A SCALABLE MONITORING SYSTEM FOR SOFTWARE DEFINED NETWORKS

Mahmoud Eissa, Ahmed Yahya, Usama Gad

Department of Electrical Engineering, Al-Azhar University,
Nasr City, Cairo-11371, Egypt.

ABSTRACT

Monitoring functionality is an essential element of any network system. Traditional monitoring solutions are mostly used for manual and infrequent network management tasks. Software-defined networks (SDN) have emerged with enabled automatic and frequent network reconfigurations. In this paper, a scalable monitoring system for SDN is introduced. The proposed system monitors small, medium, and large-scale SDN. Multiple instances of the proposed monitoring system can run in parallel for monitoring many SDN slices. The introduced monitoring system receives requests from network management applications, collects considerable amounts of measurement data, processes them, and returns the resulting knowledge to the network management applications. The proposed monitoring system slices the network (switches and links) into multiple slices. The introduced monitoring system concurrently monitors applications for various tenants, with each tenant's application running on a dedicated network slice. Each slice is monitored by a separate copy of the proposed monitoring system. These copies operate in parallel and are synchronized. The scalability of the monitoring system is achieved by enhancing the performance of SDN. In this context, scalability is addressed by increasing the number of tenant applications and expanding the size of the physical network without compromising SDN performance.

KEYWORDS

Software Defined Network; monitoring system; monitoring system architecture; network slicing; measurements data; network management applications.

1. INTRODUCTION

One of the functions of a Network Monitoring System (NMS) is to monitor an internal network for different types of issues or for managing the network. The NMS has the capability to identify and assist in resolving various issues, including slow web page downloads, lost emails, suspicious user activity, insecure network connections, and server crashes.[1].

In contrast to Intrusion Detection Systems (IDSs) or Intrusion Prevention Systems (IPSs), which primarily focus on detecting and preventing intrusions, Network Monitoring Systems (NMSs) are utilized to evaluate the network's performance during regular operations and gather measurement data for network management applications, including IDSs. To effectively execute traffic monitoring tasks, the NMS should have the capability to monitor various devices from different vendors, ranging from mobile phones to hosts, servers, routers, and switches [2].

The SDN paradigm aims to enhance flexibility by separating the control and data planes of conventional networks. The data plane is situated in forwarding devices enabled with SDN, such as SDN switches, while the control plane is logically centralized in new entities referred to as SDN controllers. This separation of planes in SDN facilitates the establishment of distinct layers

DOI: 10.5121/ijdps.2024.15101

of abstraction, enabling a high level of flexibility in network management. Figure 1 illustrates the architecture of an SDN-based network, showcasing the various entities, planes, and layers of abstraction involved.[1], [3].

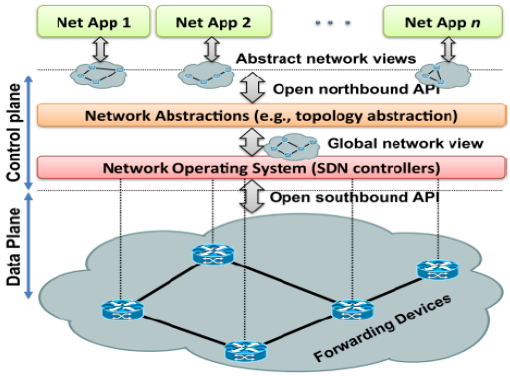


Figure 1. Software-Defined Networking architecture [1].

Application plane: SDN applications are granted resources by the SDN controller via Application Programming Interfaces (APIs). These applications encompass a range of functionalities, business applications, network management, and security management. They can be developed to identify issues, malicious activities, anomalies, or authors. [1].

Control plane: This plane is responsible for assessing the local status of the forwarding devices within the network and enforcing appropriate policies for the network's optimal functioning. Such policies may include routing, traffic engineering, or security policy enforcement. Unlike traditional networks, SDN employs flow-based forwarding decisions instead of destination-based forwarding decisions. All packets that meet a specific criterion are handled with the same actions. The control plane takes charge of determining the path of traffic and managing all network signaling activities to ensure that the network devices are properly configured [1][4].

Data plane: This plane performs traffic forwarding in network devices like switches based on the policies established by the control plane. This includes traffic filtration and various actions that can be carried out on incoming packets in a switch. It is noteworthy that in SDN, switches are of a general-purpose nature, implying that they implement flow-level policies established by the control plane and can combine actions that were previously specific to different types of network devices, such as routers and switches, in conventional networks. This plane is also referred to as the "forwarding plane."[5].

There are numerous interfaces that exist between these layers, namely the northbound and southbound interfaces. The northbound interface facilitates communication between the controller and application layers, while the southbound interface enables communication between the controller and the data forwarding layer. [6]

The SDN model is founded upon "open interfaces." The controller provides an API accessible to control applications, enabling them to verify and change the network's status. Additionally, new functionalities can be incorporated by developing new applications. The Controller uses open protocols to program network nodes, such as switches and routers, in accordance with application criteria. The most widely recognized protocol is OpenFlow, which can be deemed a standard protocol.

SDN monitoring systems should possess the ability to gather vast quantities of diverse measurement data, process them in real-time, and promptly supply the resulting knowledge to network decision-making procedures. Meeting the demands of SDN monitoring systems is a substantial challenge due to the current networks vast, dynamic traffic volumes, size, and the stringent constraints on time and hardware resources, these challenges will be tackled in this paper.

It introduces and develops a scalable monitoring system that is capable of monitoring small, medium and large size networks. This system extends its monitoring capabilities to an increased number of tenant applications, encompassing management applications. It 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 rest of this paper is organized as follows: Section 2 shows related work to this paper while Section 3 discusses the proposed architecture and solution. The implementation and validation of the proposed system are discussed in section 4. Proposed experiments for enhancing the scalability of monitoring system are discussed in section 5 while evaluations of the proposed architecture and solution are discussed in section 6. Section 7 concludes the paper.

2. RELATED WORK

Several studies have been done to monitor Software Defined Networking. However, there are still weaknesses in these approaches. The following section critically reviews existing research related to this paper.

J. Suariz proposed a flow monitoring solution for OpenFlow Software-Defined Networks that generates reports with flow-level measurements like those provided by NetFlow. To minimize the overhead on the controller and reduce the number of flow entries needed in the switch, two traffic sampling methods can be implemented in current SDN switches without any modifications to the OpenFlow specifications. These methods were then implemented in the OpenDaylight controller and their accuracy and overhead were assessed in a testbed using real-world traffic traces. [1].

B. Isyaku, M. Zahid, M. Kamat, K. A. Bakar, and F. Ghaleb presented cutting-edge research offering various solutions to address SDN performance issues stemming from the limited flow table. To overcome this, fuzzy theory and machine learning techniques can be leveraged to identify frequent flow entries that should be retained in the flow table. Consequently, this paper identifies several challenges, including update operations, resource constraints, communication overhead, and packet processing delays [3].

M. Scarlato, on the other hand, employed three tools for network monitoring in Software-Defined Networks to determine how these tools could be adapted to SDN. These tools include Ntop, Wireshark, and Argus. Their performance was evaluated, and the outcome revealed that Ntop and Argus did not behave as anticipated, as they were unable to recognize an OpenFlow traffic, despite their capability to identify NetFlow and sFlow traffic [5].

Although Wireshark has been widely utilized as a comprehensive tool, it possesses a plethora of filters specific to OpenFlow packets. Moreover, G. Tangary demonstrated that adaptive monitoring functions can be integrated into the packet-processing pipeline to maintain the accuracy of monitoring reports in the face of dynamic resource availability in the data plane [7].

M. Alsaeedi, M. Mohamad, and A. AL-Roubaiey have outlined four primary challenges in OpenFlow-SDN that contribute to traffic overhead. They also highlighted research challenges that must be addressed to develop more adaptive and scalable OpenFlow-SDN solutions under large-scale and dynamic network conditions. One of these challenges involves monitoring and classifying network flows at the application level [8].

3. PROPOSED MONITORING SYSTEM

In this section, the high level architecture, and detailed design of the proposed monitoring system are introduced.

3.1. High level architecture

The high-level architecture, as depicted in Figure (2), is arranged in a layered structure where the top layer contains management applications. Each management application has two primary elements: an information analyzer and a decision maker. The functionality of these two elements differs from one management application to another. For example, the functionality of the two elements of fault detection is different from that of anomaly detection and performance management. The information collected for each application is distinct from the others. The decision maker receives the results of the information analyzer and makes a decision that satisfy the objectives of the application. This decision should be sent to the routing element of the monitor.

The monitor software layer has an information collector, a routing element, and collected information tables. The information collector receives requests from management applications and collects the necessary measurement data. The data needed for each application differs, so the information collector can collect various types of data for each management application (calling application). All collected data is stored in the collected information tables and returned to the calling management application. The routing element receives a decision request from each application to call the southbound interface (API Function) for reconfiguring the infrastructure switches and links to satisfy the received decision. The detailed design of the information collector element is shown in the figure (3).

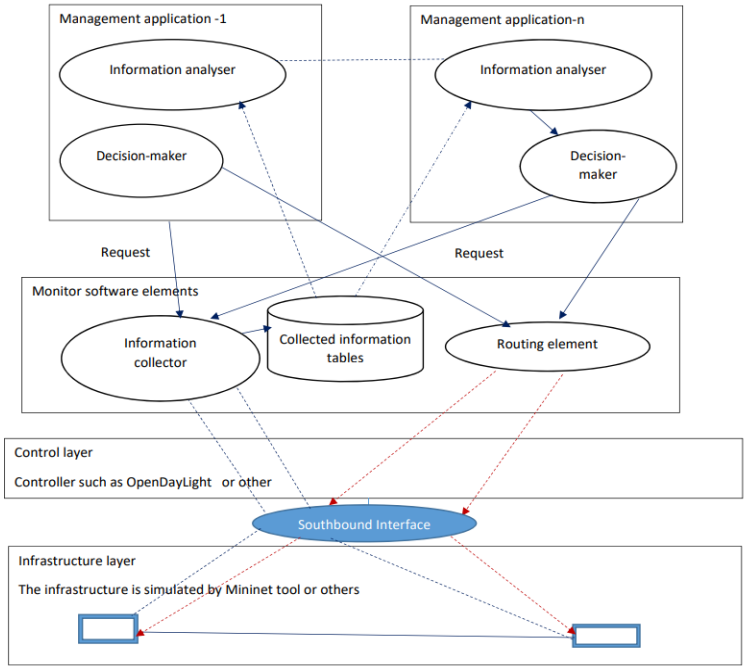


Figure 2. The proposed architecture of the monitor software system

The information collector retrieves information about the infrastructure's state by sending a call to the controller that invokes the southbound interface APIs and stores the retrieved information in the collected information tables. The information analyzer of the management application reads the collected information and analyzes it based on certain measurements. The analyzer's results are sent to the decision maker, which determines the new states of the infrastructure to solve problems such as enhancing performance. The routing element calls the controller, which in turn calls the Southbound interface APIs to implement changes in the flows. The infrastructure layer is simulated by a tool such as Mininet and controlled by a controller like Floodlight.

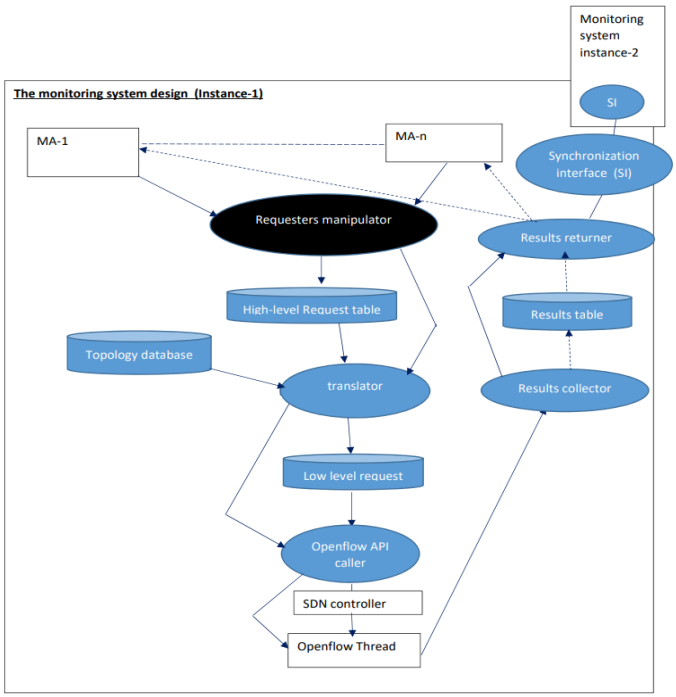


Figure 3: The proposed Architecture of monitor system design

The monitoring system consists of many modules: requester manipulator, translator, openflow API caller, results collector, and results returner. The requester manipulator receives requests from Management Applications (MAs) and stores them in the requests table. The translator module translates each high-level request using topology tables into a low-level request and stores it in the low-level request table. The OpenFlow API caller module sends the low-level request to the controller, creating an OpenFlow thread to execute the required operation function (request).

The measurement results produced by the OpenFlow threads are collected by the results collector module. The collected results are then stored in the results table. The results returner module associates the relevant high-level target and delivers (returns) each result to its MA request.

3.2. The Synchronization module (interface)

The synchronization interface is provided to synchronize monitoring information in a distributed and scalable data plane. It facilitates the exchange of monitoring data between different monitoring system instances.

3.3. The database schema of data plane topology

The topologies of data planes are represented as graphs. Each graph (data plane topology) is represented as a set of database tables. The schema of data plane topology is represented as follows:
Host (Host-ID, IPv4addres, IPv6address, Switch, Port)
Link (Direction, SourcePort, SourceSwitch, DestinationPort, DestinationSwitch, Type)
Switch (Switch-ID, IPv4address, Role)

3.4. High-Level- Requests Table schema

The requests from management applications are received by monitoring module and stored in the high-level requests table. The schema of the high level requests table is as follows:
High-level-requests (MA-id, Req-id, task, and HL- target, Intervals).
Each high level request in the schema has the ID of the management Application (MA-id) that sent the request followed by an identifier (Req-id). Each request has a task (needed measurements data function), followed by the high level name of the target (node, or link). Intervals field represents the frequencies of executing the request.

3.5. Low-level requests Table schema

This table schema is as follows:
Low-Level-requests (MA-id, Req-id, Op –code, and LL-target, Intervals).
The op-code field is the corresponding Openflow function of a task in High-Level- request.The Low-level-target (LL-target) field is the physical address (targets) corresponding to the High-Level -target in the High-Level -requests. Intervals field represents the frequencies of executing the request.

3.6. Results-table schema

This schema of this table is as follows:
MA-id, Req-id, measurments-data, and timestamp.
Where MA-id is the identifier of the calling management application.The algorithm of information collector module is shown as follows:

```
While there is a request from MAs // MA=Management Application
      Call request manipulator ()
       {
        Request manipulator ( )
           {
            Receive a request // high level request
            Store the request in HLR table //HLR=High Level Request
            Call translator ()
           }
         Translator ( )
           {
             While HLR table != empty
              { Read HLR // HLR= High Level Request
                 Search in topology database for mapping
                 Map HLR to LLR     //LLR=Low Level Request
                 Store LLR in LLR database
                Call openflowAPIcaller ( )
              }
           }
        openflowAPIcaller (
          {
             While LLR in low-level-request-table
            {
     execute openflow API equivalent to LLR

                 call results collector (
            }
          }
```

```
Results collector (
  {
    While (results
    {
        Receive results from openflow thread
    Store results in results table
Call results returner ()
                    }
                  }
      Results returner ( )
          {
           While (result table != empty
             {
Return results to its MA caller
        Call synchronization-interface ( results )
             }
           }
Synchronization-Interface ( information)
        {
   Exchange information with other monitors.
        }
```

4. IMPLEMENTATION AND VALIDATION

For implementing and testing the proposed monitoring architecture we used the following:

1. Python programming language

2. OpenFlow:

Originally, OpenFlow defined the communication protocol in SDN architectures that allowed the SDN controller to directly interact with network device forwarding planes, including physical and virtual (hypervisor-based) switches and routers. This enables the network to better adapt to changing business requirements [1].

3. Floodlight controller

The Floodlight controller is a widely used SDN controller written in Java and primarily developed by Big Switch Networks. It is responsible for controlling and managing the SDN network. The controller supports REST applications that can perform its functions. These REST applications utilize Floodlight's REST API to interact with the controller, allowing them to retrieve information from the controller and send network configurations to it. Additionally, the Floodlight controller comes equipped with a web-based GUI [9].

4. Mininet

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 [10].

4.1. Implementing a topology

To create the topology shown in Figure 4a, we implemented the Mininet emulator code in Figure 4b. The topology comprises four hosts, four switches, and eight links. We utilized the created topology to collect measurement data for validating the proposed monitoring system. Figure 5 illustrates the metadata of the hosts in the created topology, while Figure 6 shows the metadata of the topology links. Figure 7 represents the metadata of the topology switches.

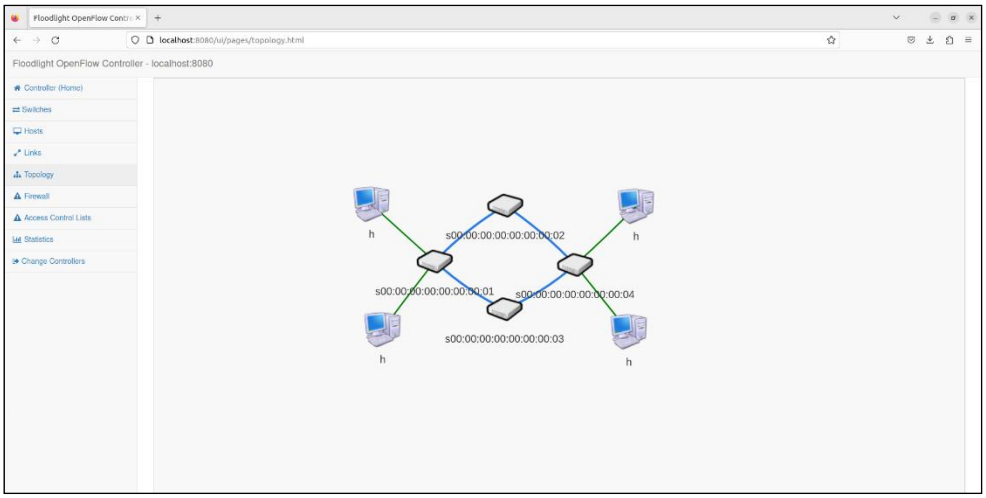
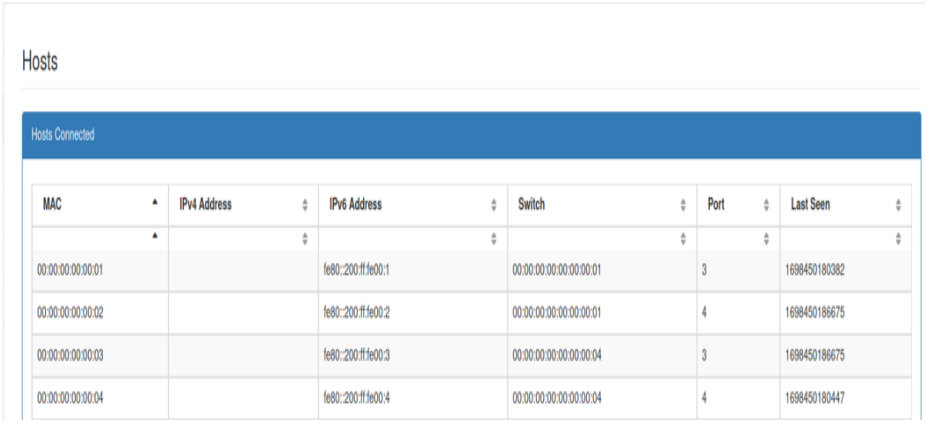


Figure 4.a: The topology graph

```
mahmoud@mahmoud:~/floodlight$ sudo mn --custom ~/onstutorial/flowvisor_scripts/flowvisor_topo.py --topo fvtopo --link tc --controller remote --mac --arp --switch ovsk,protocols=OpenFlow13
[sudo] password for mahmoud:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** 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) (10.00Mbit) (1.00Mbit) (10.00Mbit)
*** Starting CLI:
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>
```

Figure 4.b. An example for topology created using the Mininet

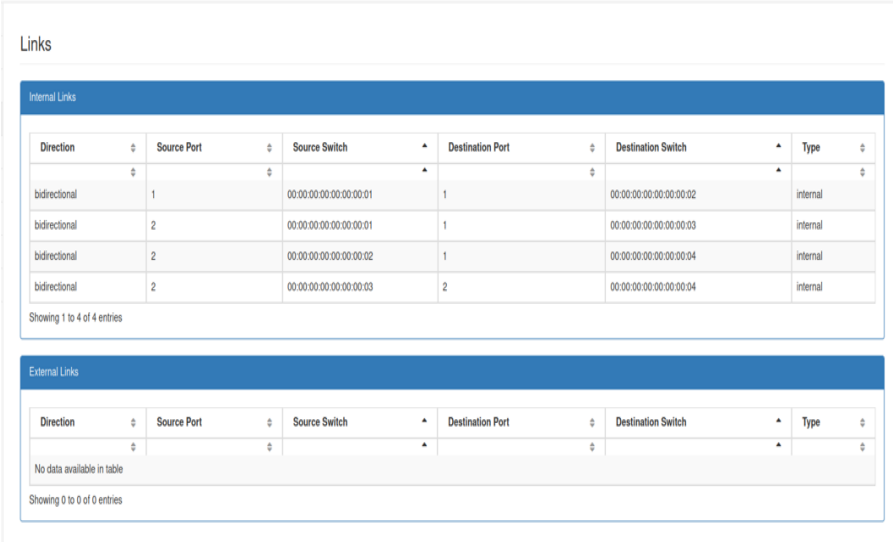


Hosts

Hosts Connected

MAC	IPv4 Address	IPv6 Address	Switch	Port	Last Seen
00:00:00:00:00:01		fe80::200:ff:fe00:1	00:00:00:00:00:00:00:01	3	1698450180382
00:00:00:00:00:02		fe80::200:ff:fe00:2	00:00:00:00:00:00:00:01	4	1698450186675
00:00:00:00:00:03		fe80::200:ff:fe00:3	00:00:00:00:00:00:00:04	3	1698450186675
00:00:00:00:00:04		fe80::200:ff:fe00:4	00:00:00:00:00:00:00:04	4	1698450180447

Figure 5. Metadata of the hosts of the created topology



Links

Internal Links

Direction	Source Port	Source Switch	Destination Port	Destination Switch	Type
bidirectional	1	00:00:00:00:00:00:00:01	1	00:00:00:00:00:00:00:02	internal
bidirectional	2	00:00:00:00:00:00:00:01	1	00:00:00:00:00:00:00:03	internal
bidirectional	2	00:00:00:00:00:00:00:02	1	00:00:00:00:00:00:00:04	internal
bidirectional	2	00:00:00:00:00:00:00:03	2	00:00:00:00:00:00:00:04	internal

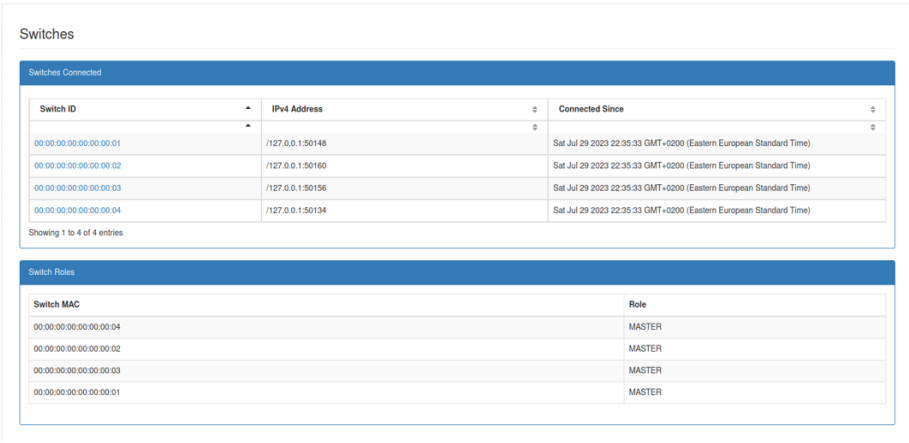
Showing 1 to 4 of 4 entries

External Links

Direction	Source Port	Source Switch	Destination Port	Destination Switch	Type
No data available in table					

Showing 0 to 0 of 0 entries

Figure 6. Metadata of the topology links.



Switches

Switches Connected

Switch ID	IPv4 Address	Connected Since
00:00:00:00:00:00:00:01	/127.0.0.1:50148	Sat Jul 29 2023 22:35:33 GMT+0200 (Eastern European Standard Time)
00:00:00:00:00:00:00:02	/127.0.0.1:50160	Sat Jul 29 2023 22:35:33 GMT+0200 (Eastern European Standard Time)
00:00:00:00:00:00:00:03	/127.0.0.1:50156	Sat Jul 29 2023 22:35:33 GMT+0200 (Eastern European Standard Time)
00:00:00:00:00:00:00:04	/127.0.0.1:50134	Sat Jul 29 2023 22:35:33 GMT+0200 (Eastern European Standard Time)

Showing 1 to 4 of 4 entries

Switch Roles

Switch MAC	Role
00:00:00:00:00:00:00:04	MASTER
00:00:00:00:00:00:00:02	MASTER
00:00:00:00:00:00:00:03	MASTER
00:00:00:00:00:00:00:01	MASTER

Figure 7. Metadata of the topology switches.

4.2. Test cases for validating the proposed monitoring system

In this section, two test cases as concrete examples of validating the monitoring system have been given.

4.3. Test case1 (normal case)

In this test case, as shown in Figure 8, all links are available. The proposed monitoring system retrieved measurement data of all links between the four switches.

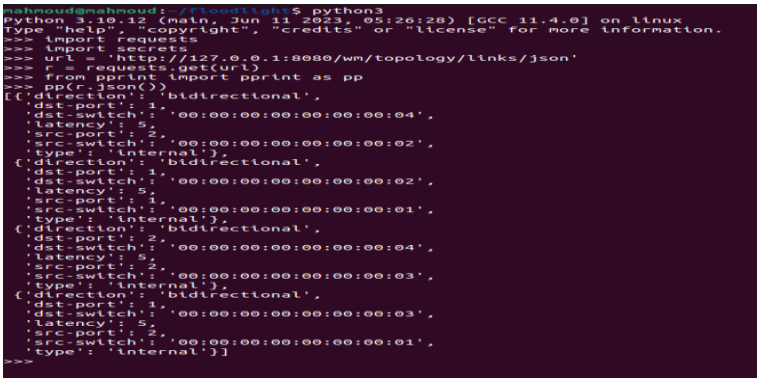


Figure 8: Retrieved measurement data by Proposed monitoring system in normal case

4.4. Test case 2 (error case)

In this test case as shown in figure 9, the link between switch1 and switch2 is down (not available) but other links are available. Therefore as shown in the Figure 9, the measurement data on the available links retrieved by the proposed monitoring system. We notice also there is no measurement data retrieved from the down link.

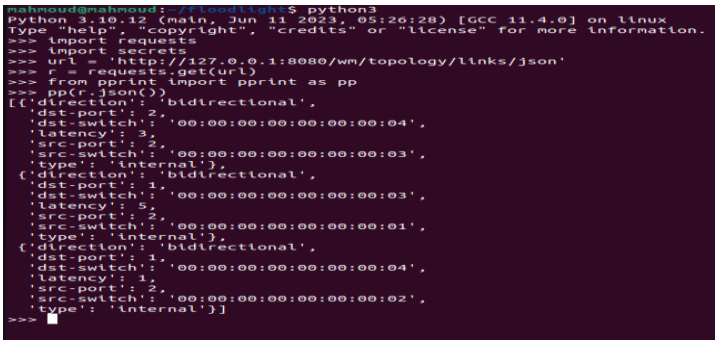


Figure 9. Retrieved measurement data by proposed monitoring system in in abnormal case (link between swtch1 and switch2 s down).

5. ENHANCING SCALABILITY OF THE PROPOSED SDN MONITORING SYSTEM

We address the scalability issue in different dimension enhancing the performance of SDN by increasing TCP throughput and reducing latency supporting a large number of tenant's applications.

The scalability of proposed monitoring system and SDN achieved by slicing the physical resources (data plane) to many slices. Each slice is controlled by a controller and monitored by a copy of the proposed monitoring system. Large number of tenants applications can run in parallel on different slices. Each application calls the proposed monitoring system that is installed on the target slice for collecting measurement data needed by the application.

In this paper to achieve the scalability by slicing the physical resources, the FlowVisor has been used. The flowvisor is a special OpenFlow controller that can slice the physical network into multiple (potentially overlapping) virtual slices.

5.1. Slicing experiment.

We used the miniNet emulator for creating the topology shown in Figure 10. The topology has 4 hosts, 4 switches and 8 links. By using the flowvisor, this topology sliced into two slices: slice-1 (upper-slice) and slice-2 (lower-slice). To use the FlowVisor to slice the topology in Figure 10 to two slices (upper and lower), we implemented the following code shown in Figure 11.

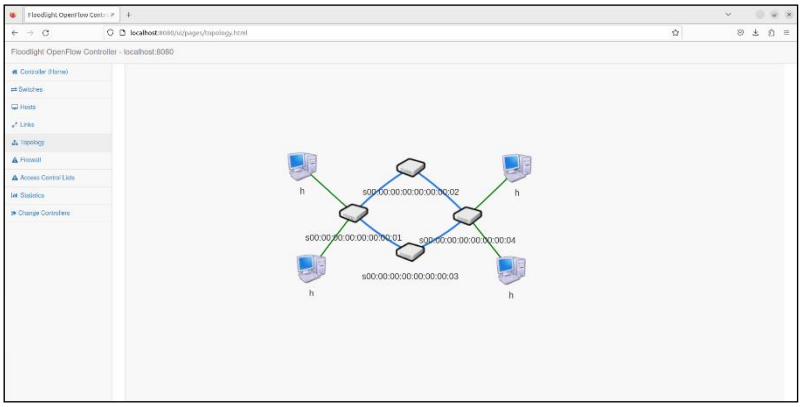


Figure 10. The non-sliced topology

Flowspace	Commands
Switch1	fvctl -f /dev/null add-flowspace dpid1-port1 1 1 in_port=1 upper=7 fvctl -f /dev/null add-flowspace dpid1-port3 1 1 in_port=3 upper=7 fvctl -f /dev/null add-flowspace dpid1-port2 1 1 in_port=2 lower=7 fvctl -f /dev/null add-flowspace dpid1-port4 1 1 in_port=4 lower=7
Switch2	fvctl -f /dev/null add-flowspace dpid2 2 1 any upper=7
Switch3	fvctl -f /dev/null add-flowspace dpid3 3 1 any lower=7
Switch4	fvctl –f /dev/null add-flowspace dpid4-port1 4 1 in_port=1 upper=7 fvctl -f /dev/null add-flowspace dpid4-port3 4 1 in_port=3 upper=7 fvctl -f /dev/null add-flowspace dpid4-port2 4 1 in_port=2 lower=7 fvctl –f /dev/null add-flowspace dpid4-port4 4 1 in_port=4 lower=7

Figure 11. The slicing code

The metadata and topologies of the two slices (slie-1 and slice-2) are shown in Figures 12.a & b, and Figures 13.a & b respectively. The slice-1 is controlled by a Floodlight controller and monitored by a copy of the proposed monitoring system. The slice-2 is controlled by another copy of Floodlight controller and monitored by another copy of the monitoring system.

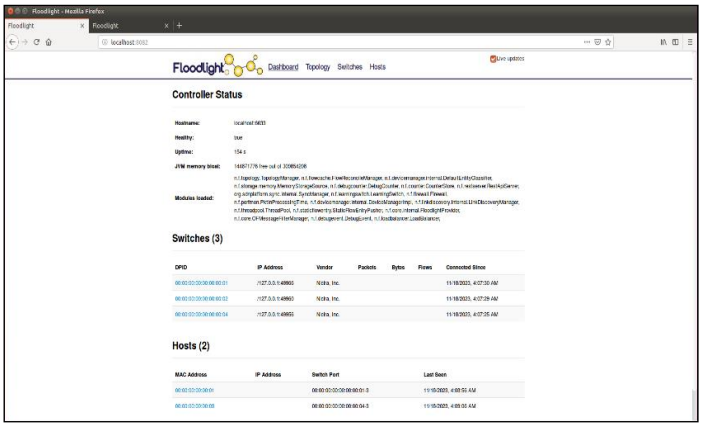


Figure (12.a). The slice-1(upper-slice) Metadata

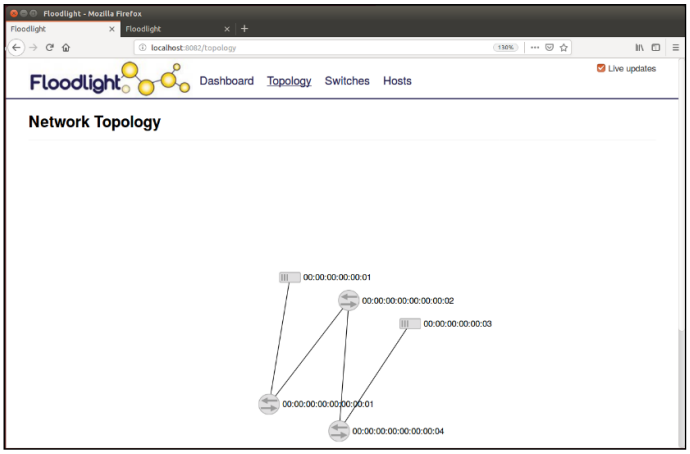


Figure (12. b). The Slice-1 (upper Slice) graph

As shown in Figures (12.a&b) the upper slice contains three switches and two hosts. The physical addresses of these switchesand hosts are shown in Figure (12. a).

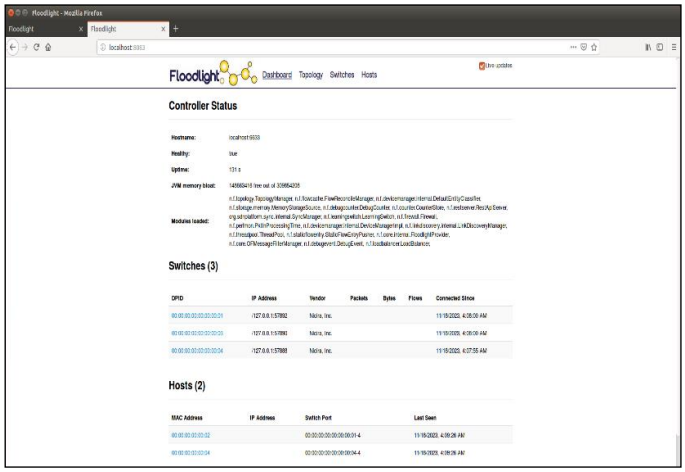


Figure (13.a) .The slice-2 (Lower Slice) metadata

As shown in Figures (13.a & b) the slice-2 contains three switches. The addresses of the switches are shown in the lower slice metadata (Figure 13.b).

5.2. Throughput and latency of non-sliced (scenario-1) and sliced (scenario-2) topologies

We have conducted two scenarios to compare between non-sliced topology performance and the two sliced topologies (higher and lower) performance. The performance metrics are TCP throughput and latency.

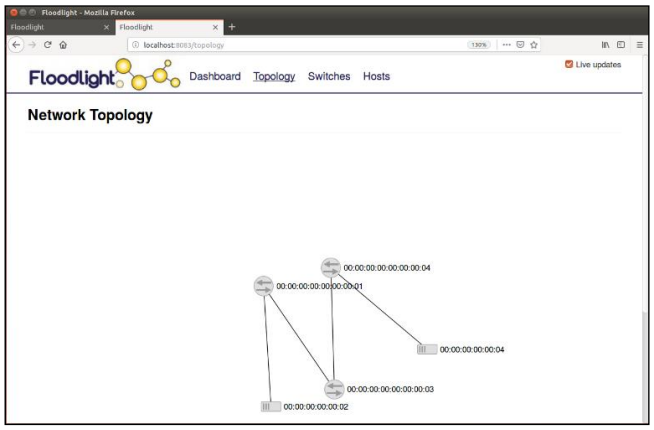


Figure (13.b). The slice-2 (Lower Slice) Graph

In the first scenario the non-slicing topology is managed by only one floodlight controller and monitored by one copy of the proposed monitoring system and many TCP-packets have been sent between switches.

As shown in Figure 14, the non-slice topology is controlled by a Floodlight controller and monitored by a copy of the proposed monitoring system that collects measurement data to tenant applications.

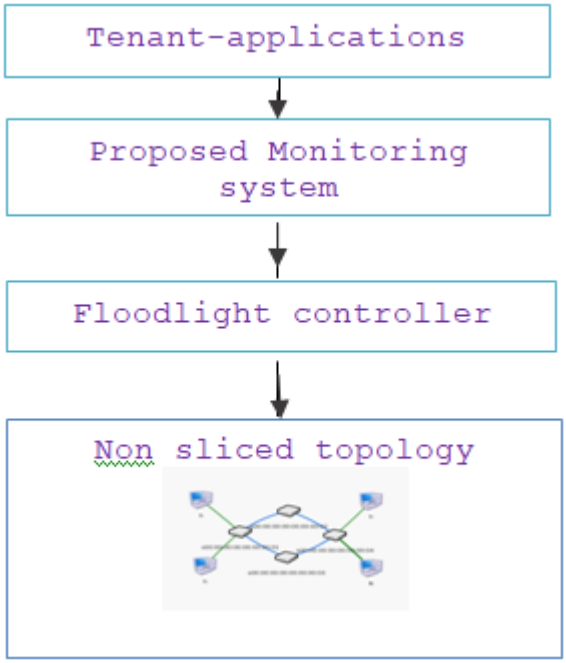


Figure 14.Tenant's applications run on the non-sliced topology

In the second scenario, two slices have been created and each slice is controlled by a separate floodlight controller and monitored by a copy of the proposed monitoring system. Each monitoring copy synchronizes with each other for data exchange

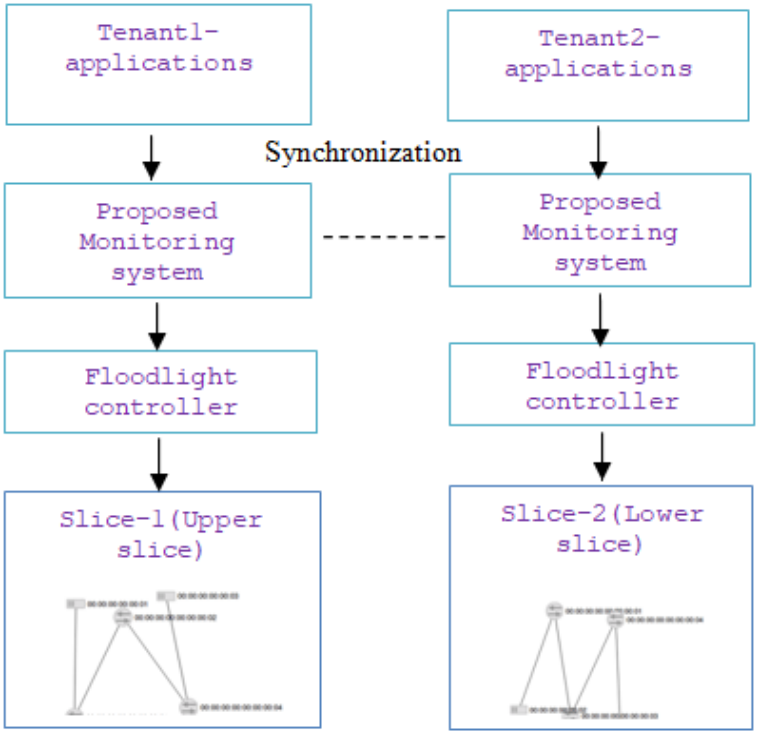


Figure 15. Tenant's applications run in parallel on the slices using different copies of the monitoring system.

As shown in Figure 15, each slice is controlled by a Floodlight controller and monitored by a copy of the proposed monitor where the two monitoring copies are synchronized. Each monitoring copy collects measurement data to tenant applications. These applications run on the two slices in parallel.

The performance of scenario-1 and scenario-2 has been measured by measuring the throughput and latency metrics. The results of measuring the throughput of each scenario are shown in Figure.16 (a,b,c,d) and results of measuring the latency of each scenario are shown in Figures 17 (a,b,c,d)

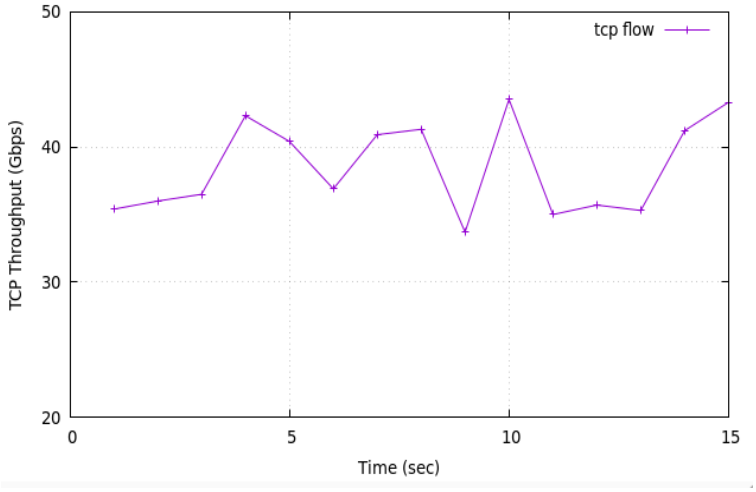


Figure.16.a: Switch2 to switch1 TCP throughput without slicing

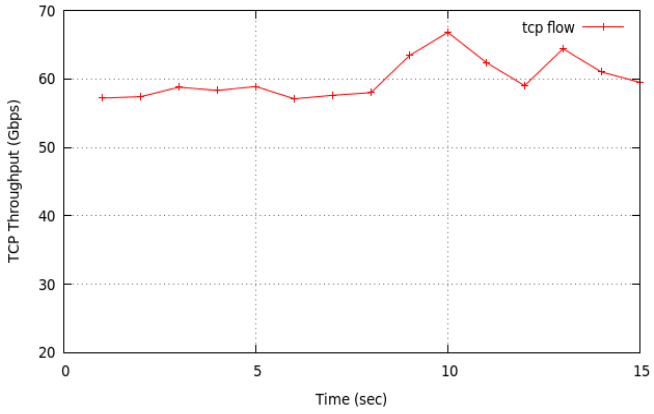


Figure.16.b: Switch2 to switch1 TCP throughput slice 1 (upper slice)

From Figure 16.a and Figure 16.b, we notice that the throughput of slice-1 (upper slice) is greater than the throughput of non-slicing topology. This is because the overhead of the slice controller is reduced due to the reduction of traffic collision. The same thing the throughput of slice2 (lower slice) is greater than the throughput of non-sliing topology as shown in Figures 16.c and 16.d

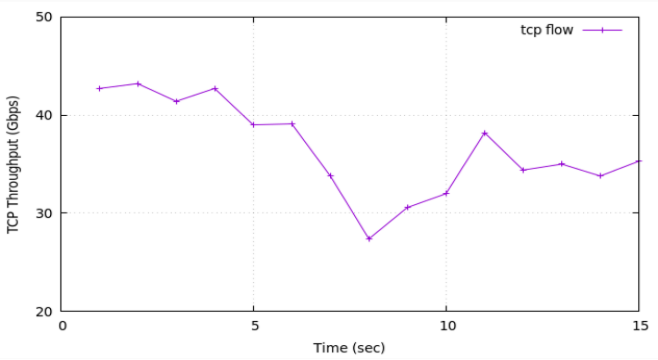


Figure.16.c: Switch3 to switch1 TCP throughput without slicing

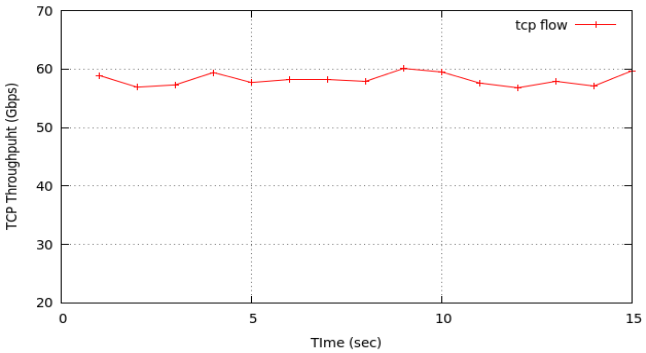


Figure.16.d: Switch3 to switch1 TCP throughput of slice2

In addition to increasing the throughput, more than one tenant applications run in parallel on the generated slices (slice-1 and slice-2 in scenario-2) as shown in Figure 15, which verifies multi-tenants applications.

The latency of scenario-1 (based non-sliced topology) and latency of scenario-2 (based on the two slices) have been measured. The results are shown in Figures 17 (a, b,c, and d).In the following figures, RTT stands for Round Trip Time.

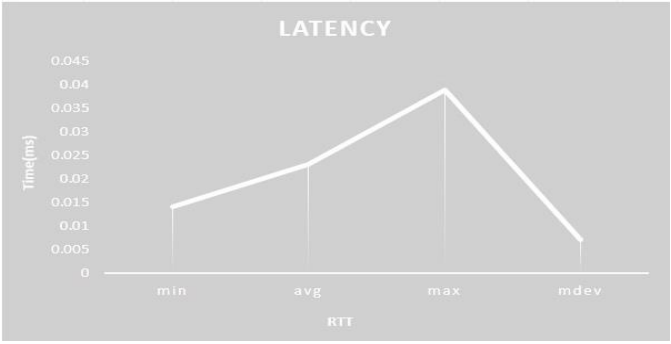


Figure 17.a Latency of s2 to s1 in non-sliced

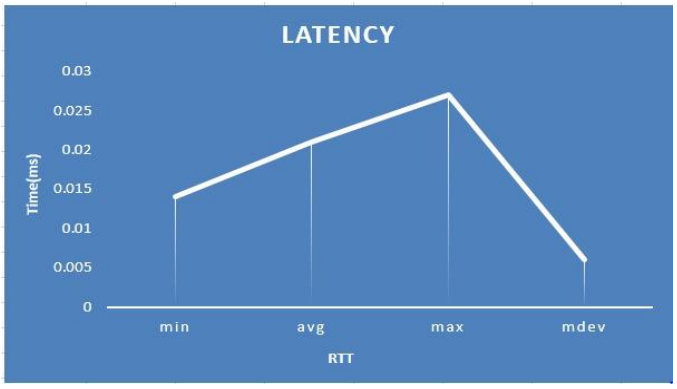


Figure 17.b Latency of s2 to s1 in slice-1

In Figure 17.a, and 17.b we can see that the latency between the two switches S2 and S1 in scenario-1 (non-sliced topology-based) is greater than the value of latency between the same two switches in slice-1 (scenario-2). The maximum value of latency in Scenario-1 (non-sliced topology-based) nearly equal 0.04ms but in slice-1 (scenario-2) is nearly equal 0.0275ms (i.e. less than 0.03ms). This means the latency (factor of performance) of the slice-1(scenario-1) is better than the latency of scenario-1(non-sliced topology-based).

Also, in Figures 17.c, and 17.d, we notice that the latency of slice 2 (scenario-2) between switch 3 and switch 1 is between 0.04ms and 0.045ms but the latency of scenario-1 (non-sliced topology-based) between switch 3 and switch 1 is nearly equals 0.07ms. This means that the latency of slice-2 (scenario-2) is better than the latency of scenario-1(non-sliced topology-based).

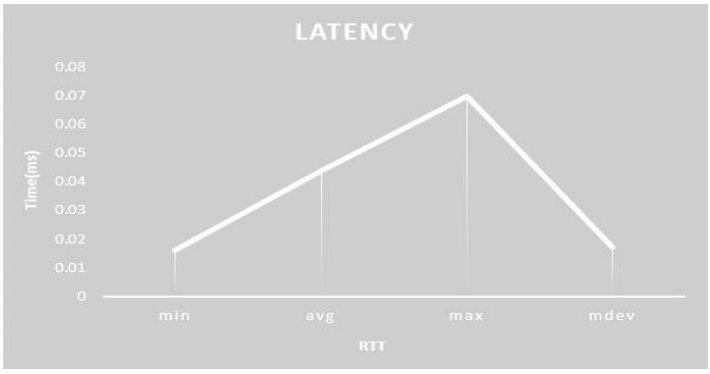


Figure 17.c Latency of s3 to s1 in non-sliced

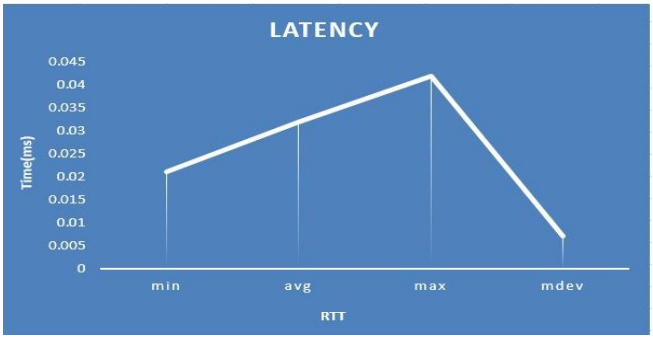


Figure 17.d Latency of s3 to s1 in slice-2

Since the throughput of each slice (scenario-2) is better than the throughput of scenario-1 (non-sliced topology-based) and the latency of each slice (scenario-2) also better than the latency of scenario-1(non-sliced topology –based), then the performance of scenario-2 (slices-based) is better than the scenario-1 (non-sliced network-based). Also the number of tenant's applications that run in parallel on the slices of scenario -2 are increased and monitored by copies of the synchronized proposed monitoring system.

From the above discussion, the proposed monitoring system is scalable in two dimensions (i) increasing the size of data-plane without sacrificing the performance. This is by slicing the data-plane where each slice monitored by a copy of the proposed monitoring system where all copies are synchronized together. (ii) Increasing the number of tenant applications that run in parallel and monitored by the proposed monitor. By experiments that have been conducted in this research we noticed that the throughput and latency of each slice is better than the non-sliced. Thus, in addition to the above two dimensions the performance (throughput and latency of SDN) of slicing scenario is better than non-sliced scenario.

6. PERFORMANCE EVALUATIONS

We have implemented and tested the proposed scalable monitoring system, which can retrieve measurement data from the simulated data plane based on the management application request. The monitor can receive a request from a management application to collect measurement data, which may differ from one application to another.

The proposed monitoring system is scalable, capable of managing large-scale networks by dividing them into multiple slices. Each slice is managed by a copy of the proposed monitoring system, and all copies are synchronized for exchanging information through the synchronization module of the monitoring system. Also, the proposed monitoring system monitors multitenant applications. The proposed scalable monitoring system has high throughput and low latency than the non-slicing based monitoring system. The limitation of the proposed monitoring system is working only on Floodlight SDN controller. In the future, it will be enhanced to work with other SDN controllers.

Based on the common criteria between our proposed monitoring system solution and other solutions we compared between our proposed scalable monitoring solution and Vivi Monita solution [11] in table -1. Also we compared our proposed monitoring solution with Tangari's Research in table-2.
In the first comparative study, we conducted a side-by-side examination of our research and the notable work conducted by Vivi Monita, as outlined in the 2023 paper titled "Network Slicing Using FlowVisor for Enforcement of Bandwidth Isolation in SDN Virtual Networks." [9]. Our study concentrates on creating a scalable monitoring system for Software-Defined Networks (SDNs), strategically employing FlowVisor to facilitate network slicing and enhance scalability without compromising performance. Conversely, Vivi Monita's research is centered on achieving tenant isolation and enforcing bandwidth isolation in SDN virtual networks through the adept utilization of FlowVisor. By closely analyzing the methodologies and outcomes of each study, we aim to detect the distinctive contributions and methodologies employed in the dynamic field of our proposed solution and Vivi Monita's Study [11]

Table 1: Acomparative study between our proposed monitoring solution and Vivi Monita solution

Aspect	Our proposed monitoring solution	Vivi Monita's Study [11]
Aims of Study	A scalable monitoring system for SDNs.	Ensuring tenant isolation in SDN using FlowVisor and an SDN controller.
Methodology	Utilized FlowVisor for effective network slicing	Two types of renters and testing procedures were used for connectivity and functionality.
Key findings	Scalable monitoring system for various SDN scales.	Correct host linkage achieved without turning on FlowVisor.
Multitenancy approach	Multiple copies run in parallel.	Hosts can only communicate within the same tenant.
Contribution	Addressing scalability concerns in SDNs.	Correct host linkage and tenant isolation through network slicing.

In the second comparison (Table 2), we compared our research on scalable SDN monitoring with Tangari's [7] work on resource-efficient monitoring for future networks. Our emphasis lies in scalability without compromising performance, while Tangari introduces innovative methodologies for accurate monitoring. We assess key aspects, including scalability models, monitoring frameworks, adaptive solutions, and performance metrics, aiming to highlight distinctive contributions and advancements in both studies, offering insights into the evolving landscape of SDN monitoring

Table 2: A comparative study between our proposed monitoring solution and **Tangari's** solution

Aspect	The proposed scalable monitoring system	Tangari's Research [7]
Objective	Scalable SDN monitoring system.	Accurate and resource-efficient monitoring for future networks.
Scalability	Increasing the number of tenant applications and the size of the physical network without sacrificing SDN performance.	Increasing network sizes, avoiding processing bottlenecks in large-scale networks.
Monitoring Framework	Slices the network using FlowVisor tool, synchronized copies using OpenFlow protocol.	Decentralized and self-adaptive monitoring framework (SAM) with modular architecture for diverse management applications.
Adaptive Solutions	Synchronized copies running in parallel, adaptive solution for efficient data extraction (SAM).	SAM for dynamic reconfiguration under dynamic traffic conditions, MONA for resilience to bottlenecks in software dataplanes.
Measurement Technique	Processing measurement data from different slices in parallel.	Classifiers used to reduce measurement data-processing costs for various monitoring queries.
Performance Metrics	Performing scalability without sacrificing SDN performance. High throughput and low latency	Enhanced monitoring accuracy levels for various measurement tasks.

7. CONCLUSION

In this paper, a scalable monitoring system for SDN is proposed.The scalability has been addressed in two dimensions without sacrificing the performance of SDN. The two dimensions are increasing the size of SDN and increasing the number of tenants and their applications.

The monitoring system is a vendor-independent, capable of retrieving measurement data from any infrastructure device, including switches, routers, links, and others. This is because the

monitoring system uses the OpenFlow protocol for retrieving measurement data, which is infrastructure-independent. The scalability has been achieved by slicing the topology of SDN to slices using flowvisor tool. Each slice has been controlled by a floodlight controller and monitored by a copy of the proposed monitoring system. MutliTenant applications run in parallel on multi-slices.

The proposed scalable monitoring system has been tested and evaluated. The results of evaluation show that the number of tenants' applications can be increased without sacrificing the performance. Also, the size of the network can be increased without sacrificing the performance by slicing the networks to slices monitored by the proposed monitoring system. The scalable monitoring system is working only with Floodlight SDN controller.In the future the proposed monitoring system will be enhanced to be adaptable (i.e. it works with more than one SDN controller).

Statements and Declarations

- Funding: no funding was received to assist with the preparation of this manuscript.
- Financial interests: The authors declare they have no financial interests.

The authors have no competing interests to declare that are relevant to the content of this article.

- Consent to participate: not applicable
- Conflict of interest: The authors declare they have no Conflict of interest.

REFERENCES

[1] Jose Su ´ arez-Varela, Pere Barlet-Ros, Flow monitoring in Software- ´Defined Networks, Computer Networks (2018), doi:10.1016/j.comnet.2018.02.020.

[2] Andres J. Aparcana-Tasayco ; Fredy Mendoza-Cardenas ; Daniel Diaz-Ataucuri "Open and Interactive NMS for Network Monitoring in Software Defined Networks" 2022 International Conference, Publisher: IEEE.

[3] Isyaku, B., Zahid, M.S., Kamat, M., Bakar, K.A., & Ghaleb, F.A. (2020). Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey. Future Internet, 12, 147.

[4] C. Lee, C. Yoon, S. Shin, and 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] Scarlato, Michele. (2014). Network Monitoring in Software Defined Network, Master on New Technologies in Computer Science. Itinerary Networks and Telematics, Università degli studi di Cagliari Cagliari, Sardinia, Italy

[6] Y. Liu, B. Zhao, P. Zhao, P. Fan, and H. Liu, "A survey: Typical security issues of software-defined networking," China Commun., vol. 16, no. 7, pp. 13–31, 2019, doi: 10.23919/j.cc.2019.07.002.

[7] Tangari, Gioacchino; (2019) Accurate and Resource-Efficient Monitoring for Future Networks. Doctoral thesis (Ph.D), UCL (University College London).

[8] Alsaeedi, M., Mohamad, M. M., & Al-Roubaiey, A. A. (2019). Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey. IEEE Access, 7, 107346–107379..

[9] Hasan ÖZER 1*, 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).

[10] http://pakiti.com/sdn-101-mininet-openflow-and-gns3/#:~:text=Mininet%20is%20a%20network%20emulator,routing%20and%20Software%2DDefined%20NetworkingM. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[11] Monita, V., Wendato, W.& Anggiratih, E. “Network Slicing Using FlowVisor for Enforcement of Bandwidth Isolation in SDN Virtual Networks “.Jurnal Ilmiah Teknik Elektro Komputer dan Informatika, Vol. 9, No. 3, September 2023.

AUTHORS

Mahmoud Eissa, a dedicated master's student in Electrical Engineering at Al Azhar University, Cairo, Egypt. He earned his Bachelor's from Future University, Egypt, with a 'good' grade in 2015 specializing in Electrical and Communication. His outstanding graduation project, "Performance Evaluation of Routing Protocols for Mobile Ad-hoc Network," received an excellent grade, showcasing his commitment to advancing the field.

Ahmed Yahya [ORCID: 0000-0002-3271-058X], is currently working as a Professor in the Department of Electrical Engineering, A-Azhar University, Cairo, Egypt. He obtained his Ph.D. from Ain Shams University, Cairo, Egypt. He has published many research papers, which contributed significantly to the development of his field. He had been working as As Professor in various reputed Institution’s in Abroad and Egypt. His research area focuses on Distributed Cloud Edge Computing, 5G Testbeds, LoRaWAN Networks, V2X Communications, Reconfigurable System-on-chip Design, Metamaterial-based Reconfigurable Antenna and Intelligent Cognitive Radio Networks.

Usama Gad, an Assistant Professor in the Department of Electrical Engineering, Electronics and Electrical Communications program, Faculty of Engineering, Al Azhar University, cairo, Egypt. He obtained his Ph.D. in Electronics and Communications in 2014, focusing on "Image Processing using Residue Number System."he earned an M.Sc. in Electronics and Communications (2008), exploring "Realization of digital filters using residue number system," and a B.Sc. with honors (2002) for his project on "Optical Voice Link."