The Indonesian Journal of Computer Science
www.ijcs.net
Volume 13, Issue 6, December 2024
https://doi.org/10.33022/ijcs.v13i6.4417

Fault Tolerance Management Implementation from Medium-to-Large-Scale Networks

Munienge Mbodila[1], Omobayo Ayokunle Esan[2], Femi Abiodun Elegbeleye[3]
mmbodila@wsu.ac.za[1], oesan@wsu.ac.za[2], felegbeleye@wsu.ac.za[3]
[1]Department of Network and Support, Walter Sisulu University, East London, South Africa
[2]Department of Network and Support, Walter Sisulu University, East London, South Africa
[3]Department of Network of Business Application and Development, Walter Sisulu University, East London, South Africa

Article Information

Received : 29 Sep 2024
Revised : 12 Oct 2024
Accepted : 30 Nov 2024

Keywords

Control placement, software-defined network, redundancy, link failure, fault-tolerance management

Abstract

As network infrastructures grow in complexity, ensuring high availability and resilience becomes critical, especially for medium-to-large scale networks. This study focuses on the development and implementation of fault tolerance management within Software-defined networking (SDN) environments, aimed at minimizing downtime and enhancing network reliability. SDN's centralized control and dynamic programmability provide an ideal framework for implementing efficient fault detection and recovery mechanisms. The proposed model leverages real-time monitoring, redundancy protocols, and adaptive rerouting strategies to mitigate the impact of node or link failures. Key components of the model include failover mechanisms, load balancing, and traffic rerouting algorithms, designed to maintain seamless network operations during failures. Through simulation and testing, the model demonstrates significant improvements in network resilience, reducing recovery time and ensuring uninterrupted service delivery. This research provides a comprehensive guide to implementing fault-tolerant networks using SDN, offering scalable solutions that can be adapted to various network sizes and configurations. The findings emphasize the potential of SDN to revolutionize fault management in modern network infrastructures, making it a crucial consideration for future network design and operations.

A. Introduction

Fault management is a term used in network management, describing the overall processes and infrastructure associated with detecting, diagnosing, and fixing faults, and returning to normal operations in telecommunication systems [5].

Fault tolerance is an essential part of the design of any communication system/network. Computer networks are built on physical infrastructure or virtualized versions of the physical infrastructure. These infrastructures are critical because business applications rely on the proper operation of such infrastructures. However, such infrastructures are prone to a wide range of challenges and attacks such as faults, failures, and errors which disrupt network service[6].

Software-defined network (SDN) is a software-centric network design paradigm that makes it possible to simplify network management by decoupling control logic from forwarding devices. [1]. The control logic is the brain of the network, deciding the best paths, preventing failure, and putting the business logic into action. This is done centrally using a computing device (server) instead of the old distributed design where each node in the network has the task of finding the best paths or recovering from failures. In addition to the control plane, there is a data plane and an administration plane. [2]. The control plane interacts with the data plane via southbound interfaces such as OpenFlow. The plane is primarily responsible for the forwarding of data packets in switching or routing through programmable devices.

To handle a received packet, switches consult the forwarding rules in their flow table(s). The packet is forwarded if a match is found; otherwise, the switch sends a packet-in message to the master controller. The controller reactively determines the path for this packet and installs new rules in the affected switches. [3]. The time from receiving the packet-in message to receiving a new response from the controller is called the flow setup time. Reducing this time is critical to network efficiency, and this is majorly affected by the load on the controllers and the propagation delay. A single-controller system provides a holistic network-wide view, with a single point of failure and lacks reliability and scalability. [4]. Therefore, multi-controller SDNs have been developed that allow the control plane to be physically distributed while keeping it logically centralized by synchronizing network status between controllers [5]. However, multi-controller SDNs bring new challenges such as network synchronization and adaptive switch-controller mappings.

A fundamental problem in multi-controller SDNs is the controller placement problem (CPP) [6]. This problem focuses on deciding the number of controllers required to meet the service level agreement for a network and where to find appropriate locations for each controller. That is, dividing the network into sub-regions (domains) while considering some quality criteria and cost constraints. [7].

Numerous research have been conducted to address this issue, the research in [8] addressed the controller placement by including the factor of load controllers using a capacitated k-center. According to the results, the new strategy reduces the number of controllers required to avoid overload and reduces the load of the busiest controllers. Also, has a smaller radius than the dynamic controller

provisioning. Similarly, research in [9] investigated the multi-controller placement problem for a Software-defined network (SDN) to minimize the propagation latency between switches and associated controllers by utilizing a modified exemplar-based clustering method that is based on affinity propagation.

Furthermore, research in [10] Introduced multi-controller optimization in SDN to address the problem of optimal controller problem that affects overall network performance by using a metaheuristic algorithm. This algorithm includes a harmonic search algorithm and a particle swarm optimization algorithm (HAS-PSO). The SDN-enable network involves the partitioning of the network with the nodes and links of the physical topology of undirected graphs given in equation (1).

$$G = (V, E) \tag{1}$$

where E is the set of physical links connecting the nodes, and V is the SDN set of nodes that represents the network Switches and controllers. Instead, the network partition can be described as in equation (2) if the number of partitions is given as k when SDN is taken into consideration.

$$SDN_i(V_i, E_i) \tag{2}$$

Equation (3) is subject to conditions in equations (3) – (7).

$$\cup_{i=1}^{k} V_i = V; \ \cup_{i=1}^{k} E_i = E \tag{3}$$

The total number of subnetworks required to cover all network nodes and linkages is represented in equation (4).

$$SDN_i \cap SDN_j = \emptyset; \ \forall_{i \neq j, \ i,j \in k} \tag{4}$$

In equation (4), ∅ Denotes the total number of subnetworks necessary to cover all network components, including nodes and links. Elements in one sub-network have the same similarity and this is represented in equations (5) and (6) respectively.

$$similarity(SDN_i) = TRUE, \ \forall_{i \neq k} \tag{5}$$

$$similarity(SDN_i \cap SDN_j) = FALSE, \ \forall_{i \neq j, i, j \in k} \tag{6}$$

The elements assigned to various subnetworks appear to have distinct properties, according to equation (5), when the network is anticipated to be partitioned so that the nodes in one cluster have a smaller latency equation (6), while the nodes in separate clusters have a bigger latency equation, this is when the similarity is defined as the latency (7).

$$SDN_I \text{ are connected regions} \tag{7}$$

All of the vertexes in one sub-network are linked together, according to equation (7). The network is divided into smaller sub-networks using the k-means clustering-based partition algorithm to reduce the maximum end-to-end distance. To be clear, the initial nodes chosen to run the clustering algorithms are referred to as the “center”. The true center, or “centroid”, of each cluster.

Our work is similar to research in [10], but in contrast to the work in which the authors use failure planning to select backup controllers and minimize two latencies, the latency from switching to the primary controller and the backup controller. Extensive simulation of the proposed method on different network failure scenarios hereby validates the performance of the new method in terms of latency and throughput. [11]. Hence, the research contributions are as follows:

- Selection of the ideal number of controllers to deploy in the best location is this controller placement problem (CPP).
- maximize network performance as a whole using the proposed method that takes into account the CPP for the distribution of the controller workload, control plane utilization, and network communication delay
- Extensive simulations are conducted using real network topologies.

B. Research Method

This section presents the research method represented by the design of the simulation steps used during the simulation in the context of this study. From a practical aspect, it is important to note that all the simulations in this study are implemented on computers using various software applications. The steps involved in the simulation are shown in Figure 1.

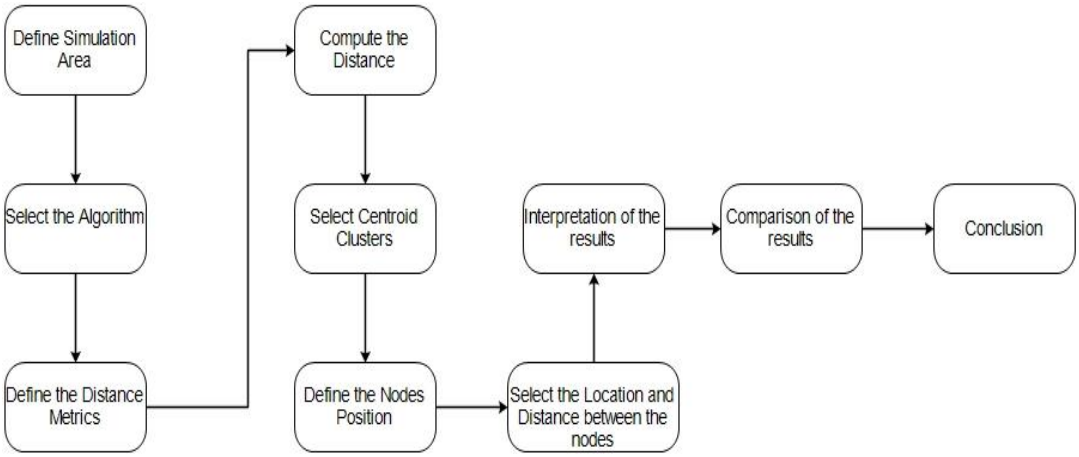

Figure1. Flow diagram of various steps involved in the simulation

To address the issue of controllers' placement in the proposed SDN-FTM architecture, we employ the *city-block* distance metric in MATLAB using the equations (8) and (9), to compute the shortest link on the network that minimizes the average propagation latency between the nodes as well as finding the best location that minimizes the maximum distance to place the controllers in the network infrastructures. When using *Cityblock* distance (also known as Manhattan distance) to calculate the sum of absolute difference *a* plus the distance in b.

In the *Cityblock* distance metric, each centroid is the component median of the point in that cluster. This is like defining the distance between points, cities, and areas (like in Manhattan) and how to move around streets and buildings instead of moving directly to one location. The *Cityblock* distance between two points, *a* and *b* with *k* dimension is denoted and calculated as in equation (8).

$$\sum_{j=i}^{k} | aj - bj | \tag{8}$$

To compute the distance using different distance metrics and to find out the centroid clusters between the controllers and switches whereby *x* represents the distance and *c* is the centroid as presented in equation (9).

$$d(x, c) = \sum_{j=i}^{k} | aj - bj | \tag{9}$$

The main objective of using the above equation is to compute the k-means clustering algorithm to define the location of the controllers as well as specify the distance between them.

The evaluation of the proposed FTM-SDN was done with the throughput which is 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 our 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. Furthermore, the latency which is the total delay in the amount of data transmitted was also used. This also serves as a benchmark for any network performance. As a result, latency and throughput were appropriate metrics for assessing the proposed system.

C. Result and Discussion

This section contains the results of research and discussion, as well as the implementation of the developed system design. The simulation software used in this research is MATLAB, 64-bit operating system, SDN Mininet, Wireshark, Hard Disk 1 T, RAM 8G, and Processor. The simulation was conducted in MATLAB to define the optimal controller locations and optimize the number of controllers to be deployed in the proposed SDN-FTM network infrastructure. The k-mean clustering and k-medoid clustering methods were used to assess the network placement locations by creating a set of random numbers using *x, and y* coordinates making use of a matrix that holds randomly generated usage numbers for each switch and defining the number of clusters to be used (essentially = to the amount of controller) on the network. To exhibit the planned network topology partition, the distance between switches and controllers must be specified as a fundamental component of this method as shown in Figure 2.

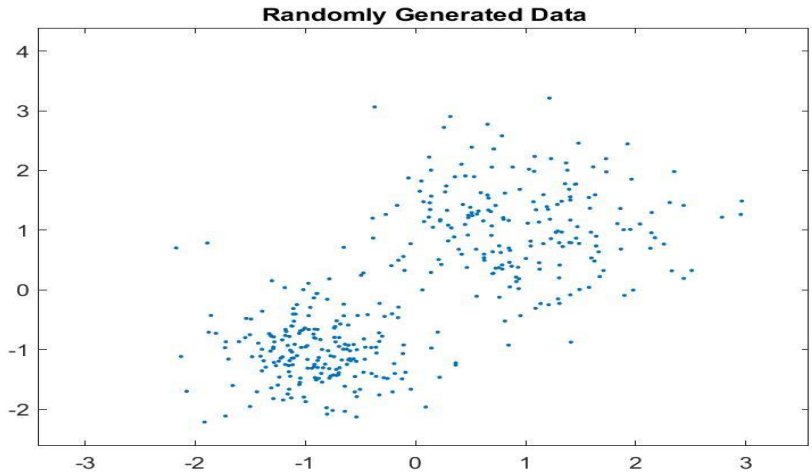

Figure 2. Placement topology formation

The plot shown in Figure 2 displays the network partition randomly generated to define the best arrangement of switches. This partition is important in determining the optimal number of controllers to be deployed and their location. In this scenario, the locations are generated randomly. For the deployment of multiple controllers, we segment the network topology in medium-large scale manageable clusters to overcome the challenge of the single controller by initializing *city-block* distance metric randomly using the k-means algorithm as shown in Table I to find out the optimal controller placement distance between several given switches in the clusters.

Table 1. Distance metrics

Replicate	Iterations	Total Sum of Distances
1	11	88.2602
2	8	92.3789
3	9	89.3037
4	6	92.6302
5	8	91.1183
6	7	96.1411
7	10	92.0779
8	9	93.9656
9	6	91.2769
10	7	99.2522
The best total sum of the distance		88.2602

Out of the 10 randomly initialized instances of the k-mean, with various iterations, the minimum sum of the distance between all points clusters switches and their centroid is 88.2602. Considering this distance metric, we plot the cluster allocations that define the best network sections using the above metric to partition them into 10 small segments (clusters) as shown in Figure 3. Each segment is represented with a specific color to allow the plotting of the centroids for the controllers' deployment. The number of clusters generated is essentially equal to the number of controllers needed to be deployed.

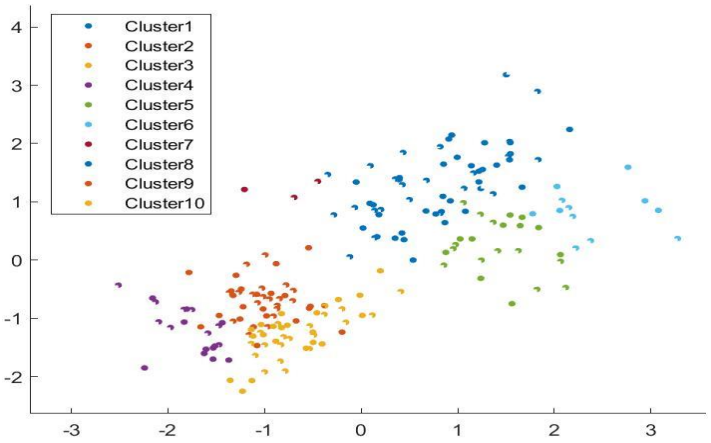

Figure 3. Cluster allocations

Reducing the average controller propagation latency is the primary goal of solving the controller placement problem (CPP), increasing reliability, and always ensuring availability in the network infrastructures. Analyzing various distance metrics after each iteration in Table II reveals that the distance between the switches and controllers is minimized. This shows that deploying more controllers increases the capacity of the controller and decreases the average propagation latency between the clusters in the network architecture as in Figure 4.

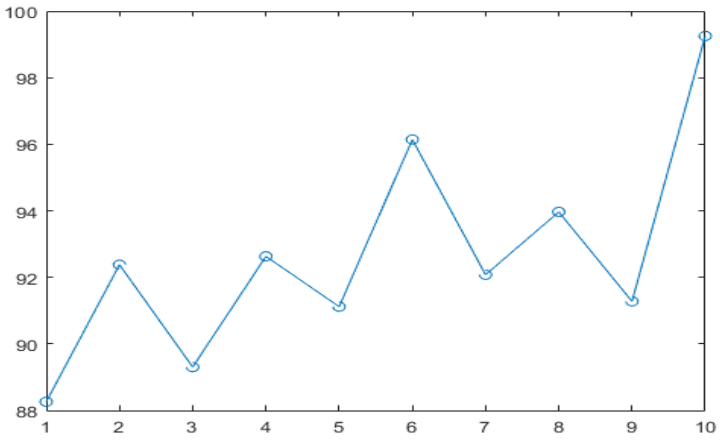

Figure 4. Propagation latency between the clusters

To confirm the above statement, we used the same distance metric in Table 1, although the number of replicates was minimized as shown in Table 2.

Table 2. Distance with fewer replicates

Replicate	Iterations	*The total sum of distances*
1	11	88.2602
2	8	92.3789
3	9	89.3037
4	6	92.6302
5	8	91.1183
The best total sum of distances		88.2602

By relating the analysis in Fig. 5 it shows that the propagation delay between the controllers is very high compared to Fig. 2.

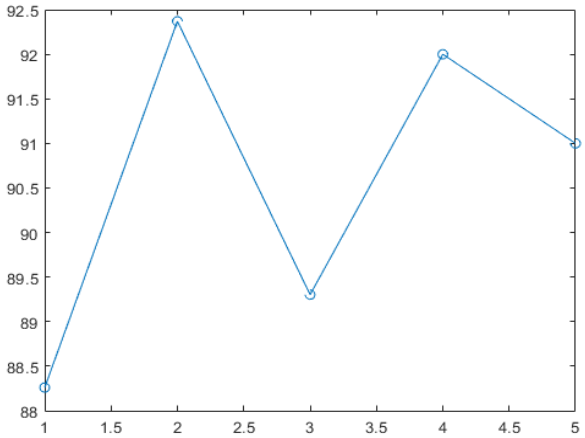

Figure 5. Propagation latency between fewer replicates

According to the literature, the switches-controller distance is a key factor that imposes fundamental limitations on the controller availability and propagation delay in the network. Looking at both analyses it is evident that, with fewer controllers deployed the capacity of the controller to handle traffic is compromised and this may increase the balanced loading of each controller in the network. But the more controllers deployed it increases the capacity of the controllers, the reliability of the network, and always ensures the availability of resources.

After obtaining the best distance metric between switches and controllers, we must specify the best location for their placement. We make use of k-*Mediods* clustering algorithms because of their robustness to plot the center of each cluster as the central location with a minimum sum of distances to the switches to place the controllers. The idea is to have a medoid as a central location that reduces network interference and delay to minimize the maximum distance metric of the switches to the closest controller C_n for better placement in the best location represented in x_n. as in Figure 6.

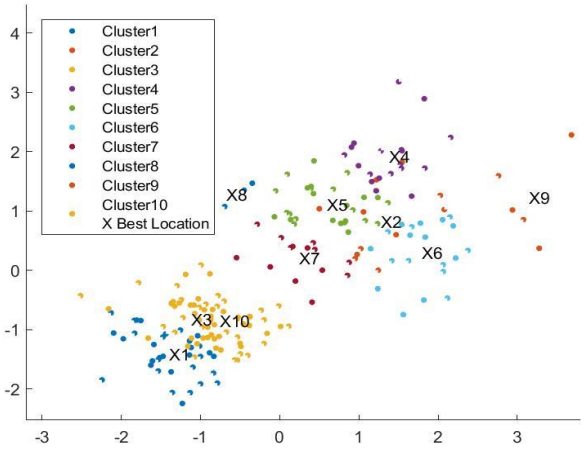

Figure 6. Best controller locations

Looking at Figure 6, we discovered that the best location for controller placement, in this case, is not fixed but was chosen randomly using the best metric as a central point of each cluster (network segment). The plotting of these best locations for placement (centroids), shows the exact locations that minimize the average latency in the design network topology where the controllers need to be placed. The plot shown in Figure 7, demonstrates the actual controller location placements whereby the optimal number of controllers, in this case, is *Cn=10*, whereby C_1 represents controller 1, C_2 controller 2, etc.

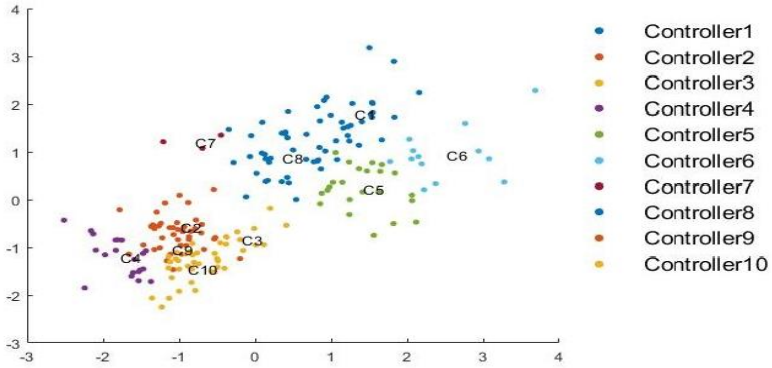

Figure 7. Controller locations

Figure 7 displays the best SDN controllers location deployment for the proposed SDN-FTM that guarantees network reliability, availability of resources always as well as best network performance.

Evaluation of the proposed FTM based on controller placement

As indicated in the previous section the deployment of controller placement addresses the issue of a single controller overseeing the network that creates a single point of failure in SDN. Utilizing more than one controller (multiple controller approach) can improve network fault tolerance, scalability, load balancing, and quality of service as well as network performance. In most cases, the deployment of a distributed control plane increases system availability and reliability, and security issues may also be improved. Furthermore, this approach affects throughput and latency performance, hence, the location of the controller (placement) inside the network is crucial to the network's functionality. We added more controllers in the proposed SDN-FTM architecture in a distributed approach compared to the one presented in the literature. Compared to the traditional SDN architecture, our proposed one can provide increased fault-tolerant features and improve independent controller management as well as throughput and latency.

Using the same OpenFlow SDN-FTM Controllers Setup in Mininet as presented in the previous section, the city-block distance metric values that were generated randomly using the k-Means algorithm are applied as shown in Table 2 to define the "x" and "y" values during the simulation for the best optimal controller placement distance between several given controllers and switches in the clusters. Additionally, we increased the number of controllers from four (4) and we have increased the IP addresses range from 172.0.0.1/10 while keeping the same number of switches and hosts as in Figure 8.

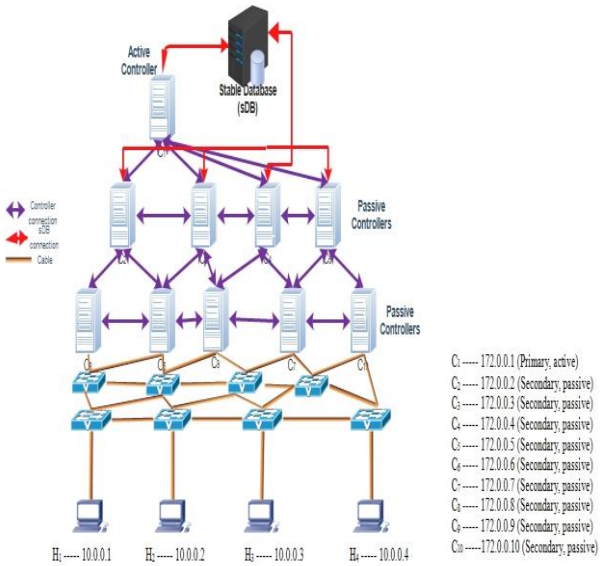

Figure 8. Architecture for medium to large-scale networks

The controller c0 in this case is chosen as the primary in this initial simulation, as shown in Figure 8 above, and is recognized and connected to all the other controllers and switches in the network in a distributed fashion. This type of connection enables synchronization whereby nodes share information about their status in the network, hence removing the single point of failure that is experienced by a centralized controller. The scenario in Figure 8, assumes the deployment of 5 Ryu's controllers and 5 OpenFlow controllers, the primary controller (active) is an OpenFlow one denoted by c1 with Ip address 127.0.0.1 and the rest are the secondary controller (passive).

Performance Analysis: In the initial stage of the simulation, using the same procedure used in the previous simulation, the initial stage switches flow table is empty, once a fault occurs on the primary controller (c1, (OpenFlow)) with IP address 127.0.0.1, assuming the controller goes down due to some network fault, error, or failure. This scenario is illustrated in Figure 9.

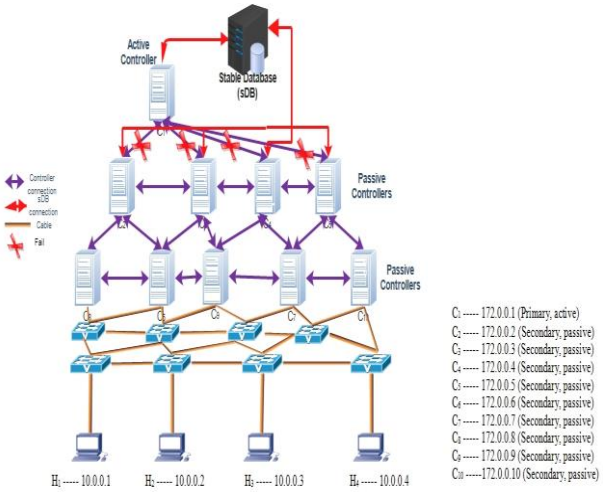

Figure 9. Controller failure in medium to large-scale networks

Using the *ovs-vsctl* command tools in Mininet as in the previous section, a new controller is selected individually controller gets the address of the newly elected primary and shares the updates. From Figure 9, the failed state indicates that the controller has failed/crashed, on the other hand, the active state indicates that the controller is functioning. To provide a summary of the performance of the individual controller during the fault tolerance, the throughput is shown in Table 3.

Table 3. Controller throughput analysis

Controller name	Type of Controller	Controller Protocol	*Status*		*IP Address*	*Packet drop (%)*	*Run-time/ms*
c1	OpenFlow	TCP	DOWN	DOWN	127.0.0.1	-	-
c2	Ryu	TCP	Active	UP	127.0.0.2	0	1.066
c3	OpenFlow	TCP	Passive	UP	127.0.0.3	0	1.01
c4	Ryu	TCP	Passive	UP	127.0.0.4	0	1.084
c5	OpenFlow	TCP	Passive	UP	127.0.0.5	0	1.032
c6	Ryu	TCP	Passive	UP	127.0.0.6	0	1.037
c7	OpenFlow	TCP	Passive	UP	127.0.0.7	0	1.013
c8	Ryu	TCP	Passive	UP	127.0.0.8	0	1.068
c9	OpenFlow	TCP	Passive	UP	127.0.0.9	0	0.986
c10	Ryu	TCP	Passive	UP	127.0.0.10	0	1.055

The result in Table 3 is represented using graphs to give a better visual illustration of the performance of the controller as in Figure 10.

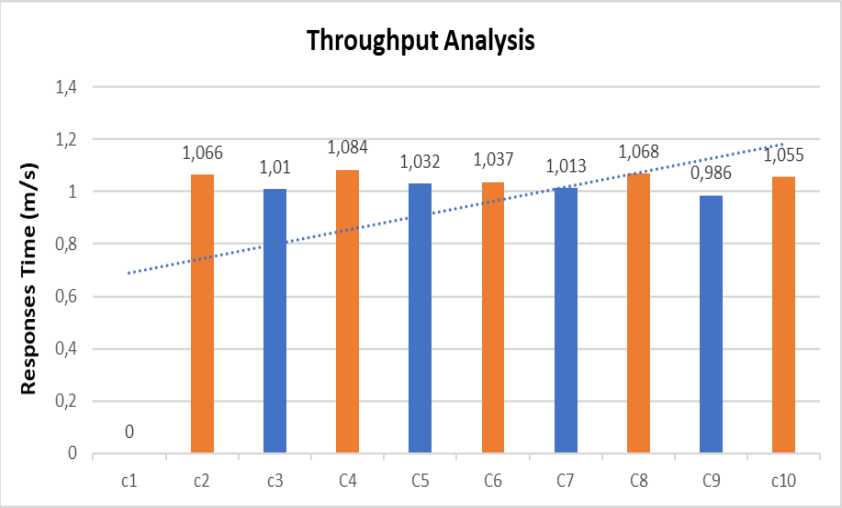

Figure 10. Controller throughput analysis

For a better understanding of the controllers' performances during the evaluation of the proposed SDN-FTM using the two scenarios during the simulation, the outcomes show that the performance of Ryu's controllers gives a better response time in case of a fault compared to the OpenFlow for every individual controller in the network. This shows that Ryu's throughput is higher than the OpenFlow, this implies that Ryu's controllers' response time during taking over of the new primary controller is faster than the one for the OpenFlow controller. This is like the simulation in the previous section, which assumed that controllers suffer in their performance. However, it is confirmed that in the proposed SDN-FTM, OpenFlow controllers suffer greatly than Ryu's. Using the round-trip time (RTT)/ms in Table 4, the researcher performed the round-trip time (RTT)/ms for Ryu and OpenFlow controllers for the proposed SDN-FTM.

Table 4. Controller latency analysis

Controller name	Type of Controller	Controller Protocol	*Status*		*IP Address*	*Packet drop (%)*	*Run-time/ms*
c1	OpenFlow	TCP	DOWN	DOWN	127.0.0.1	-	0
c2	Ryu	TCP	Active	UP	127.0.0.3	0	0.9001
c3	OpenFlow	TCP	Passive	UP	127.0.0.3	0	0.9398
c4	Ryu	TCP	Passive	UP	127.0.0.5	0	0.9011
c5	OpenFlow	TCP	Passive	UP	127.0.0.5	0	0.9379
c6	Ryu	TCP	Passive	UP	127.0.0.7	0	0.9101
c7	OpenFlow	TCP	Passive	UP	127.0.0.7	0	0.9388
c8	Ryu	TCP	Passive	UP	127.0.0.9	0	0.9001
c9	OpenFlow	TCP	Passive	UP	127.0.0.9	0	0.9398
c10	Ryu	TCP	Passive	UP	127.0.0.10	0	0.9012

To gain more understanding of the effect of latency during network fault or error for both controllers, as illustrated in Figure 11.

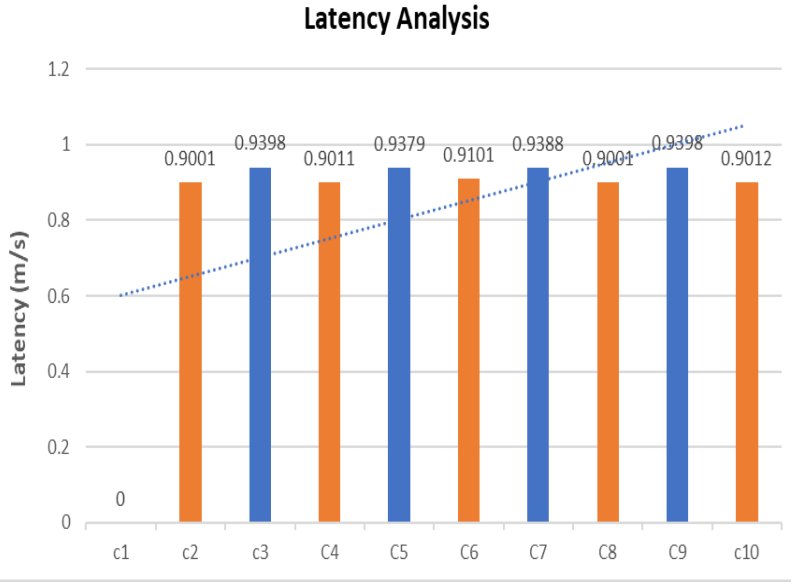

Figure 11. Architecture for medium to large-scale networks

Unlike the previous simulation, the results here are different. Ryu's controller for the proposed SDN-FTM in distributed deployment when using placement and an increased number of controllers shows a low latency rate compared to the traditional models. This implies that the performance evaluation for the latency analysis shows better performance of the system compared to the OpenFlow controllers. These results may be due to the use of multiple controllers and the selection of the best placement.

This allowed the proposed SDN-FTM to increase its performance as well as the sharing of packets sent among controllers during the selection of the primary in the event of a fault. In terms of the performance measure, the result can confirm that the proposed SDN-FTM had the best performance in terms of using Ryu controllers during fault tolerance. From the simulations, it is evident that the proposed SDN-FTM shows better performance when using Ryu's controllers than OpenFlow. This is the same phenomenon that was also discovered during the evaluation of the total latency analysis for the proposed method in both topologies

shown in the previous section. Figure 12 shows the network overview after using the *dump* command in the Mininet command line to see all the nodes and their IP address in the simulated network using Ryu's controllers.

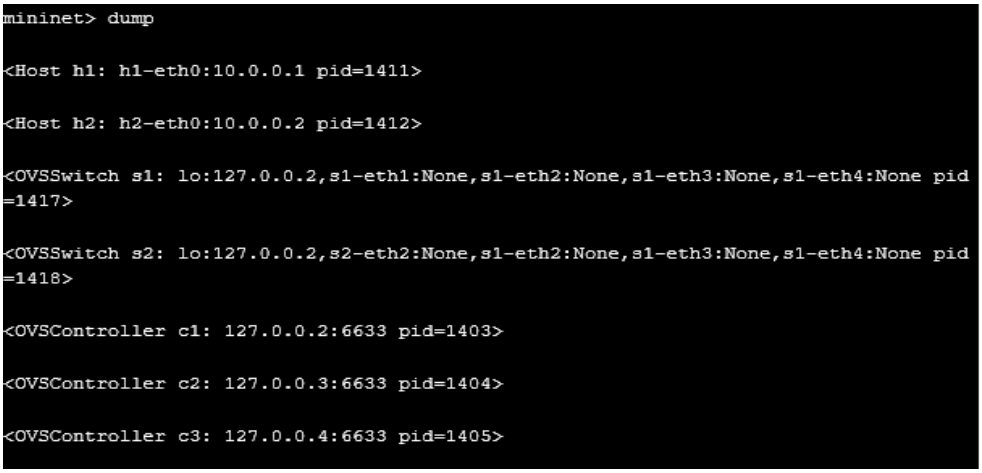

Figure 12. Network overview after election of new primary

Comparison of the Proposed SDN-FTM with other methods

The permanence of the proposed method was compared with various recent methods based on the method used, advantages, tools used, and limitations of each method. These methods include; Shortest Path First (SPF) [12], OpenFlow-based restoration [13], CORONET (Controller-based Robust NETwork) [14], Bidirectional Forwarding Direction (BFD) [15], and Dynamic Protection with the Quality of Alternative Paths (DPQoAP) [16], as shown in Table 5.

Table 5. Comparative analysis of proposed SDN-FTM with other methods

Ref	Method	*Advantage*	*Tools*	*Limitation*
[12]	Shortest Path First (SPF)	The approach works effectively by determining the path reactive for failed flow.	POX controller emulated through Mininet	Neglect the time needed for flow insertion in the switch flow table. Hence not suitable for large-scale network
[13]	OpenFlow-based restoration.	OpenFlow protocol specifies the communication between switches and controllers if the port is not available and the switch promptly sends a notification to the controller.	Mininet	The approach later calculates the new backup and installs new rules in the forwarding tables concerning the switch which takes time in real-life scenarios.
[14]	CORRONET (Controller-based Robust NETwork).	Speedy recovery for faults and scale to large-scale scale-	NOX controller emulated through Mininet.	Unreliable for the control panel and recovery time might not meet acceptable

		network.		standards.
[15]	Bidirectional Forwarding Direction (BFD).	Integrates with OpenFlow Fast-Failover group feature and switch recovery time is within 50ms.	Mininet.	Non-optimal path choice (reacting just to local failures) and resource utilization (memory overhead) especially for the large-scale network.
[16]	Dynamic Protection with the Quality of Alternative Paths (DPQoAP).	The approach considers both the quality of alternative paths and existing faults within the network.	Mininet command and Linux bridge command.	Decrease in DPQoAP intervals reduces the packet loss which leads to high cost of the system.
Ours	FTM-SDN	achieve fast, carrier-grade recovery times (less than 50 milliseconds) with multiple controllers.	MATLAB and Ryu's controller emulated through Mininet.	

The proposed SDN-FTM introduced various SA incorporate in four controllers that produced packet-in messages using network statistics while being monitored by SA and increased ten controllers with placement to test the throughput and latency performances of the controllers using Mininet. The results of the simulations show that the proposed SND-FTM has a higher throughput rate and lower latency when using multiple controllers in a distributed manner. When compared to various works in the literature such as in [9], [17], [10], and [18], one can see that the throughput performance of the proposed method is twice as high with Ryu controllers than with OpenFlow controllers. The performance of the proposed SDN-FTM improves much better than the previous framework presented within the framework of this study in the literature.

D. Conclusion

This work presented the development of an SDN Pattern Framework able to handle faults and failures in SDN network infrastructures. The flexibility of the framework to insert patterns as Drools rules in the controller shows the capability of the controller to guarantee properties and handle incidents. Moreover, the controllers' performances during the evaluation of the proposed SDN-FTM using the two scenarios during the simulation, show that the performance of Ryu's controllers gives a better response time in case of a fault compared to the OpenFlow for every individual controller in the network. This shows that Ryu's throughput is higher than the OpenFlow, this implies that Ryu's controllers' response time during taking over of the new primary controller is faster than the one for the OpenFlow controller.

Furthermore, the integration of four controllers on the proposed method produced packet-in messages using network statistics while being monitored by SA and increased ten controllers with placement to test the throughput and latency performances of the controllers using Mininet. The results of the simulations show

that the proposed SDN-FTM has a higher throughput rate and lower latency when using multiple controllers in a distributed manner.

Finally, in future work, we intend to use and extend the automated reactiveness of the framework, and to also evaluate for Byzantine Fault Tolerance and Service Function Chaining (SFC) the development of the associated S&D patterns.

E. Acknowledgment

The authors acknowledged the financial support and resources made available by the department of Network and Support, Walter Sisulu Univeristy, Potsdam Campus, East London, South Africa.

F. References

[1] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, " A survey on software-defined networking with multiple controllers," *Journal of Network and Computer Applications,* vol. 103 pp. 11-15, 2017.

[2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research challenges for traffic engineering in software-defined networks," *IEEE Network,* 2016.

[3] B. Isong, R. R. S. Molose, A. M. Abu-Mahfouz, and N. Dladlu, "Comprehensive review of sdn controller placement strategies," *IEEE Access,* vol. 8, 2020.

[4] 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.

[5] A. B. Sapkota, B. B. R. Dawadi, C. Shashidhar, and R. Joshi, "Multi-Controller Placement Optimization Using Naked Mole-Rat Algorithm over Software-Defined Networking Environment," *Hindawi Journal of Computer Networks and Communications* vol. 2022, p. 18, 2022, doi: https://doi.org/10.1155/2022/3145276.

[6] J. Zhao, H. Qu, J. Zhao, Z. Z. Luan, and Y. Gu, "'Towards controller placement problem for a software-defined network using affinity propagation," *The Institution of Engineering and Technology* vol. 53, no. 14, pp. 928–929, 2017.

[7] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An Effective Approach to Controller Placement in Software Defined Wide Area Networks," *IEEE International Conference on Communications (ICC)* pp. 1-14, 2016.

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

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

[10] N. S. Radam, S. T. F. Al-Janabi, and K. S. Jasim, "Multi-Controller Placement Optimization in SDN by the hybrid HSA-PSO Algorithm," *Computers,* vol. 11, no. 11, 2022.

[11] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A K-means-based Network Partition Algorithm for Controller Placement in Software Defined Network," *IEEE ICC 2016 - Communications Software, Services and Multimedia Applications Symposium,* 2016.

[12] J. Ali, G.-M. Lee, B.-H. Roh, D. K. Ryu, and G. Park, "Software-defined networking approaches for link failure recovery: a survey," *Sustainability* vol. 12, no. 10, p. 4255, 2020.
[13] K. Phemius and M. Bouet, "Implementing openflow-based resilient network services," *In 2012 IEEE1st International Conference on Cloud Networking (CLOUDNET),* pp. 212–214 2012.
[14] H. Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster, "Coronet: Fault tolerance for software-defined networks.," *In: 2012 20th IEEE International Conference on Network Protocols (ICNP),* pp. 1-2, 2012.
[15] Y. Yu *et al.*, "Fault management in software-defined networking: A survey," *IEEE Commun. Surveys Tutor,* vol. 21, no. 1, pp. 349–392, 2018.
[16] B. Yamansavascilar, A. C. Baktir, A. Ozgovde, and C. Ersoy, "Fault tolerance in SDN data plane considering network and application based metrics," *J. Netw. Comput. Appl.,* vol. 170, no. 102780 2020.
[17] 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.
[18] S. Lange *et al.*, "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks," *IEEE Transactions on Network and Service Management-Special Issue on Efficient Management of SDN and NFV-Based Systems,* pp. 1-14, 2015.