International Journal of Communication Networks and Information Security
2024, 16(4)
ISSN: 2073-607X,2076-0930
https://ijcnis.org/

Research Article

A Predictable performance Multi-controller-based Monitoring Framework for SDN

Mahmoud Eissa[1], Ahmed Yahya [2], Usama AbdelFattah[3]

1) Master student, Electrical Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt.
2) Professor of Electronics, Electrical Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt
3) Assisstant professor, Electrical Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt

*Corresponding Author: mahbor92@gmail.com

ARTICLE INFO

Received: 1 Sep 2024
Accepted: 11 Oct 2024

ABSTRACT

Software defined networks (SDNs) have been released for dynamic operations of the networks and for controlling scalable networks. In SDN, the operating processes are controlled by a centralized controller. The performance of the centralized controller is decreased by increasing the scale of the network. Monitoring functionality is an essential element of any network system. The monitoring performance under centralized controller is decreased with large scale networks. In this paper, a proposed monitoring framework for multi-controller software defined networks is introduced. The introduced framework enhances the reliability and performance of large-scale software defined networks. Many copies of the proposed monitoring framework can run with SDN multi-controller for monitoring large-scale networks, detecting failure in any controller, and failover of the network failure. All copies of the introduced monitoring framework run in parallel, each on a slice to enhance the performance of the SDN network. All copies receive the requests from network applications, collect considerable amounts of measurements data, process them, and return the results to the network applications.

The contribution of this paper is introducing a reliable and high-performance multi-controller-based SDN monitoring framework. The introduced monitoring framework monitors large scale SDN networks with good performance and enhances network reliability. It has the capabilities to monitor in parallel many network applications where each application runs on a network slice. Each slice is controlled by an SDN controller and monitored by a copy of the introduced monitoring framework. The copies of the introduced monitoring framework are communicated in a synchronization scheme.

Keywords: Software Defined Network, Multi-controller monitoring framework, monitoring framework architecture, network applications.

INTRODUCTION

Software-Defined Networking (SDN) represents a modern networking approach where a central software component, the SDN controller, oversees the network's overall behavior [1]. A

network monitoring system (NMS) is essential for managing and observing SDN networks. One of the key roles of an NMS is to continuously assess the internal network, identifying and addressing various issues that can arise, such as sluggish web page loading, missing emails, unusual user activity, insecure connections, or server malfunctions [1].

Unlike Intrusion Detection Systems (IDSs) or Intrusion Prevention Systems (IPSs), which concentrate on identifying and thwarting unauthorized intrusions, Network Monitoring Systems (NMSs) are primarily employed to monitor network performance under normal operating conditions. They also collect data for network management tools, including IDSs, for better network control.

To efficiently handle traffic monitoring tasks, the NMS must be compatible with a variety of devices from different vendors, including mobile devices, hosts, servers, routers, and switches [2]. SDN, as a paradigm, is founded on several key concepts, one of which is the separation of control and data planes in traditional networks. In SDN-enabled networks, the data plane resides in forwarding devices like SDN switches, while the control plane is centralized in SDN controllers. This division allows for distinct layers of abstraction, offering significant flexibility in network management.

Figure 1 illustrates the architecture of an SDN-based network, depicting its components, planes, and layers of abstraction [3]. It comprises three primary planes: the application, control, and data planes. The control plane interacts with the data plane via the southbound interface (SBI), while communication with the application plane is facilitated through the northbound interface (NBI). Additionally, east-west bound APIs have been introduced to enable communication between controllers, either within the same domain or across different domains.

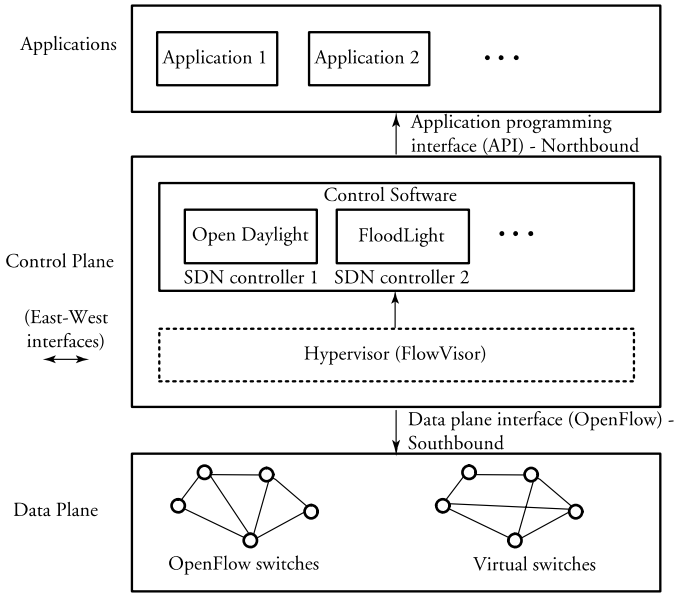


Figure 1. Software-Defined Networking architecture [2].

C**ontrol plane:**It is tasked with evaluating the local status of network forwarding devices and enforcing policies to ensure the network operates efficiently. These policies may include those related to routing, traffic engineering, and security. Unlike traditional networks, which rely on destination-based forwarding, SDN uses flow-based forwarding. This means that all packets matching a specific criterion are treated with the same set of actions. The control plane is responsible for determining traffic paths and overseeing network signaling to ensure correct device configurations are applied [1][3][4].

Data plane:It is known as the forwarding plane, is responsible for traffic forwarding in devices such as switches, based on the policies established by the control plane. This includes filtering

traffic and applying various actions to incoming packets. In SDN, switches are more generalized and can enforce flow-level policies, integrating functions that were traditionally handled separately by routers and switches in conventional networks [1][5].

Application plane:It is interact with the SDN controller through Application Programming Interfaces (APIs). These applications include business functionalities, network management, and security management, and they can be developed to detect issues, malicious activity, anomalies, or other concerns [1]. Interfaces between the layers, specifically the northbound and southbound interfaces, facilitate communication. The northbound interface connects the controller with the application layer, while the southbound interface links the controller to the data forwarding layer. OpenFlow is a standard protocol used for communication between the control and data planes in SDN-enabled networks [3][6].

The SDN model is built on the concept of open interfaces, where the controller offers an API that control applications can use to monitor and modify the network's state. It is also allows for the addition of new functionalities by developing further applications. The controller employs open protocols, such as OpenFlow, to configure network nodes like switches and routers according to the requirements of the applications, with OpenFlow being the most widely accepted standard protocol.

An SDN monitoring system must be capable of collecting, processing, and analyzing a vast range of data in real time to provide the necessary insights for effective network decision-making. However, this poses a significant challenge due to the large-scale, dynamic nature of current networks, which have high traffic volumes and strict demands on time and hardware resources.

SDN has introduced flexibility and centralized control to network management, monitoring frameworks based on single-controller architectures face several limitations when compared to multi-controller approaches. Single-controller systems often struggle with scalability, as they can become bottlenecks in large networks, leading to slower data collection and decision-making. In contrast, multi-controller architectures distribute the monitoring tasks, improving scalability and performance. Additionally, single-controller setups are more vulnerable to failure, whereas multi-controller frameworks enhance fault tolerance by providing redundancy and load balancing, ensuring more resilient network management [7].

In this paper, we address this challenge by introducing a scalable monitoring framework capable of overseeing small, medium, and large networks. This framework's monitoring capabilities can be expanded to accommodate a growing number of tenant applications, including management-related applications.It also monitors management applications and allows to generate accurate and frequent monitoring reports concerning a variety of network events and emerging conditions such as network performance bottlenecks, anomaly detection and others.

The paper contribution can be summarized as follows:

1. It presents a high-performance monitoring framework for software-defined networks.
2. It introduces a reliable multi-controller-based monitoring framework by supporting failover of the network failure
3. It addresses the performance of the monitoring and SDN in two dimensions (i) the performance of the monitoring framework with large size SDN (ii) the performance of monitoring framework with large number of applications monitored by the framework.

The rest of this paper is organized as follows: Section 2 shows related work to the paper. Section 3 discusses the proposed architecture and solution. The implementation and validation of the proposed solution are discussed in section 4. In Section 5, the validation of reliability and monitoring framework experiments are discussed. Evaluation and comparative study are discussed in section 6. Section 7 concludes the proposed work.

RELATED WORK

Numerous studies have been conducted to monitor Software-Defined Networking (SDN), yet gaps remain in these investigations. The following section critically examines the existing literature relevant to this paper.

In the work by J. Suariz [1], a flow monitoring solution tailored for OpenFlow-based SDNs is proposed. This solution produces reports that contain flow-level measurements. To minimize the overhead on the controller and decrease the number of flow entries required in switches, two traffic sampling techniques have been introduced. These methods have been implemented, validated, and assessed using the OpenDaylight controller.

B. Isyaku, M. Zahid, M. Kamat, K. A. Bakar, and F. Ghaleb proposed innovative solutions to tackle performance challenges in SDN arising from limited flow tables. They suggested leveraging fuzzy logic and machine learning techniques to identify frequently accessed flow entries that should be retained in the flow table. This research also identifies various challenges, including update operations, resource limitations, communication overhead, and delays in packet processing [3].

M. Scarlato [5] utilized three conventional network monitoring tools—Ntop, Wireshark, and Argus—and adapted them for monitoring SDNs. The performance of these adapted tools was assessed, revealing that Ntop and Argus underperformed as they could not identify OpenFlow traffic as expected.

Niu et al. [8] presented a Two-Layer Trusted Multi-Controller Architecture, a noteworthy development in SDN designed to tackle scalability and reliability challenges in multi-controller settings. This architecture employs a multi-chain approach that integrates Consistent Fault Tolerance (CFT) and Byzantine Fault Tolerance (BFT) blockchains to establish a decentralized and credible decision-making framework. It is structured into two layers: the foundational control layer, which uses the CFT blockchain for trusted distributed coordination, and the dynamic decision layer, employing the BFT blockchain for multi-party engagement in key processes like anomaly detection and recovery. This dual-layer design enables flexible adjustment of decision-making members, thereby enhancing scalability and reliability. Furthermore, it incorporates mechanisms for initial load balancing and fine-tuned dynamic load balancing to avert controller overload, along with a robust exception detection system based on switch reporting (EDSR) for managing various network anomalies.

Zhang et al. [9] conducted a thorough survey on multi-controller-based SDNs, exploring their origins, challenges, and implementation strategies. The authors noted that while multi-controller systems can enhance network performance, their scalability and reliability hinge on factors such as the number of controllers and deployment strategies. Poorly planned controller deployments can lead to uneven workloads and diminished control plane capacity, thereby undermining the anticipated scalability advantages. Limited bandwidth of connection links between switches and controllers can lead to communication problems and isolation, while controllers themselves are vulnerable to failures or attacks. Overall, the survey by Zhang et al. highlights the need for meticulous deployment strategies and the addressing of potential reliability issues to optimize performance in multi-controller SDN environments [9].

While Wireshark is widely recognized as a comprehensive tool, it has numerous filters specific to OpenFlow packets. G. Tangary demonstrated that adaptive monitoring functionalities can be integrated into the packet-processing pipeline to maintain the accuracy of monitoring reports amidst varying resource availability in the data plane [10].

In their survey paper [11], M. Alsaeedi, M. Mohamad, and A. AL-Roubaiey outlined four key challenges in OpenFlow-SDN that contribute to traffic overhead. They also highlighted research hurdles that need to be addressed to develop more adaptive and scalable OpenFlow-SDN solutions,

particularly in large-scale and dynamic network environments. One significant challenge involves monitoring and classifying network flows at the application level [11].

Moreover, in prior research, we introduced a scalable monitoring framework designed specifically for Software-Defined Networks (SDNs), which addressed the critical need for efficient, vendor-independent monitoring in multi-tenant environments. The system utilized OpenFlow to retrieve measurement data from various network components, ensuring its adaptability to diverse infrastructures. Scalability was achieved by slicing the network topology with FlowVisor and managing each slice with a dedicated Floodlight controller. This foundation forms the basis for the current study, which aims to further extend the monitoring framework's capabilities and address some of its limitations, such as compatibility with additional SDN controllers[12].

THE PROPOSED MULTI-CONTROLLER-BASED MONITORING FRAMEWORK

In this section, the layered architecture of the monitoring framework environment, and layered detailed design of the proposed monitoring system are introduced.

3.1) The layered high-level architecture of the monitoring framework environment

The high-level architecture of the monitoring framework environment, as depicted in Figure (2), is arranged in a layered structure. The architecture contains four layers: network application layer, proposed monitoring framework layer, SDN control layer, and SDN infrastructure layer (data plane). The network application layer contains applications such as fault detection, anomaly detection, performance management and others. Each network application needs specific measurements data for analyzing and taking a decision based on the functionality of the application. The measurement data are collected by the monitoring framework. The decision should be sent to the monitoring framework for reconfiguring the data plane (SDN infrastructure).The functionality of each application is different than other applications' functionalities. For example, the functionality of fault detection is different than the functionality of anomaly detection and performance management applications.

The proposed monitoring framework layer contains many copies of the monitoring framework. Each copy is synchronized with its neighbor copies for measurements data exchange. Each monitoring copy monitors a slice of the SDN infrastructure. The monitoring framework provide APIs to be used by network applications.

The SDN controller layer contains N-copies of controller. Each controller copy controls a slice of the SDN infrastructure and used by a copy of monitoring framework. The SDN infrastructure layer (data plane) contains all devices and hosts of the underlying network. There are two standard interfaces: northbound interface and southbound interface. These two interfaces are used for communication between the different layers as shown in Figure 2.

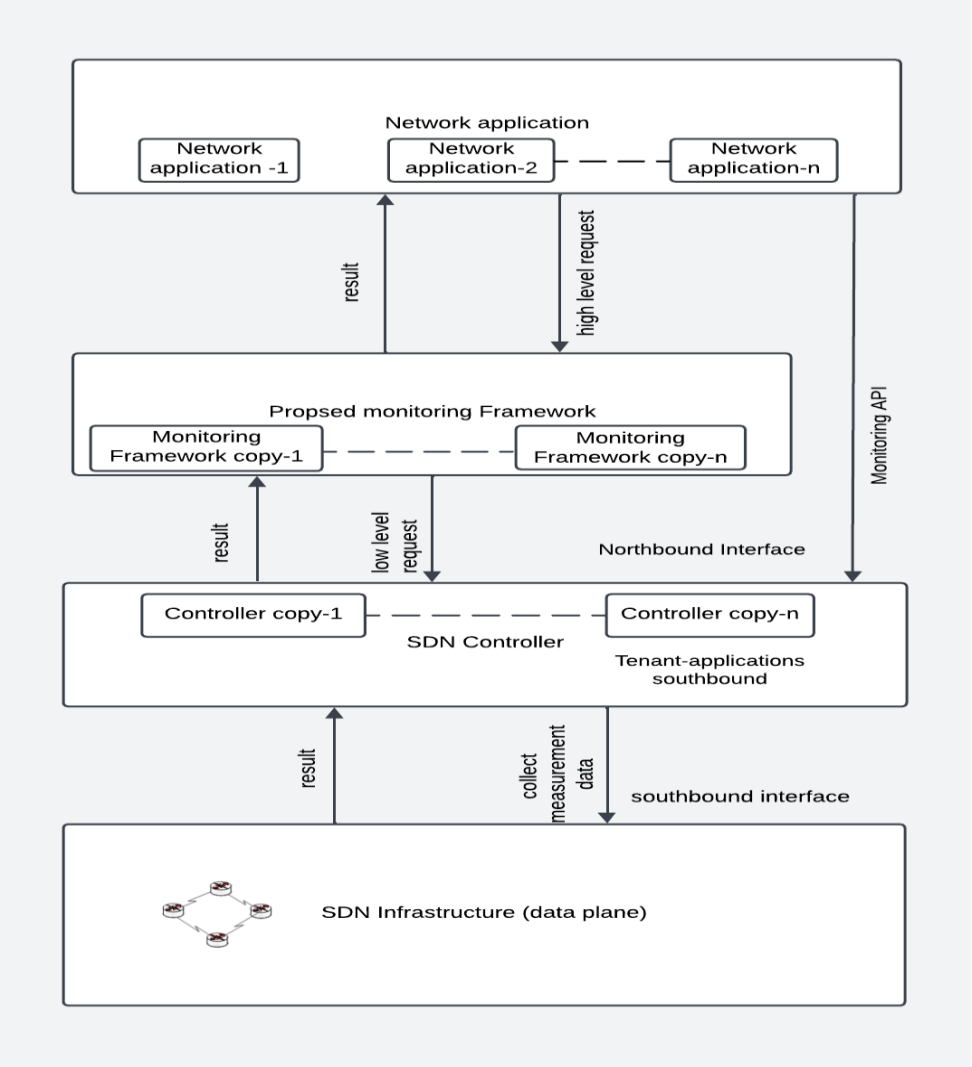


Figure 2. The layerd architecture of the monitor software framework environment

3.2) The detailed design of the proposed monitoring framework

The detailed design of the monitor framework is depicted as a layered architecture as shown in figure 3. The layers of the monitoring framework are L1, L2, and L3. There are three other layers: Layer L0 includes network applications, layer L4 includes SDN controller and layer L5 includes SDN infrastructure (data plane).

Layer L1 of the monitoring framework includes three elements: requester manipulator, results returner and synchronization module. Layer L2 contains the translator element. Layer L3 contains the measurements data Collector element. The elements of the three layers (L1-L3) in general receive requests from network applications, collect the necessary measurement data and returns the results to the calling network applications. The elements can collect various types of data for different network applications (calling applications).

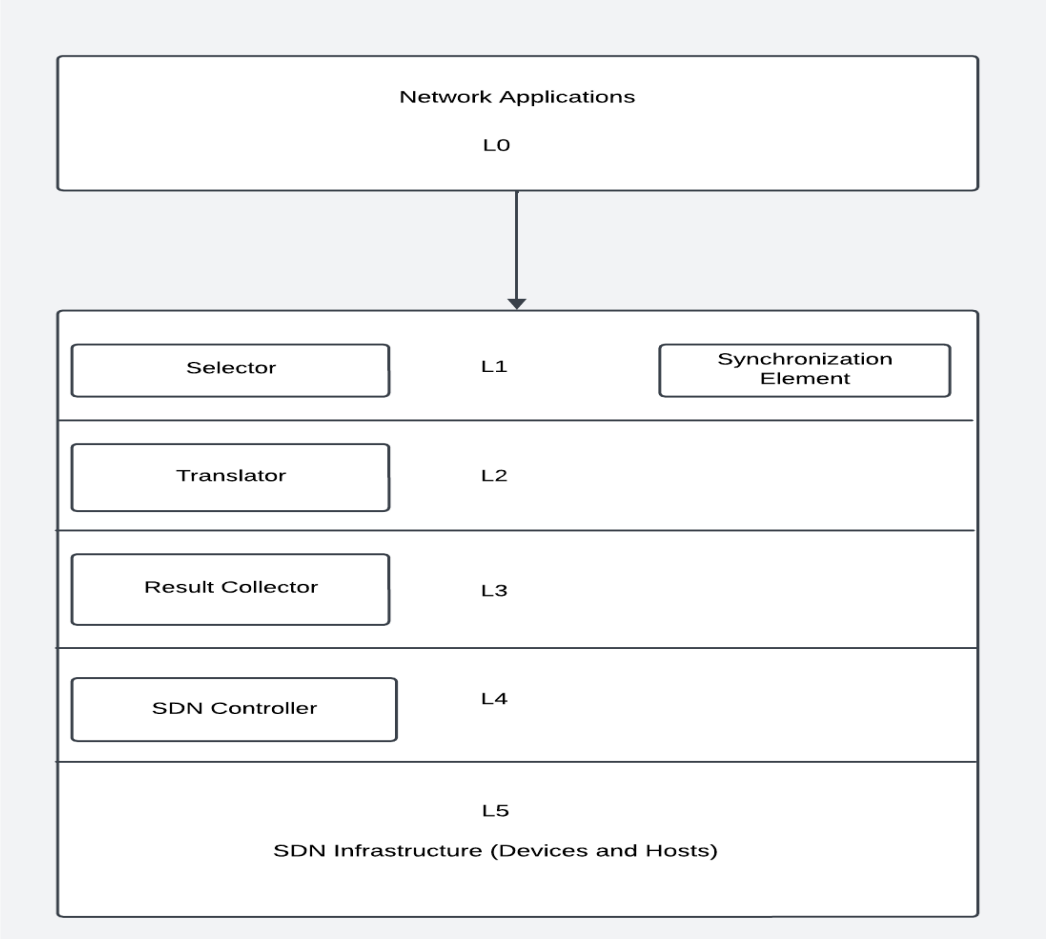


Figure 3: The layered detailed design of the proposed monitoring framework

The monitoring framework consists of many layers. Each layer has one or more module. The top layer (L1) contains selector element, and synchronization element. The second layer (L2) contains the translator module. Layer 3 (L3) contains results collector.

The selector element in layer-1 (L1) receives requests from network Applications in layer 0(L0) and selects the slice and its controller for executing the request. The translator element in layer-2(L2) translates each high-level request into a low-level request and stores the data in the database. The result collector in layer-3(L3) calls OpenFlow API and the controller interfaces to execute the required operation function (request). The measurement results collected by the result collector returned to the calling network application. The collected results are then stored in the results table.

3.3) The Synchronization module (interface)

The synchronization interface is provided to synchronize monitoring information in a SDN multi-controller and big size data plane. It facilitates the exchange of monitoring data between different monitoring framework instances (copies). Any monitoring copy can send the measured data to its neighbors monitoring copies. It also tests the health of the controller of the underlying slice. The faulty controller is detected by the synchronization element and failover of the controller failure is conducted by the synchronization element.

IMPLEMENTATION AND VALIDATION

Implementation and validation of the multi-controller-based monitoring framework.
To implement our proposed multi-controller-based monitoring framework for SDNs, we used the following tools:
1-Floodlight controller for SDN
2-Flowvisor for slicing the SDN topology
3- Mininet emulator for simulating SDN topologies.
4- SQLite database
4- Paython programming language.

In the following section we give a brief about each one:
The **Floodlight** controller is an open-source software-defined networking (SDN) controller developed by the Floodlight Project. It is designed to provide a robust and flexible platform for managing and controlling network devices, such as switches and routers, in an SDN environment. Floodlight operates by communicating with network devices using the OpenFlow protocol, enabling centralized control and dynamic management of network traffic. As a Java-based controller, Floodlight offers a modular architecture, allowing developers to extend and customize its functionality to suit specific network requirements. It supports a wide range of applications, including load balancing, network virtualization, and traffic monitoring. Floodlight is widely used in research, academia, and production environments due to its scalability, ease of use, and active community support, making it a popular choice for implementing and experimenting with SDN solutions. The Floodlight Controller is an open-source SDN controller developed by Big Switch Networks and is based on the OpenFlow protocol. One of the powerful features of the Floodlight Controller is its REST API, which provides a simple yet powerful interface for managing and monitoring an SDN network [13].

FlowVisor is a network hypervisor designed to enable network slicing in software-defined networks (SDNs). It acts as an intermediary layer between the physical network infrastructure and the control plane, allowing multiple independent network controllers to manage distinct slices of the network simultaneously. Each slice operates as if it were a fully isolated network, even though they share the same underlying hardware.

FlowVisor achieves this by intercepting and filtering OpenFlow messages between the network switches and controllers. It ensures that each controller only has visibility and control over its designated slice, thereby maintaining isolation and security across different slices. Additionally, FlowVisor enforces slice-specific policies, such as bandwidth allocation and flow rules, ensuring that each slice operates according to the predefined rules without interference from other slices. This capability makes FlowVisor particularly useful in research and multi-tenant environments where different users or applications need to control separate portions of the network without impacting each other [14].

Mininet is a network emulator that can create a virtual network comprising hosts, switches, controllers, and links. The hosts in Mininet run standard Linux network software, and the switches support OpenFlow, allowing for highly customizable routing and Software-Defined Networking. Mininet is useful for research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having an experimental network on a laptop or other PC [15].

SQLite database is a lightweight, disk-based database that doesn't require a separate server process. In python, sqlite3 is a built-in module that allows programmers to interact with SQLite database [16].For validating the proposed MCMF, we created a topology using the Mininet emulator, sliced the topology to two slices using FlowVisor where each slice is controlled by a

floodlight controller. After that we used the REST APIs for retrieving measurement information from each slice. The measured information is retrieved based on the high-level requests from network applications. In the following we discuss each action in some detail. In addition, we test failover capability, where the MCMF can monitor the slices with fault in one floodlight controller. This means that the MCMF supports the monitoring reliability of multi-controller SDNs.

4.1-Creating a topology

We used the Mininet emulator to simulate a real topology called onstutorial. The onstutorial topology is a custom network topology designed for educational and experimental purposes within the context of software-defined networking (SDN). The onstutorial topology defines a specific arrangement of network switches, hosts, and links, allowing users to experiment with different network configurations, control policies, and failover mechanisms. This topology is particularly useful for demonstrating how SDN controllers interact with network devices and manage traffic flows across the network. By using the onstutorial topology, users can gain hands-on experience in configuring and managing SDN-based networks, testing features like network slicing, redundancy, and failover, all within a controlled and reproducible environment. The onstutorial consists of four switches and 4 hosts and 8 links as shown in Figure 4.

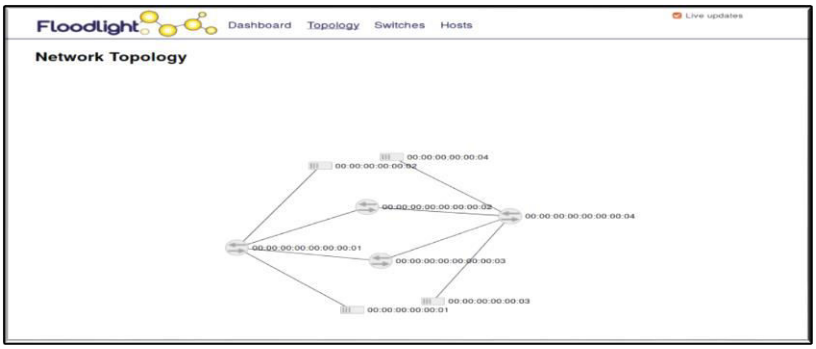
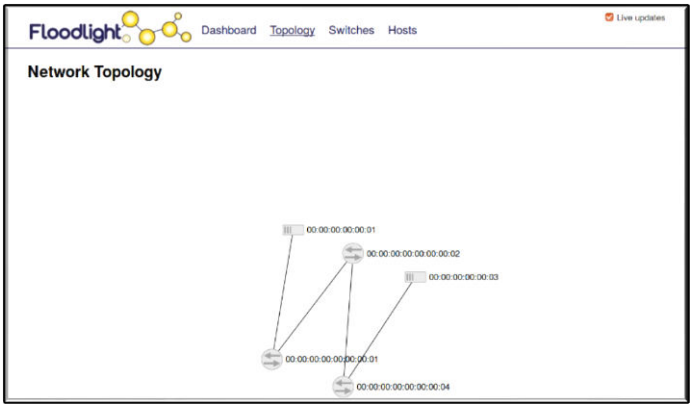


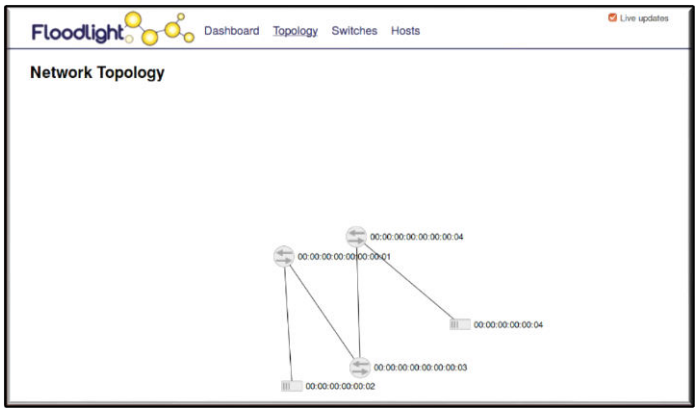
Figure 4: The simulated topology of onstutorial.

4.2- Slicing the topology into slices

Flowvisor has been used to slice our network topology into two slices as shown in Figure 5(a,b). Each slice is managed by one floodlight controller (floodlight1&floodlight2) for the sake of scalability of our proposed MCMF and SDN networks.



a) Upper slicecontrolled by floodlight1



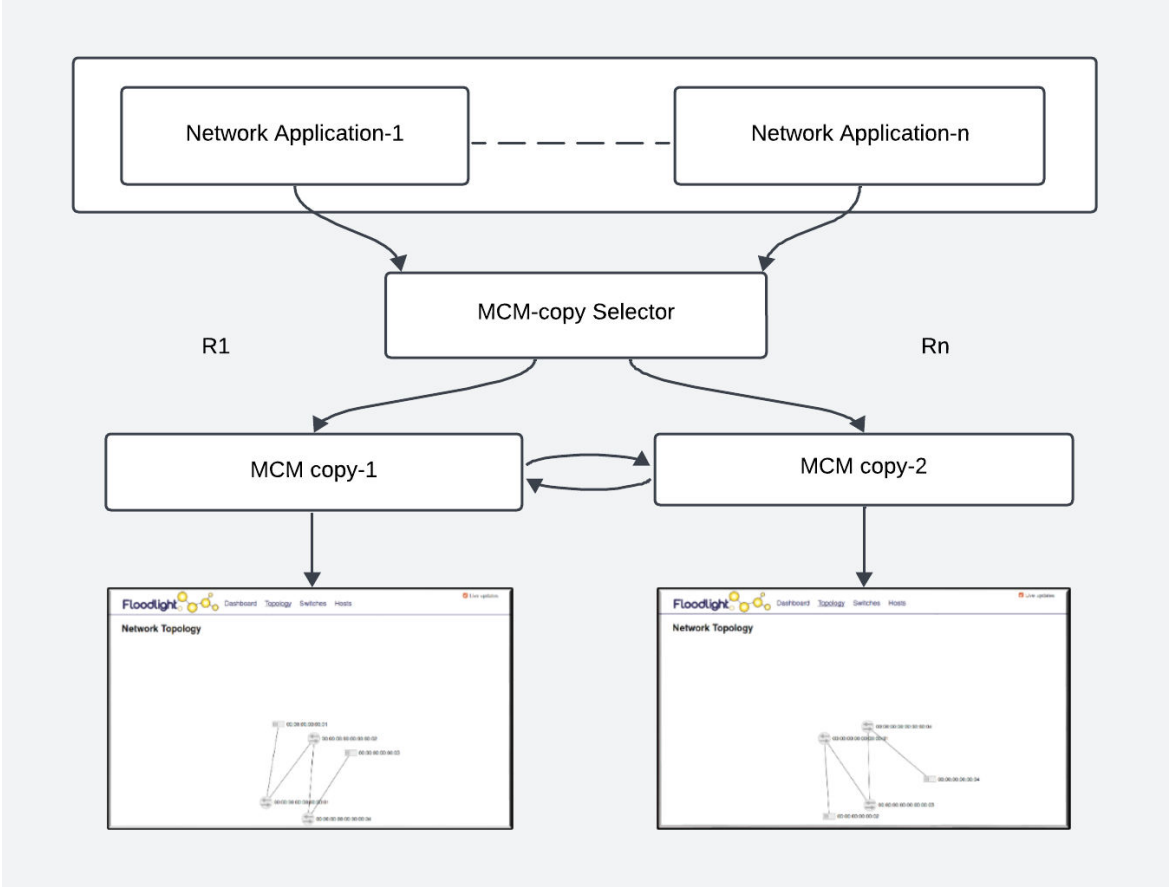
b- Lower slice is controlled by Floodlight2
Figure 5: The two slices are presented a) upper slice b) lower slice

The simulated custom topology has been divided into two slices where each slice is controlled by a different floodlight controller. The high performance with scalable software defined network is achieved by slicing the underlying topology to slices and adding flowspaces that have been assigned to the two floodlight controllers for switches configuration.

In earlierwork[12] , we have conducted a thorough evaluation of the Multi-Controller Monitoring (MCM) framework's performance, specifically focusing on throughput and latency measurements before and after the implementation of network slicing. The initial phase of the evaluation involved measuring throughput and latency in a centralized monitoring setup, where the entire topology was managed as a single entity without slicing. In this configuration, we observed that as the network size and the number of monitoring requests increased, the throughput began to degrade, and latency became significantly higher due to the centralized nature of the system.Subsequently, we applied the network slicing mechanism within the MCM framework, where the network was divided into multiple slices. Each slice was independently managed by a separate instance of the Floodlight controller, enabling distributed control and monitoring across the network. The results post-slicing demonstrated a substantial improvement in both throughput and latency. By decentralizing the monitoring process and distributing the load across multiple controllers, the system could handle a higher volume of traffic with lower delay. Specifically, the throughput increased as each Floodlight controller efficiently managed its respective slice, preventing bottlenecks associated with the centralized monitoring system. Additionally, latency was significantly reduced, as each controller was able to quickly respond to monitoring requests within its designated slice without the overhead of managing the entire network. This evaluation confirms that network slicing within the MCM framework enhances both the scalability and performance of the monitoring system, making it more capable of managing large-scale SDN topologies[12].

4.3-Deployment architecture.

Figure 6 shows the deployment architecture of multi-controller -based monitoring framework. In the figure 6 the upper slice is controlled by floodlight1 and monitored by MCMS copy1, and the lower slice is controlled by floodlight2 and monitored by MCMF copy2. Figure 7 a,b show the information about each slice and its controller after the deployment operation.



The upper slice is controlled by floodlight1 The lower slice controlled by floodlight2

Figure 6: The Deployment architecture of the upper slice, lower slice and floodlight controllers

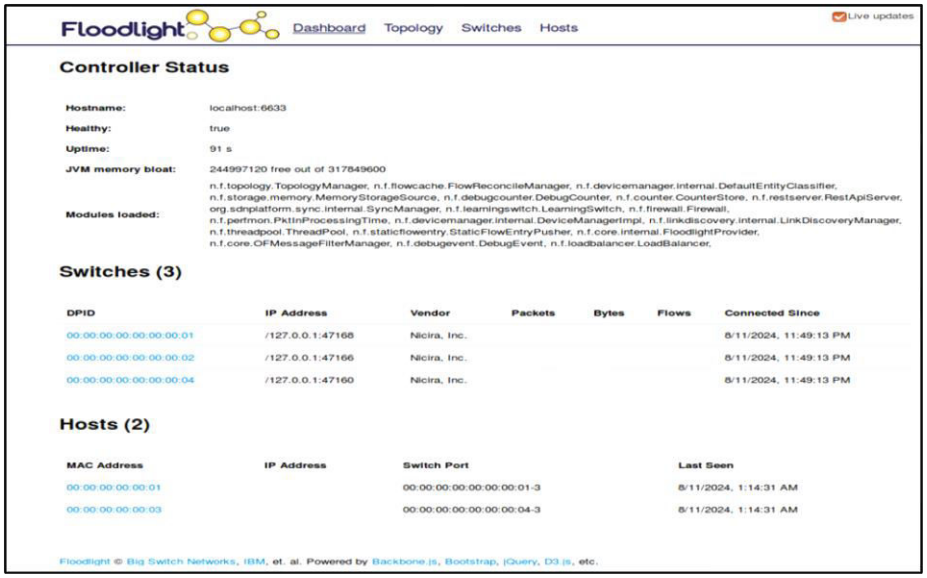


Figure 7.a: A Screen represents that floodlight1 physically controls the upper slice

Figure 7.b: A Screen represents that Floodlight2 controls the lower slice.

4.4- Ensuring the isolation between the two slices.

Four test cases have been conducted as shown in Figure 8 for making sure that the two slices are isolated and testing the reachability between hosts.

Test case 1: host1 sent pingall message to other all hosts, only the host 3 respond because it is in the same slice of host 1. No response from host 2 and host 4.

Test case 2: host2 sent pingall message to other all hosts, only the host 4 sent a response because it is in the same slice of host 2. No response from host 1 and host 3.

Test case 3: host3 sent pingall message to other all hosts, only the host 1 sent a response because it is in the same slice of host 3. No response from host 2 and host 4.

Test case 4: host4 sent pingall message to other all hosts, only the host 2 sent a response because it is in the same slice of host 4. No response from host 1 and host 3.

All the four test cases are shown in the python code in Figure 8.

```
mahmoud5@mahmoud5-VirtualBox:~$ sudo mn --custom ~/onstutorial/flowvisor_scripts
/flowvisor_topo.py --topo fvtopo --link tc --controller remote --mac --arp
[sudo] password for mahmoud5:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s1) (h3, s4) (h4, s4) (1.00Mbit) (1.00Mbit) (s1, s2) (10.00Mbit) (
10.00Mbit) (s1, s3) (1.00Mbit) (1.00Mbit) (s2, s4) (10.00Mbit) (10.00Mbit) (s3,
s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...(1.00Mbit) (10.00Mbit) (1.00Mbit) (1.00Mbit) (10.00Mbit) (10.00Mb
it) (1.00Mbit) (10.00Mbit)
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 X
h2 -> X X h4
h3 -> h1 X X
h4 -> X h2 X
*** Results: 66% dropped (4/12 received)
mininet>
```

Figure 8: Testing the isolation between the two slices upper and lower.

It appears from figure 8, that only host1 can reach host3 and vice versa. This is because the two hosts are in the same slice. Also, host2 only can reach host4 and vice versa. Host1 and host3 as appears in figure 6 are managed by flooldlight1. Host2 and host4 as appears in figure 6 are managed by floodlight2.

4.5- Validating the monitor elements of SDNs

After slicing and deployment validation, we test the elements of the proposed Multi- Controller Monitoring Framework (MCMF).
The MCMF monitor consists of four elements: selector, translator, results returner, and synchronization elements.

4.5.1-The selector element

The selector is implemented in python programming language; it selects the copy of MCMF where the high-level request is assigned. The selection is dependent on the parameters of the request (i.e. links, switches, or others). If the parameters of the request exist in upper slice, then the request is assigned to floodlight1-based MCMF copy1, else is assigned to floodlight2-based MCMF copy2.

4.5.2-The Translator element

The translator element is implemented in python programming language; it receives the request from the selector and translates it into a low-level request that makes it so easier to monitor the network. The low -level requests call the Floodlight REST APIs for collecting needed information. The translator algorithm is represented as a Flowchart diagram and shown in Figure 9. Figure 10 shows the implementation of receiving a request for retrieving information about specific switches and other elements from slice 1. The system status request implementation and results are shown in figure 11.

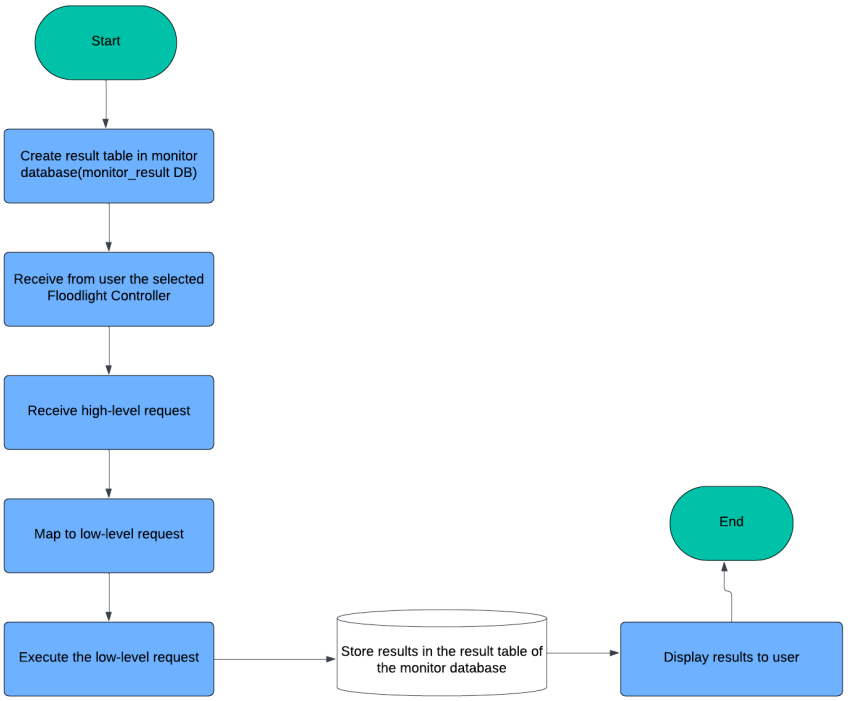


Figure 9: The Flowchart diagram of the translator

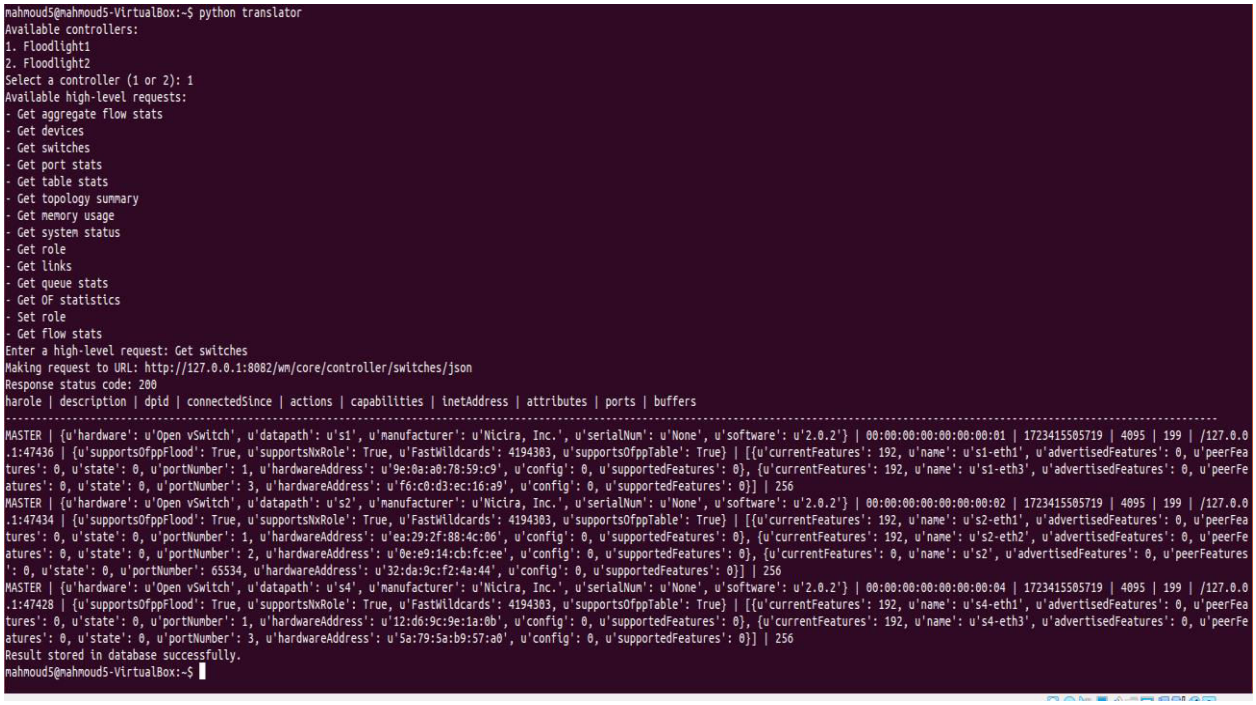


Figure 10: Example of execution of a monitoring request (Get switches) using the proposed translator

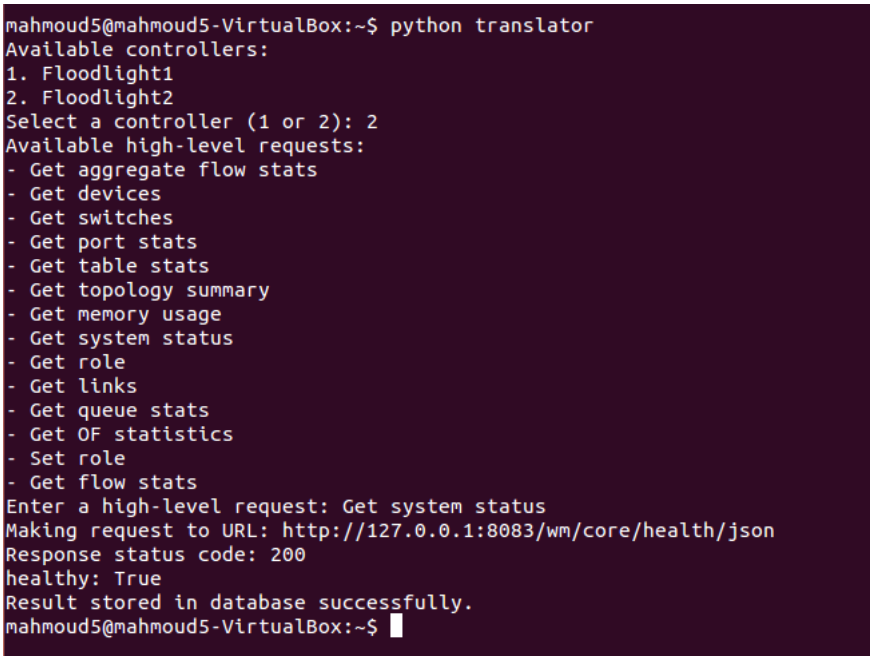


Figure 11: Example of Testing the status of the framework during running state using the proposed translator.

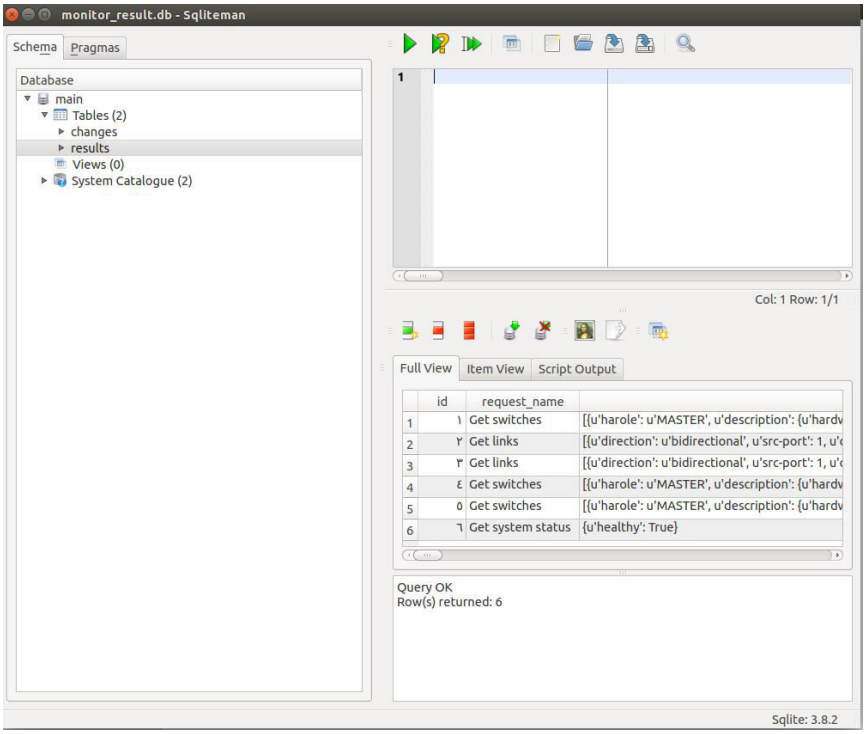


Figure 12: Shows the monitoring information stored in monitoring framework DB

The translator is implemented in a python language that runs over the SDN network and takes care of requests as presented in figures 10, and 11.

The results of the requests conducted by the translator are stored in the database of our MCMF as shown in figure 12

4.5.3-The Result Collector:

This element used to store either the results from the translator or the data that being synchronized from the synchronization element that will being explained after this one. The result collector is implemented as a python code that create the Database of our MCMF framework. The algorithm of the result collector is represented as a Flowchart in figure 13.

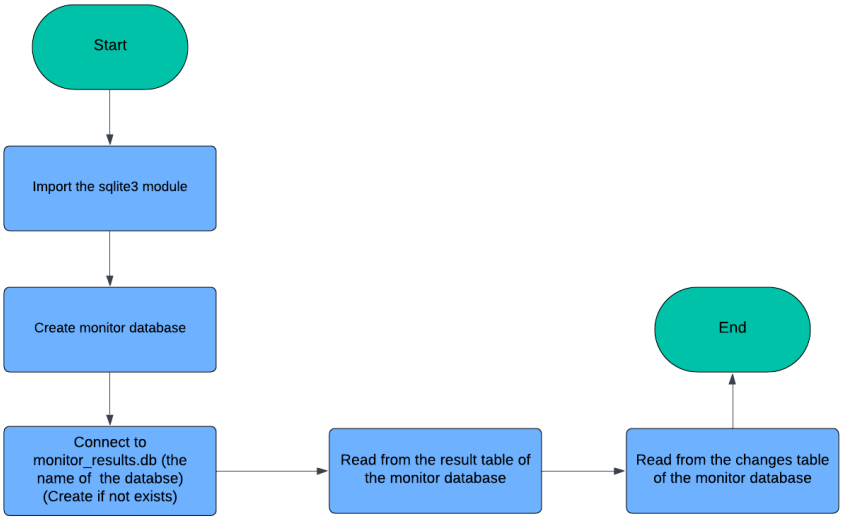


Figure 13: The Flowchart diagram of the result collector

The shown in figure 13, the results are stored in the database. These results will be used also by the synchronization element to take an action if needed.

5-Validating SDN reliability using the synchronization element:

The synchronization element is a python code that runs over the network to make synchronization between the two floodlight controllers for detecting any change of the two slices and enhancing the reliability of the SDN network. For example, if there is a failure in any floodlight controller it will be detected by the synchronization element. After detecting the failure in the floodlight, the synchronization element is taking an action for failover.

For validating the reliability, the floodlight1 is down by a command. The synchronization element detects that the floodlight1 is down then takes an action of failover by making the flooldight2 not to manage only its slice but to manage the floodlight1's slice. This means the flooldight2 manages the entire network topology by reassigning the slice of the floodlight1 to the floodight2. The floodlight2 is now the only active controller, which manages the entire topology based on the action of the synchronization element. All actions are stored in the MCMF Database. By detecting the down (faulty) floodlight and failover technique introduced by the synchronization element, the reliability of monitoring and SDN is enhanced. The algorithm of the synchronization element is represented as a Flowchart as shown in figure 14.

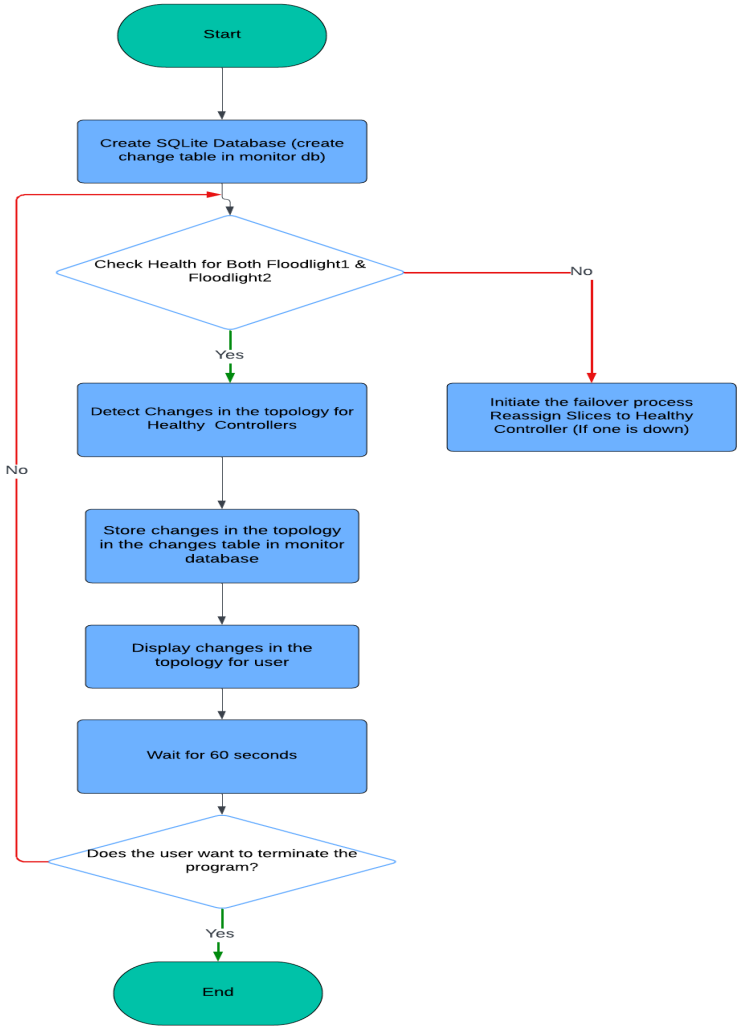


Figure 14: The Flowchart Diagram of the synchronization element

6-Evaluation of the proposed multi-controller-based monitoring framework (MCMF)

1-MCMF reliability

In our case study we introduce the MCMF ability of failover when one of the controllers is down. We tested the failover in our case study by using a command to makefloodlight1 down shown in figure 15. There are twooptions for checking the controller's health. In Option1, a request is sent to the translator element to check the controller's health . If one of the controllers is down, the translator will trigger the synchronization element to take action. The synchronization element makes the failover process that will reassign the slice of the down controller to the active one. The active controller can take over the two slices until the down controller is back to health status.
In option2: the synchronization element is synchronizing between the two controllers and the synchronized data between them is stored in the monitor database. The synchronization runs regularly until abnormal change happens in the system. The synchronization element can detect that change and then take an action. For example, in the same case study the floodlight1 is down and the synchronization element will take care of this change and make the active controller controls of the down controller slice.

```
mahmoud5@mahmoud5-VirtualBox:~$ sudo iptables -A INPUT -p tcp --dport 8082 -j DR
OP
```

Figure 15: The command that makes controller one is down for our case study

After the failover process, we tested the connection between all hosts to make sure that the two slices are controlled by the floodlight2. As we can see in figure 16, any ping message from any host reaches to all other hosts. This means that the two slices are controlled by floodlight2 with faulty floodlight1.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Figure16. Ping test after failover

Figure 17a illustrates the screen of information about the physical topology controlled by floodlight 2 after failover process. Figure 17b illustrates the physical topology controlled by floodlight2 after the failover process. The topology consists of the two slices as shown in the figure 17b.

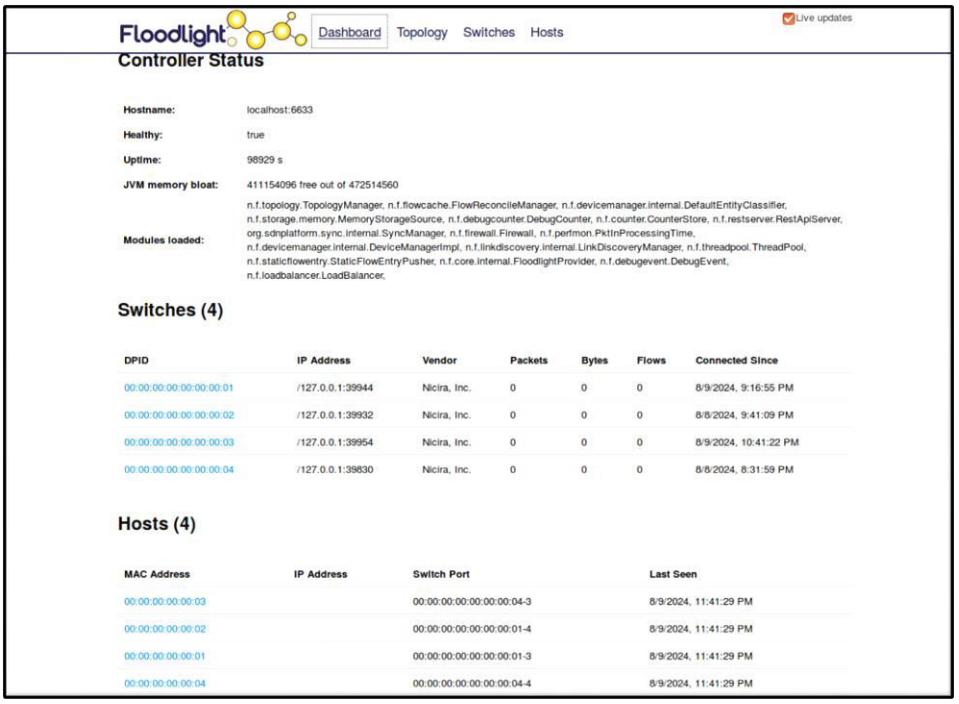


Figure17a. Failover active controller gui

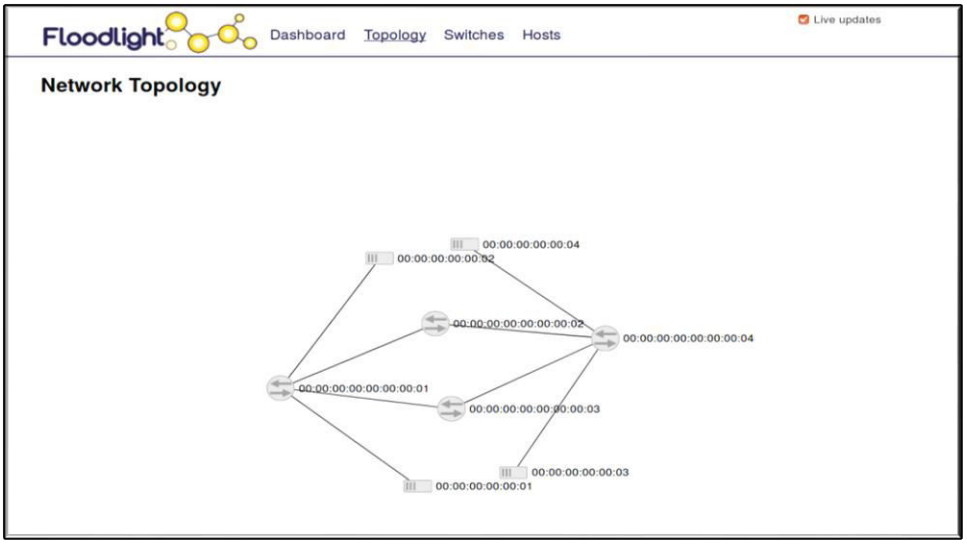


Figure17b. Topology of active controller after failover

EVALUATION AND COMPARATIVE STUDY

We have implemented and tested the proposedhigh performance monitoring system that run on SDN multi-controller, which can retrieve measurement data from the simulated data plane based on a network application request. The monitor can receive a request from a network application to collect measurement data, which may differ from one application to another.

We have developed and rigorously tested the Multi-Controller Monitoring framework (MCMF), a high-performance solution designed to operate across SDN multi-controllers. The MCMF framework excels in retrieving measurement data from the simulated data plane based on specific network application requests, which may vary depending on the application's requirements. This framework is engineered for large-scale network management, with capabilities to divide the network into multiple slices, each managed by a synchronized instance of the MCMF framework. This synchronization module ensures seamless information exchange between instances, thereby facilitating efficient monitoring of multitenant network applications. The MCMF framework demonstrates superior throughput and lower latency compared to traditional centralized monitoring frameworks that lack slicing. However, the current implementation is limited to operating on the Floodlight SDN multi-controller, with plans to extend support to other SDN multi-controllers in the future.

The Multi-Controller Monitoring (MCM) framework is designed to be versatile and adaptable, enabling it to function seamlessly with various vendors of data plane devices, including switches and routers. As long as these devices support the OpenFlow protocol, the MCM framework can effectively integrate with them. This compatibility ensures that network administrators can leverage the advantages of the MCM framework across a wide range of networking equipment without being limited to specific vendor solutions. By utilizing the OpenFlow protocol, the MCM framework can facilitate consistent and efficient monitoring and management of network resources, thereby enhancing the overall performance and scalability of Software-Defined Networking (SDN) environments, regardless of the underlying hardware.

In a comparative study, we evaluated the MCMF framework against the Two-Layer Trusted Multi Controller Architecture, which is grounded in a multi-chain approach to enhance scalability and reliability in SDN environments. While our MCMF framework emphasizes synchronization and high-performance monitoring across network slices, the Two-Layer Trusted Architecture focuses on decentralization, decision-making credibility, and the use of CFT and BFT blockchains to ensure reliable multi-controller coordination. By examining the key aspects and performance outcomes of each system, we highlight the unique strengths and methodologies of the MCMF framework in contrast to the sophisticated multi-chain mechanisms employed in the Two-Layer Trusted Multi-Controller Architecture.

Table 1: a comparative study between our proposed monitoring solution and Two-Layer Trusted Multi-Controller Architecture[7].

Aspect	MCMF framework	Two-Layer Trusted Multi-Controller Architecture
Purpose and Design Focus	Manages and synchronizes two SDN controllers for network monitoring, failover, and redundancy	Ensures scalability and reliability in SDN multi-controller environments using a multi-chain approach.
Scalability Mechanisms	Achieves scalability through synchronization of two controllers with failover capability	Utilizes Initial Load Balancing Mechanism (ILBM) and Fine-Grained Dynamic Load Balancing for scalability.
Reliability and Exception Handling	Provides reliability through a failover mechanism where one controller can take over if the other fails.	Uses a sophisticated reliability mechanism with exception detection based on switch reporting (EDSR).
Decentralization and Decision-Making	Operates in a semi-centralized manner, focusing on redundancy between two controllers	Emphasizes decentralization with CFT and BFT blockchains for distributed coordination and decision-making.
Performance Guarantees	Ensures performance by maintaining network stability through dual-controller management.	Focuses on high performance through dynamic load balancing and multi-chain architecture.
Simulation and Results	Design suggests effective management of failover situations, though specific simulation results are not provided.	Simulation results demonstrate prevention of controller overload and high performance with credibility and decentralization.

CONCLUSION

In this paper, a reliable and high-performance monitoring framework for Multi-controller SDN has been proposed. It runs on SDN multi-controller, where a copy of the monitoring system runs on a copy of the multi-controller. Each copy of the monitoring system can exchange the measurement data with its neighbor copies using the synchronization module in each copy. The synchronization element of the proposed monitor detects the failure in any controller and failover the failure by assigning the slices of the faulty floodlight to another health floodlight controller. The proposed framework manages and monitors large scale SDN networks, where these networks can be sliced to two or more slices. Each slice is monitored by a copy of the monitoring system and controlled by an SDN controller. The proposed framework monitors a large number of network applications, where each application is monitored by a copy of the monitoring framework and controlled by SDN controller.

The monitoring framework is also a vendor-independent, capable of retrieving measurement data from any infrastructure device, including switches, routers, links, and others. This is because the monitoring framework uses the OpenFlow protocol for retrieving measurement data, which is infrastructure independent. The high performance of multi-controller-based monitoring framework has been achieved by slicing the topology of SDN to slices using flowvisor tool. Each slice has been controlled by an instance of floodlight controller and monitored by a copy of the proposed monitoring framework. The different network applications run in parallel on multi-slices controlled by multi-controller and monitored by different copies of the proposed monitoring framework. Also, the size of the network can be increased without sacrificing the performance by slicing the networks to slices monitored by the proposed monitoring framework.

The scalable monitoring framework works only with Floodlight SDN multi-controller.

In the future the proposed monitoring framework can be built using large language models (LLMs). Also in future work, we aim to explore a more adaptive approach by investigating various SDN controllers beyond Floodlight, which will enhance the versatility of our research. Additionally, integrating machine learning techniques for predictive analytics will enable more intelligent network management, allowing for proactive adjustments based on traffic patterns and potential network disruptions. This direction not only aims to improve the overall performance and reliability of SDN implementations but also addresses the evolving demands of dynamic network environments. By focusing on these enhancements, we anticipate significant advancements in the capabilities of SDN architectures, leading to more resilient and efficient networking solutions.

REFERENCES

[1] J. Suarez-Varela, P. Barlet-Ros, “Flow monitoring in Software-Defined Networks,” Computer Networks, vol. , no. , pp. , 2018, doi:10.1016/j.comnet.2018.02.020.

[2] A. J. Aparcana-Tasayco, F. Mendoza-Cardenas, D. Diaz-Ataucuri, “Open and interactive NMS for network monitoring in Software Defined Networks,” 2022 International Conference, IEEE, vol. , no. , pp. , 2022.

[3] N. F. Mir, Computer and Communication Networks, 2nd ed. 2015, Prentice Hall.

[4] C. Lee, C. Yoon, S. Shin, S. K. Cha, “INDAGO: A new framework for detecting malicious SDN applications,” 2018 IEEE 26th Int. Conf. Netw. Protoc., pp. 220–230, 2018, doi:10.1109/ICNP.2018.00031.

[5] M. Scarlato, “Network monitoring in Software Defined Network,” Master on New Technologies in Computer Science, Itinerary Networks and Telematics, UniversitàdegliStudi di Cagliari, Cagliari, Sardinia, Italy, 2014.

[6] Y. Liu, B. Zhao, P. Zhao, P. Fan, H. Liu, “A survey: Typical security issues of Software-Defined Networking,” China Communications, vol. 16, no. 7, pp. 13–31, 2019, doi: 10.23919/j.cc.2019.07.002.

[7]M. N. Yusuf, K. B. A. Bakar, B. Isyaku, B. Mukhlif, F. Fadhil, "Distributed controller placement in Software-Defined Networks with consistency and interoperability problems," *Journal of Electrical and Computer Engineering*, vol. 2023, article ID 6466996, 33 pages, 2023. doi: 10.1155/2023/6466996.

[8] X. Niu, J. Guan, X. Gao, S. Jiang, “Scalable and reliable SDN multi-controller system based on trusted multi-chain,” 2022 IEEE 33rd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), Kyoto, Japan, 2022, pp. 758–763, doi: 10.1109/PIMRC54779.2022.9977820.

[9] J. Zhang, Q. Chen, X. Liu, “Multi-controller based Software-Defined Networking: A survey,” IEEE Communications Surveys & Tutorials, 2023, doi: 10.1109/COMSUR.2023.3302194.

[10] G. Tangari, “Accurate and resource-efficient monitoring for future networks,” Doctoral thesis (Ph.D.), UCL (University College London), 2019.

[11] M. Alsaeedi, M. M. Mohamad, A. A. Al-Roubaiey, “Toward adaptive and scalable OpenFlow-SDN flow control: A survey,” IEEE Access, vol. 7, pp. 107346–107379, 2019.

[12]Mahmoud Eissa, Ahmed Yahya, Usama Gad, “A Scalable Monitoring System for Software Defined Networks,” International Journal of Distributed and Parallel Systems (IJDPS), vol. 15, no. 1, pp. 1–14, 2024, doi: 10.5121/ijdps.2024.15101.

[13] Hasan ÖZER, İbrahim Taner OKUMUŞ “A Scalable and Efficient Port-Based Adaptive Resource Monitoring Approach in Software Defined Networks” The Journal of Graduate School of Natural and Applied Sciences of Mehmet Akif Ersoy University 13(1): 9-26 (2022).
[14] M. T. Kurniawan, M. Fathinuddin, H. A. Widiyanti, G. R. Simanjuntak, “Network slicing on SDN using FlowVisor and POX controller to traffic isolation enforcement,” 2021 International Conference on Engineering and Emerging Technologies (ICEET), Istanbul, Turkey, 2021, pp. 1–6, doi: 10.1109/ICEET53442.2021.9659765.
[15] K. Patel, J. Chaudhari, H. Mewada, H. Jayswal, R. Patel, D. Kirange, “Shortest path forwarding in Software-Defined Networks using RYU controller,” International Journal of Electrical and Electronics Engineering, vol. 11, pp. 299–305, 2024, doi: 10.14445/23488379/IJEEE-V11I5P127.
[16] S. Suraya, M. Sholeh, “Designing and implementing a database for thesis data management by using the Python Flask framework,” International Journal of Engineering, Science and Information Technology (IJESTY), vol. 2, no. 1, pp. 1–10, 2022.