Docker Swarm-based SDN Multi-Controller Architecture for Enterprise Networks

Bilal Babayigit
Department of Computer Engineering
Erciyes University
Kayseri, 38039 Turkey

Mohammed Abubaker
Department of Computer Science
Palestine Technical College – Deir El-Balah
Gaza, Palestine

ABSTRACT
Software-defined network (SDN) is a new paradigm in computer networking that aims to simplify network management and to enable agile network evolvement by decoupling control and data planes using a single centralized controller. However, one of the weaknesses in SDN is the use of a single centralized controller, as it is unable to handle the flow of data processing and it is vulnerable to a single-point failure, especially as the network grows larger. A promising solution to address this problem is the use of multi-controller system. However, it is a critical factor for network performance and reliability to ensure the high availability of the controllers and their forwarding devices in multi controller architecture. Failures in communication between controllers and forwarding devices can lead to network downtime and service interruptions, which makes the high availability a fundamental requirement for enterprise networks. To solve this high availability issue in operations of SDN and to eliminate the single points of failure of multiple connections, this paper proposes a highly available multi controller system for SDN that uses a cluster of Docker containers by utilizing Swarm mode to build and manage this cluster. The proposed scheme is designed and implemented on a real system, and its functionality is validated. The result shows that by using this high availability approach, the single points of failure are overcome and the SDN multi-controller management is enhanced.

General Terms
Computer Networks, Data centers, Data Management, Computer Science and Engineering.

Keywords
Software-Defined Network, Multi-Controller System, High Availability, Swarm Mode.

1. INTRODUCTION
Availability is the amount of time of an online operational service without interruption, and it is generally expressed by the percentage of uptime achieved per year [1, 2]. Today uptime for most social services like Facebook, Twitter, YouTube, etc. fall between 99% and 100%. Particularly, in the real world, even 99.99% equates to 52 minutes and 33 seconds of downtime per year [3]. While even these popular social services have such high availability, infrastructure systems need to have higher accessibility. But it is difficult to capture 100% availability. Because as the uptime percentage goes up, to capture high availability becomes cost prohibitive or exponentially more expensive. Also, there can be a collapse in performance of the servers because of a sudden spike in traffic or a sudden power outage, or even the servers can be malfunctioned regardless of the applications hosted in the cloud or a physical machine. These situations are unavoidable. Hence, infrastructure devices and applications require agile networks instead of complex and heterogeneous environments, and to guarantee the service integrity and continuity they require highly available and scalable network architectures.

To satisfy highly available, agile, and flexible network requirements, a novel networking paradigm named as Software-Defined Networking (SDN) is recently emerged [4–6]. The brain of the SDN is the centralized controller named as SDN controller. One of the issues in the SDN controller architecture is the single points of failure that negatively impact the overall availability of the network. To handle these issues, improve SDN availability, and enable continuously processing controller, deploying multi-controller SDN architecture [7–9] can be used as a disaster plan. However, controller failures will cause disconnections among the controllers and the switches. Therefore, these failures can cause high availability problems. The motivation of this study is to overcome the high availability issues of multi-controller SDNs. For this purpose, a distributed system consists of multi-controller and Docker [10] working together can be used. The structure that will manage this distributed system is Docker Swarm [11].

Docker Swarm is a tool for creating and managing a cluster of Docker machines, in which a group of Docker engines are connected together to form a network of many Docker nodes that can be represented as a single virtual system. In Docker machine, Swarm mode functionality is built into the Docker engine, which can be enabled to con-vert the individual Docker machines into Swarm mode. The Swarm mode architecture in Docker includes, among other Docker nodes, a Swarm manager node that essentially initializes and controls the entire swarm and ensures that services on Docker nodes run properly.

The main contributions of this paper can be summarized as follows:

- This paper proposes a cooperative scheme that utilizes a combination of multi-controller and Docker-Swarm mode for SDN environments.
- The proposed scheme is designed to enhance the management and high availability of multi-controllers in SDN, using a new working container to replace the collapsed controller.
- The orchestration of multi-controllers is achieved through Docker-Swarm mode, which provides advanced features for scaling, load-balancing, and fault tolerance.
- The proposed scheme is implemented and validated in a real-world environment to demonstrate its effectiveness and efficiency in improving SDN performance and reliability.

The rest of this article is organized as follows: Section 2 presents the related works in SDN. The preliminary concepts for this article are given in Section 3. Section 4 explains the proposed system and discusses the results. The conclusions are provided in Section 5.

2. RELATED WORKS

The authors [12] proposed a simulated solution to the single-point failure issue in SDN by introducing a distributed SDN using an OpenDayLight controller. This work involves forming a cluster of distributed controllers, where each controller controls its domain and can share the load within the network. However, this work used a GNS3 network simulator and was not implemented in real-world SDN networks to improve their reliability and security. Moreover, the OpenDayLight is complex to install and maintain, which can make it challenging for users to set up and operate.

Other work in [13] presented a hierarchical SDN model to reduce the load of the root controllers in comparison with the classical hierarchical model. this was done by eliminating the intra-domain information which each subordinate controller shares with its root controller. However, this work does not provide a detailed evaluation of the proposed hierarchical SDN model. Additionally, it does not discuss the impact of the proposed scheme on the performance of the network.

Other researchers [14] proposed an adaptively adjusting and mapping controllers (AAMcon) to design a stateful data plane in distributed SDN controller architecture for data center networks. However, the experiments were conducted in a simulated environment, and it is unclear how their proposed method would perform in a real-world environment.

An architecture for distribution of controllers in a SDN based on the division the SDN control plane into areas was proposed in [15]. In which, one of the areas were selected as a designated controller to maintain a global view of the network for all area controllers. However, this approach assumed that the network topology is known in advance, which may not always be the case in real-world scenarios.

3. PRELIMINARY

With the aim of making this paper more self-contained, the following describes the background settings and information required in the context of a multi-controller system for SDN using a cluster of Docker containers by enabling Swarm mode in Docker engines.

3.1 SDN controller structure

The single and multi-controller structures are given in Fig. 1. The main disadvantage of a single controller is that this structure is not able to handle the huge demands on processing when the network grows larger, especially in SDN with large scale networks. A possible solution to this problem in large scale networks is to use a multi-controller structure. Although, multi-controller structure can handle heavy traffic loads, to cope with a down scenario of any controller is not possible. Thus, the system cannot ensure and guarantee the network's high availability.

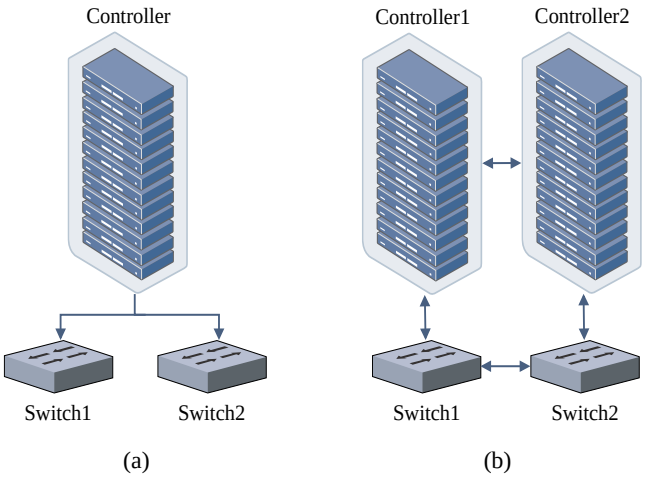

Fig. 1. The controller structure: (a) Single, (b) Multi.

3.2 Mininet Emulator

Mininet emulator [16] is used to create virtual networks. Mininet can serve to develop Python codes for any user specific topology and can also connect data plane elements to a remote controller. The switches and hosts are connected by Mininet.

3.3 Docker installation

Docker is a tool that facilitates the creation, deployment, and execution of applications through the use of containers, which bundle all the necessary dependencies to allow these applications to work efficiently in different environments. The docker host system is depicted in Fig. 2. The main elements of the docker host system are container, image, and repository.

Container: It is the name given to each of the processes that are run in isolation from each other in the Linux kernel by the Docker Daemon. The equivalent of the operating system running on the physical machine in classical virtualization is the container.

Image: It is the name given to the binary that determines the Container which is based on the operating system or other Image, the structure of the file system and its content, the program that will be run, and the content which is based on a text-based Dockerfile.

Repository: It is a structure created by a group of Images. Different Images in a repository are tagged so that different versions can be managed.

In case of not being able to reach the controller, a structure that can manage all of them is required, as different containers will be run on more than one server. The docker swarm mode is the technology used here.

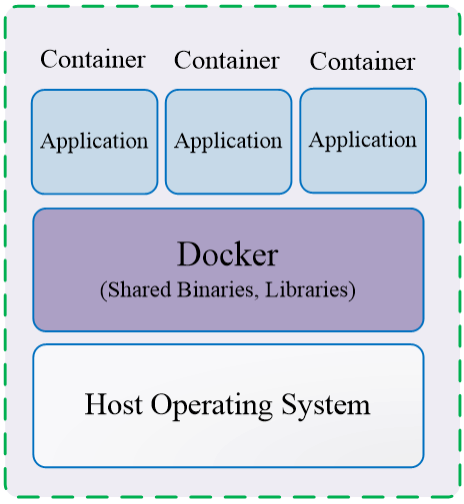

Fig. 2. The docker host system.

3.4 Docker-swarm mode

Docker swarm provides a native clustering solution for Docker containers. It converts multiple Docker hosts into a single virtual server. This enables clustering with the built-in swarm orchestration. One or more Docker nodes can be included in a Swarm. These nodes can be either managers or workers. Manager node performs cluster management operations, such as maintaining cluster state, scheduling services, and deploying endpoints in Swarm mode, while the worker node executes containers but do not make scheduling decisions [17].

4. THE PROPOSED MULTI-CONTROLLER SYSTEM AND THE RESULTS

To implement a highly available SDN system, a multi controller structure using a cluster of Docker containers by enabling Swarm mode included in Docker engines is proposed in this paper. This structure provides a solution to the challenge of building a highly available SDN system by distributing the controllers among several Docker containers. Fig. 3 illustrates the simulation topology of the proposed SDN multi-controller architecture. The distributed controllers are implemented based on Floodlight [18] to configure a multiple controller setup and to enable controllers to communicate with each other. Floodlight controller can synchronize updates with each other from the other controller. The Floodlight user interface of the topology given in Fig. 3 is shown in Fig. 4.

The proposed multi-controller structure using Docker Swarm mode provides a flexible, scalable, and fault-tolerant architecture that can ensure high availability in SDN. By using the concept of containers, it is possible to deploy multiple controllers easily and quickly. Moreover, the use of Floodlight controllers enables distributed controllers to synchronize updates with each other. The proposed architecture can provide reliable and robust communication between the controllers and their forwarding devices.

The proposed system is implemented using Mininet, a network emulator that allows users to create virtual networks. The codes of the topology created using Mininet are shown in Fig. 5. The aim of these codes is to define which switch will be connected to which controller. First, c1 and c2 controllers are specified with their IP and port numbers. Then, four switches are created and named as s1, s2, s3, s4, respectively, and connect two hosts to each one of the switches. After the codes are run, the topology is created and connected to the controllers. Although the system works, what happens if there is a problem with the controller created. The use of Docker swarm mode solves this challenge, in which the controllers are distributed among several Docker containers to ensure high availability. The structure of the docker-swarm mode used in this study is shown in Fig. 6.

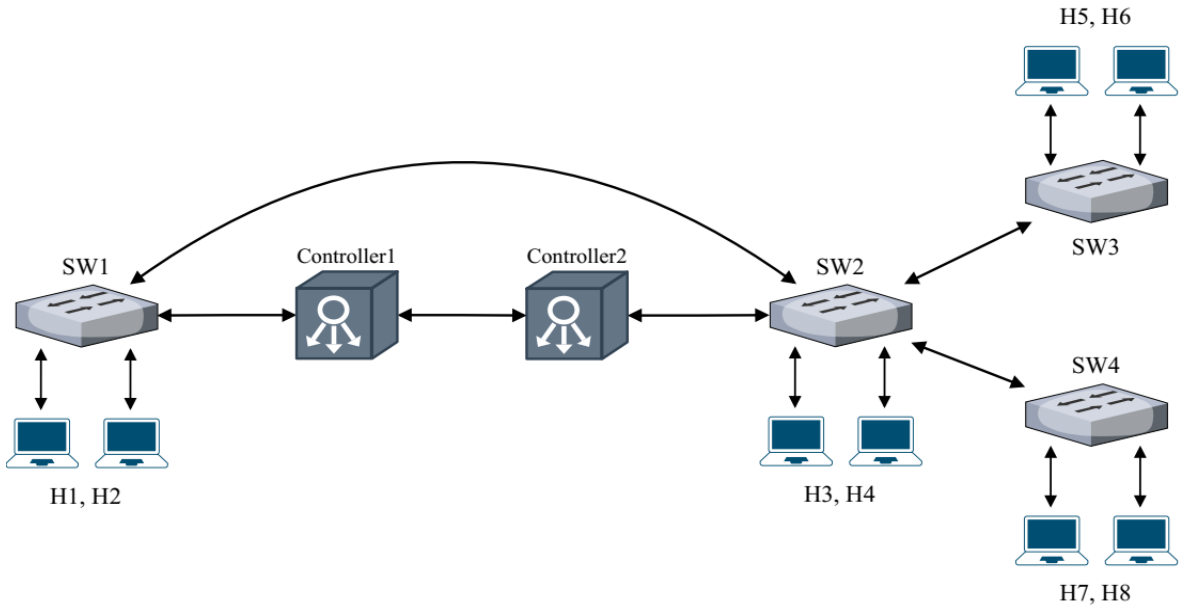

Fig. 3. The simulation topology of the proposed multi-controller system using docker-swarm mode (SW: switch, H: Host).

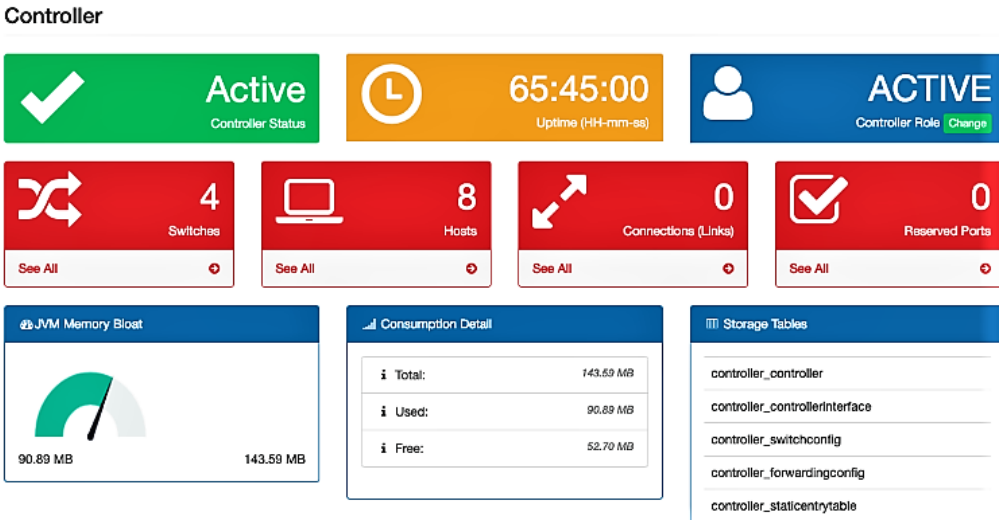

Fig. 4. The Floodlight user interface.

```
import time
from mininet.net import Mininet
from mininet.node import Controller, OVSKernelSwitch, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info

setLogLevel('info')

def addHost(net, N):
    name = 'h%d' % N
    ip '10.0.0.%d' % N
    return net.addHost(name, ip=ip)

net = Mininet(controller=RemoteController, switch=OVSKernelSwitch)
c1 = net.addController('c1', controller=RemoteController, ip='192.168.1.51', port=6653)
c2 = net.addController('c2', controller=RemoteController, ip='192.168.1.52', port=7753)

print '*** Creating Switches'
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
s3 = net.addSwitch('s3')
s4 = net.addSwitch('s4')


print '*** Creating Hosts'
hosts1 = [ addHost(net, n) for n in 3, 4 ]
hosts2 = [ addHost(net, n) for n in 5, 6 ]
hosts3 = [ addHost(net, n) for n in 7, 8 ]
hosts4 = [ addHost(net, n) for n in 9, 10 ]

print '*** Creating Links'
for h in hosts1:
    s1.linkTo(h)
for h in hosts2:
    s2.linkTo(h)
for h in hosts3:
    s3.linkTo(h)
for h in hosts4:
    s4.linkTo(h)
```

Fig. 5. The topology created using Mininet.

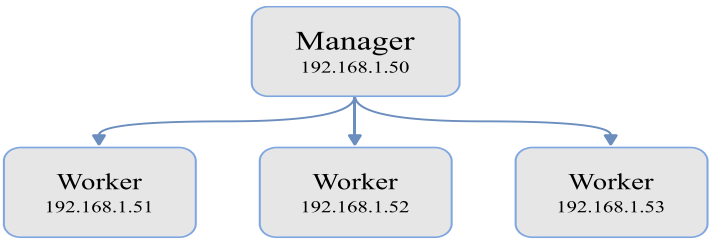

Fig. 6. The structure of the docker-swarm mode.

```
FROM openjdk:8-alphine

WORKDIR /root/controller

COPY floodlight /root/controller


ENTRYPOINT ["java", "-jar", "target/floodlight.jar"]
```

Fig. 7. The docker file.

Name	Stack	Image	Scheduling Mode		Published Ports
controller-1	-	controller:latest	replicated	1 / 1 Scale	6653:6653 8090:8080
controller-2	-	controller:latest	replicated	1 / 1 Scale	7753:7753 8091:8080
web-site	-	httpd:latest	replicated	1 / 1 Scale	180:80 4443:443

Fig. 8. The docker-swarm services

In the structure given in Fig. 3, the controllers are not installed directly on a server, but Docker is started as a service. To create the service, the controller application and the port settings are made for synchronization, and the image file is created with the help of the Docker file as given in Fig. 7. Then, by using this image, the controllers as a service in Docker Swarm mode are started as shown in Fig. 8. This makes the controller always accessible on different devices and guarantees that any node is always active. If a container cannot be accessed in any way, the manager node reinstates the new one. This offers the controllers to operate synchronously with the help of Floodlight multi-controller mode. As a result, the system continues to operate actively without any shutdowns. The implemented proposed system is given in Fig. 9.

The implemented system, as can be seen in Fig. 9, consists of 4 units of ODROID-MC1 which has 8 CPU and 2GB of DRAM. The use of ODROID-MC1 units for implementing system offers a powerful and cost-effective solution for running Docker-swarm applications. Each unit is connected to a switch device and in totally 32-core cluster structure run Docker-swarm applications. This enables the system to handle high loads and ensure high performance. As illustrated in Fig. 6, the IP for the manager ODROID-MC1 is 192.168.1.50, while the IPs for the worker ODROID-MC1s are 192.168.1.51, 192.168.1.52, and 192.168.1.53, respectively. The necessary port configurations are made on containers so that the controllers can communicate with each other. In the event of a problem in any of the workers, a backup of the failed controller occurs in the other worker (Fig. 10). For example, if controller1 fails (2nd column of Fig.10), controller1 starts again in another worker (3rd or 4th columns of Fig. 10). The reason for this is the production of 1 replica from the image of controller1. It is observed that the proposed multi-controller system using docker-swarm mode achieves a highly available network and eliminates the single points of failure. Thus, the proposed system represents a significant contribution to the field of computer networks for deploying SDN systems at scale, and it offers a powerful solution for addressing the challenges of network availability and reliability.

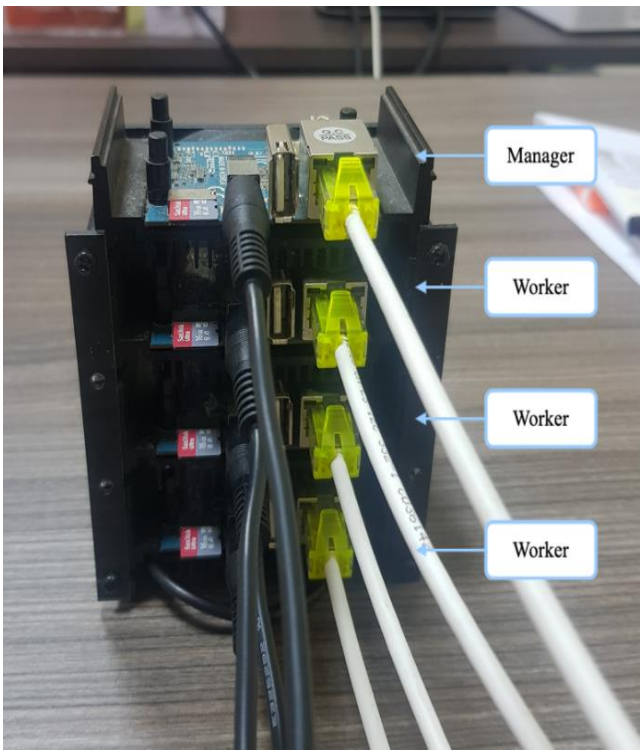

Fig. 9. The implemented proposed multi-controller system using docker-swarm mode.

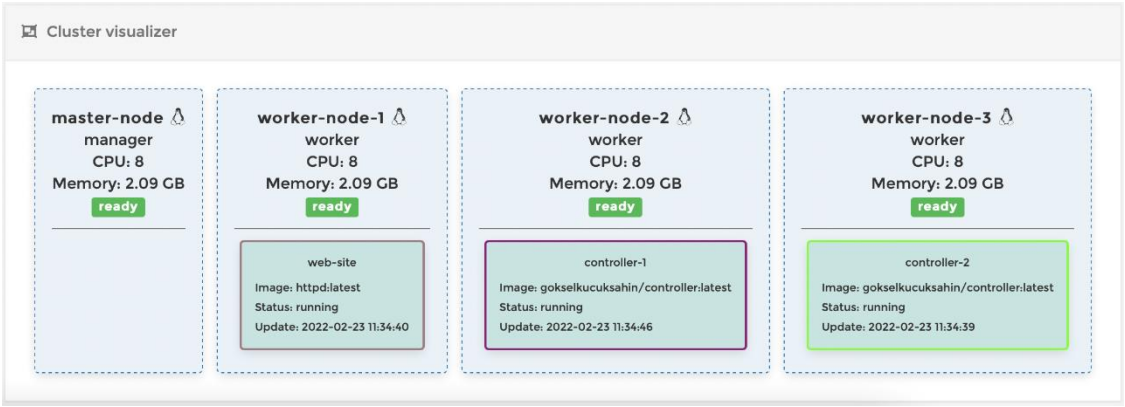

Fig. 10. Visualized cluster structure of the implemented working system

One of the strengths of the proposed system is its cost effectiveness. The utilization of ODROID-MC1 units, with their powerful specifications and affordable pricing, provides a convincing solution for running Docker-swarm applications. This combination of performance and cost-efficiency makes the system an attractive option for deploying SDN systems at high scale. Table 1. shows the components used in the proposed system.

Table 1. Components used in the proposed system with their configuration

Component	Configuration
ODROID-MC1 Units	4 units with 8 CPUs, 2GB DRAM
Cluster Structure	32-core cluster structure
Manager ODROID-MC1	IP: 192.168.1.50
Worker ODROID-MC1s	IPs: 192.168.1.51, 192.168.1.52, 192.168.1.53
Redundancy	1 replica produced from the image of controller1

Table 2. summarizes the key performance metrics and advantages of the implemented multi-controller system using Docker-swarm mode. The system exhibits high availability, ensuring that network services are actively accessible without any shutdowns. Moreover, it demonstrates reliability by eliminating single points of failure and implementing fault tolerance mechanisms. The performance of the system is optimal, offering efficient handling of network loads and scalability to accommodate growing demands.

Table 2. Analysis of performance metrics

Metric	Value
Availability	High
Reliability	High
Scalability	High
Fault Tolerance	Yes
Single Points of Failure	Eliminated
Network Load Handling	Efficient
Cost-Effectiveness	Yes

5. CONCLUSIONS

SDN has become an emerging trend in networking technologies due to its potential benefits in network reliability and flexibility. SDN controllers can control the data traffic efficiently, however, they can have a severe problem if the controllers are down. In case of controller failure, the entire network would be disrupted. The main challenge is to provide a system with high availability to solve this problem. In this paper, a system of highly available architecture for SDN has been presented and implemented to solve the problem of the single points of failure in multi-controller structure. The proposed implementation enables Docker Swarm mode in Docker engines to form a cluster of multiple Docker containers in which one controller is started as a service, and when the first controller fails, the other controller operates in a container. Thus, this architecture enables continuous operation of the controller. As a result, the proposed scheme provides a high availability control plane for multi-controller.

In the proposed scheme, during the switch of the controllers, there can be some data loss. Therefore, in the future work, it will be considered as a system with two services operating simultaneously, which has a master-slave container structure.

6. REFERENCES

[1] Hairong Sun., Han, J. J., and Levendel, H. 2003. Availability requirement for a fault-management server in high-availability communication systems. IEEE Trans Reliab. 52(2): 238–244. doi: 10.1109/TR.2003.812624.

[2] Toy, M. 2017. High Availability Layers and Failure Recovery Timers for Virtualized Systems and Services. Procedia Comput Sci. 114: 126–131. doi: 10.1016/j.procs.2017.09.028.

[3] Hiles, A. 2022. Five nines: chasing the dream? Available at http://www.continuitycentral.com/feature0267.htm [Accessed 04/15/2023].

[4] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. 2008. OpenFlow. ACM SIGCOMM Comput Commun Rev. 38(2): 69–74. doi: 10.1145/1355734.1355746.

[5] Kiriha, Y., and Nishihara, M. 2013. Survey on Data Center Networking Technologies. IEICE Trans Commun. E96.B(3): 713–721. doi: 10.1587/transcom. E96.B.713.

[6] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., and Chou, W. 2014. A roadmap for traffic engineering in SDN-OpenFlow networks. Comput Networks. 71: 1–30. doi: https://doi.org/10.1016/j.comnet.2014.06.002.

[7] Wang, H., Zhang, P., Xiong, L., Liu, X., and Hu, C. 2016. A secure and high-performance multi-controller architecture for software-defined networking. Front Inf Technol Electron Eng. 17(7): 634–646. doi: 10.1631/FITEE.1500321.

[8] Hu, T., Guo, Z., Yi, P., Baker, T., and Lan, J. 2018. Multi-controller Based Software-Defined Networking: A Survey. IEEE Access. 6: 15980–15996. doi: 10.1109/ACCESS.2018.2814738.

[9] Hu, T., Yi, P., Guo, Z., Lan, J., and Zhang, J. 2018. Bidirectional Matching Strategy for Multi-Controller Deployment in Distributed Software Defined Networking. IEEE Access. 6: 14946–14953. doi: 10.1109/ACCESS.2018.2798665.

[10] Docker. 2022. Docker: Enterprise Container Platform. Available at https://www.docker.com/ [Accessed 04/15/2023].

[11] DockerSwarm. 2022. Swarm mode overview. Available at https://docs.docker.com/engine/swarm/ [Accessed 04/15/2023].

[12] Oleiwi, W. K., and Abdullah, A. A. 2021. Design and Implementation of Distributed Controller Clustering for Solving the Issue of Single Failure in SDN Networks. Webology. 18(2): 1365–1378. doi: 10.14704/web/v18i2/web18395.

[13] Amiri, E., Alizadeh, E., and Raeisi, K. 2019. An Efficient Hierarchical Distributed SDN Controller Model. In: 2019 5th Conf. Knowl. Based Eng. Innov. IEEE; pp 553–557.

[14] Liu, W., Wang, Y., Zhang, J., Liao, H., Liang, Z., and Liu, X. 2020. AAMcon: an adaptively distributed SDN controller in data center networks. Front Comput Sci. 14(1): 146–161. doi: 10.1007/s11704-019-7266-6.

[15] Mattos, D. M. F., Duarte, O. C. M. B., and Pujolle, G. 2016. A resilient distributed controller for software defined networking. In: 2016 IEEE Int. Conf. Commun. IEEE; pp 1–6.

[16] Mininet. 2022. Mininet. Available at http://mininet.org/ overview/ [Accessed 04/15/2023].

[17] DockerDocumentation. 2022. How nodes work. Available at https://docker-docs.netlify.app/engine/ swarm/how-swarm-mode-works/nodes/ [Accessed 04/15/2023].

[18] Floodlight. 2022. Floodlight Open Flow Controller. Available at https://floodlight.atlassian.net/wiki/spaces/ floodlightcontroller/overview [Accessed 04/15/2023].