MASKAY 16(1), May 2026
ISSN 1390-6712

Recibido (Received): 2025/07/03
Aceptado (Accepted): 2025/09/29

Evaluation of performance and scalability in SDN networks using sFlow

Evaluación del rendimiento y escalabilidad en redes SDN usando sFlow

Sebastian Ruiz, Esthefanny Bonilla, Madelaine Muñoz

***Abstract*—Software-Defined Networking (SDN) separates the data plane from the control plane, enabling centralized and flexible network management. Traditional networks face significant challenges in terms of scalability and flexibility; therefore, implementing efficient traffic monitoring techniques is crucial. Using a protocol like sFlow within an SDN network helps optimize performance and resource management, facilitating the analysis of network scalability and efficiency. In this work, we investigate the integration of the sFlow protocol into an SDN architecture to collect real-time network status data and evaluate its scalability. A custom topology was created using the Mininet simulation environment and the Ryu controller, with the sFlow protocol and sFlow-RT deployed to capture traffic metrics, including packet loss, jitter, throughput, and total traffic. The results enable an analysis of how network scalability is affected as the number of devices increases. As more devices connect, network capacity begins to decline, especially when over thirty devices are connected, leading to higher packet loss and reception failures. This behavior heavily depends on how efficiently the controller manages the network.**

***Index Terms*—SDN network, Ryu controller, sFlow protocol, Real-time monitoring, sFlow-RT.**

***Resumen*—Las redes definidas por software (SDN) permiten separar el plano de datos del plano de control, permitiendo una gestión centralizada y flexible de la red. Las redes tradicionales enfrentan grandes desafíos relacionados con la escalabilidad y la flexibilidad; por lo tanto, es necesario implementar técnicas de monitoreo eficiente del tráfico. Al implementar una herramienta como sFlow en la red SDN, se permite optimizar el rendimiento y la gestión de los recursos, permitiendo analizar la escalabilidad de la red. En este trabajo, se analiza la integración del protocolo sFlow en una arquitectura SDN con el objetivo de recolectar información en tiempo real sobre el estado de la red y analizar la escalabilidad. Se desarrolló una topología con el entorno de simulación Mininet y el controlador Ryu, implementando el protocolo sFlow y sFlow-RT para capturar métricas de tráfico como paquetes descartados, jitter, throughput, tráfico total. Los resultados obtenidos permiten analizar la escalabilidad de la red al aumentar el número de dispositivos. Al incrementar la cantidad de dispositivos, la capacidad de la red empieza a disminuir, especialmente a partir de tener más de treinta dispositivos conectados, lo que conduce a tener más pérdida de paquetes y fallas en la recepción. Este comportamiento depende mucho de la eficiencia en la gestión del controlador.**

***Palabras Claves*—Red SDN, controlador Ryu, protocolo sFlow, monitoreo en tiempo real, sFlow-RT.**

Sebastian Ruiz is with the Engineering, computer science, modeling, electronics and systems engineering department, Università della Calabria, Calabria, Italy (email: rzjmsp97r14z605w@studenti.unical.it)
Esthefanny Bonilla is with the Escuela Superior Politécnica de Chimborazo, Riobamba, Ecuador (e-mail: bonillaesthefanny99@gmail.com)
Madelaine Muñoz is with the Escuela Superior Politécnica de Chimborazo, Riobamba, Ecuador (email: madelaine.munioz@espoch.edu.ec)

I. Introduction

SOFTWARE-Defined Networking (SDN) technology was first presented in Ethane [1], a re- search project carried out in the laboratories of Stanford University. Since then, it has grown in importance worldwide in the field of networking technologies, establishing itself as one of the key solutions on the Internet [2]. Over the years, SDN has reinforced the concept of separation between information handling and forwarding roles (CNC separation). In this model, data layer switches are limited to the task of packet forwarding. At the same time, the control plane centralizes data collection and decision-making through a controller that oversees the entire network. Communication between the two planes occurs through a standard interface, where the controller sends forwarding policies and regulations to subordinate switches, which are responsible for implementing these guidelines. This design enhances operational efficiency in network management and, simultaneously, significantly reduces the costs associated with network administration. Several companies and enterprises have already developed and launched their own SDN solutions to reduce costs and increase network utilization [3].

The rapid advancement of network technologies, combined with the increasing complexity of interactions between society and the digital environment, has significantly expanded the magnitude and impact of the Internet. Against this backdrop, the conventional network architecture and the processing capacity of devices are being subjected to increasingly demanding requirements. In this context, SDN emerges as an innovative solution by decoupling the control function of traditional networks, thereby establishing a logical and centralized control plane. This plane enables the design and management of services and policies in an integrated manner, generating flow tables that are distributed to the network devices. This form of operation significantly increases the efficiency of network resource utilization and contributes to optimizing operating costs.

DOI: 10.24133/maskay.v16i1.4203

Several cloud service providers that have integrated and used SDN in a significant way include Google, Facebook, and Microsoft, as they have publicly highlighted their use. In addition, major telecom operators such as AT&T, Deutsche Telekom,

NTT and Comcast have shared their intentions to adopt SDN-based solutions, especially in their access networks. However, they do so with caution, as most of their projects employ hybrid methods [3].

SDN (Software-Defined Networking) systems [4] utilize basic switches with OpenFlow and sFlow support, along with an advanced software core controller that enables network engineers to monitor and create traffic in new ways easily. These OpenFlow switches connect to traffic analysis solutions such as sFlowRT, sFlowTrend, Ganglia, among others [5]. The network of sFlow-enabled switches sends measurement datagrams to one or more collectors. The sFlow collectors enable an SDN application to gain visibility into the network traffic. sFlow and OpenFlow [6] offer complementary capabilities in software-defined networking environments.

The rest of the document is organized as follows. In section II, the SDN background is described, and SDN, sFlow, and OpenFlow technologies are described in the context of flow monitoring. In Section III, the methodology and the network topology used to implement sFlow and obtain the monitoring metrics are developed. Section IV shows the comparative results and graphs obtained with the RYU controller. Section VI concludes the article.

II. Background Of SDN

A. *Software-Defined Network*

The introduction of SDN in traditional networks provides a new approach to the design, implementation, and management of networks. This technology separates the control plane, which manages traffic control and routing, from the data plane, which is responsible for packet forwarding. This enables networks to be centrally programmed through software, rather than relying on manual configurations [7].

SDN enables network administrators to manage, configure, and optimize networks using software programming, rather than relying on physical hardware devices such as routers and switches. This provides greater control over traffic management, resulting in increased agility, efficiency, and flexibility. SDN uses north-bound and south-bound application programming interfaces (APIs), where the north-bound API facilitates communication between applications and the control layer. In contrast, the south-bound API connects the infrastructure and control layers [8].

In [9], it talks about the northbound interface, which connects network applications to the controller. This interface provides routing information for applications, manages infrastructures and controllers, and provides information about policies governing communications between controllers and applications.

The southbound interface connects the controller to network devices. This interface is essential in that it can be easily and quickly reconfigurable, as it must cope with the dynamism and flexibility of the change in the control plane, must be able to make traffic safe and isolated from other networks to guarantee the required constraints, and must share the characteristics of the physical resources available. The general architecture of the SDN network is shown in Fig. 1.

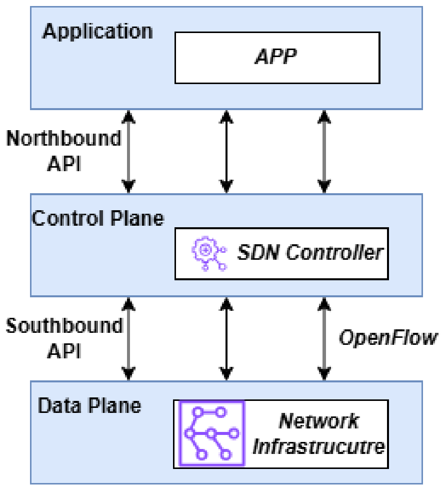


Fig. 1. Architecture of SDN network

B. *OpenFlow*

OpenFlow is a network communication protocol used between controllers and forwarders in the SDN architecture. An OpenFlow controller serves as the brain of the SDN architecture, located in the control layer, and instructs data forwarding through the OpenFlow protocol. The OpenFlow switch is responsible for forwarding in the data plane. It exchanges messages with the controller through a secure channel to receive forwarding instructions and report its status [10].

OpenFlow utilizes the concept of a flow table, where forwarders transmit data packets, and controllers implement flow tables within the forwarders through OpenFlow interfaces, thereby achieving control over the forwarding plane. To forward packets, switches maintain flow tables consisting of flow entries. Each flow entry includes match fields such as the incoming interface port, source and destination ports, IP addresses, MAC addresses, and network protocol type.

When a switch receives a data packet from a host, it performs a lookup in its flow table using the packet's header characteristics [1]. If the packet matches a flow rule, the table counter increases, and the specified actions are applied. If no matching entry is found, the packet flow is forwarded via the OpenFlow protocol to the controller, which decides what action to take and returns the data path. The controller can either drop or forward the packets. Fig. 2 shows the packet processing flow in an OpenFlow switch, where the Packet-In and Packet-Out messages are special control messages that enable interaction between the OpenFlow switch.

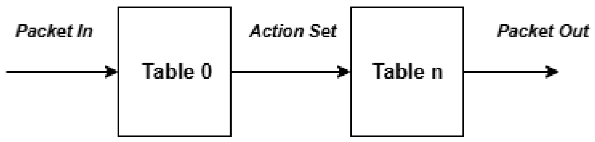


Fig. 2. Flows and tables OpenFlow

C. Sampled Flow (sFlow)

SFlow is a network traffic monitoring protocol that uses statistical sampling to capture and analyze data packets in real-time. sFlow monitoring technology collects network packet samples and sends them in a UDP datagram to a monitoring station, known as a collector, via port 6343 [11].

The header of an sFlow packet consists of four fields, as shown in Fig. 3. The Ethernet Header, which contains the source and destination MAC addresses, as well as frame control information. IP Header, which contains the source and destination IP addresses, as well as routing-related information. UDP Header, which contains information such as source and destination ports, and the sFlow Datagram Header, which contains the sample data.

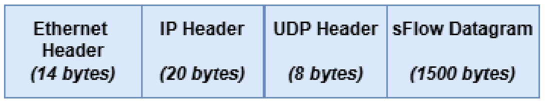


Fig. 3. sFlow packet header

The topology of sFlow consists of two main components: a sFlow agent, which is integrated into devices and enables packet capture and metrics collection from network devices, and then sends them to the sFlow collector. The collector is a server that receives and analyzes the data sent by the sFlow agent [12]; it can be software such as sFlow-RT.

sFlow-RT is an open-source product with an integrated OpenFlow controller, enabling the monitoring of sFlow agents embedded in network devices and converting the collected measurements into actionable metrics accessible via a RESTful Flow API [13].

In a sFlow system, the sFlow agent collects traffic statistics from an interface by sampling packets and encapsulating the statistics into sFlow packets. When the sFlow packets expire, the sFlow agent sends the sFlow packets to the sFlow collector. The sFlow collector analyzes the sFlow packets and displays the analysis results [14]. The expiration period is 1 second.

III. Methodology

A. Network topology

For implementing the topology, use Mininet version 2.3.1b4, which supports both Python 3 and Python 2. The switches used are Open vSwitch version 2.13.8 (ovs-vsctl), which is included by default in Mininet. These switches are configured with the sFlow protocol version 6.04, which enables the collection and transmission of network monitoring metrics using the sFlow-RT tool version 3.1.1711.

The SDN network is managed by a Ryu version 4.34 controller, which is responsible for creating the flow tables and allowing communication between hosts.

The topology employs a tree-like structure with three levels: the first level comprises the root switch, the second level features two child switches, and the third level consists of four switches. The network comprises a total of seven connected switches, with two hosts connected to each switch on the third level and one host connected to each switch on the second level, resulting in a total of eight hosts. The available computing resources constrain the bandwidth of the links connecting the switches.

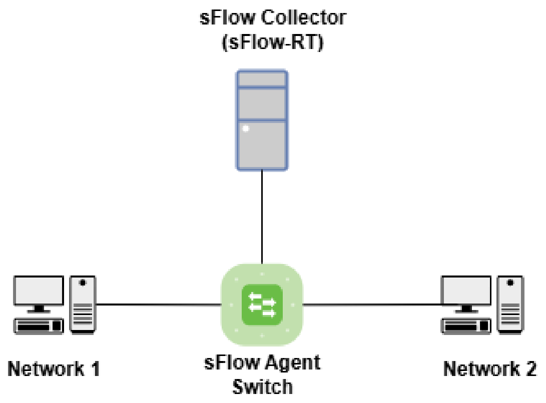


Fig. 4. sFlow topology

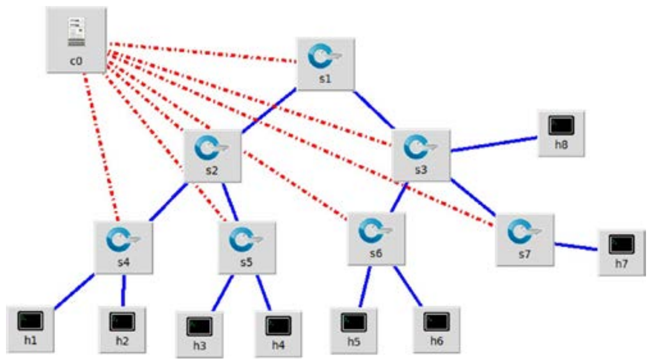


Fig. 5. Network topology

To create the topology, the following parameters were configured. Static IP addresses were assigned to the hosts, while the MAC addresses were set by default by Mininet. Furthermore, the bandwidth of the links connecting the switches was defined. Regarding the controller, the port, controller type, and the IP address for listening and communication were configured. For the sFlow Agent, the collector address to which the data is sent for processing was specified, as well as the Sampling, Polling, and Header parameters. These parameters are summarized in Table I.

TABLA I
Main Network Parameters

Configuration	Parameter
Number host	8 (h1-h8)
Number Switch	7 (s1-s7)
IP Address Host	10.0.0.0/8
MAC Address Host	Default
Links (Bandwidth)	Delimited by the device Type Controller
IP Address Controller	127.0.0.1
Port Controller	6653

sFlow Collector IP	127.0.0.1
sFlow Sampling	10
sFlow Polling	20 seconds
Number Switch	7 (s1-s7)
sFlow Header	128 bytes

To analyze the scalability of the network, it is necessary to increase the number of switches, which modifies the base topology according to the requirements of the devices under evaluation. To achieve this, Mininet commands were used, enabling the creation of various topologies based on the number of devices.

Using the command mn –topo tree, depth=3, fanout=2 –switch ovsk –controller remote, it is possible to create tree topologies, where depth specifies the number of levels in the tree, fanout defines the number of child switches per parent switch, switch ovsk defines the switch type (Open vSwitch), and controller remote specifies the controller type. Meanwhile, with the command mn –topo linear,n –switch ovsk –controller remote, linear topologies can be created, where n is the number of devices.

With these two command lines, different topologies are generated for evaluation, considering networks with 10 to 100 switches to assess their performance. The configured parameters according to the number of switches are shown in Table II.

TABLA II
PARAMETERS FOR THE CREATION OF THE TOPOLOGIES

Number Switch	Configuration
10	n=10
20	depth=2,fanout=4
30	depth=2,fanout=5
40	n=40
50	n=50
60	n=60
70	depth=2,fanout=8
80	n=80
90	depth=2,fanout=9
100	n=100

B. *Performance metrics*

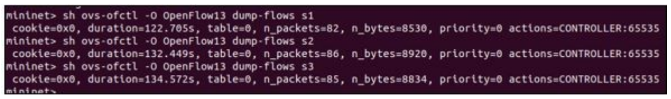
The implementation of sFlow allows the collection and analysis of various metrics in the SDN network. The primary metrics evaluated in this work are jitter, throughput, dropped packets, and sFlow packets.

- Throughput refers to the amount of data successfully transmitted within the SDN network.
- Jitter is the delay experienced by packets in reaching their destination.
- Dropped packets refer to the number of packets that are discarded before being delivered.
- sFlow packets represent the number of sampled packets within the network.

To obtain these metrics, measurements were performed using the iperf command, which allows evaluating network performance by generating TCP or UDP traffic. To generate traffic, two sessions must be created: one with the host acting as the client and another with the host acting as the server. In our main scenario, shown in Fig. 5, we generated TCP and UDP traffic for 30 seconds to obtain the metrics above. For measuring overall network performance, traffic was generated for 60 seconds.

C. *RYU Controller*

To implement the RYU controller, a flow rule was configured to forward all packets to the controller. When a packet arrives at the controller, the source MAC address and the incoming port are inspected to determine how to process it within the controller and to begin building the MAC address tables. If the destination address is registered in the MAC address table, the packet is forwarded to the corresponding port; otherwise, flooding is performed, sending the packet to all ports except the one it arrived on. Once the destination is known, the controller installs flow rules in the switch so that subsequent packets with similar characteristics are handled directly by the switch, rather than being sent to the controller. In Fig. 6, the flow tables created by the RYU controller in each switch are shown.

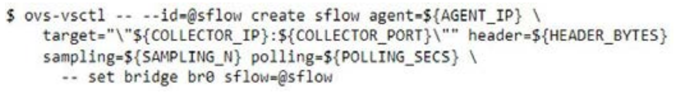


```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=122.705s, table=0, n_packets=82, n_bytes=8530, priority=0 actions=CONTROLLER:65535
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s2
 cookie=0x0, duration=132.449s, table=0, n_packets=86, n_bytes=8920, priority=0 actions=CONTROLLER:65535
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s3
 cookie=0x0, duration=134.572s, table=0, n_packets=85, n_bytes=8834, priority=0 actions=CONTROLLER:65535
mininet>
```

Fig. 6. Flow tables

D. *sFlow*

By default, sFlow is not enabled on devices. To activate this protocol on the switches, the parameters specified in Table I must be configured. These parameters are: Agent, which defines the IP address of the device generating the flow data; Target, which specifies the IP address and port of the server that receives and analyzes the traffic data; Sampling, which sets the rate at which packets are sampled; Polling, which determines the interval in seconds for sending traffic statistics to the collector; and Header, which defines how many bytes of the packet header are included in each sample. The command line to configure sFlow is shown in Fig. 7.



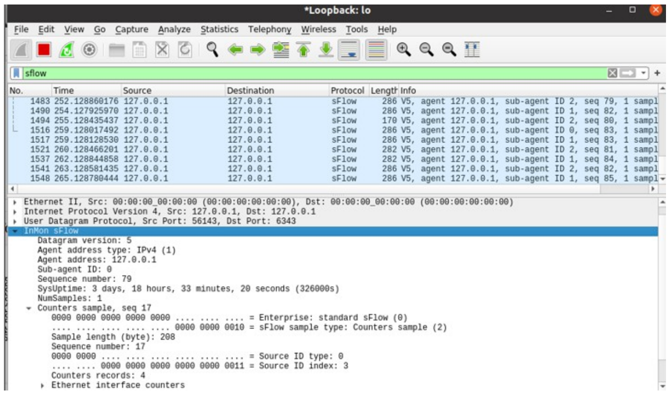
```
$ ovs-vsctl -- --id=@sflow create sflow agent=${AGENT_IP} \
    target="\"${COLLECTOR_IP}:${COLLECTOR_PORT}\"" header=${HEADER_BYTES}
    sampling=${SAMPLING_N} polling=${POLLING_SECS} \
      -- set bridge br0 sflow=@sflow
```

Fig. 7. sFlow command line

To verify that the sFlow protocol is correctly established, we can use the Wireshark software, which allows us to capture and examine network traffic in real-time. By filtering the data, we can observe that the packets passing through the network contain a new header corresponding to the sFlow protocol, as shown in Fig. 8.

The configuration of these parameters enables the collection of metrics sent by the switches. These metrics are sent from the sFlow agent to the collector, which processes the data and allows monitoring through tools such as sFlow-RT. By default, sFlow uses port 6343.

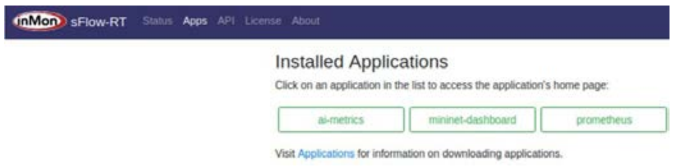
sFlow-RT includes a REST API that allows us to query and extract specific sFlow metrics. This tool also provides an embedded dashboard with the main metrics, as shown in Fig. 9. To access this dashboard, the sFlow-RT tool must be executed and then accessed through the browser using the IP address configured in the collector, in this case, 127.0.0.1.


*Loopback: lo
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
sflow

```
No.   Time          Source     Destination  Protocol  Length Info
 1483 252.128860176 127.0.0.1  127.0.0.1    sFlow     286 V5, agent 127.0.0.1, sub-agent ID 2, seq 79, 1 sampl
 1490 254.127925970 127.0.0.1  127.0.0.1    sFlow     286 V5, agent 127.0.0.1, sub-agent ID 1, seq 82, 1 sampl
 1494 255.128435437 127.0.0.1  127.0.0.1    sFlow     170 V5, agent 127.0.0.1, sub-agent ID 2, seq 80, 1 sampl
 1516 259.128017492 127.0.0.1  127.0.0.1    sFlow     286 V5, agent 127.0.0.1, sub-agent ID 0, seq 83, 1 sampl
 1517 259.128128530 127.0.0.1  127.0.0.1    sFlow     286 V5, agent 127.0.0.1, sub-agent ID 1, seq 83, 1 sampl
 1521 260.128466201 127.0.0.1  127.0.0.1    sFlow     282 V5, agent 127.0.0.1, sub-agent ID 2, seq 81, 1 sampl
 1537 262.128844858 127.0.0.1  127.0.0.1    sFlow     282 V5, agent 127.0.0.1, sub-agent ID 1, seq 84, 1 sampl
 1541 263.128581435 127.0.0.1  127.0.0.1    sFlow     286 V5, agent 127.0.0.1, sub-agent ID 2, seq 82, 1 sampl
 1548 265.128780444 127.0.0.1  127.0.0.1    sFlow     286 V5, agent 127.0.0.1, sub-agent ID 1, seq 85, 1 sampl

▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ User Datagram Protocol, Src Port: 56143, Dst Port: 6343
▾ InMon sFlow
    Datagram version: 5
    Agent address type: IPv4 (1)
    Agent address: 127.0.0.1
    Sub-agent ID: 0
    Sequence number: 79
    SysUptime: 3 days, 18 hours, 33 minutes, 20 seconds (326000s)
    NumSamples: 1
  ▾ Counters sample, seq 17
      0000 0000 0000 0000 0000 .... .... .... = Enterprise: standard sFlow (0)
      .... .... .... .... .... 0000 0000 0010 = sFlow sample type: Counters sample (2)
      Sample length (byte): 208
      Sequence number: 17
      0000 0000 .... .... .... .... .... .... = Source ID type: 0
      .... .... 0000 0000 0000 0000 0000 0011 = Source ID index: 3
      Counters records: 4
    ▸ Ethernet interface counters
```

Fig. 8. sFlow packets



Installed Applications

Click on an application in the list to access the application's home page:

Visit Applications for information on downloading applications.

Fig. 9. sFlow-RT dashboard

IV. Results

To analyze the performance and scalability of SDN networks, two scenarios will be conducted. The first will verify the operation of the sFlow protocol using our network with a tree topology composed of seven switches, and the second will involve generating different types of networks with Mininet commands to vary the number of switches and assess scalability.

To analyze the operation of the sFlow protocol and obtain key metrics, such as jitter, throughput, and dropped packets, we first examined our main tree-type topology. To generate UDP and TCP traffic, the iperf command was used, creating traffic for 30 seconds. The generated traffic was transmitted through four hosts connected to different child nodes at the third level.

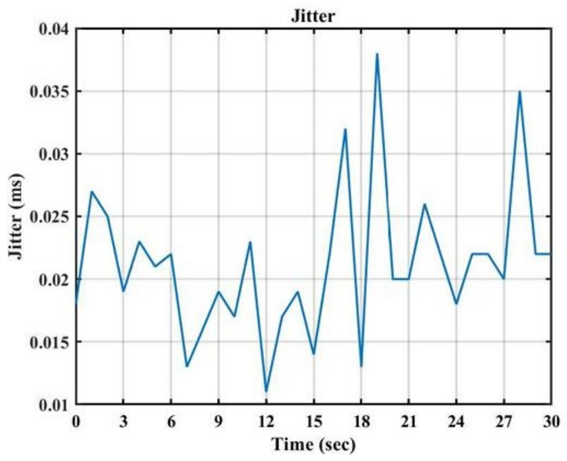


Fig. 10. Jitter

Fig. 10 shows the jitter measured in the SDN network. To measure this parameter, UDP traffic was generated between the network hosts, resulting in different delay values. These values range from 0.03 ms to 0.038 ms.

This parameter analyzed depends on the flow tables created by the controller. If the switches do not have the flow rules for the packets in their tables, the packets are sent to the RYU controller (Packet-in) for processing, and the flow rule is installed and sent back to the switches (Packet-out). This round-trip processing between the controller and the switches increases the delay. If the controller takes a long time to decide what to do with the packets, it also increases the packet delay. If the flow is already specified in the switch tables, these packets are not sent to the controller and are forwarded directly, thus reducing the delay.

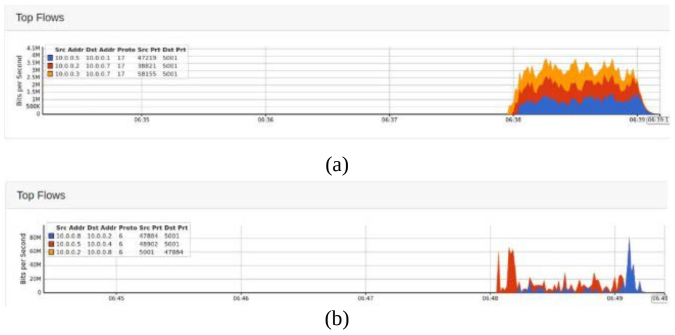


Fig. 11. Throughput;(a) UDP Traffic; (b) TCP Traffic

Fig. 11 shows the network throughput generated by the four hosts. The first image illustrates the throughput during UDP traffic generation, while the second corresponds to the throughput using TCP traffic. These data are obtained from the sFlow metrics, which are sent to the collector, processed, and visualized through the sFlow-RT dashboard, as shown in the figure. In addition to reporting the throughput values, the figure provides information about the source and destination IP addresses generating the traffic, as well as the ports in use.

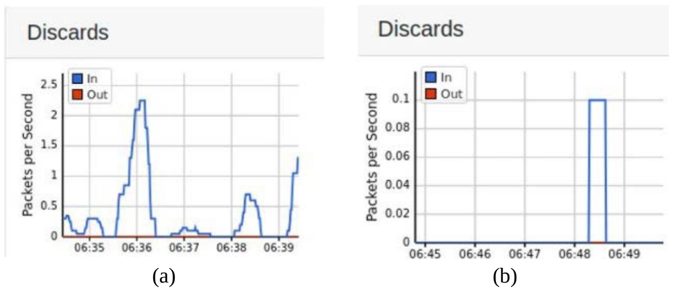


Fig. 12. Discard Packet;(a) UDP Packets; (b) TCP Packets

sFlow-RT, having several built-in dashboards, also allows visualizing the packets discarded per second when transmitting UDP or TCP traffic, as shown in Fig. 12. These discarded packets may be caused by congestion in the switches, resulting in packets being lost and not processed correctly, or by improperly configured flow rules in the RYU controller, leading to packet loss. As shown in the first figure, the discarded packets under UDP traffic appear randomly, while for TCP traffic, the packet loss is constant. All these values are captured by the sFlow datagrams and sent to the controller for analysis and processing.

Fig. 13 shows the total amount of traffic flowing through the switch interfaces during the 30-second traffic generation. These measurements are obtained from sFlow datagrams, which are sent to the collector for visualization and analysis. This traffic is represented in bits per second. In the first figure, we can observe the total traffic generated by UDP packets, which remains constant over the 30 seconds of traffic generation. In contrast, the second image shows TCP traffic, which is more random compared to UDP.

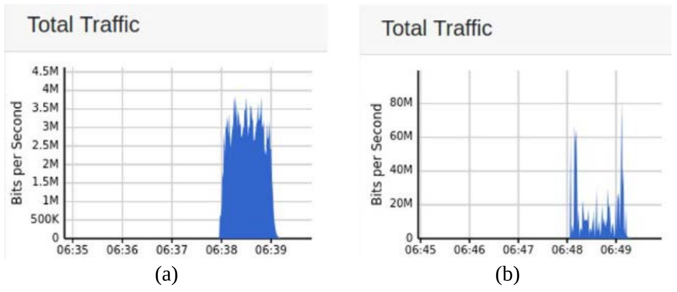


Fig. 13. Total Traffic;(a) UDP Traffic; (b) TCP Traffic

To analyze scalability, different topologies are implemented considering the number of switches to be evaluated. For this purpose, the configurations defined in Table II are utilized. As in the first scenario, jitter, throughput, discarded packets, and sFlow packets will be assessed. To obtain these parameters, tests were conducted by generating TCP and UDP traffic for 60 seconds.

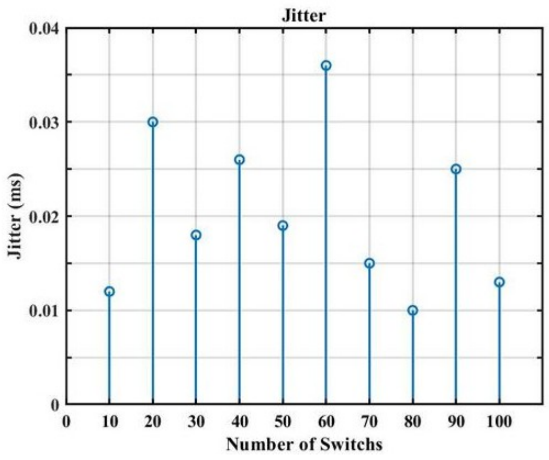


Fig. 14. Jitter with respect to the number of devices

Fig. 14 shows the analysis of jitter as the number of devices increases. As observed, jitter increases significantly when there are 20 and 60 connected devices. This is because, in these cases, the implemented topologies produce more delay between the links, causing the jitter value to rise, while for the other configurations, the increase is moderate.

Fig. 15 illustrates the number of discarded packets as the network size increases. As observed, the more devices there are in the network, the more flow rules the controller has to configure, leading to more discarded packets and increased traffic congestion in the switches. In this case, an exponential behavior is observed as the number of devices increases.

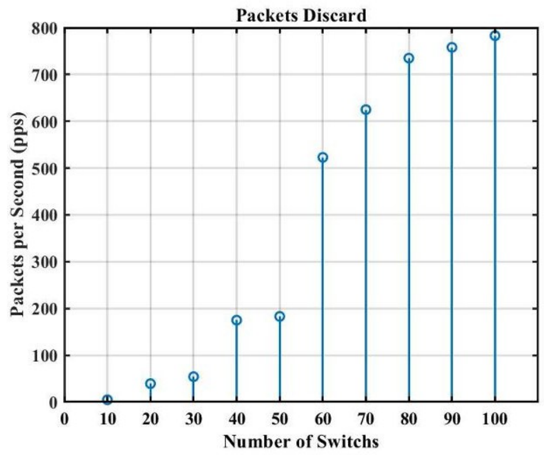


Fig. 15. Packets respect the number of devices

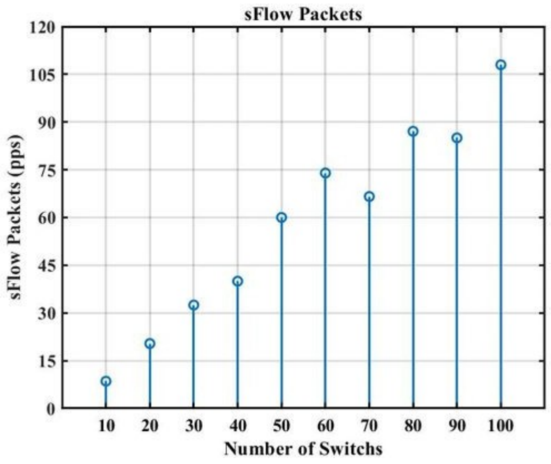


Fig. 16. sFlow Packets respect the number of devices

sFlow packets are sent by the switches and contain all the monitoring information collected by the sFlow protocol. These packets are sent every second and are used to analyze network performance or detect failures. Fig. 16 shows that as the number of switches increases, the number of sFlow packets generated also increases, with minor nonlinear variations, such as in the cases of having 70 and 90 devices. The number of packets generated will depend on the number of devices and the type of topology used.

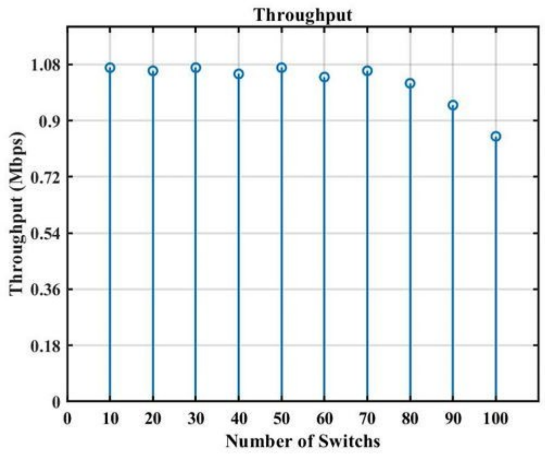


Fig. 17. Throughput with respect to the number of devices

Fig.17 shows the amount of valuable data that the network can transmit per unit of time. As the number of devices increases, this parameter decreases because the controller receives more Packet-In requests to process and needs more time to respond to each flow. This consumes additional bandwidth that could otherwise be used for valuable data, slowing down the processing between the switch and the controller and resulting in a reduction of throughput.

V. Discussion

The contribution of this work lies in demonstrating how monitoring protocols can be implemented in SDN networks to monitor and analyze the variation of parameters, such as throughput, jitter, and discarded packets, with increasing numbers of devices. This provides valuable information for sizing controllers and planning scalable network topologies.

During the tests, it was observed that throughput begins to decrease once 30 devices are connected. This is due to the controller's saturation when creating tables and processing packets; as the number of devices increases, the controller receives more flows and may become overloaded or even fail. Consequently, if the controller becomes saturated, it will start discarding packets. When analyzing the discarded packet metric, we can observe that starting from 60 devices, the controller begins to experience failures and does not properly send out packets, which may lead to connectivity loss between devices. If we analyze the sFlow packets, we can see that the more devices are connected, the more information is sent to the controller. It can be determined that from 80 devices onward, the controller receives a larger number of packets. This metric allows us to monitor the controller's performance and anticipate packet drops primarily.

The jitter metric depends on the topologies implemented, since as the number of devices increases, packets experience greater delays traveling from their source to their destination, resulting in higher latency between links, as observed with 20 and 60 devices. These results confirm that as the number of devices increases, the Ryu controller may encounter failures in table creation and packet forwarding.

Upon analyzing the results obtained from previous research, it is evident that using a single controller is not an efficient approach; therefore, relying solely on a single controller in an extensive network can lead to packet loss. Hence, it is necessary to employ multiple controllers to improve performance and achieve load balancing. This enables greater scalability in the network and improved packet management, resulting in a reduced number of discarded packets and enhanced throughput [15].

VI. Conclusion

This work analyzes the performance and scalability of Software-Defined Networks (SDN) using the sFlow monitoring protocol. The primary metrics obtained were throughput, discarded packets, jitter, and total network traffic. The results show that as the network scales, the number of discarded packets and sFlow messages increases, leading to congestion and packet loss, while the overall network capacity decreases. These behaviors are strongly influenced by the efficiency of the controller and the implemented topology. However, this study is limited to a single centralized controller (Ryu) and a specific telemetry protocol. As future work, it is suggested that multi-controller architectures be evaluated to enable better load balancing and traffic management among controllers. Additionally, it is recommended to adopt other telemetry protocols, such as INT, which allows for obtaining more complex and realistic metrics

References

[1] A. M. D. Tello and M. Abolhasan, "Sdn controllers scalability and performance study," in the 13th International Conference on Signal Processing and Communication Systems (ICSPCS), 2019, pp. 1–10.

[2] M. Wang, Y. Lu, and J. Qin, "Source-based defense against ddos attacks in sdn based on sflow and som," *IEEE Access*, vol. 10, pp. 2097–2116, 2022.

[3] O. V. Peterson, Cascone and Davie, "Software-defined networks: A systems approach: Use cases," https://sdn.systemsapproach.org/uses.html, 2022, accessed: 2025-07-02.

[4] O. D. Adeniji, "Scalable flow based management scheme in software define network (sdn) using sflow," WSEAS Transactions on Computers, vol. 22, pp. 64–69, 2023.

[5] InMon Corp., "sflow-rt: Real-time network analytics engine," https://sflow-rt.com/, 2025, accedido el 02 julio 2025.

[6] A. Ram, M. P. Dutta, and S. K. Chakraborty, "A flow-based performance evaluation on ryu sdn controller," *Journal of The Institution of Engineers (India): Series B*, vol. 105, pp. 203–215, 2024.

[7] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737

[8] M. A. Diouf, S. Ouya, J. Klein, and T. F. Bissyande´, "Software security in software-defined networking: A systematic literature review," arXiv preprint arXiv:2502.13828, 2025.

[9] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, p. 102563, 2020.

[10] U. I. Sunday and S. D. Akhibi, "Application of software-defined networking," *European Journal of Computer Science and Information Technology*, vol. 10, no. 2, pp. 27–48, 2022.

[11] X. Liu, X. Zhang, Z. Li, Z. Zhang, and Z. Zhang, "Towards sflow and adaptive polling sampling for deep learning-based intrusion detection," *Future Generation Computer Systems*, vol. 101, pp. 1–9, 2019.

[12] S. Hublikar, V. Eligar, and A. Kakhandki, "Detecting denial-of-service attacks using sflow," in Inventive Communication and Computational Technologies, G. Ranganathan, J. Chen, and A. Rocha, Eds. Singapore: Springer Singapore, 2020, pp. 483–491.

[13] sFlow RT, "sflow-rt real-time analytics," 2025, accedido: 27 de abril de 2025. [Online]. Available: https://sflow-rt.com

[14] M. Ilham and N. R. Rosyid, "Pengembangan aplikasi pemantauan jaringan berbasis web pada software-defined networking dengan protokol sflow," *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 8, no. 6, pp. 1117–1126, 2021.

[15] M. M. Elmoslemany, A. S. Tag Eldien, and M. M. Selim, "Performance analysis in software defined network (sdn) multi-controllers," *Delta University Scientific Journal*, vol. 6, no. 1, pp. 181–192, 2023.