Al-Khwarizmi Engineering Journal
ISSN (printed): 1818 – 1171, ISSN (online): 2312 – 0789
Vol. 21, No. 2, June, (2025), pp. 93–105

Al-Khwarizmi Engineering Journal

# Recent Tools of Software-Defined Networking Traffic Generation and Data Collection

**Tabarak Yassin Khudhair[1*] and Omar Ali Athab[2]**
[1,2]*Department of Information and Communications Engineering, Al-Khwarizmi College of Engineering, University of Baghdad, Baghdad, Iraq*
Corresponding Author's Email:Tabarak.taha1603@kecbu.uobaghdad.edu.iq

(Received 13 November 2023; Revised 7 June 2024; Accepted 26 June 2024; Published 1 June 2025)
**https://doi.org/10.22153/kej.2025.06.002**

## Abstract

Software-defined networking (SDN) has proven its superiority in addressing ordinary network problems, such as scalability, agility and security. This advantage of SDN comes because of its separation of the control plane from the data plane. Although many studies have focused on SDN management, monitoring, control and improving quality of service, only a few are focused on presenting what is used to generate traffic and collect data. The literature also lacks comparisons amongst the tools and methods used in this context. Therefore, this study introduces the recent tools used to simulate, generate and obtain traffic statistics from an SDN environment. In addition, the methods used in SDN data gathering are compared to explore the capability of each one and hence, determine the suitable environment for each method. The SDN testbed is simulated using Mininet software with tree topology and OpenFlow switches. An RYU controller was connected to control forwarding. The famous tools iperf3, Ping and python scripts are used to generate network datasets from selected devices in the network. Wireshark, the RYU application and the ovs-ofctl command are used to monitor and gather the dataset. Results show success in generating several types of network metrics to be used in the future for training machine or deep learning algorithms. Therefore, when generating data for the purpose of congestion control, iperf3 is the best tool, whilst Ping is useful when generating data for the purpose of detecting distributed denial-of-service attacks. RYU applications are the most suitable monitoring tool for obtaining all network details, such as the topology, characteristics and statistics of the components. Many obstacles and mistakes are also explored and listed to be prevented when researchers try to generate such datasets in their next scientific efforts.

***Keywords:*** *SDN; OpenFlow1.3; Ryu controller; Mininet; iperf3; Wireshark; Python.*

## 1. Introduction

Software-defined networking (SDN) is a networking paradigm that decouples the network's control plane from the data plane [1]. In traditional networks, the control plane and data plane are coupled. In other words, the network devices, such as routers and switches, are responsible for making decisions about how to forward traffic [2].

The separation of the control and data planes in SDN provides a number of advantages, such as the ease of making changes to network configuration. The changes can be made to the centralised controller rather than to individual devices, which can be particularly beneficial in many environments, such as data centres [3], [4], 5G [5] and IoT networks [6], where the number and types of workloads can change frequently. SDN also allows for the network to be programmed using software applications.

Many studies have been conducted in the field of generating and collecting SDN flows. Mu [4] used the built-in iperf tool with a Python script to simulate 25.6 MB and 256 KB flows using the transmission control protocol (TCP)/internet protocol in 1 Gb/s links. The configuration overhead was reduced, and execution time was improved by 50%, but how a built-in iperf was used inside a Python script is not explained. Jiang et al. [6] utilised the iperf tool to generate traffic and send it in links that have a bandwidth (BW) of 1 Gbps and

a round-trip time (RTT) of 0.3 ms to achieve high throughput with increasing packet loss rates and RTT to control congestion; however, the controller's type or version is not referred to. Zhao et al. [7] employed the iperf tool to generate traffic between pairs of hosts in intervals equal to 10 s, and the flow load increased from 0 Mbit/s to 25 Mbit/s to create congestion in the network. SDN controller messages were also used to monitor network status and request information about the topology, quality of service (QoS) and links. Given their efforts, congestion was successfully controlled and reduced, and network performance was increased. However, the specific built-in or external tool that was used, the version of the tool and the protocol used in their work are not explained. Wu et al. [8] used the iperf tool to simulate user datagram protocol (UDP) traffic between three host pairs at the same time with a changed transmission rate to predict link congestion and reach 95.4% accuracy. Meena et al. [9] investigated Wireshark, the ovs-ofctl dump-flows instruction and the Pingallfull command to reduce the first packet forwarding time and succeeded in reducing 83% of the forwarding time. Wireshark was used to capture and analyse packets, the ovs-ofctl dump-flows instruction to capture and analyse flow entries of OpenFlow switches, and a built-in Pingallfull command on the Mininet to measure RTT, yet why the Pingallfull command was selected instead of Ping and the advantages of using this tool are not explained. Ali et al. [10] applied a built-in iperf3 tool inside Mininet to generate TCP and UDP traffic between hosts and measure the throughput and transfer data to enhance traffic load balancing, even though the BW and transfer duration of the generated traffic is not determined by them. Nougnanke [11] employed the iperf tool to measure delay, jitter and BW to control congestion. Diel et al. [12] used the iperf3 tool to generate 5 GB of flows and send them in links with 1 Gbps of capacity, and the RYU controller was used to collect the sent and received numbers of packets, bytes, drops, errors and collisions to avoid congestion in the data centre, which succeeded in reducing flow completion time by half, but the specific tool or method used to collect data was not clarified. Mohsin and Hamad [13] implemented a Ping command inside a Python script to generate internet control message protocol (ICMP) flood traffic between hosts at different rates for single topology, linear topology and linear with multicontroller topology, respectively, to detect and eliminate DDoS attacks. Increased switches in single topology reduce load, eliminate attack effects quickly and minimise detection and mitigation time. Increased controllers enhance detection and mitigation by minimising error rates, but how this method was implemented is not stated.

This research aims to address the shortcomings found in the literature, such as the lack of details about the tools and methods used to generate data in SDN networks. Failing to explain how to hire many tools or how to utilise different tools to accomplish the same task is a shortcoming of many researchers. Moreover, this work focuses on avoiding practical programming obstacles during data generation, such as the mismatch between the versions of all of the software and the programming languages used in each of the network components. This study also addresses technical errors to achieve an acceptable reliability in the data, e.g. time synchronisation when using more than one tool between specific host pairs, when used as a dataset input in machine learning and deep learning training.

Section 2 explains the SDN layers and devices used in the structure of the network. Section 3 demonstrates the work infrastructure and design. Section 4 presents the simulated topology used in this study. Section 5 shows the methods and tools used to generate traffic in SDN. Section 6 states methods and tools used to monitor network statistics and performance. Section 7 conducts a comparison between all the methods and tools included in this study. Section 8 contains the main conclusion of the study.

## 2. SDN Layers and Devices

The SDN network system model consists of three layers: the data layer, the control layer and the application layer [14]. The application layer contains network services and applications that interact with the control layer [٥].

The control layer provides a centralised global view of the entire network, represented by controllers, that provides updates and collects data from network switches [12]. Given a centralised view of every device in the network, all network characteristics and metrics can be monitored and measured. The controller gathers real-time statistics from all the switches through OpenFlow messages that inquire the state of the switches periodically [16], allowing the controller to be directly programmable and have full management of the switches [17]. The controller manages the switches through the southbound interface, whilst the application layer communicates with the controller through the northbound interface. The northbound interface personalises the application of the SDN network [18].

The data layer consists of OpenFlow switches and terminal hosts that transmit packets according to the rules imposed by the control layer [19]. SDN uses the OpenFlow protocol to deal with OpenFlow switches, allowing the controller and switches to understand each other. OpenFlow switches consist of several flow tables [3]. A flow table is composed of multiple flow entries that include the following: match fields, used to match flows; priority, used to match the priority of the flow; counters, to be refreshed to match the packets; instructions, to determine the action taken; timeouts, the total time or idle time before the timeout ends; and cookies, the implicit value of the data assigned by the controller [20].

## 3. Infrastructure and Design of the Work

The work was divided into four main parts: topology simulation, traffic generation, data collection and tool comparison, as shown in Figure 1. The topology simulation part consists of a description of the simulator, starting from a simulation of the topology, then its connection to the controller, until the operation of all the network components.

The second part explains how traffic is generated using different tools such as Iperf3, Ping, and Python scripts.

Afterwards, in part three, data are collected on different network devices using many applications and tools such as Wireshark, SDN controllers and a Mininet simulator.

In the final part, all the tools that generate traffic were compared, and the data are collected separately.

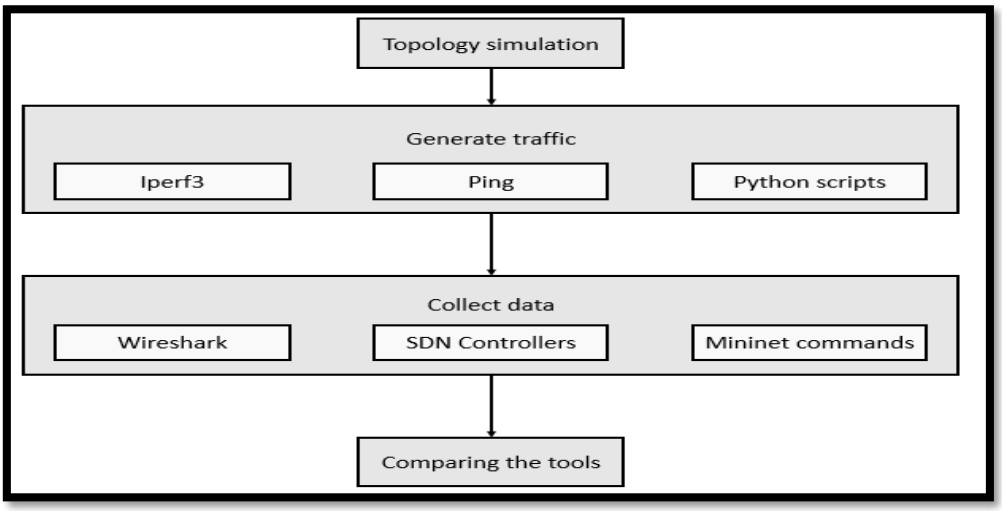



**Fig. 1. Workflow diagram**

## 4. Simulation

This research was performed to generate and collect flow statistics from the SDN switches. The topology used in this study was created in a Mininet simulator [21], and the OpenFlow protocol v1.3 was used. Version 2.3.1b4 of Mininet, a software simulator, was installed on a Linux operating system and initialised for the SDN network simulation. It can simulate many network topologies, such as star, linear, tree, etc. An RYU controller is an SDN framework software, version 4.34 was used in this work. The RYU controller is coded with Python and installed separately from Mininet but has the ability to connect with Mininet. The RYU controller was selected instead of the default POX controller because of its ability to support all OpenFlow protocol versions. As shown in Figure 2, the network was made in the form of a tree topology in depth 3, which contains one controller, seven switches and eight hosts. The switches are provided by Mininet and run using Open vSwitch version 2.13.8. All switches have three ports, except S1, which has two ports, which work as sending and receiving ports. The hosts are PCs provided by Mininet to send traffic.

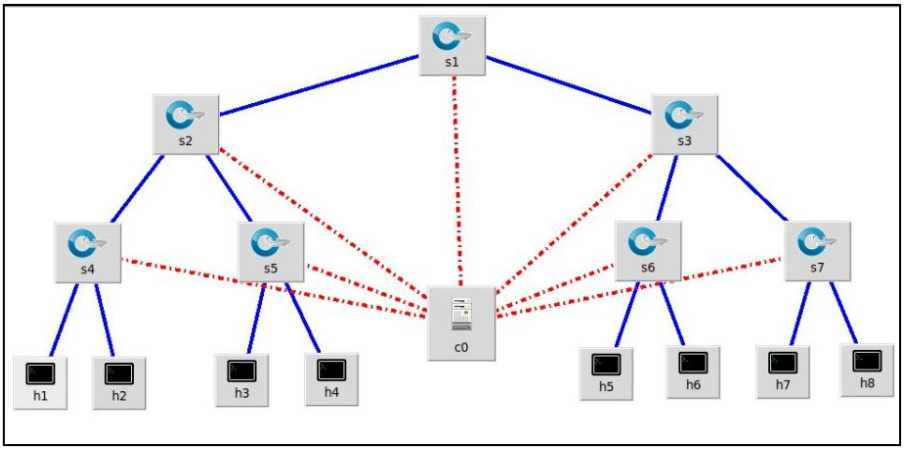



**Fig. 2. SDN topology**

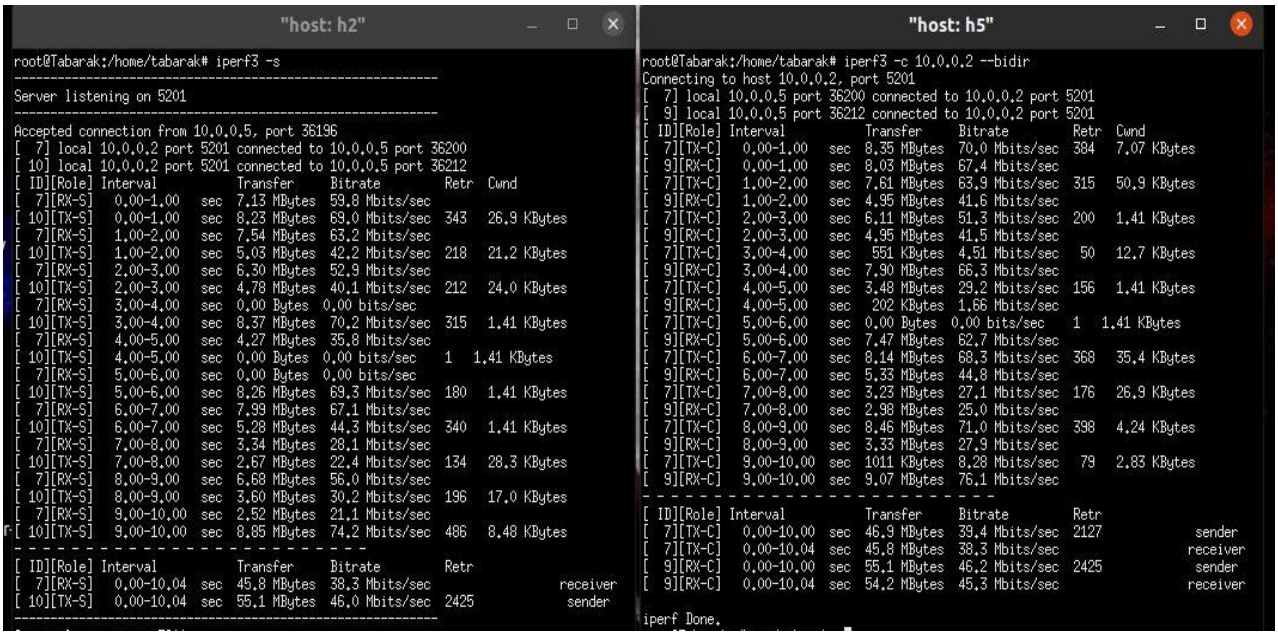
## 5. Methodologies to Generate SDN Traffic

Many tools are built-in or built-out in the Mininet simulator to generate traffic between end hosts and measure specific network performance attributes. The most common and widely used tools are iperf, Ping and Python scripts.

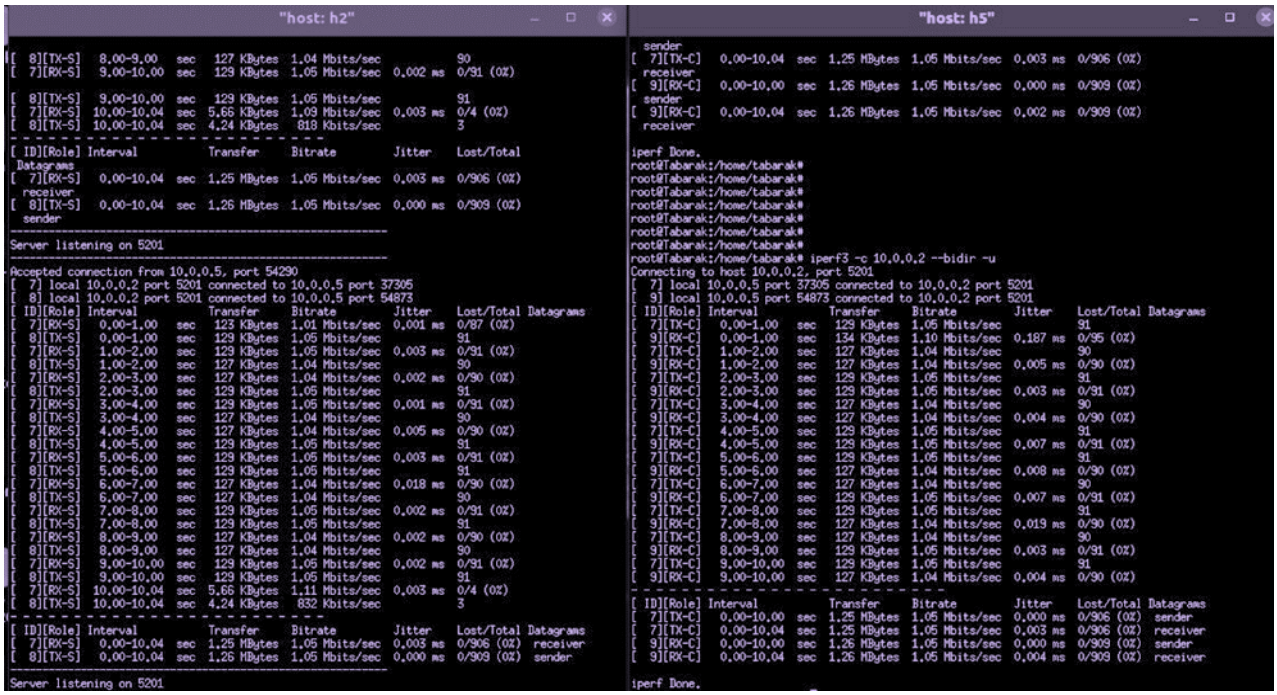
### 5.1. Iperf3

Iperf [22] is a built-out Mininet tool that generates and simulates TCP and UDP traffic between any pair of hosts, one of which is a server and the other a client. The iperf tool can also measure network performance parameters such BW, jitter and transferring data .

Figure 3 shows a sample of the generated traffic using the iperf3 tool between h2 as a server and h5 as a client; given that they are located in two different subtrees, this forces the traffic to pass through most of the switches to reach the client. TCP is used in part A of Figure 3, and UDP is used in part B. The ID in the list represents the identification number of the connection. The 'Interval' is the time interval to report the throughput. 'Transfer' represents how much data was transferred in each interval. 'Bitrate' is the measured throughput in each interval. 'Retr' is the number of TCP segments retransmitted in each interval. 'Congestion Window' (CWND) indicates the congestion window size in each interval. 'Jitter' represents the difference in packet delay. Finally, 'Lost/Total' indicates the number of lost datagrams over the total number of datagrams .




**(a) TCP traffic**



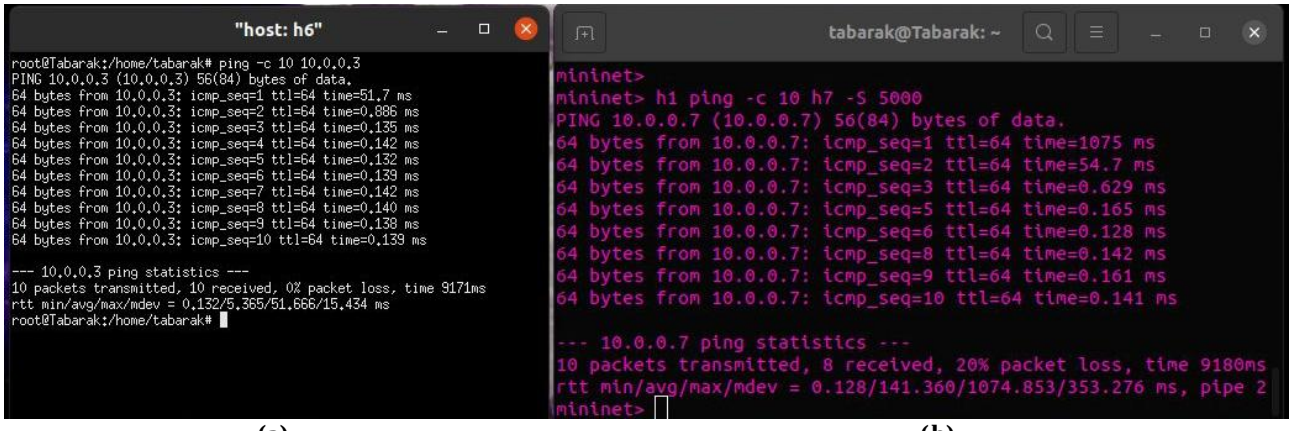

**(b) UDP traffic**

**Fig. 3. Output of Iperf3 tool to generate (a) TCP traffic, (b) UDP traffic**

## 5.2. Ping

Ping is another tool that generates ICMP traffic between hosts and sends ICMP Echo Request and Echo Reply messages over the network to a targeted host. Ping can be used to determine the number of transmitted and received packets, packet loss percentage, total time of transmission and the RTT for end-to-end network hosts .

Figure 4 shows the output result of using the Ping tool in the SDN topology mentioned in Figure 2. As shown in Figure 4-a, h6 Pings in 10 packets to h3 by using its IP, whereas in Figure 4-b, a built-in Ping tool is used between h1 and h7 by sending 10 packets with a 5 KB data size. Hosts are selected to measure the performance between different subtrees on each side.




**(a)** **(b)**

**Fig. 4. Using a Ping tool to generate traffic: (a) external host terminal, (b) built-in tool**

## 5.3. Python Scripts

The Mininet simulator supports the ability to run built-out Python scripts sequentially within each endpoint host, thus allowing TCP or UDP traffic to be generated and sent to the predefined destination at the given port number.

Figure 5 shows an example of a TCP client – server program. The program is a simple e-mail application where the server is running on host 3 and the clients, on hosts 4 and 8, connect to the server to view received e-mails or send a new one.

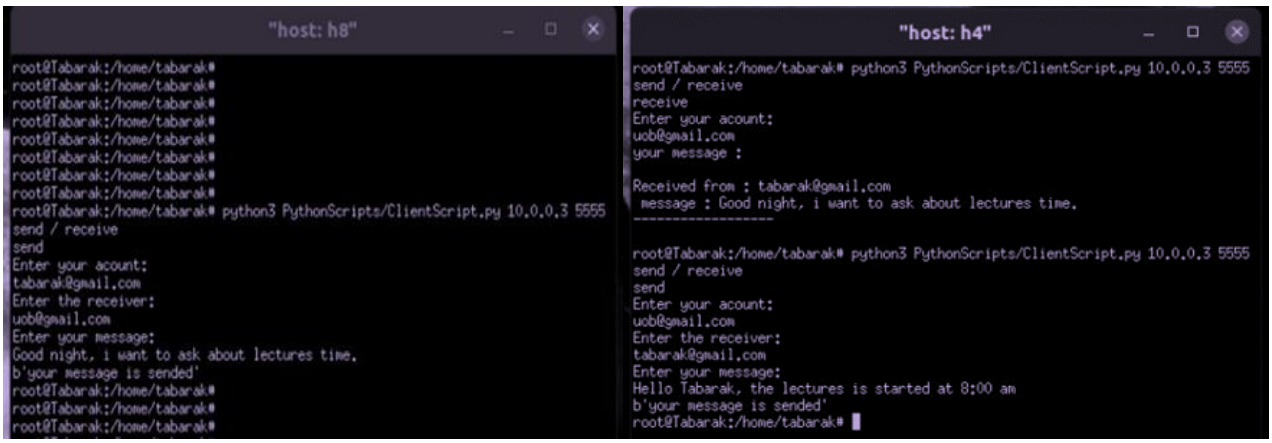



**Fig. 5. Running python script to generate traffic between hosts**

## 6. Methods of Traffic Monitoring

Many different monitoring techniques are used with Mininet to capture statistics about the SDN network by using some of the OpenFlow messages specified in the OpenFlow specification [23]. The most common methods used to obtain network statistics are the controller itself and the wireshark tool.

### 6.1. Wireshark Tool

Wireshark [24] is a packet-capture tool. Wireshark contains three panes called the Packet List pane, the Packet Details pane, and the Packet Bytes pane. The Packet List pane displays packets from a selected interface. The Packet Details pane displays the protocol fields used in the selected packet from the Packet List pane. Finally, the Packet Bytes pane displays the data of a packet in a hex dump .

Figure 6 shows a Wireshark window that runs in the s1 to display OpenFlow packets between the switch and the controller in the topology of Figure 2. As shown, the Packet List pane displays the packet number, capture time of a packet, IP address and port number of the source and destination, length of packet and the packet message type. When selecting a packet, more details about the header and data are presented in the Packet Details and Packet Bytes panes .

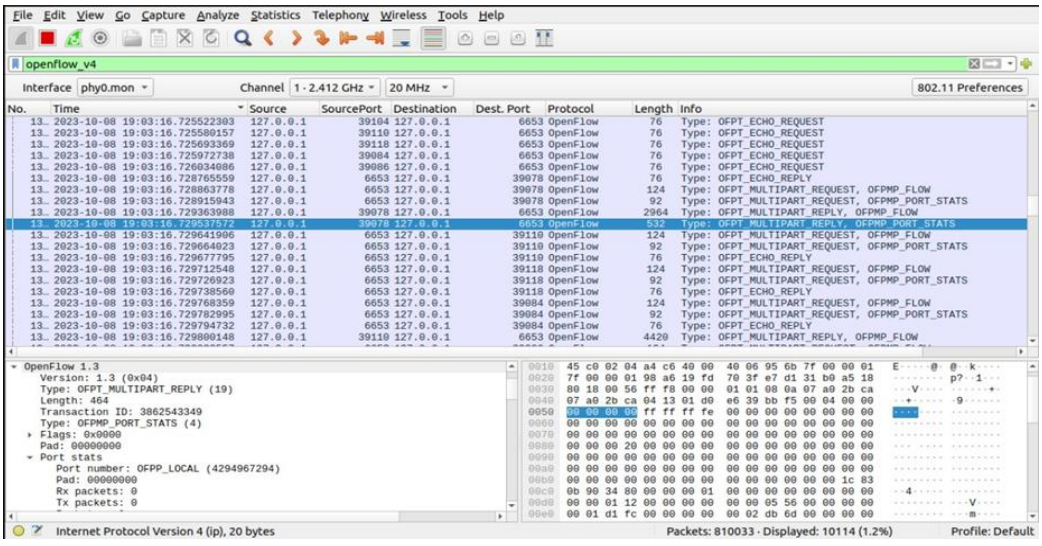


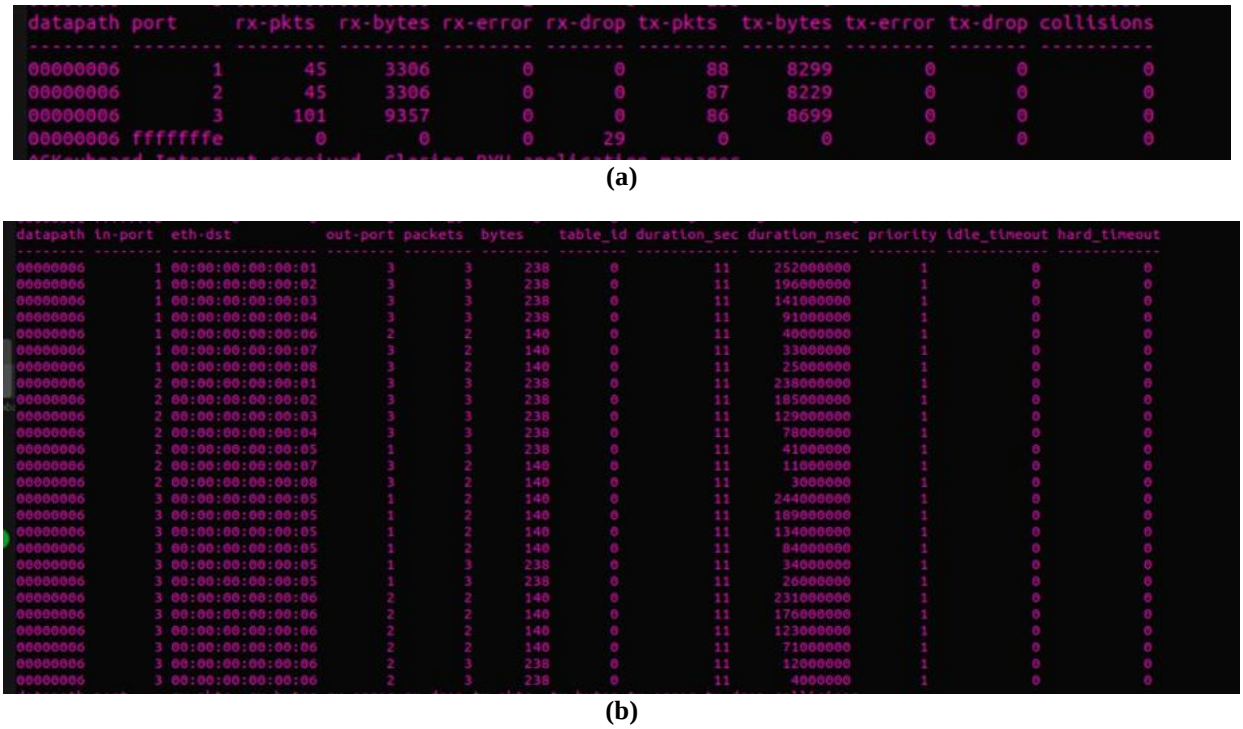

**Fig. 6. Output of Wireshark tool**

## 6.2. SDN Controller Applications

The controller is connected to SDN switches through the OpenFlow protocol. The controller sends request messages to acquire statistics about the port, flow entry, queue, group, meter and topology statistics. When a switch receives a 'request' message, it sends out a response with a 'reply' message. This message holds the statistics collected from the switch counters and can also query about the link load, network topology and network QoS parameters .

### 6.2.1. Simple_monitor_13.py

The controller sends 'OFPortStatsRequest' and 'OFPFlowStatsRequest' messages and receives 'OFPortStatsReply' and 'OFPFlowStatsReply' messages that include information about both port statistics, the transmitted and received number of dropped packets, bytes, errors and collisions. It also receives messages about the flow statistics, the duration of the flow, packet and byte count .
Figure 7 shows the output of using the RYU Simple_monitor_13.py script to determine which attributes' statistics are needed in 'OFPortStatsReply' and 'OFPFlowStatsReply' messages for switch 6 in Figure 2.



| datapath | port | rx-pkts | rx-bytes | rx-error | rx-drop | tx-pkts | tx-bytes | tx-error | tx-drop | collisions |
|---|---|---|---|---|---|---|---|---|---|---|
| 00000006 | 1 | 45 | 3306 | 0 | 0 | 88 | 8299 | 0 | 0 | 0 |
| 00000006 | 2 | 45 | 3306 | 0 | 0 | 87 | 8229 | 0 | 0 | 0 |
| 00000006 | 3 | 101 | 9357 | 0 | 0 | 86 | 8699 | 0 | 0 | 0 |
| 00000006 | fffffffe | 0 | 0 | 0 | 29 | 0 | 0 | 0 | 0 | 0 |

**(a)**

| datapath | in-port | eth-dst | out-port | packets | bytes | table_id | duration_sec | duration_nsec | priority | idle_timeout | hard_timeout |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000006 | 1 | 00:00:00:00:00:01 | 3 | 3 | 238 | 0 | 11 | 252000000 | 1 | 0 | 0 |
| 00000006 | 1 | 00:00:00:00:00:02 | 3 | 3 | 238 | 0 | 11 | 196000000 | 1 | 0 | 0 |
| 00000006 | 1 | 00:00:00:00:00:03 | 3 | 3 | 238 | 0 | 11 | 141000000 | 1 | 0 | 0 |
| 00000006 | 1 | 00:00:00:00:00:04 | 3 | 3 | 238 | 0 | 11 | 91000000 | 1 | 0 | 0 |
| 00000006 | 1 | 00:00:00:00:00:06 | 2 | 2 | 140 | 0 | 11 | 40000000 | 1 | 0 | 0 |
| 00000006 | 1 | 00:00:00:00:00:07 | 3 | 2 | 140 | 0 | 11 | 33000000 | 1 | 0 | 0 |
| 00000006 | 1 | 00:00:00:00:00:08 | 3 | 2 | 140 | 0 | 11 | 25000000 | 1 | 0 | 0 |
| 00000006 | 2 | 00:00:00:00:00:01 | 3 | 3 | 238 | 0 | 11 | 238000000 | 1 | 0 | 0 |
| 00000006 | 2 | 00:00:00:00:00:02 | 3 | 3 | 238 | 0 | 11 | 185000000 | 1 | 0 | 0 |
| 00000006 | 2 | 00:00:00:00:00:03 | 3 | 3 | 238 | 0 | 11 | 129000000 | 1 | 0 | 0 |
| 00000006 | 2 | 00:00:00:00:00:04 | 3 | 3 | 238 | 0 | 11 | 78000000 | 1 | 0 | 0 |
| 00000006 | 2 | 00:00:00:00:00:05 | 1 | 3 | 238 | 0 | 11 | 41000000 | 1 | 0 | 0 |
| 00000006 | 2 | 00:00:00:00:00:07 | 3 | 2 | 140 | 0 | 11 | 11000000 | 1 | 0 | 0 |
| 00000006 | 2 | 00:00:00:00:00:08 | 3 | 2 | 140 | 0 | 11 | 3000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:05 | 1 | 2 | 140 | 0 | 11 | 244000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:05 | 1 | 2 | 140 | 0 | 11 | 189000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:05 | 1 | 2 | 140 | 0 | 11 | 134000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:05 | 1 | 2 | 140 | 0 | 11 | 84000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:05 | 1 | 3 | 238 | 0 | 11 | 34000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:05 | 1 | 3 | 238 | 0 | 11 | 26000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:06 | 2 | 2 | 140 | 0 | 11 | 231000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:06 | 2 | 2 | 140 | 0 | 11 | 176000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:06 | 2 | 2 | 140 | 0 | 11 | 123000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:06 | 2 | 2 | 140 | 0 | 11 | 71000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:06 | 2 | 3 | 238 | 0 | 11 | 12000000 | 1 | 0 | 0 |
| 00000006 | 3 | 00:00:00:00:00:06 | 2 | 3 | 238 | 0 | 11 | 4000000 | 1 | 0 | 0 |

**(b)**

**Fig. 7. Simple_monitor_13.py outputs for (a) port statistics and (b) flow statistics**

### 6.2.2. Gui-topology.py

Gui-topology.py is a distinct RYU application that views the topology's switches, ports of each link and the switch ID as data-path-id (dpid). This script also provides information about flows for each switch. Figure 8 shows the information that was provided from the gui-topology application.

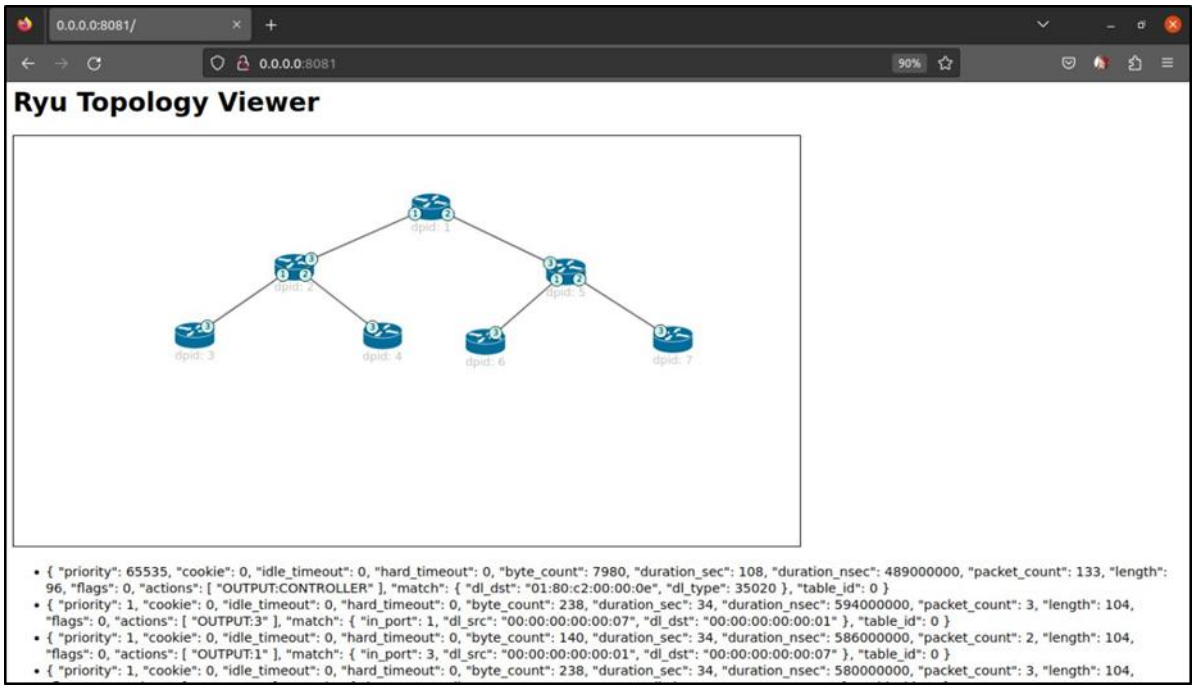



**Fig. 8. Gui-topology application for viewing SDN topology and flow statistics**

### 6.2.3. Flowmanager.py

Flowmanager.py [25] is a web application that views the switches and hosts of the topology, ports in each switch and host, switch id and host mac address. This application can also provide information about switches. Figure 9 shows the topology that was presented using the flowmanager.py application.

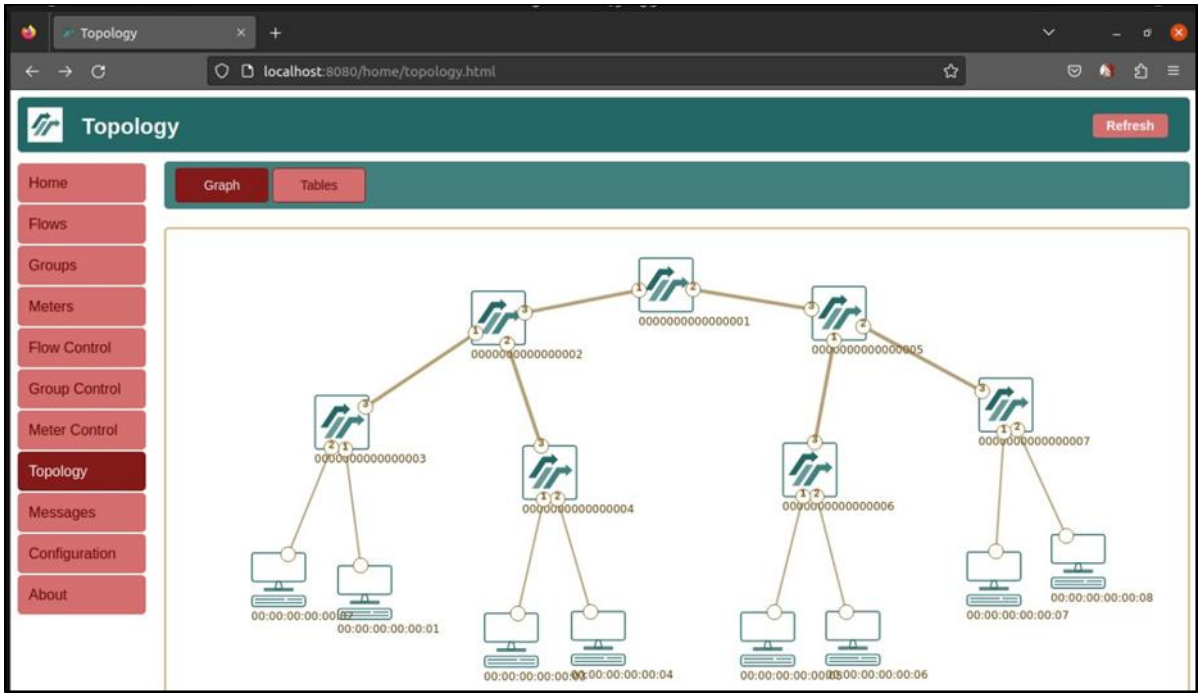



**Fig. 9. Flowmanager application for viewing SDN topology and switch statistics**

## 6.3. Mininet Commands

Mininet offers a built-in command-line interface (CLI) for managing virtual networks and communicating with them. Network topologies, network starts and stops, network device configurations and connectivity tests are all accomplished with the use of CLI commands .

### 6.3.1. Built-in Iperf

Figure 10 shows the result of using a built-in iperf tool to measure the BW of TCP and UDP protocols between two pairs of hosts from Figure 2. When comparing the results with the results of Figure 3, the same results are present, but a built-in tool divided the BW into uplink and downlink.

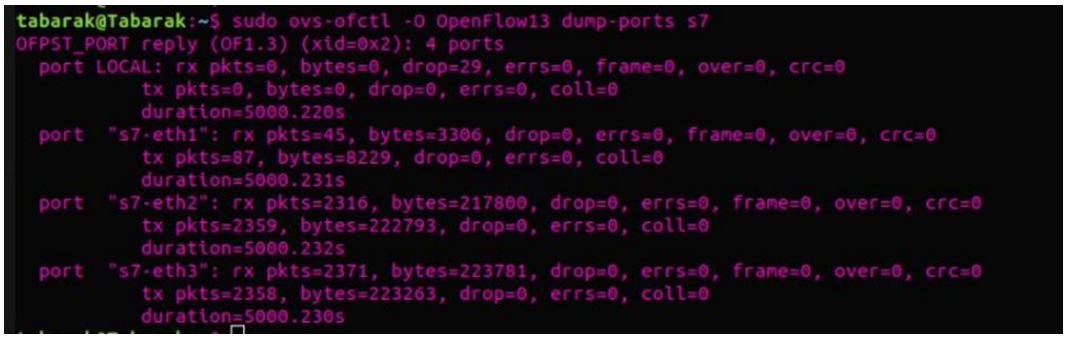

**(a) TCP traffic** **(b) UDP traffic**

**Fig. 10. Use of a built-in Iperf tool to measure the BW of (a) TCP traffic and (b) UDP traffic**

### 6.3.2. Built-in Ovs-ofctl

One of the most important tools for controlling and adjusting Open vSwitch (OVS) switches in Mininet networks is the ovs-ofctl command. It offers an extensive feature set for specifying switch parameters, collecting performance data and managing switch behavior. By understanding and utilising the ovs-ofctl command, users can effectively manage and optimise Mininet networks for various testing and experimentation purposes

Figure 11 shows the results when ovs-ofctl commands are called in for port statistics in part A and flow statistics in part B.




**(a)**


**(b)**

**Fig. 11. View port and flow statistics by using the ovs-ofctl command: (a) port statistic, (b) flow statistic**

## 7. Discussion and Comparisons

In recent years, SDN has gained considerable attention and popularity due to its ability to integrate with machine learning and deep learning, which enables SDN to leverage intelligent algorithms and data analytics to optimise network performance and enhance security. For that, an efficient dataset should be gathered to train the learning algorithm. However, no current application or tool can collect all the required datasets in various SDN networks. For that, this study offers the most popular methods and tools to generate various data and collect the required metrics.

Table 1 compares the tools presented in Section 5. As shown in Table 1, the iperf3, Ping and Python scripts can be used to generate traffic in SDN. Scripts written using the Python language are used to generate traffic between hosts and are sometimes used to save the results from other tools and applications in a comma-separated values file (CSV) or json file, whilst iperf3 and Ping are used to generate traffic in SDN. It shows that iperf3 is widely used to generate traffic for many network applications, such as congestion control and load balancing, whilst for DDos attacks, the Ping tool is widely used to generate attacks. Therefore, the selection of the tool depends on what the application demands.

**Table 1,**
**Comparison of tools that generate data in this study**

| Tool | Packet generated type | Work style | Tool type | Application |
|---|---|---|---|---|
| Iperf3 | TCP and UDP segments | Client–server pair | Built-out | Congestion controlling, predicting and avoiding.<br>Traffic load balancing |
| Ping | ICMP Datagrams | Source–destination pair | Built-in and built-out | Detect DDoS attacks. |
| Python scripts | HTTP, DNS, FTP, and SMTP messages | Client–server pair | Built-out | Flow management.<br>Detect DDoS attacks |

Table 2 compares the tools presented in Section 6. As shown in Table 2, the OpenFlow command ‘ovs-ofctl’ is used to monitor and configure switches. It can also view the states of the switches such as table entries, features and configurations to analyse the flow in the switches. Wireshark is used for capturing packets and viewing their details to control forwarding time. Simple_mointor_13 is an RYU application script that is written in Python and is used to send stats and request messages from the controller to the switches to inform about every feature and entry. When the controller receives stats and reply messages, it updates its records and stores the new records in a CSV or json file. Gui-topology is another RYU application that is used to view the topology that was implemented. Gui-topology only views switches and prints their ID and uses ports to link between them. Gui-topology can also view a summary of flow statistics for each switch. Flowmanager is another RYU script that is used to view network topology. It differs from gui-topology by showing hosts and printing their mac or IP address. Whilst iperf and Ping are used to generate traffic in SDN, each one measures different metrics. Iperf measures BW, throughput, the number of retransmitted segments, CWND, jitter and the number of lost datagrams, whilst Ping measures RTT, the number of transmitted and received packets, packet loss and the time of transmission. As a result, the application or the metrics the researcher wishes to measure or examine will determine which tool is best.

**Table 2,**
**Comparison of tools that collected data in this study**

| Tool | Metrics collected | Monitoring work style | Tool type | Application |
|---|---|---|---|---|
| Wireshark | Header and data fields of the protocol | Anywhere in network | Built-out | Reduce forwarding time |
| Simple_mointor_13 | Switch statistics such as flow, port, queue, VLAN, meter table, etc. | Centralised controller | Built-out RYU | Congestion controlling and avoiding. |
| Gui-topology | Switch ID, port number, and flow statistics | Centralised controller | Built-out RYU | Topology discovering. |
| Flowmanager | Switch ID, hosts Mac or IP address, and port numbers. | Centralised controller | Built-out RYU | Topology discovering. |
| Iperf3 | Bandwidth, throughput, number of retransmitted segments, CWND, jitter, and number of lost datagrams | End-to-end hosts | Built-in Mininet | Congestion controlling, predicting and avoiding. Traffic load-balancing |
| Ping | RTT, number of transmitted and received packets, packet loss, and time of transmission. | End-to-end hosts | Built-in Mininet | Detect DDoS attacks. |
| ovs-ofctl command | Switch statistics like ports, flows, queues, and VLAN | Switch ports | Built-in Mininet | Reduce forwarding time |

## 8. Conclusion

A Mininet simulator is used to create SDN topologies and connect to controllers. This tool can integrate with many tools to simulate real networks. One of these tools is iperf3, which is used to generate traffic between two hosts and measure the number of bytes that can be sent, BW, CWND, number of retransmitted segments, jitter and the number of lost datagrams. Another important tool is Ping, a tool that sends ICMP packets between two hosts and measures the RTT, time-to-life, number of segments and the time of transfer. Python scripts can also run on the Mininet and generate traffic between hosts. All three of the previous tools are used to generate traffic, but Mininet also has tools to capture traffic and view information, such as the RYU controller application. Wireshark is one such tool that captures packets and displays their headers and data. Ovs-ofctl is another built-in command that is integrated into Mininet to view statistics about the network, such as ports, queues, etc. Gui-topology and Flowmanager are RYU controller applications used to monitor and view network topology, switches' ids, host mac addresses and port numbers.

This study was conducted to view the most used tools that connect with the SDN environment to generate traffic and collect data. After explaining each tool, comparisons were added to clarify the difference in work style, type and the applications of each tool. In conclusion, each application requires a specific tool to generate and collect data. Therefore, recommending one tool to select when working in an SDN network is not advisable, and the choice depends on the final goal of the work.

## Acknowledgments

This work is supported by the information and communications engineering department, Al-Khawarizmi Engineering College, University of Baghdad.

## References

[1] A. Sha, S. Madhan, S. Neemkar, V. B. C. Varma and L. S. Nair, "Machine Learning Integrated Software Defined Networking Architecture for Congestion Control," In: 2023 Inter-national Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), Ballar, India, pp. 1-5, 2023, doi: 10.1109/ICDCECE57866.2023.10151339 .

[2] A. T. Albu-Salih, S. A. H. Seno, and S. J. Mohammed, "Dynamic Routing Method over Hybrid SDN for Flying Ad Hoc Networks," Baghdad Sci.J, vol. 15, no. 3, 0361, Sep. 2018.

[3] T. E. Ali, A.H. Morad, and M. A. Abdala, "SDN Implementation in Data Center Network," Journal of Communications, vol. 14, no. 3, 223–228, March 2019.

[4] T. Y. Mu, "Toward Self-Reconfigurable Parametric Systems: Reinforcement Learning

Approach," Doctoral thesis, Western Michigan University, Dec. 2019 .

[5] N. S. Soud and N. A. S. Al-Jamali, "Intelligent Congestion Control of 5G Traffic in SDN using Dual-Spike Neural Network," Jcoeng, vol. 29, no. 1, pp. 110–127, Jan. 2023 .

[6] H. Jiang, Q. Li, Y. Jiang, G. Shen, R. Sinnott, C. Tian, and M. Xu, "When machine learning meets congestion control: A survey and comparison," Computer Networks, vol. 192, 2021, 108033, ISSN 1389-1286 .

[7] J. Zhao, M. Tong, H. Qu, and J. Zhao, "An Intelligent Congestion Control Method in Software Defined Networks," 2019 IEEE 11th International Conference on Communication Software and Networks (ICCSN), pp. 51-56, Chongqing, China, 2019, doi: 10.1109/ICCSN.2019.8905364 .

[8] J. Wu, Y. Peng, M. Song, M. Cui and L. Zhang, "Link Congestion Prediction using Machine Learning for Software-Defined-Network Data Plane," 2019 International Conference on Computer, Information and Telecommunication Systems (CITS), pp. 1-5, Beijing, China, 2019, doi: 10.1109/CITS.2019.8862098 .

[9] R. C. Meena, M. Bundele , and M. Nawal, "SDN-CIFE: SDN-Controller with Instant Flow Entries to Improve First Packet Processing Period," Test Engineering and Management, vol. 83, pp. 911-919, March-April 2020, Available at SSRN: https://ssrn.com/abstract=3569521.

[10] T. E. Ali, A.H. Morad, and M. A. Abdala, "Traffic management inside software-defined data centre networking," Bulletin of Electrical Engineering and Informatics, vol. 9, no. 5, pp. 2045-2054, October 2020.

[11] K. B. Nougnanke, "Towards ML-based Management of Software-Defined Networks," Doctoral dissertation, Université Paul Sabatier-Toulouse III, 2021.

[12] G. Diel, C. C. Miers, M. A. Pillon and G. P. Koslovski, "Data classification and reinforcement learning to avoid congestion on SDN-based data centers," GLOBECOM 2022 - 2022 IEEE Global Communications Conference, Rio de Janeiro, Brazil, pp. 2861-2866, 2022, doi: 10.1109/GLOBECOM48099.2022.10000708 .

[13] M.A. Mohsin, and A.H. Hamad, "Performance evaluation of SDN DDoS attack detection and mitigation based random forest and K-nearest neighbors machine learning algorithms," Revue d'Intelligence Artificielle, vol. 36, no. 2, pp. 233-240, 2022, https://doi.org/10.18280/ria.360207

[14] I. M. Ali and M. I. Salman, "SDN-assisted Service Placement for the IoT-based Systems in Multiple Edge Servers Environment," Iraqi Journal of Science, vol. 61, no. 6, pp. 1525–1540, Jun. 2020.

[15] A. Sharma, V. Balasubramanian, and J. Kamruzzaman, "A Novel Dynamic Software-Defined Networking Approach to Neutralize Traffic Burst," Computers, vol. 12, no. 7, p. 131, Jun. 2023, doi: 10.3390/computers12070131 .

[16] A. R. Mohammed, S. A. Mohammed and S. Shirmohammadi, "Machine Learning and Deep Learning Based Traffic Classification and Prediction in Software Defined Networking," 2019 IEEE International Symposium on Measurements & Networking (M&N), pp. 1-6, Catania, Italy, 2019, doi: 10.1109/IWMN.2019.8805044.

[17] T. E. Ali, Y.-W. Chong, and S. Manickam, "Comparison of ML/DL Approaches for Detect-ing DDoS Attacks in SDN," Applied Sciences, vol. 13, no. 5, pp. 3033, Feb. 2023, doi: 10.3390/app13053033 .

[18] O. F. Hussain, B. R. Al-Kaseem, and O. Z. Akif, "Smart Flow Steering Agent for End-to-End Delay Improvement in Software-Defined Networks," Baghdad Sci.J, vol. 18, no. 1, pp. 0163, Mar. 2021, https://doi.org/10.21123/bsj.2021.18.1.0163.

[19] M. I. Salman et al, "A Software Defined Network of Video Surveillance System Based on Enhanced Routing Algorithms," Baghdad Sci.J, vol. 17, no. 1(Suppl.), pp. 0391, Mar. 2020 .

[20] M. H. Khairi, S. H. Ariffin, N. M. Latiff, and K. M. Yusof "Generation and collection of data for normal and conflicting flows in software defined network flow table," Indonesian J. Electr. Eng. Comput. Sci, vol. 22, no. 1, pp. 30., 2021.

[21] http://mininet.org/

[22] https://iperf.fr/iperf-doc.php

[23] https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html

[24] https://www.wireshark.org/

[25] https://github.com/martimy/flowmanager

# الأدوات الحديثة للشبكات المعرفة بالبرمجيات توليد حركة المرور وجمع البيانات

**تبارك ياسين خضير[1]* عمر علي عذاب[2]**

[1،2] *قسم هندسة المعلومات والاتصالات، كلية الهندسة الخوارزمي، جامعة بغداد، بغداد ، العراق*

*البريد الالكتروني: Tabarak.taha1603@kecbu.uobaghdad.edu.iq

**المستخلص**

أثبتت الشبكات المُعرّفة بالبرمجيات (SDN) تفوقها في معالجة مشاكل الشبكات التقليدية، مثل قابلية التوسع والمرونة والأمان. وتعود هذه الميزة إلى فصلها مستوى التحكم عن مستوى البيانات. ورغم أن العديد من الدراسات ركزت على إدارة الشبكات المُعرّفة بالبرمجيات ومراقبتها والتحكم فيها وتحسين جودة الخدمة، إلا أن القليل منها ركز على عرض ما يُستخدم لتوليد حركة المرور وجمع البيانات. كما تفتقر الأدبيات إلى مقارنات بين الأدوات والأساليب المستخدمة في هذا السياق. لذلك، تُقدم هذه الدراسة أحدث الأدوات المستخدمة لمحاكاة وتوليد والحصول على إحصاءات حركة المرور من بيئة الشبكات المُعرّفة بالبرمجيات. بالإضافة إلى ذلك، تُقارن الأساليب المستخدمة في جمع بيانات الشبكات المُعرّفة بالبرمجيات لاستكشاف إمكانيات كل منها، ومن ثم تحديد البيئة المناسبة لكل طريقة. تمت محاكاة منصة اختبار الشبكات المُعرّفة بالبرمجيات باستخدام برنامج Mininet مع طوبولوجيا الشجرة ومفاتيح OpenFlow. تم توصيل وحدة تحكم RYU للتحكم في إعادة التوجيه. تُستخدم الأدوات الشهيرة iperf3 وPing ونصوص Python لإنشاء مجموعات بيانات الشبكة من أجهزة مُحددة في الشبكة. يُستخدم برنامج Wireshark وتطبيق RYU وأمر ovs-ofctl لمراقبة مجموعة البيانات وجمعها. تُظهر النتائج نجاحًا في توليد أنواع متعددة من مقاييس الشبكة لاستخدامها مستقبلًا في تدريب خوارزميات التعلم الآلي أو التعلم العميق. لذلك، يُعدّ iperf3 الأداة الأمثل لتوليد البيانات للتحكم في الازدحام، بينما يُعدّ Ping مفيدًا لتوليد البيانات للكشف عن هجمات رفض الخدمة الموزعة. تُعد تطبيقات RYU أداة المراقبة الأنسب للحصول على جميع تفاصيل الشبكة، مثل طوبولوجيا وخصائص وإحصائيات المكونات. كما تم استكشاف العديد من العقبات والأخطاء وسردها لتجنّبها عند محاولة الباحثين توليد مثل هذه المجموعات من البيانات في جهودهم العلمية القادمة.