Journal of Information Systems and Informatics
Vol. 6, No. 3, September 2024 e-ISSN: 2656-4882 p-ISSN: 2656-5935
DOI: 10.51519/journalisi.v6i3.818 Published By DRPM-UBD

Optimization of Network Performance in Complex Environments with Software Defined Networks

Munienge Mbodila[1], Omobayo Ayokunle Esan[2], Femi Abiodun Elegbeleye[3]

[1,2]Departement of Network and Support, Walter Sisulu University, East London, South Africa
[3]Departement Business Application and Developments, Walter Sisulu University, East London, South Africa
Email: [1]mmbodila@wsu.ac.za, [2]oesan@wsu.ac.za, [3]felegbeleye@wsu.ac.za

Abstract

Software-defined networks (SDN) have emerged as a promising approach to address the limitations of conventional networks. Its architecture can be implemented using either a single controller or multiple controllers. Although a single controller is inadequate for managing networks, deploying multiple controllers introduces the challenge of controller placement (CPP) in a network environment. To address these issues, this study presents a Software Defined Networks-Fault-Tolerant Method (SDN-FTM) where, in the event of a network failure, the SDN controller automatically reroutes traffic through an alternate, pre-configured network path, thereby maintaining uninterrupted service. The proposed SDN-FTM was tested and evaluated in real-time using Mininet simulation tools on a real-life small scale network data from tracking unit department in Walter Sisulu University (WSU), with a focus on performance measures such as latency and throughput. From the result obtained, the proposed method produced throughput and latency on Ryu with 2.15m/s and 18408m/s respectively. Furthermore, the findings indicate that Ryu controllers generally outperform OpenFlow controllers in terms of throughput, while OpenFlow controllers exhibit lower latency. The proposed method demonstrates significant improvements in network management by providing a robust solution for maintaining high network availability and performance in the presence of faults.

Keywords: Software-Defined Network, Controller, Latency, Throughput, Software-Defined Network-Fault Tolerance

1. INTRODUCTION

Today, many organizations, individuals, and ecosystem applications are required to have fast, reliable, and efficient networks that can handle voluminous traffic, which can be deployed to large dynamic applications and services [1]. Server virtualization networks and networked data centers are increasingly common. In modern network infrastructures, ensuring seamless connectivity and high performance in complex environments is critical, particularly when network faults

occur [2]. SDN has gained significant traction as an innovative paradigm [3]. Its architecture, which features a central controller that separates the control and data planes, uses high-level abstractions that are communicated to network devices through a southbound interface. However, employing a single controller in a complex, large-scale network can pose challenges due to the risk of a single point of network failure. Additionally, as more switches connect to the controller, its performance may degrade and impeding scalability [4].

As the network is anticipated to play a crucial role in the Internet of Things, the use of SDN seeks to enhance its usability and applicability [5]. However, employing a single controller in a large and complex SDN network can be challenging due to the risk of creating a single point of failure [5]. Furthermore, as the number of switches linked to a controller increases, the message volume may exceed the controller's capacity, hindering SDN scalability. Consequently, multiple controllers are deployed in SDN networks to address these issues. However, if more than one controller needs to be installed, the appropriate number of switches must be allocated to each controller because, upon a packet arriving at a switch's ingress port, the flow table is queried based on the information contained in the incoming packet header to find a matching flow entry [6].

The packet is routed to the SDN controller if no flow entry is found. Once the destination has been determined, the SDN controller adds a flow entry to the switch. Because of this, if the switches are not correctly assigned to the controller, the delay in this process increases [7] [8]. One important area of research in SDN is controller placement; potential solutions involve trying to balance load balancing and communication [9]. In [10], the authors started researching the issue of controller placement. The literature claims that a crucial problem in an SDN environment is controller placement. The placement of controllers has been examined by numerous researchers. The issue of controller load in SDN controller placement was investigated using a capacitated k-center in research by [11]. The use of the propagation latency of the k-center method to deploy a network controller to desired locations was introduced [12]. From a medium-sized network, the simulation's result indicates that the latency from each node to the controller of a single node meets the response time while propagation delay is not guaranteed by the approach.

To minimize propagation latency between the switches and the controller and improve the throughput of the network, this study employs a Software Defined Networks-Fault-Tolerant Method (SDN-FTM) on a real-life small-scale network data from the tracking unit department in Walter Sisulu University (WSU) where, in the event of a network failure, the SDN controller automatically reroutes traffic through an alternate, pre-configured network path, thereby maintaining uninterrupted service. Hence, this study aims to optimize network performance in

complex environments using software-defined networks [13] . Drawing upon the provided background information, this study raises the following inquiry: How can the network in a complex environment be sustained when one network is faulty? The proposed method is developed with the consideration of the above-mentioned gaps and the associated research question, which led to the following contributions,

1) a network partition concept is introduced to address the issue of controller placement and reduce the end-to-end latency.
2) to reduce controller queuing latency and improve the throughput in multi-controller placement.
3) Real network topologies from the Internet Topology are used in extensive simulations.

This paper proceeds as follows. In this section 2, the methodology is covered. The results are presented and evaluated in section 3. Section 4 presents the discussion. Section 5 concludes the paper.

2. METHODS

This section described the method applied to solve problems of single controller placement failure including procedures, and evaluation methods. To achieve the research objective stated in Section 1, the flow chart which depict the basic idea of the proposed work shown in Figure 1.

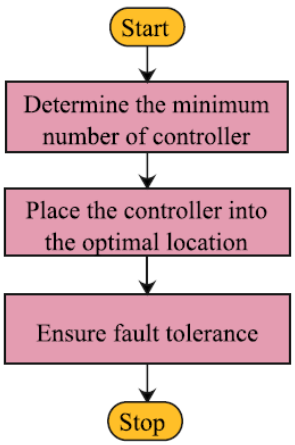


Figure 1. A basic Flowchart of the proposed SDN-FTM method adapted from [14]

Figure 1 shows all the stages that are used in the simulation of this research, these stages are further explained in detail in the following sections.

2.1 Stage 1: Determination of the Optimal Number of Controller

For the Controller Placement Problem (CPP) in SDN, the process for determining the number of controllers in the SDN Mininet, Linux deployment is outlined. The network requires a set of (k) controllers to be installed. The intention here is to select the best positions from a possible set of controller locations (c), ensuring that these positions do not share an edge with certain controls in the switch (s). A network split with a high-degree node technique is utilized to identify the critical network controllers [15] [16]. The SDN-FTM method for CPP in SDN outlines potential nodes for controller positions, aiming to provide the shortest possible reaction time for controllers and associated switches while maintaining optimal response time.

The method addresses controller placement using a high node degree technique, which is repeated to identify suitable controller instances from the switch (s) until nodes in set of controller location (C) dominate all nodes in switch (s) or until the maximum number of controllers is reached. Then, the independent dominating set approach is applied to divide the network into multiple domains, ensuring that the controller distribution is as close to the minimal response time as possible. This method locates high-degree nodes that act as cluster centers, then creates network subgroups for forwarding nodes. The key selection criterion is the node with the highest degree and the shortest total distance to other nodes in the network.

2.2 Stage 2: Controller Placement

In this simulation, we utilized 1, 2, 3, and 4 controllers alongside 5 switches, using a spanning-tree topology. Since one of the objectives of this study is to optimize the controller's location, hence there is need to develop an effective placement strategy. The control signaling latency is reduced by including the limited capacity of the server, the number of controllers needed for fault tolerance, and inter-controller communication. Each controller manages a different part of the network, meaning it only has visibility into a segment of the network. To consolidate flow data across the network, each pair of controllers is synchronized, which involves two-way communication with at least two messages per synchronization.

2.3 Stage 3: Fault Tolerance

Here, the proactive and reactive approaches to failure recovery is utilized. In proactive recovery, decision rules are predetermined, while in the reactive method, recovery decisions are made based on the situation after a failure occurs. In the reactive approach, the controller installs flow-based rules in response to events

reported by switches, which can lead to substantial overhead on the switches, reducing their performance in terms of latency and throughput. Additional factors considered when selecting the best candidate controller during the recovery phase include controller load, reliability, and optimization of global or local assignments. Our proposed method accounts for reliability by considering the topology of the subnetwork connected to each controller, the load, and the failure probability of both connected and interconnected links. It selects a more reliable controller to reassign the switches of the failed controller accordingly. This reassignment can be optimized either globally or locally. Global optimization is achieved if all switches managed by the failing controller can be reassigned to another controller simultaneously. In contrast, local optimization involves reassigning switches one by one, which can increase latency. In our method, switch-controller reassignment is handled globally. The optimal placement of multiple controllers minimizes the maximum latency between nodes and the controller while maximizing fault tolerance in case of controller failures. If the primary controller fails, all nodes automatically connect to the nearest available controller in the network. By optimizing controller placement and considering worst-case latency scenarios, the method enhances network resiliency. Additionally, if the primary route link fails, the system detects this and switches to the closest alternate route, ensuring network communication continues with acceptable quality.

2.4 Practical Illustration of Proposed Method with Scenarios

To further evaluate the performance of the proposed method, we implement the method on two different scenarios as shown in following sections**.**

2.4.1 Scenario 1: SDN-FTN on Small Scale Evaluation

We start the simulation by using a small-scale network with OpenFlow and Ryu's controllers running the application on all the controllers using the python command in Mininet. The command used ensure the specification of only one controller in the network that is chosen to play the role of the primary in the topology. In the simulation, the four controllers are connected to virtual switches which in turn are linked to a host. Using the standard topology throughout the simulations and changing the controllers' type during the evaluation of fault tolerance. The controllers are running on the virtual machine using IP range 127.0.0.1/4 whereby the first address is used to identify the primary controller in the network. Different color codes to differentiate the connection between nodes in the network as shown in Figure 2.

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

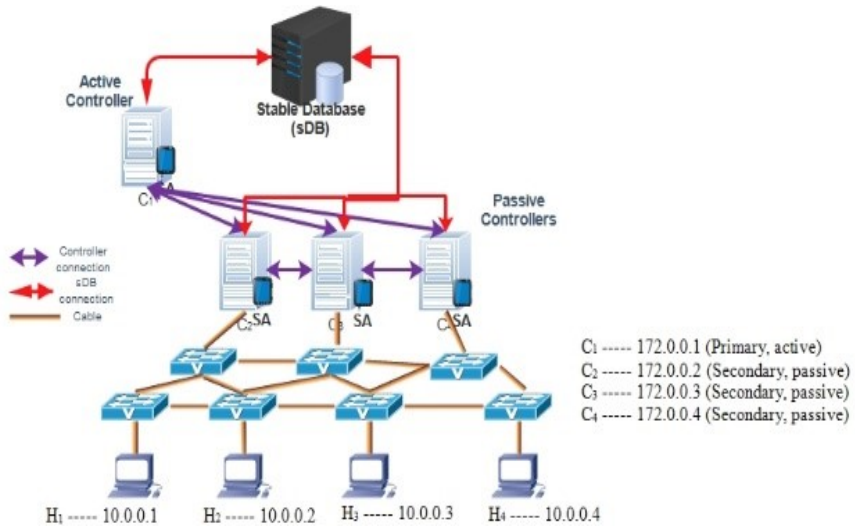


Figure 2. Small-Scale Network Evaluation

In this first simulation as shown in Figure 2, controller C_1 is selected as the primary (active controller) and is identified and connected to all the other controllers and switches in a circular scheme in the network. This circular scheme connection enables synchronization so that the nodes in the network can share information about each other state in real-time to allow the system's functionality in the event of a failure. In this case, the scheme allowed controllers to share network updates among themselves and if a decision needed to be taken in the control plane. The controllers use the information or packets sent from the switches to provide a global view of the state of the network, and updates.

In the initial stage of the simulations, the switches flow table is empty, when using the Mininet *ping* test command from the host machine, the virtual switches send a packet-in message to the controller to check in the data path, the application running on the primary controller responds with the amount of packet transmitted and received at the time it takes. As soon as the switches flow table is occupied by the correct flow information from all the nodes on the network establish communication between them. The illustration of the *ping* test command in Mininet from the host (h1) to the primary controller (C_1 with IP address 127.0.0) is as shown in Figure 3.

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

```
mininet> h1 ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.086 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.111 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.086 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.092 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.111 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.090 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.054 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.110 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.096 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.111 ms
^C
--- 127.0.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9214ms
rtt min/avg/max/mdev = 0.054/0.094/0.111/0.016 ms
mininet>
```

Figure 3. The Ping Command Test for Host (h1) to the Primary Controller c0 in Mininet

2.4.2 Scenario 2: SDN-FTN on Primary Controller Failure

Using the same scenario in Figure 3, with the assumption that the primary controller C_1 have IP address 127.0.0.1, goes down due to some network fault, error, or failure as shown in Figure 4.

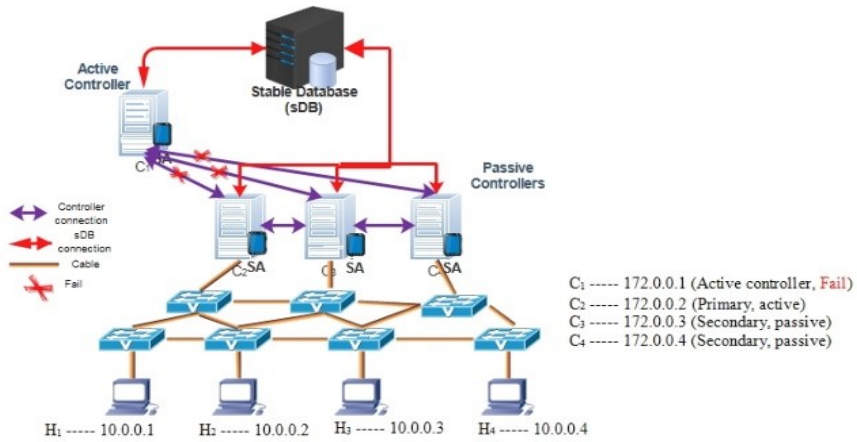


Figure 4. Primary Controller Failure

In Figure 4, there is a controller fault that occurs when the controller does not receive incoming demands from the switches or fails to send messages to respond to the switches' demand or an error caused by the node/Link that occurs when a controller receives an incorrect message in response to the switches message the network cannot result in total failure. From the above scenario, C_1 was playing the role of the primary in all Ryu controllers set up topology, after a fault or a controller failure, the other controllers (secondary) elect one of them to take over as a new primary. This process is done automatically using the control programmable capability in Mininet. Utilizing the *ovs-vsctl* command tools in Mininet the designation of the new controller is done as shown in Figure 5.

Mininet Script to change the Primary Controller

```
$ ovs-vsctl set-controller C1 "tcp: 127.0.0.2: 6633"
$ ovs-vsctl set-controller C2 "tcp: 127.0.0.2: 6633"
$ ovs-vsctl set-controller C3 "tcp: 127.0.0.2  6633"
$ ovs-vsctl set-controller C4 "tcp: 127.0.0.2: 6633"
```

Figure 5. Mininet Setup to Change the Primary Controller

In this scenario, each controller gets the address of the newly elected primary controller and shares the information about the network, and the same information is stored in the sDB. Referring to a scenario in Figure 5, after C_1 is dead, C_2 takes over the network immediately as the primary controller by using a request message, initiated by the OpenFlow to enable the controller to request a switch for port statistics using the *Ryu.ofproto.ofproto_v1_3_OFPPortStatsRequest* command in Mininet. Once the selection of the new primary is completed, new information is updated and sent to all the nodes in the network; then the detail of the new primary is stored in the sDB. The primary controller is denoted by C_1 and any other controller C_n is secondary. For the simulation in Mininet, we make use of hosts as H_1 up to H*n* in the application plane layer as devices connected to the switches. The code shows the new network topology in the first scenario with all controllers after the election of the new primary using python code in Mininet in Figure 6.

Controller Topology After Election of the new Primary in Mininet

```
"controllers": [
          {
        "opts": {
           "controllerProtocol": "TCP",
           "controllerType": "ref",
           "hostname": "c1",
           "remoteIP": "127.0.0.1",
           "remotePort": 6633
           "switchStatus":" DOWN",                    //Controller DOWN
        },
        "x": "405.0",
        "y": "290.0"
       {
        "opts": {
           "controllerProtocol": "TCP",
           "controllerType": "ref",
           "hostname": "c2",                          //New controller assigned
           "remoteIP": "127.0.0.2",
           "remotePort": 6633
           "switchStatus":" UP",
        },
        "x": "350.0",
        "y": "170.0"
     },
     {
        "opts": {
           "controllerProtocol": "TCP",
           "controllerType": "ref",
           "hostname": "c2",
           "remoteIP": "127.0.0.3",          //Secondary controller
           "remotePort": 6633
           "switchStatus":" UP",
        },
        "x": "210.0",
        "y": "215.0"
                                  ……                  //Continuity of the code
                                  ……                  //Continuity of the code
     }
```

Figure 6. Topology after the Election

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

Using the *ping* test command in Figure 7 in Mininet from the host machine to the virtual switches in the data path, which sends a packet-in message to the controller that plays the role of the new primary, the application running on the new primary controller responds by identifying the updated flow tables information from the switches and the communication is established between the nodes in the network.

```
mininet> h1 ping 127.0.0.2
PING 127.0.0.2 (127.0.0.2) 56(84) bytes of data.
64 bytes from 127.0.0.2: icmp_seq=1 ttl=64 time=0.133 ms
64 bytes from 127.0.0.2: icmp_seq=2 ttl=64 time=0.105 ms
64 bytes from 127.0.0.2: icmp_seq=3 ttl=64 time=0.094 ms
64 bytes from 127.0.0.2: icmp_seq=4 ttl=64 time=0.112 ms
64 bytes from 127.0.0.2: icmp_seq=5 ttl=64 time=0.102 ms
64 bytes from 127.0.0.2: icmp_seq=6 ttl=64 time=0.111 ms
64 bytes from 127.0.0.2: icmp_seq=7 ttl=64 time=0.090 ms
64 bytes from 127.0.0.2: icmp_seq=8 ttl=64 time=0.111 ms
64 bytes from 127.0.0.2: icmp_seq=9 ttl=64 time=0.096 ms
64 bytes from 127.0.0.2: icmp_seq=10 ttl=64 time=0.112 ms
^C
--- 127.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9194ms
rtt min/avg/max/mdev = 0.090/0.106/0.133/0.011 ms
mininet>
```

Figure 7. The Ping Command Test After the Election of the New Primary from Host (h1) to C_2

Once the new topology is created, the virtual switches update the new flow table containing new information about the network state. The new details about the number of packets transmitted, received, and lost, as well as the new time, which is updated and stored in the sDB.

2.5 Simulation System and Estimation Indexes

To simulate the proposed SDN-FTM in Mininet, two different controllers namely OpenFlow and Ryu with various parameters, metrics, and nodes were deployed on the network. As discussed above, the standard-based topology for the proposed SDN-FTM simulations in Mininet consists of controllers (OpenFlow and Ryu), legacy switches, and hosts. All these nodes used the link to establish a connection between them on the network as well as specific decryption. That identified them. To configure the controller, there is a need to define its type, the type of protocol, create its name, IP address as well as the port number used by that specific controller. The configuration of OpenFlow SDN-FTF Controllers in Mininet up is shown in Figure 8.

OpenFlow SDN-FT Controllers Setup in Mininet

```
controllers:
    {
       "opts": {
          "controllerProtocol": "TCP",
          "controllerType": "ref",
          "hostname": "c1",
          "remoteIP": "127.0.0.1",
          "remotePort": 6633,
          "ControllerStatus":"UP",
       },
       "x": "355.0",
       "y": "173.0",
    }
```

Figure 8. Controller Setup in Mininet

p-ISSN: **2656-5935** http://journal-isi.org/index.php/isi e-ISSN: **2656-4882**

Figure 8 shows a piece of code that represents the basic setup of the OpenFlow controllers used in the simulations. In the virtual machine, we have defined the protocol for the controller, the type of controller used, the name of the controller, the IP addresses assigned to the controller the status of the controller as well as the port that the controller is using. The "x" and "y" values represent the placement between the controllers in the network. This same procedure is repeated several times for each SDN controller setup in the simulation.

As indicated in the previous section, the topology used for this simulation adopts a multi-controller scheme with ten (10) SDN controllers, two switches, and two hosts attached to them. To set up a host, we specify the hostname (h1, h2, …h_n) as well as assign IP addresses between the range of 10.0.0.1/2 since we only used two hosts. Host h1 was assigned the IP address 10.0.0.1 and host h2 10.0.0.2. all these hosts were connected to two legacy switches denoted by s1-eth1 and s2-eth2 respectively and all useful links to establish a connection between them. To define the placement between the nodes on the network we used the "x" and "y" values during the simulation. The piece of code illustrating the node setup as shown in Figure 9.

Node Setup in Mininet

```
"hosts":[
{
"number": "1"
      "opts": {
        "hostname h1
        "IP" : "10.0.0.1"
           "nodeNum": "1"
           "ached": "host"
}
"x":"68.7.0
"y":"410"
}
{
"number": "1"
      "opts": {
        "hostname h2
        "IP": "10.0.0.2"
           "nodeNum": "2"
           "ached": "host"
}
"x": "227.0
"y": "485.0"
}
],
"links":[]
"switches": [
{
"number": "1",
"opts": {
  "controller" : [],
"hostname": "s1",
"nodeNum": "2"
"switchType": "legacySwitch"
"switchStatus": "UP"

},
"x" "347.0"
"y": "3310"
}
```

Figure 9. Controller Setup in Mininet

Among the connected controllers on the network, only one plays the role of the primary controller and the others are acting as secondary controllers as shown in Figure 9. For this simulation, four controllers were used for the simulation. However, the architecture of the SDN-FTM can support small and medium networks. This approach was based on the rationale to avoid network failure using a single controller and to achieve efficient network management. To address this, challenge this research adopted a multiple-controller approach to overcome this limitation.

The proposed SDN-FTM design consists of four sets of controllers (C1…Cn) interconnected between them so that secondary controllers can have the network view from the primary that maintains the network. We have used different colors for the link to show the connection between the controllers themselves and the switches. These controllers used IP addresses ranging from 172.0.0.1/4 for the first scenarios and 172.0.0.1 /10-second ones. In addition, we make use of two switches (s1 and s2) that are connected to all the controllers and share the IP address of the controller that plays the role of the primary while the IP image of all the secondary is also shared with them. The primary controller (c1) shared network updates with all the secondary controllers connected to it to alert them of any changes on the network using replication. In our case, the primary controller is identified as c1, and any chosen secondary controller as Cn.
For the simulation in Mininet, we make use of hosts denoted as H_n whereby *n* represents the host number in the application plane layer as devices connected to the switches as illustrated in Figure 9.

2.6 Performance Evaluation

An appropriate performance measure is chosen to ensure the effectiveness of the proposed method for controller placement. The metrics used to evaluate the performance of the proposed systems are throughput and latency.

Throughput: The total size of information sent each period. In an SDN platform, the performance of the controller has a significant impact on throughput and latency in the context of the proposed architecture. The throughput of the controller determines the rate of flow response, and similarly, the controller's latency performance criterion is determined by the time it takes to respond to a flow request. Additionally, since it has an impact on throughput and latency performance, the location of the controller (placement) inside the network is crucial to the network's function.

Latency: The total delay in the amount of data transmitted. This also serves as a benchmark for any network performance. As a result, latency and throughput were appropriate metrics for assessing the proposed system.

3. RESULTS AND DISCUSSION

This section presents the simulation setup and the result of simulation conduction as shown in the following sections.

3.1. Simulation Setup

The simulation software used in this research is Linux-Ubuntu 20.04 LTS, 64-bit operating system, SDN Mininet, Wireshark, Hard Disk 1 T, RAM 8G, Processor: Intel® Core™ i7-45705 CPU @ 4.90G, Processor: Intel® Core™ i7-45705 CPU @ 4.90G, OpenFlow Controller, Ryu Controller. The simulation parameter used in this research is shown in Table 1.

Table 1. Simulation parameters

Parameter	Values	Details
Controller	127.0.0.1……127.0.0.4	OpenFlow and Ryu
Nodes	s1-eth1, s2-eth2 (Switches) Name C0 ……. Cn (Controllers) h1……...hn (Hosts)	Nodes are devices connected to the network such as hosts, controllers, switches
Protocol	TCP SSL	protocol used during the simulation.
IP Base	10.0.0.1……10.0.0.2	It identified all the nodes connected to the network
OpenFlow Ports	6633, 6634, 6635, 6636	The port functions as an interface between the swathes and controllers.
Link	the link is used to establish a connection between the host, switches, and controllers	different link colors in the defined connection between different controllers

The nodes detailed used in the simulation are shown in Table 2.

Table 2. Node details

Node Types	Description	Details
Controller	A piece of hardware or software that controls how data and applications move through a network and serves as the network's central nervous system.	• controller type: OpenFlow and Ryu • Protocol IP range 127.0.0.1/4

Node Types	Description	Details
Switch	A device that helps the connection of the controllers and hosts.	• legacy Switch denoted by sw1….sw_n
Host	Workstation connected to the legacy switches	• hostname (h1, h2, …hn) • IP addresses range 10.0.0.1/2

3.2 Simulation 1: Evaluation of Small-Scale Network

The outcomes of the proposed FTM's simulated evaluation in a small-scale network scenario are shown in this section. As indicated in the previous section that the evaluation in the small network makes use of modified controllers with SA. The controllers were evaluated by the latency and throughput performance. To measure latency and throughput performance for the SDN-FTM in Mininet, we make us of the standard topology presented in Figure 3 which is based on a small-scale Network scenario. The controller running on the primary switch reacted to packet-in messages from the hosts connected to the virtual switches. After evaluating the performance of the controllers (Ryu and OpenFlow). The summary in terms of the runtime and round-trip time that determine the controller performances during the evaluation of the FTM is shown in Table 3.

Table 3. Node details

Controller ID	Throughput (m/s)	Latency (m/s)	Type Controller
c1	-	-	OpenFlow
c2	1.066	9194	Ryu
c3	0.985	9207	OpenFlow
c4	1.084	9214	Ryu

To have a clear picture of the controllers' performances conducted the simulation using small-scale network topology, the throughput and the latency performance results of the scenario using both Ryu and OpenFlow controllers are shown in Figures 10 and 11 respectively.

From Figure 10, the performance of every single controller during the simulation using the proposed FTM in a small-scale network is presented. The controller denoted by c1 and c3 were configured as OpenFlow controllers and c2 and c4 as Ryu's controllers. From the outcomes in Figure 10, one can observe that the individual controllers' performances of OpenFlow suffer because of the overhead and placement changes due to the selection of the new primary controller after the fault has occurred. In the above analysis, the throughput response time for the c2 and c4 (Ryu's) controllers is higher than for the c2 (OpenFlow) controllers. This implies that during the takeover of the new primary controller, each controller

suffers in their performance either higher load balancing or lower load balancing that caused their throughput to be high or low. It can be noticed that the OpenFlow controller throughput analysis is lower than Ryu's controllers.

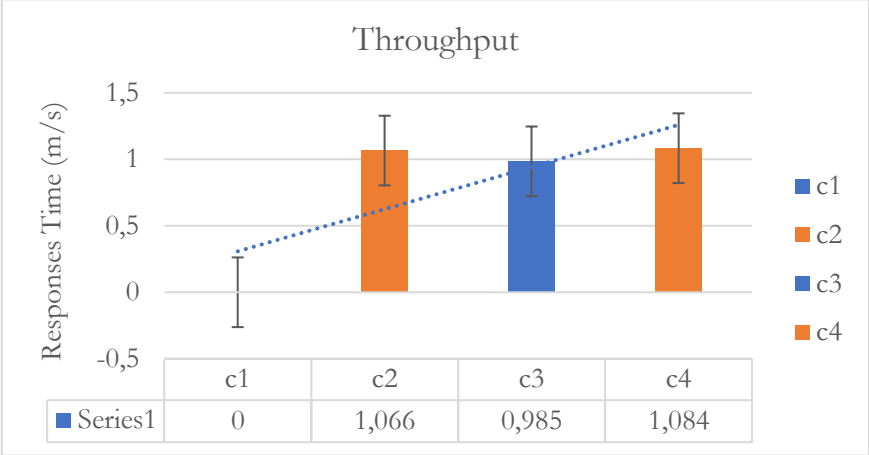
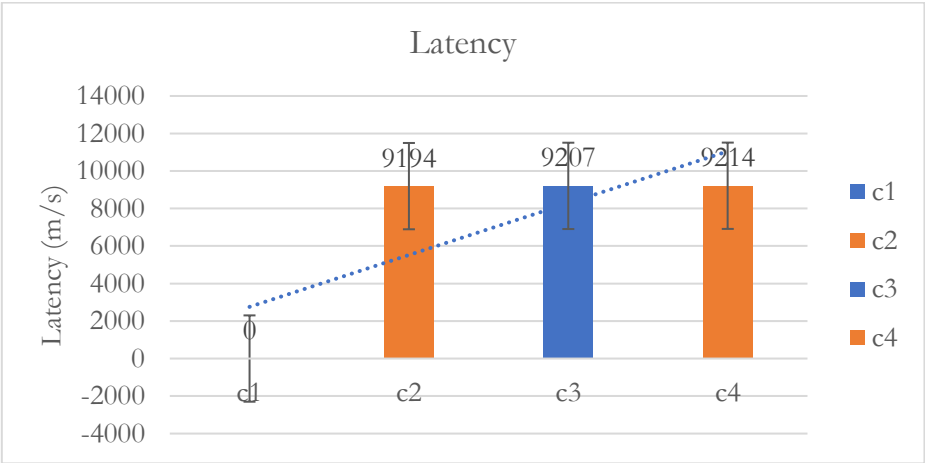


Figure 10. Individual Controller Throughput Analysis

The one for c2 and c4 (Ryu's) controllers is lower than c2 (OpenFlow). This stage c1 is considered as a fault and not having any connection in the network. The same phenomenon was also discovered during the evaluation of the latency for the SDN-FTM and the controller latency analysis as shown in Figure 11.



Figu**re 11.** Individual Controller Latency Analysis

Figure 11 shows the latency analysis of individual controllers which indicates that the performances of c2 (Ryu) are less than c3 (OpenFlow) and c4 (Ryu) is better

than c3 (OpenFlow). This suggests that each controller experienced performance issues during the selection of the new primary, nevertheless, their latency is not that much higher as compared to the throughput between the two. This implies that the network availability after the new primary is taken over is not delayed. After a fault was identified in c1, the SA listened to incoming requests using IP and port network statistics to identify the controller with less load balancing so that it could take the responsibility of the primary controller. The SA's role is to listen on a certain SDN port at an IP address, while the other reaches out to the other to establish a connection of sorts. The one in the primary controller forms to the listener while the secondary reaches out to respond to the request. The values of controller performance in terms of response time (m/s) and latency (m/s) are shown in Table 4.

Table 4. Node details

Type Controller	Throughput (m/s)	Latency (m/s)
OpenFlow	2.15	18408
Ryu	0.985	9207

Table 4 shows the summary result of the general network performance evaluation for the throughput and the latency analysis comparison based on small-scale network topology to identify the best topology performance for the proposed SDN-FTM from the result is displayed in Figures 12 and 13 respectively.

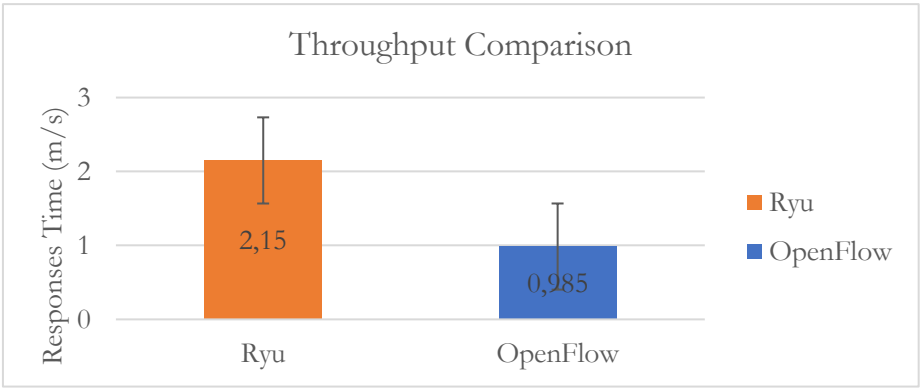


Figure 12. Controller Throughput Comparison

Figure 12 revealed that Ryu's controllers have an increased throughput performance compared to the OpenFlow controllers. It is evident that from the experiences, the general network throughput analysis for the proposed SDN-FTM for small scall network has the best performance results and shows the best scalability compared to the other. It shows that Ryu's controllers for the proposed SDN-FTM performance have a throughput analysis twice higher than OpenFlow controllers. This indicates that the quantity of data sent in the network small scale

for the proposed SDN-FTM is being processed faster in a specific period during fault-tolerant compared to OpenFlow controllers. This higher rate of throughput shows that messages sent on the network arrive at the destination successfully and are being delivered with less delay time. These results are indifferent to the low rate of throughput shown in the OpenFlow controllers. Likewise, the general network performance evaluation for the latency analysis for the proposed SDN-FTM is presented in Figure 13.

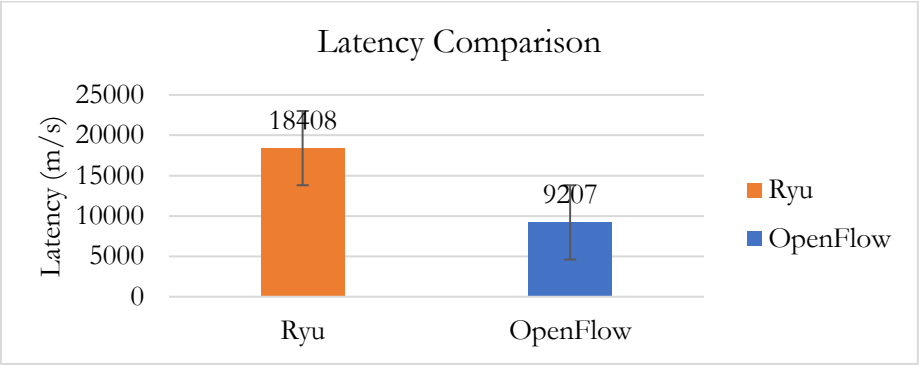


Figure 13. Controller Latency Comparison

Figure 13 shows that the OpenFlow controllers have a low latency rate compared to Ryu's controllers. This low latency demonstrates that in a small-scale network for the proposed SDN-FTM, OpenFlow controllers' responses after failover are better than Ryu's. This is evident that their latency and throughput performance is not the same in the small-scale network scenario.

3.3 Comparative Analysis of the Proposed Method with Other Methods in Literature

This section compares the performance of the proposed model with other state-of-the-art SDN models that are currently in use based on the following: problem addressed, method used, network, evaluation metric, and implementation tools, as shown in Table 5.

Table 5. Comparative analysis of the proposed method with other related methods

Citation	Problem Addressed	Method	Network	Evaluation Metric	Implementation tools
[17]	Scalability problems due to a single centralized	Greedy model	WAN	Number of required controllers, their location and load balancing	Simulations: MATLAB 2018a and CPLEX 12.6, and Internet

Citation	Problem Addressed	Method	Network	Evaluation Metric	Implementation tools
	controller in SDNs				Topology Zoo
[18]	Fault-tolerant controller placement problem	Heuristic algorithm	WAN	Number of controllers, location of controllers, and reliability of the controller placement	Simulations: LINUX, Mininet, and Internet Topology Zoo
[19]	Multi-controller placement	Hybrid harmonic search algorithm and particle swarm optimization algorithm (HSA-PSO)	WAN	Propagation latency, Round Trip Time (RTT), matrix of Time Session (TS), delay, reliability, and throughput	Simulations: CloudsimSDN network simulator, Intel (R) Core (TM) i7-4590S @ 3.00 GHZ
[20]	Control layer is completely decoupled from the data layer in the network	Binary linear programming model	WAN	Controller type, place and minimum number of required controllers, controller processing capacity, and the setup cost	Simulations: Internet2OS3E
Proposed method	Controller placement and Fault tolerance in the network environment	SDN-FTM	WAN	number of required controllers, their location, throughput, and latency	Simulations: Linux-Ubuntu 20.04 LTS, Mininet, Intel® Core™ i7-45705 CPU 4.90G, OpenFlow Controller, Ryu Controller

Relying on a single centralized controller in Software-Defined Networks can lead to scalability issues introduced in [17], to enhance scalability and maintain low

latency in wide area networks, the use of multiple controllers is recommended. The approach determined the optimal number of controllers, and their placement, and ensured effective load balancing among them to prevent overloading by using greedy algorithms. Authors in [18] addressed the issue of fault-tolerant controller placement problem with a heuristic algorithm that calculates placements with the necessary level of reliability. Also, research in [19] solved the issue of Multi-controller placement in SDN by utilizing a Hybrid harmonic search algorithm and particle swarm optimization algorithm (HSA-PSO) on WAN. The implementation evaluation was tested on propagation latency, Round Trip Time (RTT), matrix of Time Session (TS), delay, reliability, and throughput. Although the proposed problem addressed is similar to research done in [19], however, the proposed SDN-FTM method has the advantage of optimizing the network in a situation when a single network failure occurs which consequently improves the scalability and throughput of the network [21] [22].

4. CONCLUSION

In conclusion, this study presents the design, evaluation, and simulation of the SDN-Fault Tolerance Model (FTM) using a small-scale network scenario. The proposed SDN-FTM was tested and evaluated in real-time using Mininet simulation tools, with a focus on performance measures such as latency and throughput. Key findings indicate that Ryu controllers generally outperform OpenFlow controllers in terms of throughput, while OpenFlow controllers exhibit lower latency. This suggests that while Ryu controllers can handle more data efficiently, OpenFlow controllers may be more responsive in terms of communication delays. The comparative analysis across small, medium, and large-scale network scenarios reveals consistent trends in performance, with Ryu controllers maintaining higher throughput and OpenFlow controllers showing lower latency across different network scales. These results underscore the scalability and robustness of the proposed SDN-FTM architecture. Despite the promising results, the study is limited by the scope of the network scenarios tested and the specific simulation environment used. Future research could explore larger-scale networks, additional performance metrics, and further refinement of the fault tolerance mechanisms.

REFERENCES

[1] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The Controller Placement Problem in Software Defined Networking: A Survey," *IEEE Network,* pp. 21-27, 2017.

[2] M. Mbodila, B. Isong, and N. Gasela, "Towards a Cost-Effective SDN-Enabled on-Demand Security Services Framework," in *2023 International*

Conference on Electrical, Computer and Energy Technologies (ICECET), 2023: IEEE, pp. 1-6, 2023.

[3] L. Mamushiane, J. Mwangama, and A. Lysko, "Controller Placement Optimization For Software Defined Wide Area Networks (SDWAN)," *Council for Scientific and Industrial Research (CSIR),* 2019.

[4] K. A. Rasol and J. Domingo-Pascual, "Multi-level Hierarchical Controller Placement in Software Defined Networking," *12th International Networking Conference. INC 2020". Berlín: Springer,* pp. 131-145, 2020.

[5] A. K. Tran, M. J. Piran, and C. Pham, "SDN Controller Placement in IoT Networks: An Optimized Submodularity-Based Approach," *Sensors* vol. 19, no. 5474, pp. 1-27, 2019, doi: doi:10.3390/s19245474.

[6] A. Jalili, M. Keshtgari, and V. Ahmadi, "Controller Placement in Software-Defined WAN Using Multi Objective Genetic Algorithm," *International Journal of Mechatronic,Electrical and Computer Technology,* vol. 5 no. 18, pp. 2655-2663, 2015.

[7] X. You, Y. Feng, and K. Sakurai, "Packet in message based DDoS attack detection in SDN network using OpenFlow," in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, 2017: IEEE, pp. 522-528, 2017.

[8] F. Elegbeleye and S. Rananga, "IoT Device Cost Effective Storage Architecture and Real-Time Data Analysis/Data Privacy Framework," *International Journal of Industrial and Manufacturing Engineering,* vol. 17, no. 7, pp. 288-298, 2023.

[9] J. Zhao, H. Qu, J. Zhao, Z. Luan, and Y. Guo, "Towards controller placement problem for software-defined network using affinity propagation," *Electronics Letters,* vol. 53, no. 14, pp. 928–929, 2017.

[10] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *in Proc. First workshop on Hot topics in software-defined networks,* pp. 7-12, 2012.

[11] G. Yao, J. Bi, Y. Li, and L. Guo, "On the Capacitated Controller Placement Problem in Software-Defined Networks," *IEEE communication Letters,* vol. 18, no. 8, pp. 1339-1342, 2014.

[12] B. Heller, R. Sherwood, and N. Mckeown, "Controller Placement Problem," *in Proc. HotSDN,* pp. 7–12, 2012.

[13] S. Čaušević and M. Begović, "Optimizing Traffic Routing in Different Network Environments Using the Concept of Software-Defined Networks," in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019: IEEE, pp. 409-414.

[14] R. Salam and A. Bhattacharya, "Efficient greedy heuristic approach for fault-tolerant distributed controller placement in scalable SDN architecture," *Cluster Computing (2022)* vol. 25, pp. 4543–4572, 2022.

[15] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in SDN," *IEEE communications surveys & tutorials,* vol. 22, no. 1, pp. 472-503, 2019.

[16] F. A. Elegbeleye, M. Mbodila, O. A. Esan, and I. Elegbeleye, "Cost-effective internet of things privacy-aware data storage and real-time analysis," *Int J Artif Intell,* vol. 13, no. 1, pp. 247-255, 2024.

[17] M. Khorramizadeh and V. Ahmadi, "Capacity and load-aware software-defined network controller placement in heterogeneous environments," *Comput. Commun., ,* vol. 129, pp. 226-247, 2018.

[18] F. J. Ros and P. M. Ruiz, "On reliable controller placements in software defined networks," *Comput. Commun., ,* vol. 77, pp. 41-51, 2016.

[19] N. S. Radam, S. T. F. Al-Janabi, and K. S. Jasim, "Multi-Controllers Placement Optimization in SDN by the Hybrid HSA-PSO Algorithm," *Computers* vol. 11, no. 7, p. 111, 2022, doi: 10.3390/computers11070111.

[20] A. Naseri, M. Ahmadi, and L. PourKarimi, "Placement of SDN controllers based on network setup cost and latency of control packets," *Computer Communications,* vol. 208, no. 1, pp. 15-28, 2023, doi: 10.1016/j.comcom.2023.05.015.

[21] M. A. Aglan, M. A. Sobh, and A. M. Bahaa-Eldin, "Reliability and scalability in SDN networks," in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*, 2018: IEEE, pp. 549-554, 2018.

[22] F. A. Elegbeleye, M. Mbodila, A. Mabovana, and O. A. Esan, "Data privacy on using four models-a review," in *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2022: IEEE, pp. 1-9, 2022.