FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 9 No. 3, March, 2025, pp 154 - 163
DOI: https://doi.org/10.33003/fjs-2025-0903-3215

IMPLEMENTATION AND COMPARISON OF SOFTWARE-DEFINED NETWORK CONTROLLERS IN VARIOUS SIMULATED NETWORK ENVIRONMENTS

***[1]Osuolale Abdramon Tiamiyu, [1]Samuel Olusayo Onidare, [1]Hakeem Babalola Akande, [2]Oluwaseun Tolani Ajayi and [1]Abdraheem Ojonugwa Ogbotobo**

[1]Department of Telecommunication Science, University of Ilorin
[2]Department of Electrical and Computer Engineering, Illinois Institute of Technology

*Corresponding authors' email: tiamiyu.oa@unilorin.edu.ng; ozutiams@yahoo.com

ABSTRACT

In today's ever-evolving networking landscape, Software Defined Networking (SDN) has emerged as a paradigm-shifting technology that promises greater flexibility, agility, and control over network infrastructures. However, how easy are the configuration, extensibility, and programmability of these SDN controllers, considering the practical implications for network administrators and developers? Despite the growing adoption of SDN, there is limited research on the comparative performance of controllers across multiple simulation environments. This study aims to explore the practical implementation and comparative evaluation of different SDN Controllers within diverse simulated network environments. Prominent controllers such as POX and Faucet are meticulously configured and deployed in simulated network environments created using GNS3 (Graphical Network Simulator-3), OPNET (Optimized Network Engineering Tools), NS3 (Network Simulator-3), OMNET++ (Objective Modular Network Testbed in C++) and MININET platforms. Furthermore. the study employs a range of performance metrics like controller latency, network throughput, packet loss, CPU (Central Processing Unit) and memory utilization to assess the efficacy and efficiency of each SDN Controller. The Results indicated that Mininet provided the lowest latency, whereas OPNET demonstrated better scalability for large-scale networks. NS3, though useful for SDN network design and visualization, exhibited higher CPU and memory utilization that might limit its scalability for large-scale SDN controller simulations. While GNS3 offered a balanced performance and resource utilization, making it a suitable choice for SDN controller simulation that prioritizes realistic network modelling, OMNET++, on the other hand, exhibited moderate performance metrics with efficient resource utilization, making it suitable for SDN controller simulations requiring a balance between performance and resource efficiency.

Keywords: SDN Controller, Network Simulator, POX, FAUCET, Performance Metrics, OpenFlow

INTRODUCTION

Software-Defined Networking (SDN) is transforming network architecture by separating the control plane from the data plane, offering centralized management and dynamic network resource control (Adekunle & Oluwaseyi, 2023). The core component of SDN is the SDN controller, which acts as the brain of the network, providing network visibility, managing traffic, and implementing policies.

In global telecommunications networks, SDN, technologically, is a game-changer. Its adoption has emerged as a probable paradigm change in the perpetually changing landscape of modern network architecture. SDN allows for centralized network control that eventually improves the flexibility, scalability, and manageability of networks. The SDN controller optimizes traffic flow and resource allocation, acting as the network's central management unit. Its selection is an important SDN component (Franco-Almazan et al (2019)). SDN gives flexibility, in addition to scalability and programmability, to the existing networks. Separation of the control plane and data plane of SDN allows central management of the network resources and automation of the network configuration, provisioning, and orchestration by network administrators.

Traditional networking uses dedicated network devices, like routers and switches, to manage network traffic. It has distributed architecture and it is hardware-based, unlike SDN that is software-based. Its topologies are based on the intimate connection of network devices and their management software. These drawbacks of the traditional networking topologies are addressed by the emergence of SDN as a feasible method. In typical networks, network components (e.g., switches, routers, and firewalls) handle data forwarding and execute network policies. The control software that configures and regulates these devices is widely distributed. Thus, automating network administration tasks and enhancing network performance becomes difficult. Having a centralized controller in charge of managing the network architecture, SDN overcomes these limitations. A standard protocol like OpenFlow is been used by the controller to communicate with the network devices, thus the network could be designed and configured in a more flexible and efficient manner.

In the early 2000s, the separation of the control and data planes in networking devices was suggested by academics and this brought about the concept of SDN (Rego et al (2018)). However, the first real-world use of SDN by researchers came to light only in 2008 (McKeown et al (2008)). The OpenFlow protocol was suggested as a standard for communication between the controller and the network devices to allow the controller to manage the network infrastructure centrally.

Any program code that is available for use/modification by public or other developers is tagged open source. The introduction of several open-source SDN controllers like OpenDaylight, Floodlight, and Ryu (for customizing, automating and monitoring networks of various size and scale), give the development of SDN a further pace. Developers can create original network applications and manage network services using the framework being offered by these controllers.

SDN is revolutionizing the way networks are designed, deployed, and managed. By separating the control plane from the data plane, SDN allows network administrators to

implement network policies and configuration changes in a centralized and automated manner, thereby reducing the risk of human error and improving network agility. Furthermore, SDN allows network administrators to manage network resources and optimise network performance easily by providing a unified view of the network infrastructure. SDN controllers help network administrators to identify and resolve network bottlenecks and proactively monitor network performance. Nevertheless, SDN has its own drawbacks and issues that need to be resolved. One of the challenges facing SDN is the need for specific knowledge and experience to implement and operate the technology. To use SDN, thorough knowledge and understanding of network protocols, programming languages, and software-defined networking topologies are necessary. Also, the possible security issues linked to the centralized administration of the network infrastructure pose another challenge for SDN (Imran et al (2021)). SDN controllers and switches may target of cyber-attacks, thereby jeopardizing the network's security and integrity. When it comes to handling huge networks with thousands of devices, SDN has scaling issues. As the number of network devices rises, the controller could compromise the stability and performance of the network and, thus, become a bottleneck.

Despite challenges like Interoperability, Security, and Scalability that are being faced by SDN, SDN continues to evolve and gain adoption due to its benefits in automation, flexibility, and efficiency. Its applications continue to expand in areas like AI-driven networking, autonomous systems, and next-generation telecommunications (e.g. Data Centers and Cloud Computing, Enterprise Networks, Internet of Things (IoT) Networks, Cybersecurity and Network Monitoring). SDN has evolved from a research concept to a fundamental networking technology, and with AI, cloud-native applications, and 5G/6G, SDN continue to redefine the future of networking.

In simulated network environments, a lot of studies have been conducted on the performance of different SDN controllers. One of such studies is a study by (Singh et al (2022)). In their work, the performance of the OpenDaylight and ONOS (Open Network Operating System) controller was evaluated using the Mininet tool. They found out that both controllers had similar performance in terms of latency and throughput, however, in terms of reliability, the ONOS is slightly better. In a simulated network environment also, authors in (Chouhan et al (2019)) compared the performance of the Floodlight and Ryu controllers. In their study, it was found that Ryu had a higher throughput and lower latency than Floodlight, which had better reliability.

The studies conducted by (Salman et al (2016)) focused on the functionality of different SDN controllers. The ability of different controllers to support different networking protocols and integrate them with other tools was evaluated by the authors. It was concluded that ONOS (Open Network Operating System) and OpenDaylight had extensive support for different protocols and tools, while Ryu and Floodlight had limited support.

Authors in (Abuarqoub (2020)) studied the ease of use of different controllers, including their installation, configuration, and management. They concluded that ONOS and OpenDaylight had the most straightforward installation and configuration processes, while Floodlight and Ryu had more complex processes. (Tello & Abolhasan, 2019) in their study evaluated the scalability of different controllers, and their ability to handle increasing numbers of network devices and traffic. It was found that ONOS and OpenDaylight had the best scalability, while Floodlight and Ryu had limited scalability.

Despite extensive research on SDN controllers, existing studies lack a direct comparison of controllers across multiple simulated environments. This study bridges this gap by evaluating POX and Faucet in diverse network conditionsWith reference to the aforementioned, it was deduced that the existing literature on the implementation and comparison of different SDN controllers in a simulated network environment provides valuable insights into the performance, functionality, ease of use, and scalability of these controllers. However, there is still a need for further research in this area, particularly in terms of comparing the performance of different controllers under different scenarios and conditions. In a nutshell, despite extensive research on SDN controllers, existing studies lack a direct comparison of controllers across multiple simulated environments. This study bridges this gap by evaluating POX and Faucet in diverse network conditions.

MATERIALS AND METHODS

Implementation and comparison of SDN controllers in various simulated network environments is crucial for understanding how different SDN controllers perform under various conditions. Simulated environments provide a controlled and cost-effective way to evaluate SDN controllers before deploying them in real-world scenarios. This study encompasses multiple factors such as network topology, scale, performance metrics, and the specific SDN controller features that align with the needs of a given environment. The structured approach to implementing and comparing SDN controllers would focus on key aspects like deployment, performance, and scalability in simulated environments.

The rationale for selecting the POX and FAUCET controllers for this study is a decision rooted in their relevance to SDN, distinct characteristics and the goals of this study. Selecting these controllers reflects their strengths, diversity of features, and their significance to SDN research and implementation. Consideration for choosing the POX and Faucet controllers is that POX is easy to use while FAUCET is not only difficult to learn but also to use. FAUCET is complex compared to POX which is normally deployed for small networks and for rapid prototyping of new SDN applications; very unlike FAUCET which is preferred for large networks or for deploying complex SDN applications (Bosi et al (2024)).

Simulation Environment

Comparison of SDN controllers, POX and FAUCET, and their implementations necessitate a rigorous methodology that encompasses a variety of simulation environments. These environments serve as the experimental platforms for analyzing the controllers' performance, scalability, and behaviour across diverse network scenarios. In this study, five simulation environments are employed; and they are GNS3, NS3, OPNET, OMNeT++, and Mininet.

GNS3

GNS3 is a dynamic network simulation tool that allows the building of complicated network topologies by combining real and virtual networking devices. GNS3 with its user-friendly graphical interface is a versatile simulation software for simulating networks that are made up of switches, routers and firewalls. This environment excels in simulating real-world network experiences by allowing the use of actual networking hardware alongside virtualized network components (Gil et al (2014)).

NS3
NS3 is a sophisticated network simulation software for analyzing network protocols, algorithms, and applications. It simulates complicated networking behaviours using a discrete-event simulation paradigm, with a focus on wired, wireless, and mobile networks. Its flexibility and ability to mimic large-scale networks make it a useful tool for understanding complex network dynamics (Tiamiyu, 2012).

OPNET
OPNET, known for its accuracy and ability to precisely model real-world networks, enables academics and engineers to investigate network performance, traffic patterns, and protocol behaviour, thus, providing a comprehensive framework for communication network modelling and analysis. (Tiamiyu, 2012). This environment excels at simulating diverse network technologies, from wired to wireless, and can especially be useful for evaluating the real-world impact of SDN controllers.

OMNeT++
OMNeT++ is a simulation framework that is flexible and extendable for simulating discrete event-based systems such as communication networks. OMNeT++ has a modular architecture that allows for the creation of complex network scenarios and the integration of additional models. OMNeT++ is widely used to study the performance of network protocols, applications, and distributed systems, making it a suitable platform for evaluating SDN controller behaviour in various contexts (Tiamiyu, 2012).

Mininet
Being an open-source network emulator built for SDN application quick prototyping and development, Mininet focuses on replicating a network of hosts, switches, and controllers on a single physical system. It is especially well-suited for testing and experimenting with the features and performance of SDN controllers. Mininet is a popular choice for SDN research and instruction because of its capacity to quickly design configurable network topologies. (Gupta et al (2022)).
These simulation tools collectively offer a diverse range of features, enabling comprehensive analysis and comparison of the SDN controllers. They allow a comprehensive evaluation of the controllers' behaviour, efficiency, and adaptability in various networking scenarios. Table 1 shows the comparison of the selected simulation tools for SDN scenarios.

Table 1: Comparison of selected simulation tools for SDN scenarios

Simulation environment	Analysis of SDN scenarios
Mininet	Good for small and medium-sized networks. Easy to use and can be used to create virtual networks with OpenFlow switches.
GNS3	Can be used to simulate SDN scenarios, but it is not as lightweight as Mininet
NS3	The preferred choice for researching and developing detailed simulations of real-world networks. Though it can be used to simulate SDN scenarios, it is not as easy to use compared to Mininet.
OMNET++	Similar to NS3 in terms of its features and complexity. Good for researchers and developers who need a powerful and customizable network simulator. Can also be used to simulate SDN scenarios, but it is not as easy to use as Mininet
OPNET	Very powerful simulation tool that can be used to simulate large and complex network scenarios. However, it is also very expensive. Can also be used to simulate SDN scenarios, but it is not as easy to use as Mininet.

Implementation Details
POX controller on MININET
Figure 1 shows the Pox Controller installation and setup on the Mininet.

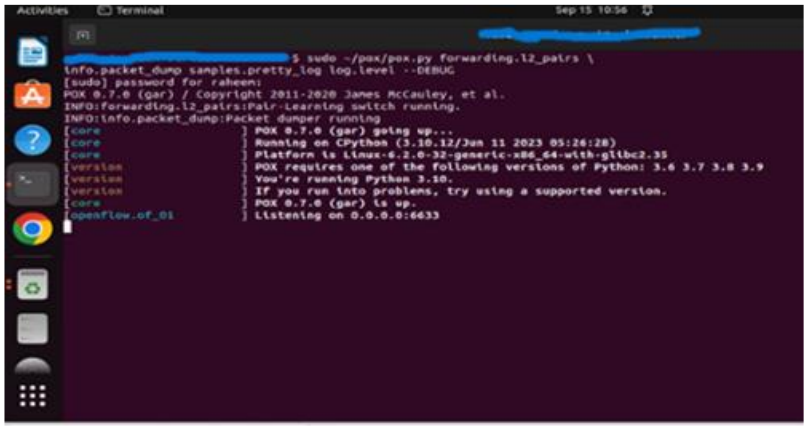

Figure 1: Pox Controller installation and setup on Mininet

The POX controller was setup and running to actively listen for incoming OpenFlow connections on all available network interfaces (0.0.0.0) on port 6633 by running on the terminal the command for starting up the POX controller "sudo ~/pox/pox.py forwarding.l2_pairs \info.packet_dump samples.pretty_log log.level –DEBUG". The POX controller is acting as an L2 learning switch which implies that the controller is implementing a basic networking functionality in an SDN environment in a way similar to that of a traditional Ethernet switch. "listening on 0.0.0.0:6633" is a phrase that

usually refers to the status message or log entry created by the POX controller software when it boots up and starts listening for incoming connections on a specified network address and port.

Explanation of the command in detail:

pox controller: POX is an open-source networking software platform that can be used to design and deploy SDN controllers (this statement simply indicates that the POX controller software is running.

"listening on" indicates that the POX controller is currently listening for incoming network connections.

"0.0.0.0": In networking, 0.0.0.0 IP address indicates the availability of all network interfaces on the host. When the software is configured to listen on 0.0.0.0, it means that the controller is listening for incoming connections on all available network interfaces, which allows it to accept connections from any remote IP address.

"6633": This is the port number on which the POX controller is listening for incoming connections. Port 6633 is commonly used for SDN controllers (Ligia et al (2014)).

Thereafter, the command "sudo mn –topo single,20 –mac –switch ovsk –controller remote" was used in creating an SDN network on Mininet. The command created 20 hosts to be connected to a single switch, OVSK (open virtual switch kernel) which is used because of its compatibility with the SDN framework; '—mac' which assigned mac addresses to each host and '—controller remote' connected the SDN network to an SDN controller that is listening for a connection.

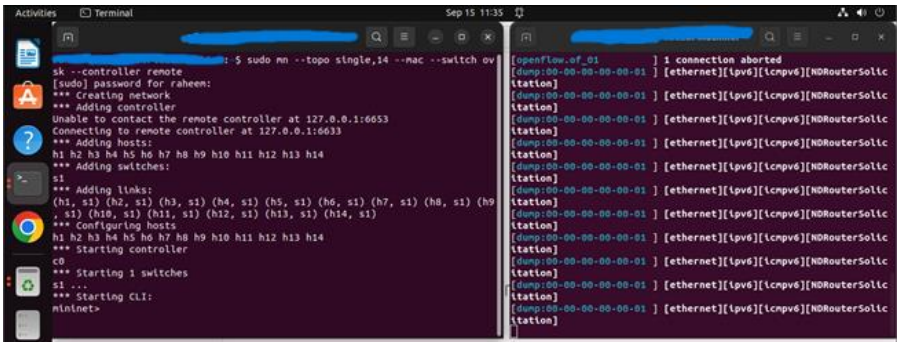

Figure 2: SDN network and Pox Controller connected

POX controller on NS3

Running an SDN network in an NS3 simulator is quite different, the reason being that a C++ script is required.

Figure 03 shows the running of a script named "ofswitch13-first" which contains code for an SDN network of a remote controller, a switch and 14 hosts.

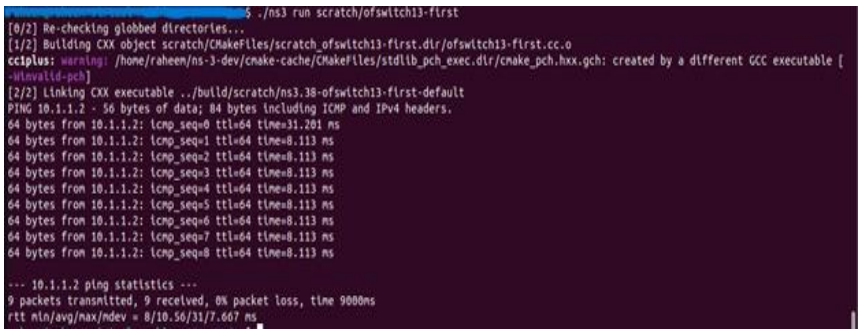

Figure 3: Running a C++ script for an SDN network on NS3

The command "./ns3 run scratch/ofswitch13-first" explained:

"./ns3": The "./" at the beginning specifies the current directory, and "ns3" is the command used to start ns-3.

"run": This is a command that tells ns-3 to execute a specific simulation scenario or script.

"scratch/ofswitch13-first": This is the path to the script to be executed. In ns-3, simulation scenarios are often defined in script files located in the "scratch" directory.

Figure 4 shows the visualization of the SDN script using NetAnim.

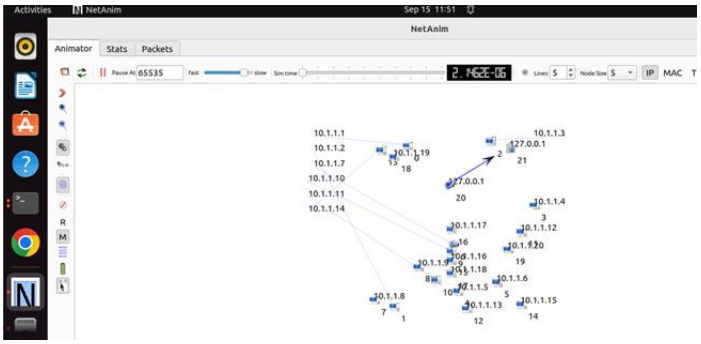

Figure 4: Visualizing SDN script with NetAnim

Running the script on the NS3 terminal isn't the end, in an attempt to visualize the C++ script for the SDN network, NetAnim which is an offline animation tool was used to animate the script. Attached to each node are its ip and MAC addresses.

FAUCET controller on GNS3

On GNS3 the faucet SDN controller and POX controller were installed and configured as a Docker Container (Figure 05).

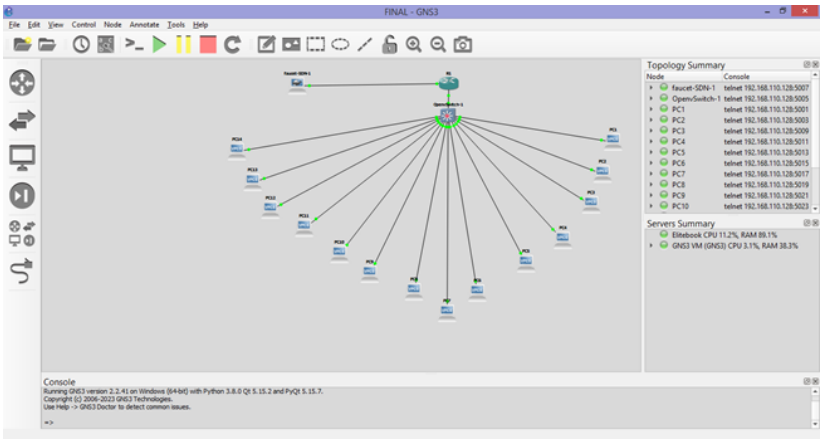

Figure 5: SDN network setup on GNS3

An SDN using a Faucet SDN controller and OpenvSwitch within the GNS3 environment was set as follows:

Install GNS3: Download and installation of GNS3 from the official website (https://www.gns3.com/).

Download the required Software: Faucet SDN Controller was downloaded as a Docker container from the official Faucet GitHub page (https://github.com/faucetsdn/faucet).

OpenvSwitch Appliance: Opening GNS3, and going to the "Appliances" tab, the OpenvSwitch appliance was downloaded from the GNS3 online appliance store.

Setting Up GNS3 Environment by:

Launch GNS3 and create a new project for the SDN network.

Add a few hosts. These will represent devices connected to the SDN network.

Add the OpenvSwitch appliance.

Test the SDN network by sending traffic between the VMs. The faucet manages the network according to the configuration.

FAUCET controller on OPNET

Setting up an SDN network on OPNET was entirely different because a node had to be configured as a sensor, another node configured as an SDN controller, third one configured as an actuator (Figure 6).

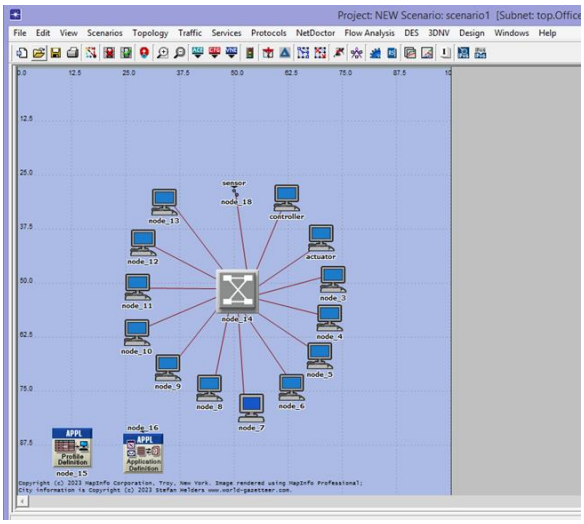

Figure 6: SDN network setup on OPNET 14.5

Configuring a network node as an actuator, sensor, and controller in the context of SDN refers to the role that node plays inside the SDN architecture. SDN separates the control plane (which decides how to forward traffic) from the data plane (which forwards traffic). This division enables more flexible network management and automation. Explanation of what each of these roles entails is as follows:

Controller:

A controller oversees the making of a high-level choice about how network traffic should be forwarded. It instructs SDN switches on how to handle packets and flows by communicating with them via a control protocol, OpenFlow. Controllers serve as the heart of an SDN network. They give a consolidated perspective of the network and oversee network-wide rules, traffic engineering, and network service implementation. Controllers can be set up to make dynamic routing decisions, manage network security, and other tasks.

Actuator:

Actuators are devices or entities in the data plane that physically carry out the SDN controller's instructions. It

could, for example, be a switch that adjusts its forwarding table in response to controller instructions. Depending on instructions from the SDN controller, an actuator is a network device or element that executes actions. For example, the actuator modifies network configurations, changes routing paths, and takes other activities to enforce network policies.
Actuators are needed to convert controller choices into real network activities.
Sensor:
A sensor is a network devices or element that detect and respond to changes in their environment. It collects and transmits data to the SDN controller regarding network conditions, traffic, and performance. Monitoring networks, sensors send real-time data to the controller, allowing it to make more cogent decisions.
Sensors allow the controller to respond to changing network circumstances by offering it network telemetry data. Sensors, for example, may report bandwidth usage, latency measures, security events, or other relevant metrics. The controller can use this data for network optimization, defect detection, and security enforcement.
SDN controller on OMNET++
Creating an SDN network in OMNET++ is quite complex. It involves using the INET FRAMEWORK and configuring the OpenFlow protocol to support an SDN network (Figure 7).

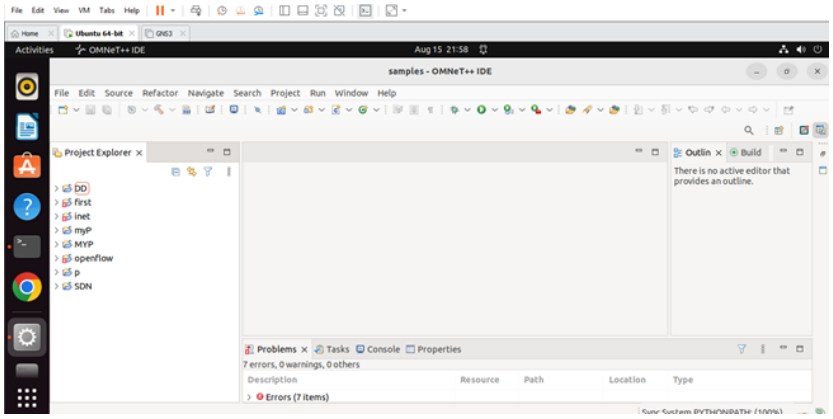

Figure 7: OMNET++ environment setup

A comprehensive description of how an SDN network was set up in OMNeT++ is as follows:
OMNeT++: OMNeT++ was installed and configured.
INET Framework: the INET Framework, which is an extension for OMNeT++ designed for network simulations was installed.
OpenFlow: a protocol used for SDN communication between the controller and switches.
Setting Up the Environment:
Create a New OMNeT++ Project:
Start by creating a new OMNeT++ project where the SDN simulation will be developed.
Import INET Framework:
Import the INET Framework into your OMNeT++ project by copying the INET Framework files into the project directory (this could also be done by adding it as a project dependency in the IDE (if supported)).
Define Network Topology:
This includes specifying the nodes (switches, hosts), links, and connections.
Configure OpenFlow on SDN Switches:
In INET, SDN switches are typically implemented as OpenFlow switches. The OpenFlow protocol is configured on the SDN switches. This includes specifying the OpenFlow version (e.g., OpenFlow 1.3), controllers' IP address, and port number.
Implementing the SDN Controller:
Develop the SDN Controller:
using the INET Framework's modules and components, a script was written for the pox and faucet controller.
Connect the Controller to Switches:
Connection is established between the SDN controller and the OpenFlow-enabled switches in the topology.
Controller Logic:
The controller was configured to act as a router, load balancer or firewall.
Running the Simulation

RESULTS AND DISCUSSION

MININET

Before measuring any performance metric, the PINGALL command was used to ensure that the hosts in the network communicate with each other. Figure 08 shows the result after testing connectivity among the hosts on the Mininet.

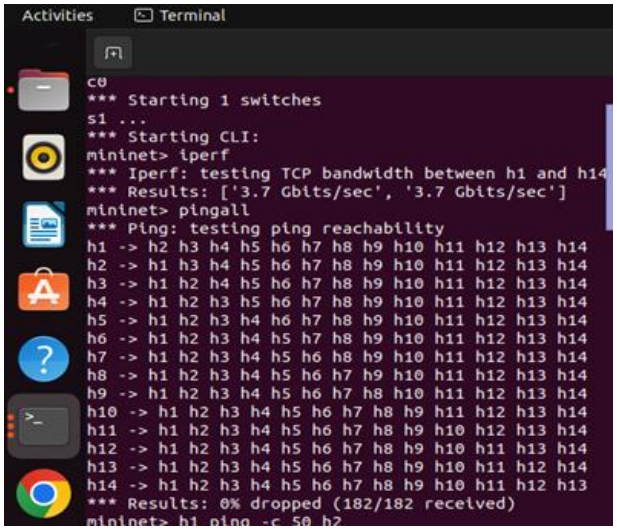

Figure 8: Connectivity among hosts on the Mininet

Figure 9 shows the results of packet loss and Latency on the Mininet.

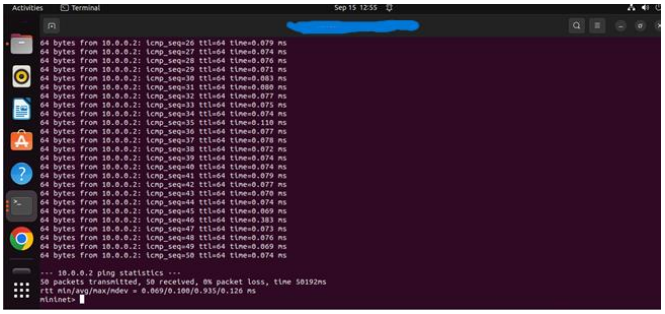

Figure 9: Packet loss and Latency on Mininet

50 packets transmitted: This indicated that the sender sent a total of 50 ICMP (Internet Control Message Protocol) echo request packets to the destination host.

50 received: This means that all 50 of the ICMP echo request packets sent were successfully received by the destination host. In other words, none of the packets were lost in transit.

0% packet loss: This is a summary statement based on the "50 packets transmitted" and "50 received" values. It means that there was no packet loss during the test. All transmitted packets were successfully received; thus, the packet loss rate is 0%.

Time 50192ms: This indicated the total time taken for the entire ping test. In this case, it took 50,145 milliseconds (or approximately 50.2 seconds) to send all 50 packets and receive responses from the destination host.

Rtt min/avg/max/mdev = 0.069/0.10/0.935/0.126 ms:

rtt: Stands for "round-trip time," which is the time it takes for a packet to travel from the sender to the receiver and back. It is measured in milliseconds (ms).

Min: The minimum round-trip time observed during the test. In this case, the minimum round-trip time was 0.069 ms.

Avg: The average round-trip time calculated from all the packets sent and received. In this case, the average round-trip time was 0.100 ms.

Max: The maximum round-trip time observed during the test. In this case, the maximum round-trip time was 0.935 ms.

Mdev: Stands for "mean deviation." It is a measure of the variation or dispersion of round-trip times. In this case, the mean deviation was 0.126 ms. Figure 10 shows the CPU and memory utilization by the Mininet.

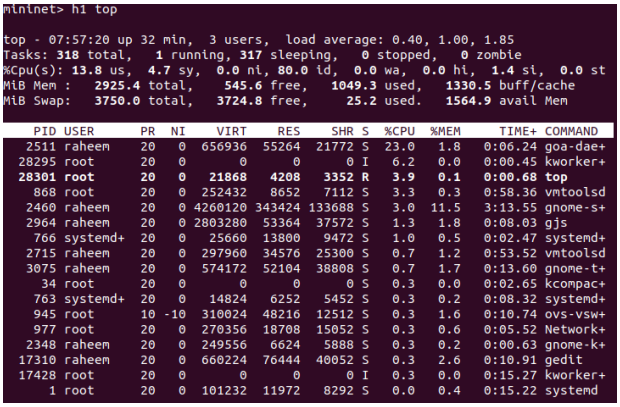

Figure 10: CPU and memory utilization by the Mininet

NS3

Figure 11 shows the results of the packet loss and latency on NS3.

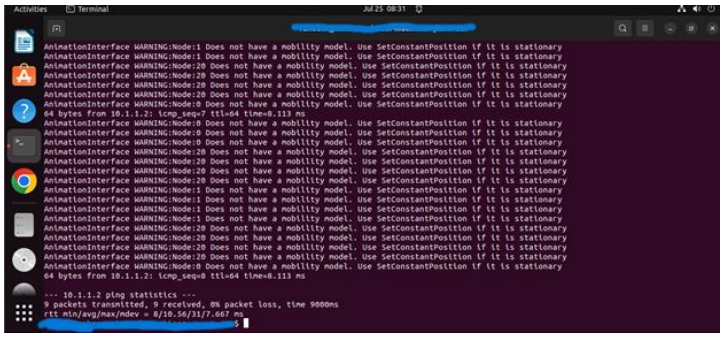

Figure 11: Packet loss and latency on NS3

GNS3

Figure 12 shows the Faucet controller configuration on GNS3, while Figure 13 shows the configuration of routers to generate Packet loss and Latency results.

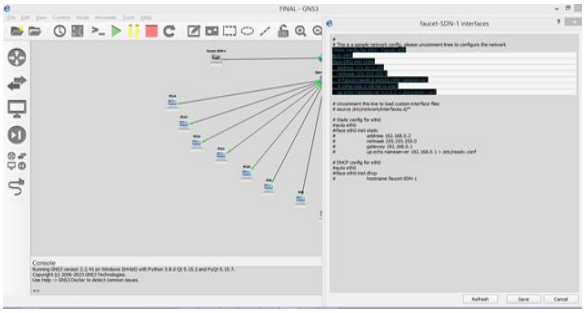

Figure 12: Faucet Controller Configuration

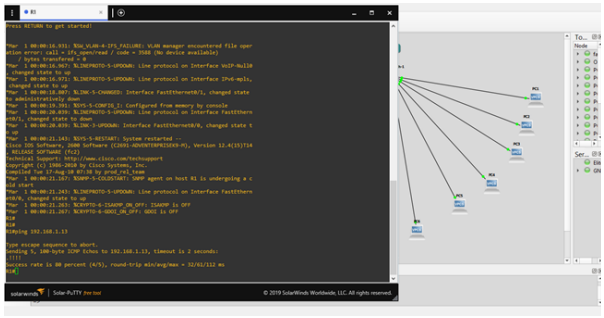

Figure 13: Routers' configuration generating Packet loss and Latency results

OPNET

Figure 14 shows the attributes configuration for the sensor, actuator and controller on OPNET 14.5.

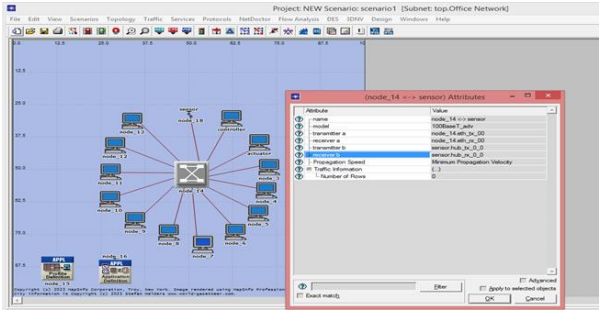

Figure 14: Sensor, actuator and controller

Figure 15 shows the configuration of DES statistics on OPNET 14.5 to generate results for performance metrics.

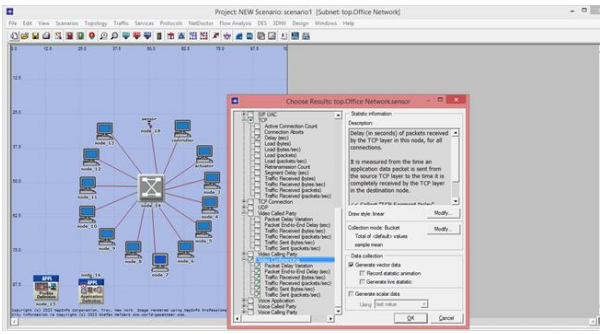

Figure 15: DES statistics configuration on OPNET 14.5 to generate results for performance metrics

Table 2 shows all the results on performance metrics from all the simulation environments.

Table 2: Performance metrics from all the simulators

Metric	MININET	NS3	GNS3	OPNET	OMNET++
Throughput	6.2Gb/s	0.99Mb/s	350mb/s	4.5mb/s	150mb/s
Packet loss	0%	0%	20%	0.3%	1%
Latency	0.190ms	10.56ms	21ms	1.5ms	6ms
CPU utilization	5.9%	30%	2%	45%	35%
Memory utilization	11.1%	60%	19.2%	-	50%

Analysis

Throughput:

Mininet demonstrates the highest throughput, making it the best choice when a high data transfer rate is of utmost priority. However, it is a lightweight simulator that is not designed for simulating large networks with a lot of traffic. GNS3, OPNET and OMNET++ also offered decent throughput. Though NS3 has the lowest throughput. yet it is a more complex simulator that is designed for simulating large networks with a lot of traffic.

Packet loss:

OPNET, Mininet and OMNET++ exhibited low packet loss rates, and this is very good as the low packet loss is very crucial for data integrity. Thus, making any of them a preferred choice when low packet loss is paramount. NS3 and GNS3 have higher packet loss rates, which would be a concern for applications sensitive to data loss.

Latency:

High latency affects real-time applications adversely, thus, making GNS3 and NS3 that had higher latency values unpreferred choice for real-time applications. On the other hand, Mininet demonstrated the lowest latency, and this is beneficial for low-latency network designs.

CPU and Memory Utilization:

Very unlike the Mininet and GNS3 have moderate resource utilization which makes them suitable for projects with moderate hardware constraints, NS3 and OPNET had the highest CPU and memory utilization which could be a concern if resource efficiency is of concern to the user. However, Mininet strikes a balance between resource utilization and performance.

Comparison of the SDN controllers

Table 3 shows the comparisons of the SDN controllers being investigated in this study.

Table 3: SDN Controllers Compared

Factor	POX	FAUCET
Community support	Large and active community	Large and active community
Complexity	Light weight	Complex
Ease of use	Easy to use	More difficult to learn and use
Deployment	Good for small networks and for rapid prototyping of new SDN applications	Good for large networks or for deploying complex SDN applications
Features	Basic features	A wide variety of features
Flexibility	Flexible	Flexible
Programming language	Python	Java

Comparison of the selected Simulation Environment

Table 4: The Selected Simulation Environments Compared

Simulation environment	Analysis of SDN scenarios
Mininet	Good for small and medium-sized networks. Easy to use and can be used to create virtual networks with OpenFlow switches.
GNS3	Could be used to simulate SDN scenarios, but it is not as lightweight as Mininet
NS3	Good for researchers and developers who need to create detailed simulations of real-world networks. Could be used to simulate SDN scenarios, though it is not as easy to use, compared to Mininet.
OMNET++	Similar to NS3 in terms of its features and complexity. A great choice for researchers and developers who need a powerful and customizable network simulator.
OPNET	Very powerful and could be used to simulate large and complex networks. However, it is not as easy to use as Mininet, as observed when it was being used to simulate SDN scenarios in this study.

CONCLUSION

Throughout the study, valuable insights into the strengths and weaknesses of various simulators when applied to SDN controller implementations were gained. The results of the evaluation showed that the performance of the SDN controllers varies depending on the network simulator and the complexity/size of the network topology. Though Mininet could be a preferred choice because of its ease of use and that it is very good for simulating small and medium-sized networks, when there is a need to simulate large and complex networks, OPNET might be the best choice, not minding the fact that it is expensive compared to others. In a nutshell, the complexity of the network and or the resources available to the developer (both the hardware and the Software) play a vital role in the choice of the simulator for the implementation of an SDN controller. Nevertheless, Mininet provided competitive performance metrics, making it a cost-effective solution for simulating SDN controllers with moderate hardware requirements. NS3, though useful for SDN network design and visualization, exhibited higher CPU and memory utilization that might limit its scalability for large-scale SDN controller simulations. While GNS3 offered a balanced

performance and resource utilization, making it a suitable choice for SDN controller simulation that prioritizes realistic network modelling, OMNET++, on the other hand, exhibited moderate performance metrics with efficient resource utilization, making it suitable for SDN controller simulations requiring a balance between performance and resource efficiency.

REFERENCES:

Abuarqoub, A. (2020). A Review of the Control Plane Scalability Approaches in Software Defined Networking. *Future Internet*, 12(3), 49. https://doi.org/10.3390/fi12030049

Adekunle O. O., & OluwaseyiO. (2023). A SECURITY ARCHITECTURE FOR SOFTWARE DEFINED NETWORK (SDN). *FUDMA JOURNAL OF SCIENCES*, *2*(2), 28 - 36. Retrieved from https://fjs.fudutsinma.edu.ng/index.php/fjs/article/view/1347

Bosi, L. L., Mendes, A. C., & Salles, R. M. (2024). A Review on the Overall Performance of SDN Controllers. In 2024 11th International Conference on Software Defined Systems (SDS). IEEE. 156-163. https://ieeexplore.ieee.org/abstract/document/10883890

Chouhan, R. K., Atulkar, M., & Nagwani, N. K. (2019). Performance Comparison of Ryu and Floodlight Controllers in Different SDN Topologies. *2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE)*. https://doi.org/10.1109/icatiece45860.2019.9063806

Franco-Almazan, A., Fernandez-Soriano, N., & Vidal-Beltrán, S. (2019). A comparison of Traditional Network and Software-defined Network schemes using OpenFlow protocol. *WSEAS Transactions on Computers*, 18, 210-216.

Gil, P., Garcia, G. J., Delgado, A., Medina, R. M., Calderon, A., & Marti, P. (2014, October). Computer Networks Virtualization with GNS3. In *Proc IEEE Frontiers in Education Conference* (pp. 2141-2144).

Gupta, N., Maashi, M. S., Tanwar, S., Badotra, S., Aljebreen, M., & Bharany, S. (2022). A Comparative Study of Software Defined Networking Controllers Using Mininet. *Electronics*, 11(17), 2715. https://doi.org/10.3390/electronics11172715

Imran, Ghaffar, Z., Alshahrani, A., Fayaz, M., Alghamdi, A. M., & Gwak, J. (2021). A topical review on machine learning, software defined networking, internet of things applications: Research limitations and challenges. *Electronics*, *10*(8), 880.

Ligia Rodrigues Prete, Shinoda, A. A., Schweitzer, C. M., & de Oliveira, R. L. S. (2014). Simulation in an SDN network scenario using the POX Controller. *2014 IEEE Colombian Conference on Communications and Computing (COLCOM).* https://doi.org/10.1109/colcomcon.2014.6860403

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., & Turner, J. (2008). OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communications Review*, Apr. 2008.

Rego, A., Garcia, L., Sendra, S., & Lloret, J. (2018). Software Defined Network-based control system for an efficient traffic management for emergency situations in smart cities. *Future Generation Computer Systems*, 88, 243-253.

Salman, O., Elhajj, I. H., Kayssi, A., & Chehab, A. (2016). SDN controllers: A comparative study. *2016 18th Mediterranean Electrotechnical Conference (MELECON)*. https://doi.org/10.1109/melcon.2016.7495430

Singh, A., Kaur, N., & Kaur, H. (2022). Extensive performance analysis of OpenDayLight (ODL) and Open Network Operating System (ONOS) SDN controllers. *Microprocessors and Microsystems*, 95, 104715. https://doi.org/10.1016/j.micpro.2022.104715

Tello, A. M. D., & Abolhasan, M. (2019). SDN Controllers Scalability and Performance Study. *2019 13th International Conference on Signal Processing and Communication Systems (ICSPCS)*. https://doi.org/10.1109/icspcs47537.2019.9008462

Tiamiyu, A. O. (2012). Comparative Analysis of Imitation Modeling Software Supporting Trusted Routing (Сравнительный Анализ Средств Имитационного Моделирования ТКС, Поддерживающих Доверенную Марщрутизацию). In *Topical issues on problems of information security: collection of scientific articles (Актуальные Проблемы Информационной Безопасности, Сборник Научных Трудов)* Stelmashonok E. V. (ed.) 49−53. http://infosec.spb.ru/wp-content/uploads/2014/05/SbornikNTr_20121.pdf