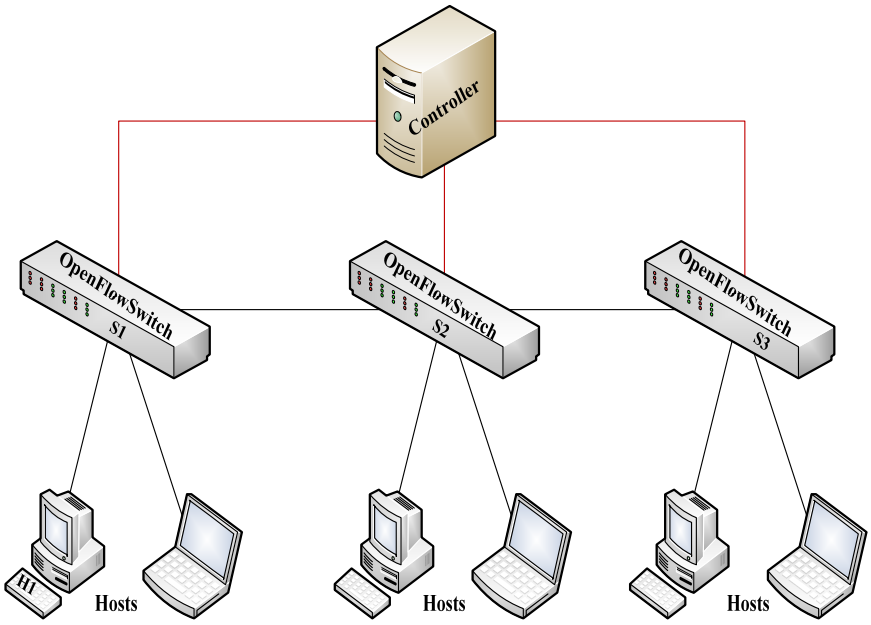

Figure 5. Multiple SDN Switches Scenario (M3S).

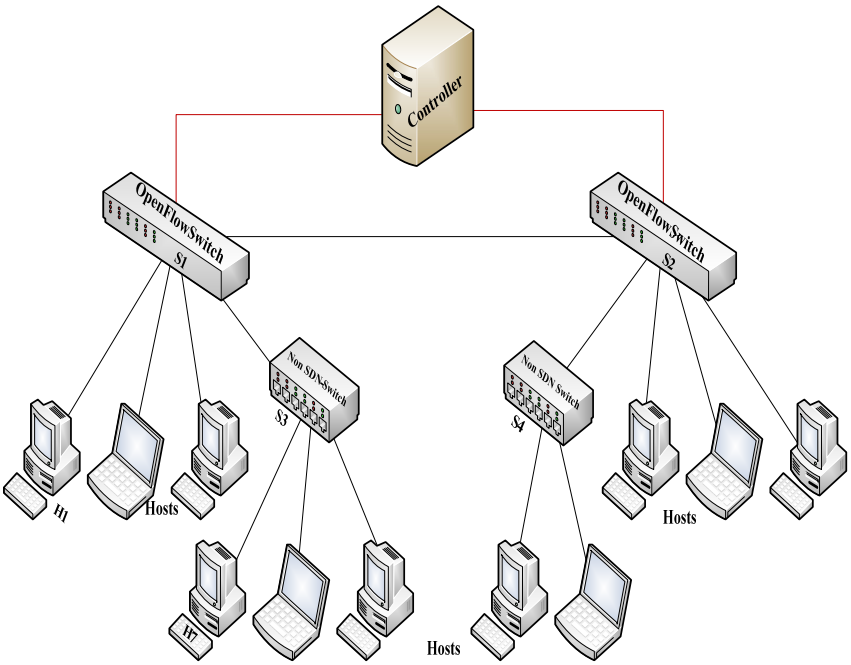

Figure 6. Hybrid SDN Switch Scenario (H3S).

5.3. HostLink Table Data in the Different Scenarios

The implementation of and experimentation with the proposed system used 4S M3S and H3S networks. It identified the linked host in the network topology by handshaking between the SDN switch and controller and saved the extracted host information in the HostLink Table [28]. Details of connected hosts were verified using a Wireshark network traffic analyzer. The results given in Table 1 show that the system performed as intended and discovered the networked hosts effectively.

Table 1. HostLink Table Status.

Host MAC	SDN Switch Port ID	SDN Switch ID
Network Scenario: H3S		
00:00:00: 00:00:03	1	1
00:00:00: 00:00:07 00:00:00: 00:00:09 00:00:00: 00:00:08	2	
00:00:00: 00:00:06 00:00:00: 00:00:05 00:00:00: 00:00:04 00:00:00: 00:00:10 00:00:00: 00:00:11	3	
00:00:00: 00:00:01	4	
00:00:00: 00:00:02	5	
00:00:00: 00:00:04	1	2
00:00:00: 00:00:05	2	
00:00:00: 00:00:06	3	
00:00:00: 00:00:10 00:00:00: 00:00:11	4	
00:00:00: 00:00:03 00:00:00: 00:00:02 00:00:00: 00:00:01	5	

00:00:00: 00:00:09		
00:00:00: 00:00:08		
00:00:00: 00:00:07		
Network Scenario: M3S		
00:00:00: 00:00:03	1	1
00:00:00: 00:00:06		
00:00:00: 00:00:05		
00:00:00: 00:00:04		
00:00:00: 00:00:02	2	
00:00:00: 00:00:01	3	
00:00:00: 00:00:02	1	2
00:00:00: 00:00:01		
00:00:00: 00:00:03	2	
00:00:00: 00:00:04	3	
00:00:00: 00:00:06	4	
00:00:00: 00:00:05		
00:00:00: 00:00:05	1	3
00:00:00: 00:00:06	2	
00:00:00: 00:00:03	3	
00:00:00: 00:00:02		
00:00:00: 00:00:01		
00:00:00: 00:00:04		
Network Scenario: 4		
00:00:00: 00:00:02	1	1
00:00:00: 00:00:01	2	
00:00:00: 00:00:03	3	
00:00:00: 00:00:04	4	

5.4. Number of Flow Entries

The 4S M3S and H3S network scenarios were used with various configurations, i.e., with different SDN switches, hosts, and controllers for the implementation and instant detection of networked hosts, as well as with host detail mining and with and without installing flow entries into the SDN Switch flow table. Several of the flow entries installed in the forwarding devices are given in Table 2. The flow entries installed during testing could be retrieved with Ovs-ofclt dump flow instructions. We found that the system defined and set up flow entries into SDN switches as per the requirements of each approach [29].

Table 2. Flow Entries Generated for Network Scenarios.

Network Scenario	Flow Entries Number			
	SDN Switch			Sum
	S1	S2	S3	
4S	16	-	-	16
M3S	21	30	21	72
H3S	89	83	-	172

5.5. Flow Entry with Varying Host Numbers

We carried out tests in the 4S network scenario using different hosts to verify the flow entries of the SDN switch flow tables. The results showed that SDN-CIFE performed correctly with varying numbers of hosts in the network [30], defining and setting up flow

entries in the SDN Switch as required. The numbers of flow entry numbers in the S1 SDN switch flow table during testing using different host numbers are given in Table 3.

Table 3. Flow Entries using different Host Numbers.

Test No	Flow Entries	Host Numbers
1	225	15
2	900	30
3	2025	45
4	3600	60
5	5625	75
6	8100	90
7	11,025	105
8	14,400	120
9	18,225	135
10	22,500	150

5.6. Bandwidth Test at Different Hops

A system bandwidth delivery analysis was carried out using an H3S network scenario with single, dual, and multiple connections and a different number of hops. The test was conducted without and with the SDN-CIFE system. The results indicated that after implementing the proposed approach, the bandwidth delivery improved with three or more hops. The results given in Figure 7 and Table 4 were collected during testing.

Table 4. Bandwidth Delivery Test.

Connection Scenario		Hops (B/w in Gbps)			
		1	2	3	4
With SDN-CIFE	Single	47.70	45.50	37.80	33.50
	Dual	25.00	24.10	17.90	17.70
	Multiple	9.00	8.90	7.20	7.20
Without SDN-CIFE	Single	50.00	41.00	42.30	35.30
	Dual	18.60	22.30	17.70	17.60
	Multiple	9.70	7.20	8.00	7.80

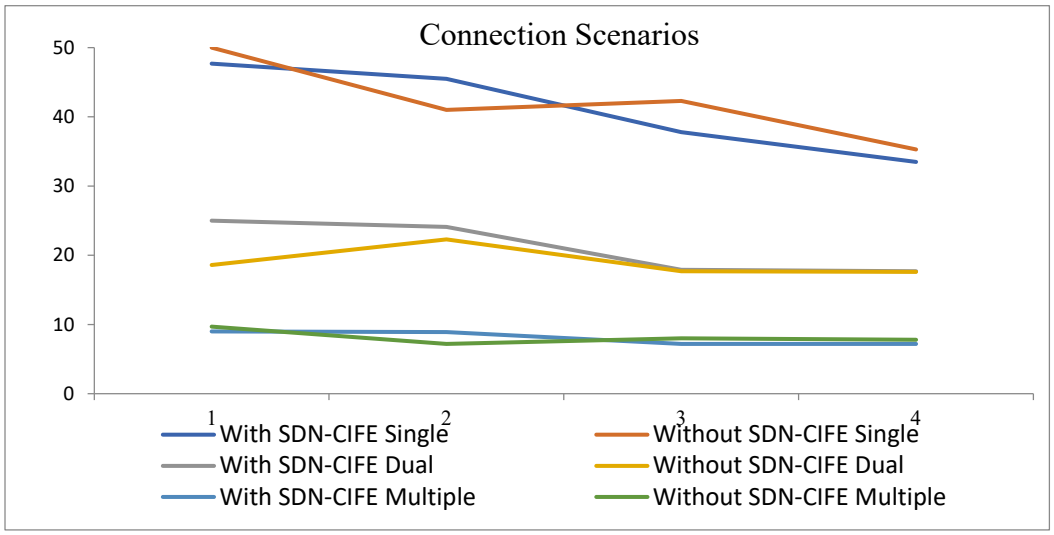

Figure 7. Bandwidth Delivery Test for H3S.

5.7. First Packet Processing Performance:

First, the packet processing time was tested by issuing ICMP echo requests and echo reply messages. We used the "pingallfull" command in the Mininet simulator command line to test the round trip time (RTT). The total time was calculated from the echo request issue until the destination host received an echo reply message.

Table 5 data shows that the M3S network scenario reduced the time required to forward and receive the packets of the connected hosts by 92.54%, while H3S demonstrated a reduction of 91.41%. Our approach achieved more than 83% time reduction with 150 hosts in the 4S network scenario. The experiment results show that our approach, by forwarding and processing the first packet in every network scenario, achieved significant time reductions.

Table 5. First Packet Processing Performance.

Network Scenario	RTT (in ms)		Time Saved
	Without SDN-CIFE	With SDN-CIFE	
SH3	505.623	43.432	91.41%
M3S	214.572	15.999	92.54%
4S	51.809	7.157	86.19%
4S with 150 hosts	130,608.116	20,911.016	83.99%

5.8. Source Address Validator

At the controller, this module checks the source host address against the HostTable to ensure that it is correct. If the controller determines that the source host information is genuine, it sends the packet to the destination switch port; otherwise, it discards the packet and configures the flow-entry to discard similar packets at the switch port level. For PacketIn events, Algorithm-4 creates a module called VALIDATEADDRESS. The event message is used as an input. Using the packet's source host and switch information, it compares the values to those in the HostTable. If the source host and switch information match, the packet is sent; otherwise, the FLOWENTRY-MANAGER method is called to discard the packet and set up a flow entry to discard any future packets with an identical source and destination.

5.9. Flow Entry Manager

The major function of the Flow Entry Manager (FEM) is to manage and configure flow inputs into the OFSwitch. It centers on the RPs of OFSwitches and creates the flow entry required to send host packets to their destination. Depending on the findings of the source-address validator module, it will then disable the flow entry and discard any malicious packets. We use POX and RYU controllers in the system implementation; their flow entry fields are named differently. Figure 6 provides a concise overview of the various fields and their definitions. The flow input fields for various packet types are listed in Table 6. The flow entry is removed automatically when the idle timeout period has elapsed, based on the value stored in the corresponding system variable. If the flow is no longer necessary, it can be removed from the flow table. The HOSTTABLE-HANDLER, HOSTDETECTOR, and VALIDATEADDRESS modules carry out the procedure to establish or terminate flow entry into OFSwitch. The IP, MAC, Switch ID, and Switch Port of the calling host are required, as is the cause for the module's activation.

Table 6. Number of Switches, Hosts, Ports, and Links.

Net Scenario	Hosts	Ports	SW	Links
Tree,3,5	125	185	31	155
Tree,5,3	243	483	121	363
Tree,7,2	128	380	127	254
Hybrid	30	48	10	39

When a new port or host (reason = INSERT) is discovered in the network, FEM creates a flow entry with a high priority to forward the OutPort. When the corresponding network node or port is deleted (reason = DELETE), the entry is erased. When the status of a port or host changes (reason = MODIFY), the existing flow item is deleted, and a new one is added. When the controller detects a malicious packet, the system inserts a flow entry with a very low priority, Idle TimeOut, to cause the packets to be dropped. The Idle TimeOut value is set by the system administrator to meet systemwide security objectives (20 ms, 30 ms, etc.).

In Figure 8, Tree,5,3, all the leaf and root OFSwitches are connected to six other switches. Three interconnected OFSwitches are shown in layer 3 with the notation SW34–SW36; layer 5 and 4 are represented by SW5 and SW4, respectively. The hosts are linked to the OFSwitches at level five. Except for the leaf and root OFSwitches, each OFSwitch in Figure 9 Tree,7,2 is connected to two other switches. The two connected OFSwitches are shown in layer 3 in SW33-SW34, while levels 4–6 are represented by SW4, 4–5 by SW6, and layers 6–7 by SW7. Hosts communicate with OFSwitches on the seventh level. As can be seen in Figure 10, the three-tiered Tree,3,5 architecture features fully linked second-tier OFSwitches at every node. Five interconnected OFSwitches can be seen in the diagram at position SW21–SW25, while position SW3 represents the OFSwitches in the third layer. Hosts connect to the OFSwitches at the third-level.

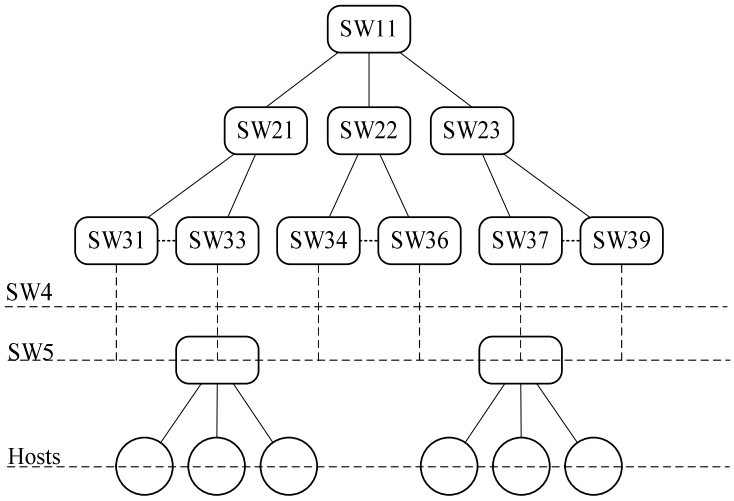

Figure 8. Network Scenario 1: Tree,5,3.

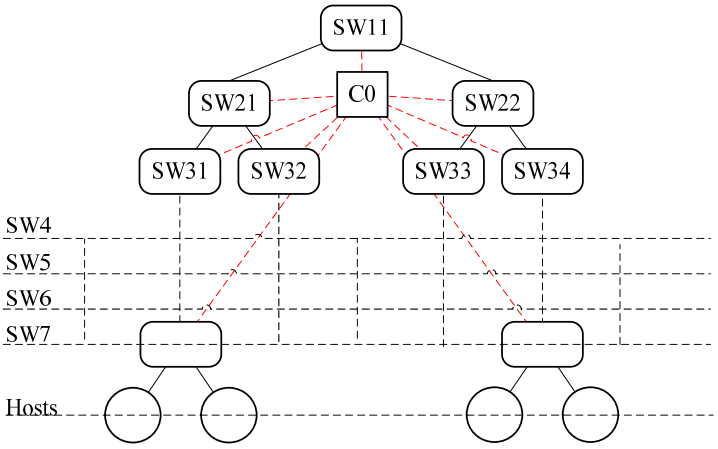

Figure 9. Network Scenario 2: Tree,7,2.

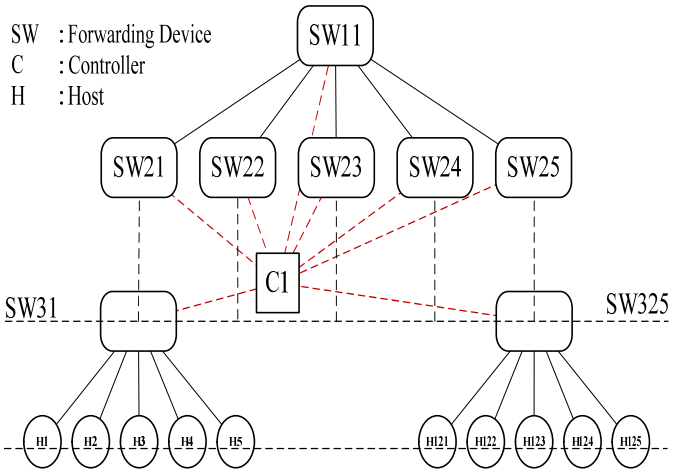

Figure 10. Network Scenario 3: Tree,3,5.

Figure 11 depicts the architecture of a hybrid network [8] using eight OFSwitches and two legacy switches. We dubbed this configuration "Hybrid" since it combines elements of both OFSwitches and non-OFSwitches. The functionality of HyPASS is evaluated in this scenario with non-SDN switches. S1–S2, S4–S7, and S9–S10 in the scenario diagram are all OFSwitches, whereas S3 and S8 are not.

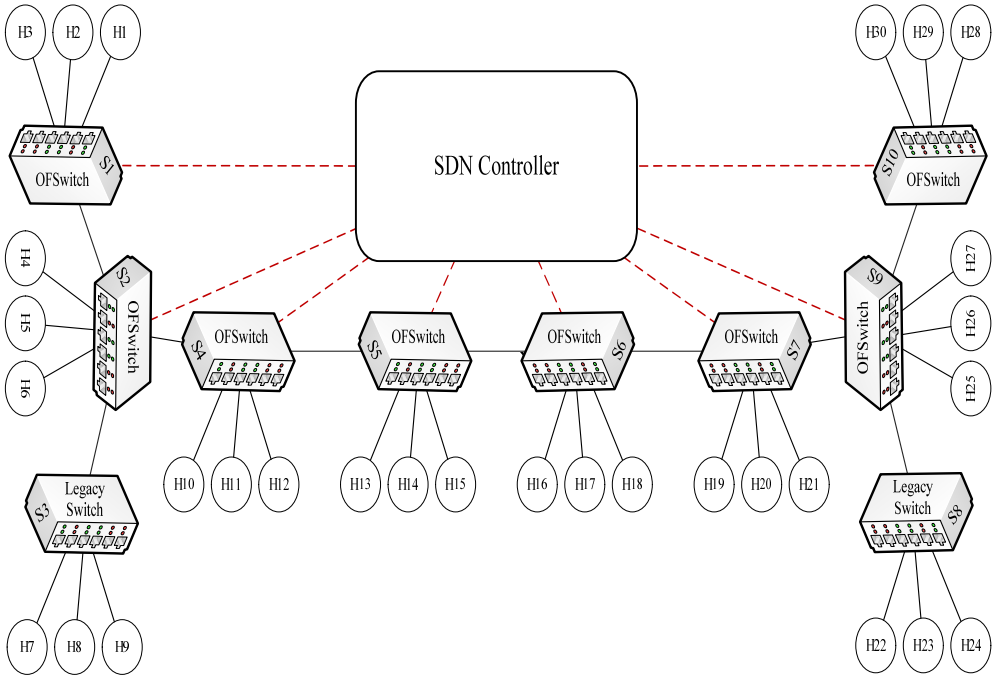

Figure 11. Network Scenario 4: Hybrid.

At the absolute least, one OFSwitch is linked to each and every switch. Connected hosts and switches are represented by the numbers H1 through H30. Information on the hardware and software, SDN controllers, software library, network analyzer tools, and other resources used for SDN validation [31] is summarized in Table 7. During testing, we simulated assaults using the Scapy software library.

Table 7. Performance and Comparative analysis of SDN Source Address Prevention Techniques. (S = Supported, C = SDN Programming/Applications/Services Development, 4 = IPv4).

Exiting Work	Complexity	Filtering Accuracy	Binding/Filter Parameter	Development Cost	IP Support	Messages Used	Dynamic Network	Validation
[1]	Moderate	Less	Network IP-Prefixes	C	4	SNMP packets	S	Network
[6]	Simple	Moderate	IP and Switch Port	C	4	AAM packets	S	Switch Port
[7]	Simple	High	IP, MAC, and Switch Port	C	4	ARP packets	S	Switch Port
[2]	Moderate	Less	Network IP-Prefixes/IGuarantee Header and encryption	C	4	SNMP packets	S	Network
[3]	Moderate	High	MAC	C	4	Packet_In messages	S	Switch Port
[5]	Moderate	Moderate	IP, MAC	C and Inspector	4	Packet_In messages	S	Special Device
[4]	Moderate	High	IP, MAC	C	4	Packet_In messages	S	Switch Port
[23]	Moderate	High	MAC, IP	C	4	ARP messages	S	Switch Port
[24]	Moderate	High	MAC, IP	C	4	TTL Value	S	Host Level
Proposed Method	Simple	High	MAC, IP	C	4	Special ARP packets	S	Controller and Switch Port

6. Conclusions

To improve the first packet processing and forwarding time, we proposed an SDN-CIFE system. During our experiments, the system installed all necessary flow rules in the SDN switch and processed and forwarded the packets based on these rules. The flow rules direct the SDN switch to the correct forwarding route for a destination, but the first packet of connected hosts is not forwarded to the controller. During the handshaking between the SDN switch and controller, the system identifies the connected hosts, prepares the HostLink Table, and loads the required flow entries. Flow entry generation, bandwidth performance, first packet processing, and forwarding time duration are just a few of the parameters that have been tested. Following the implementation of SDN-CIFE, the first packet processing time duration was reduced by more than 83%, while the overall time was reduced by less than 17%. This performance may differ depending on the environment, network topology, and the number of connected hosts. In the future, the SDN-CIFE approach could be used to create security systems, policies, and applications.

Future Work

SAVSH does not record which hosts are linked to it in its "sink" tree. It is not able to provide a blueprint for locating hosts. The method proposed in this paper does not configure the necessary controls and rules prior to traffic creation by the linked host. It represents a new method to identify a live host before real traffic is generated. Before the hosts produce actual traffic, researchers may decide to proactively build up safety standards and control procedures. Both SIPAV-SDN and SDN-SAVI rely on the ARP/AAM messages sent by a host to determine the host configuration. To learn about the host's IP, MAC address, switch port, etc., researchers may devise an in-house method for generating encrypted messages.

Author Contributions: Methodology, R.C.M., P.K.S. and N.V.; Software, S.B.; Validation, A.A.M. and R.H.J.; Investigation, A.K., P.K.S. and S.B.; Resources, N.V.; Data curation, R.C.M. and P.K.S.; Writing—original draft preparation, N.V.; Writing—review & editing, A.A.M., A.K. All authors have read and agreed to the published version of the manuscript.

Funding: Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R151), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ahmad, S.; Mir, A.H. Scalability, Consistency, Reliability and Security in Sdn Controllers: A Survey of Diverse Sdn Controllers. *J. Netw. Syst. Manag.* **2021**, *29*, 9. https://doi.org/10.1007/s10922-020-09575-4.
2. Ahmed, B.; Ahmed, N.; Malik, A.W.; Jafri, M.; Hafeez, T. Fingerprinting Sdn Policy Parameters: An Empirical Study. *IEEE Access* **2020**, *8*, 142379–142392. https://doi.org/10.1109/access.2020.3012176.
3. Al-Harbi, A.; Bahnasse, A.; Louhab, F.E.; Talea, M. Towards an Efficient Resource Allocation Based on Software-Defined Networking approach. *Comput. Electr. Eng.* **2021**, *92*, 107066. https://doi.org/10.1016/j.compeleceng.2021.107066.
4. Alharbi, T. Deployment of Blockchain Technology in Software Defined Networks: A Survey. *IEEE Access* **2020**, *8*, 9146–9156. https://doi.org/10.1109/access.2020.2964751.
5. Barguil, S.; Lopez, V.; Manta-Caro, C.; De Lerma, A.M.L.; De Dios, O.G.; Echeverry, E.; Fernandez-Palacios, J.P.; Karvonen, J.; Kemppainen, J.; Maya, N.; et al. Field Trial of Programmable L3 Vpn Service Deployment Using Sdn-Based Multi-Domain Service Provisioning over Ip/Optical Networks. *IEEE Netw.* **2021**, *35*, 217–224. https://doi.org/10.1109/mnet.011.2100006.
6. Chica, J.C.C.; Imbachi, J.C.; Vega, J.F.B. Security in Sdn: A Comprehensive Survey. *J. Netw. Comput. Appl.* **2020**, *159*, 102595. https://doi.org/10.1016/j.jnca.2020.102595.
7. Patel, N.J.; Jhaveri, R.H. Trust based approaches for secure routing in VANET: A survey. *Procedia Comput. Sci.* **2015**, *45*, 592–601. https://doi.org/10.1016/j.procs.2015.03.112.
8. Meena, R.C.; Bhatia, S.; Jhaveri, R.H.; Cheng, L.; Kumar, A.; Mashat, A. HyPASS: Design of hybrid-SDN prevention of attacks of source spoofing with host discovery and address validation. *Phys. Commun.* **2022**, *55*, 101902. https://doi.org/10.1016/j.phycom.2022.101902.
9. Das, R.K.; Pohrmen, F.H.; Maji, A.K.; Saha, G. Ft-Sdn: A Fault-Tolerant Distributed Architecture for Software Defined Network. *Wirel. Pers. Commun.* **2020**, *114*, 1045–1066. https://doi.org/10.1007/s11277-020-07407-x.
10. Dawadi, B.R.; Thapa, A.; Guragain, R.; Karki, D.; Upadhaya, S.P.; Joshi, S.R. Routing Performance Evaluation of a Multi-Domain Hybrid Sdn for Its Implementation in Carrier Grade Isp Networks. *Appl. Syst. Innov.* **2021**, *4*, 46. https://doi.org/10.3390/asi4030046.
11. Deb, R.; Roy, S. A Software Defined Network Information Security Risk Assessment Based on Pythagorean Fuzzy Sets. *Expert Syst. Appl.* **2021**, *183*, 115383. https://doi.org/10.1016/j.eswa.2021.115383.
12. Djeldjeli, Y.; Zoubir, M. Cp-Sdn: A New Approach for the Control Operation of 5g Mobile Networks to Improve QoS. *Eng. Technol. Appl. Sci. Res.* **2021**, *11*, 6857–6863. https://doi.org/10.48084/etasr.4016.
13. Geng, H.; Yao, J.; Zhang, Y. Single Failure Routing Protection Algorithm in the Hybrid Sdn Network. *Comput. Mater. Contin.* **2020**, *64*, 665–679. https://doi.org/10.32604/cmc.2020.09912.
14. Hamdan, M.; Hassan, E.; Abdelaziz, A.; Elhigazi, A.; Mohammed, B.; Khan, S.; Vasilakos, A.V.; Marsono, M. A Comprehensive Survey of Load Balancing Techniques in Software-Defined Network. *J. Netw. Comput. Appl.* **2021**, *174*, 102856. https://doi.org/10.1016/j.jnca.2020.102856.
15. He, H.; Song, Y.; Xiao, T.; Rehman, H.U.; Nie, L. Design of Software-Defined Network Experimental Teaching Scheme Based on Virtualised Environment. *Appl. Math. Nonlinear Sci.* **2021**, *6*, 181–192. https://doi.org/10.2478/amns.2021.2.00005.
16. Hou, J.; Zhang, M.; Zhang, Z.; Shi, W.; Qin, B.; Liang, B. On the Fine-Grained Fingerprinting Threat to Software-Defined Networks. *Futur. Gener. Comput. Syst.* **2020**, *107*, 485–497. https://doi.org/10.1016/j.future.2020.01.046.
17. Huang, X.; Zeng, M.; Xie, K. Intelligent Traffic Control for Qos Optimization in Hybrid Sdns. *Comput. Netw.* **2021**, *189*, 107877. https://doi.org/10.1016/j.comnet.2021.107877.
18. Ibrar, M.; Wang, L.; Muntean, G.-M.; Akbar, A.; Shah, N.; Malik, K.R. Prepass-Flow: A Machine Learning Based Technique to Minimize Acl Policy Violation Due to Links Failure in Hybrid Sdn. *Comput. Netw.* **2021**, *184*, 107706. https://doi.org/10.1016/j.comnet.2020.107706.
19. Islam, T.; Islam, N.; Refat, A. Node to Node Performance Evaluation through Ryu Sdn Controller. *Wirel. Pers. Commun.* **2020**, *112*, 555–570. https://doi.org/10.1007/s11277-020-07060-4.
20. Kang, H.; Yegneswaran, V.; Ghosh, S.; Porras, P.; Shin, S. Automated Permission Model Generation for Securing Sdn Control-Plane. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 1668–1682. https://doi.org/10.1109/tifs.2019.2946928.
21. Lee, S.; Kim, J.; Woo, S.; Yoon, C.; Scott-Hayward, S.; Yegneswaran, V.; Porras, P.; Shin, S. A Comprehensive Security Assessment Framework for Software-Defined Networks. *Comput. Secur.* **2020**, *91*, 101720. https://doi.org/10.1016/j.cose.2020.101720.

22. Bello, L.L.; Lombardo, A.; Milardo, S.; Patti, G.; Reno, M. Experimental Assessments and Analysis of an Sdn Framework to Integrate Mobility Management in Industrial Wireless Sensor Networks. *IEEE Trans. Ind. Inform.* **2020**, *16*, 5586–5595. https://doi.org/10.1109/tii.2020.2963846.
23. Mahrach, S.; Haqiq, A. Ddos Flooding Attack Mitigation in Software Defined Networks. *Int. J. Adv. Comput. Sci. Appl.* **2020**, *11*, 693–700.
24. Martinez-Yelmo, I.; Alvarez-Horcajo, J.; Carral, J.A.; Lopez-Pajares, D. Ehddp: Enhanced Hybrid Domain Discovery Protocol for Network Topologies with Both Wired/Wireless and Sdn/Non-Sdn Devices. *Comput. Netw.* **2021**, *191*, 107983. https://doi.org/10.1016/j.comnet.2021.107983.
25. Nguyen, H.N.; Tran, H.A.; Fowler, S.; Souihi, S. A Survey of Blockchain Technologies Applied to Software-Defined Networking: Research Challenges and Solutions. *IET Wirel. Sens. Syst.* **2021**, *11*, 233–247. https://doi.org/10.1049/wss2.12031.
26. Papavassiliou, S. Software Defined Networking (Sdn) and Network Function Virtualization (Nfv). *Futur. Internet* **2020**, *12*, 7. https://doi.org/10.3390/fi12010007.
27. Rischke, J.; Sossalla, P.; Salah, H.; Fitzek, F.H.P.; Reisslein, M. Qr-Sdn: Towards Reinforcement Learning States, Actions, and Rewards for Direct Flow Routing in Software-Defined Networks. *IEEE Access* **2020**, *8*, 174773–174791. https://doi.org/10.1109/access.2020.3025432.
28. Srisamarn, U.; Pradittasnee, L.; Kitsuwan, N. Resolving Load Imbalance State for Sdn by Minimizing Maximum Load of Controllers. *J. Netw. Syst. Manag.* **2021**, *29*, 46. https://doi.org/10.1007/s10922-021-09612-w.
29. Wei, S.H.; Chin, T.S.; Kwang, L.C. Cost-Location Aware Heuristic Algorithm for Hybrid Sdn Deployment. *Ann. Math. Artif. Intell.* **2021**, *89*, 875–897. https://doi.org/10.1007/s10472-021-09750-6.
30. Yao, Z.; Yan, Z. A Trust Management Framework for Software-Defined Network Applications. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e4518. https://doi.org/10.1002/cpe.4518.
31. Jhaveri, R.H.; Patel, S.J.; Jinwala, D.C. Improving route discovery for AODV to prevent blackhole and grayhole attacks in MANETs. *INFOCOMP J. Comput. Sci.* **2012**, *11*, 1–12.