Reliability and Scalability in SDN Networks

Mohamed A. Aglan
Ain Shams University
Cairo, Egypt
offengaglan@gmail.com

Mohamed A. Sobh
Ain Shams University
Cairo, Egypt
mohamed.sobh@eng.asu.edu.eg

Ayman M. Bahaa-Eldin
Misr International University
On leave from Ain Shams University
ayman.bahaa@eng.asu.edu.eg

***Abstract* — Software Defined Networking (SDN) is a software-based solution which setup a unified control plane, as an intelligent network manager, to control the whole network devices that only forward data, instead of controlling each device alone which wastes time and effort. The conversion from the traditional networks to the SDN network has significant challenges that need a deep consideration. We consider the reliability and the scalability as critical factors that affect the SDN network. Generally, the criteria of reliability and the scalability of network are the following: the desired data rate at the data delivery, the high availability of the controller by eliminating the single point of failure, and finally the multiple topologies that are ancillary for scalable network. These criteria must be achieved in the SDN network also. We examine in details multiple controller architectures (i.e., centralized, distributed and hierarchical). We propose a framework for controller failover that increases the availability of SDN network and makes its framework more reliable and scalable.**

***Index Terms*— Software Defined Networking (SDN), Reliability, Scalability, Controller Architecture, Controller Failover.**

I. INTRODUCTION

The amount of data increases significantly, irrespective the network techniques that deal with such a big data. Especially, with the advent of IOT age and computer clouding solutions that deal with the big data [1], therefore network management is no longer something that can be avoided, but it is the core of the business needs, to operate the network services successfully, with a costly effective manner, and to facilitate the service provisioning, as fast as it can. There has been a dramatic shift in the way businesses operate. There are no physical offices or geographic boundaries constraints. Resources must now be seamlessly available at any time and in any place.

Software Defined Networking (SDN) depends on contemporary techniques which separate the control plane from the data plane that is considered a white box device that forward the data, without the ability to make a decision. Making decision is restricted only to the controller plane. This approach helps to automate multiple network devices, to innovate various programs that depend on a comprehensive view to the entire network state and to integrate the different applications easily. For instance, load balancing and routing programs can be integrated sequentially [2].

The central control architecture of the network is the default structure for the Software Defined Networking. This centralization model is useful for the future of the network. It needs to be enhanced and refined, for example by introducing virtualization, selective delegation, federation of responsibility, as well as hybrids of SDN and traditional networking [3].

The SDN network has critical issues that should be resolved. We focus on the reliability and the scalability, as two related issues that affect the performance of SDN network. SDN network is scalable if the control plane architecture will be able to operate, admin, maintain and provide the network services with the same expected quality while increasing the complexity of network. SDN network is reliable if the controller achieves the required quality while keeping its high availability. We infer that the scalability and reliability of SDN network mean a dedicated quality with any size of network at any time.

SDN, in the default structure, moves the control plane out of data plane and uses only one controller to make a unified decision. But as the size of the network scales up, the centralized architecture cannot meet the demand. The policy permits or denies the flows in the network devices in a logically centralized controller (e.g., Ryu [4], NOX [5]), which has the full and absolute control over all network elements through protocols such as OpenFlow.

The centralized SDN enables the network as a service with a fast provision in an automated manner, allows the network innovations and automates the management of large networks. On the other hand, it also results in a scalability problem due to latency (e.g., the flow setup delay), the capacity of the controller, and the control channel (the interconnection between a switch and its controller) capacity [6]. For example, according to [16], the maximum bandwidth for flow-setup payloads between the switch and the controller is around 17 Mbps, which restricts the speed of flow setups in the switch (roughly 275 flow setups per second). Furthermore, the delay for flow statistics gathering can be intolerable. (15 seconds are required to collect the flow statistics from a flow table with 78,000 flow entries [7].) Therefore, scaling the control plane of SDN is issue when SDN is applied in large networks [8]. We can use the scalability

metric of distributed system to quantify the scalability of SDN control plane. The scalability metric for distributed system is based on throughput. The distributed system is scalable if the throughput is maintained as the system scale changes.

The control plane can be deployed on one or several SDN controllers, which is running on dedicated servers [9]. The set of hardware and software components for providing of centralized network management in SDN is a control platform. The controller builds a unified network graph, which is stored in its network information base (NIB). Using network view controller, applications control network devices and data flows. That is why SDN network performance, reliability and scalability are defined by control platform characteristics.

The riskiest disadvantage of SDN is that the controller is a single point of failure and, therefore, the central SDN architecture is not reliable. The Controller failure can be caused by various reasons: hardware failure or damage, operating system crash, operating system freezing, power failure, abnormal termination of the controller process, network disconnection, memory overflow and a security breach on the controller [18], [19], [20] and [21].

In this paper we deal with control plane by OpenFlow protocol, as the one of the most promising implementations of SDN approach [10]. OpenFlow protocol contains rules that organize the relation between the control plane and the data plane. The control plane in OpenFlow detects and controls the OpenFlow switches by adding or editing or deleting a record in the flow up tables to control and route the data traffic. The controller can connect to applications by northbound interface for network monitoring, accounting and policy management. OpenFlow controller can install flows in OpenFlow switches for data flows supporting predictive, reactive, and proactive flow installation modes.

We focus on comparing the efficiency of SDN control plane with different architectures. In general, the existing SDN control plane architectures can be classified into three pure categories: centralized, distributed and hierarchical (refer to Section II for more details). A straightforward question arises of which control plane architecture best fits when a SDN network is to be deployed. On the other hand (in section III), we propose a prototype model for a failover controller that depends on a primary controller replicated with a slave controller working as a stand by controller in a separated site to become also, a disaster backup which gives a reliable and scalable SDN framework. The rest of this paper is organized as follows. In Section II, we compare the three architectures of the controller plane and abstract them in terms of flow setup and statistic collection to clarify which one of them is more suitable for the reliable and scalable network. In Section III, we analyze the different active/standby strategies to provide a controller failover, and propose a fault-tolerant control plane architecture for enterprise software-defined networks that provides the network ability to fast recovery of the control plane, with a control recovery procedure and a procedure for network view synchronization between active and standby controller instances. Finally, we summarize our conclusions in Section IV.

II. SDN CONTROLLER ARCHITECTURE

The controller layer is the middle layer between the application layer and the Data layer as shown in fig.1.

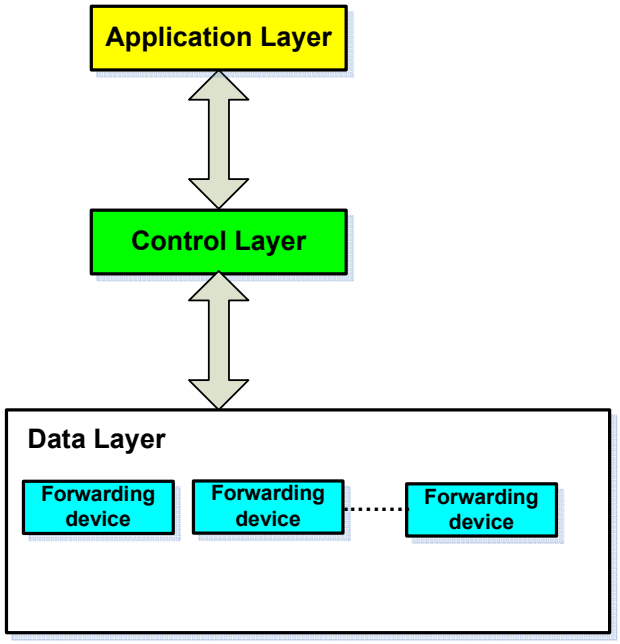

Fig. 1. SDN layers.

The SDN controller replies the requirements coming from the applications and execute a behavior over the forwarding devices, while providing network statistics up to north bound interface drivers in the various SDN applications. As discussed in Section I, the scalability of the control plane is restricted by the capability of the single controller and the processes overhead between controllers and switches. Specifically, the bottleneck for scaling the size of the SDN network lies in flow setup and statistic collection [11], [12].

It is better to build a dedicated management network which is separated from the data network. It offers reliability, interference avoidance, Ease of network planning and secured and robust environment. A dedicated management network has undeniable advantages, for areas of a network in which management is critical, for example, the backbone at a service provider or even a large enterprise. We will describe the controller architectures and show the scalability and reliability degree in each architecture. The philosophy of the separated control has a harmony with the philosophy of the separated network for management.

A. *Centralized Controller*

It is the default architecture in SDN, which depends on a single controller to control the whole data forwarding switches. All flow initialization requests are sent to this controller, and it sends a reply messages to one or many related data forwarding switches according to its source code logic. As for statistics collection, the controller will periodically send state request instruction to all switches in its control to gather resource states. Receiving the state request message, the switch will send state respond messages back to the controller. Examples of the centralized control plane are Ryu, NOX, etc. as shown in Fig. 2.

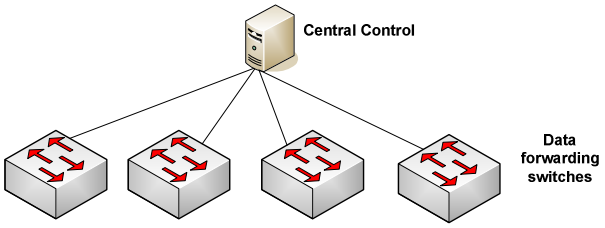

Fig. 2. Central SDN controller.

The central controller is very efficient in the small or the medium scale, but it suffers from overflow in the large scale that represents a high degree of complexity. The service level objectives that were promised are not being met due to service failure or even a sensible delay that doesn't suit the new era of computer clouding and internet of things. The central controller may be not available, due to cabling failure or system crash. It is considered in this case a single point of failure that all the system, or in other words, the whole open flow switches free from restriction, or can't forward any flow. If we make the controller as a logical firewall and it is shut down for any electrical issues, the data plane devices will not forward any flow, because they miss the leader, which degrades the system. The central controller doesn't have a plan B, as a backup or an alternative controller to take over the primary controller. The central controller doesn't have a disaster backup controller in a different site to safe the important configurations, in case of any environmental situations, such as, a huge fire that affect the central controller which make the SDN platform in a great danger. The previous factors show that the central controller in SDN network is not scalable or reliable and doesn't suit the enterprise network, or the SDN-WAN. We need to try extending the controller architecture to a more robust Architecture such as a hierarchical architecture or distributed architecture. We analyze these architectures in part B and part C.

B. *Hierarchical controller*

The traditional, three-layer hierarchical design model divides the network into core, distribution, and access layers, and allows each portion of the network to be optimized for specific functionality. We can imitate this architecture, in a collapsed model with only two layers in the hierarchical control plane (e.g., Kandoo [13]). As shown in fig. 3. , the controllers are organized in a hierarchical structure. There are two levels of the control plane. The top level is only one core controller while the second level has multiple distribution controllers. Each distribution controller controls a particular area of OpenFlow switches, and each one of them is physically separated from the others. There is no explicit connection between the distribution controllers to eliminate network loop.

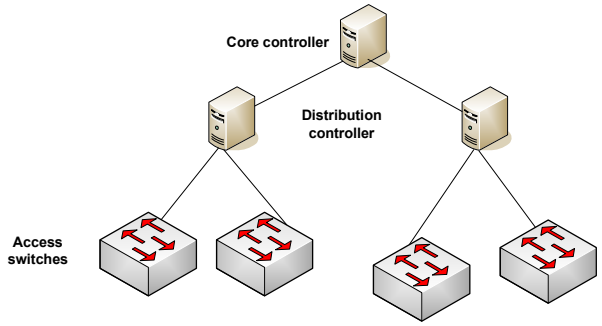

Fig. 3. Hierarchical SDN controller.

The core controller is connected to each distribution controller, as a Maestro that regulate the relation between the distribution controllers. In this scenario, the root controller role depends on the source node and the destination node. If the source node and destination node in the same area that belong to one distribution controller, the distribution controller will make a decision and add it in the OpenFlow switches for further similar requests. If the source node and destination node in different areas that belong to different distribution controllers, the distribution controllers will forward the request to the core controller, and it will make a decision and add it in the distribution controller that the sender belongs to it. The distribution controller looks like a layer2 switch that allow to forward a dedicated virtual lan and forward the unknown vlan to the top multilayer switch which work as a virtual router to the different vlans. In this case, the core controller is considered as multilayer switch. Network state in the hierarchical control architecture will be more complicated. Any physical change, due to a port failure or just a replacement has to be known to the distribution controller and to the core controller. The hierarchical controller architecture has a great capability to guarantee more scalability, because its ability to delegate some tasks from the core controller to the distribution controllers that also, guarantee a more reliability than the central controller, but the root controller is also considered, a single point of failure. This issue can be healed by another root controller. Each root controller can be the owner to a dedicated open flow switches and a backup to the other open flow switches. This scenario lead to a very traditional problem in network, which called "loop". Network loop occurs when there a closed loop wiring, or in another meaning, more than one path

between devices. The common solution to eliminate the loop is Spanning Tree Protocol which disables one port and enables the other port. The STP increases the complexity of configuration and increases the messages between the controllers in the negations and the periodic check. It is complicated to configure in SDN network. You have to deal with HP switches, as an example, that supports open flow protocol and STP protocol to achieve the required task which restricts innovation and the white box switches based on NetFPGA [14]. The hierarchical architecture has an efficient scalable and reliable solution, but it has a complexity in configuration.

C. *Distributed controller*

In the Peer to Peer distributed control plane, as shown in fig.4. , there are multiple controllers in one layer. Each controller has a full functionality and can communicate with other controllers to execute queries and inform them an update. Each controller is semiautonomous, which is commonly termed federated controllers. Each controller has a significant autonomy that controls a dedicated network device from the whole network and these controllers have a peer relationship. The controller has either the local view of the sub-network in its control or the global view of the whole network. In the P2P control plane with local view, the controller can cooperate with other controllers to execute a flow request and thus set up global flows. Specifically, when a controller receives a request which means that the flow isn't in the flow table of the OpenFlow switch, the controller checks whether the destination node is under its responsibility. If it is, the controller adds a new flow in the flow up table, in the OpenFlow switch. Otherwise, a request is forwarded to one of the federated controllers. The federated controllers repeat the previous operation until any controller can find a route to the destination node. As for statistics collection, each controller only needs to collect the statistics of the sub-network it controls. (i.e., the controller sends state request messages to the switches in its control and the switches respond with state response messages.). In the Peer to Peer distributed control plane with global view, the controllers work as integrated managers. Each controller has identical responsibilities in the network and thus can process all the flow initialization requests generated by the switches in its control area. Specifically, the controller can add a flow record in any switch regardless of whether or not they are in its control. The statistics collection in P2P with global view is exactly the same as in P2P control plane with local view. Although the p2p with global view solves the single point of failure, it suffers from the massive congestion between the various controllers. Every update in any controller is needed to be replicated to the other controllers which reduce the efficiency of controllers as managers to the data path infrastructure. The hierarchical control plane has the best scalability and reliability performance and the P2P with local view achieve second performance, with regard to the size and diameter. The centralized and P2P with global view control planes are similar as the worst scalability performance. The previous architectures have a notable advantage but they degrade network in various degrees when a burst traffic occurs and makes a flow setup delay intolerable. In the next section we propose a failover technique to eliminate the previous issues.

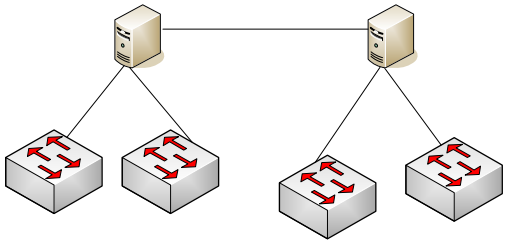

Fig. 4. Distributed SDN Controller

III. FAILOVER ARCHITECTURE

In spite of SDN advantages, one of the serious problems of SDN is that the controller is a critical point of failure and, therefore, the controller decreases the overall network availability. A Controller failure can be caused by various reasons: failure of the server where a controller is running, the server operating system failure, power outage, abnormal termination of the controller process, network application failure, network attacks on the controller and many others. The decrease in controller availability affects negatively and directly on SDN scalability and reliability. We make a prototype of a failover solution to make a robust SDN controller designed to provide uninterrupted network services. Reliability, recoverability and continuous operations are primary characteristics of a highly available solution by failover.

A. *Failover strategies for control platform*

The failover strategy depends on a backup controller connected to the network devices as will as the primary controller. They need to be coordinated by a shared data store, such as NAS storage to replicate the services and states, or by private cloud. Another optimization by making each controller has its own data store with raid system to offer hardware redundancy and make a coordination by an atomic messaging system that keeps all of the servers in synchronization, such as zookeeper Apache [15]. Final solution can be achieved by ryu controller, we can dedicate from start a master controller and a slave controller by some rules and checking the availability of the master controller by standard messages between controller and switch. If the data path doesn’t find a way to connect to the master control, and in the same time the slave control request periodically to check its role, the switch replies the slave that the data path is changed, then the slave controller should be a master

controller and the master one should be the slave. The third way is a brilliant, but it suffers from the increasing load on the switch because, the multiple ryu controllers can't connect directly in a standard way. The controllers can be active or standby mode in the control plane. An active controller directly receives and processes OpenFlow messages from network devices. A standby controller duplicates the functionality of the active controller but receives and processes OpenFlow messages from network devices only in case of active controller failure. In case of primary controller failure, the standby controller automatically takes over network infrastructure control and data flows management. This procedure is called controller failover. Controller failback is the reverse procedure to failover. This procedure is used when the primary controller is restored. Failover strategy can be classified into three types: cold standby, warm standby and hot standby. The cold standby strategy use an additional powered off controller connected to the active primary controller. The standby controller is stateless. In case of primary controller failure a standby controller is powered on and its services and applications start (including topology discovery service to form network view) from scratch. The initial configuration and policies is cloned to the cold standby controller before unloading it. The update in configuration isn't been replicated to the secondary controller, so if the primary controller fails, the secondary controller has an initial configuration that lost completely the final state of primary controller. It is an unreliable to depend on this strategy because it will take time to configure the controller to reach the final state of topology and requirement of network devices. The failover controller Warm standby strategy includes periodically primary controller state replication to standby controllers and automatically failover procedure. The warm standby strategy is usually provided by hardware and software redundancy. In case of primary controller failure the standby controller replaces a failed controller and continues to operate on the basis of its previous state. It depends on periodic replication. The lost part of the control plane state is those state changes which were between the last state synchronization procedure and the primary controller failure. Hot standby strategy includes full state synchronization of the primary and the standby controllers and automatically failover procedure. No loss of the controller state provides the minimum recovery time. State of the primary controller is replicated to the standby controller for any change in it, as an incremental replication simultaneously. In case of primary controller failure the standby controller replaces a failed controller and continues basing on the current state. The hot standby strategy is implemented by software and hardware redundancy. The failover time is very fast without any impact on user. It is clear that, the hot standby strategy is the most efficient failover technique, because it keeps the complete network state before failure.

B. Scalability and Reliability metrics according to failover strategy

1) Health good rate to measure the Percentage of Time with Good Health in the failover system.
2) Duration of Failover includes the time a system requires to recover from failures. The duration of the failover process can be a relatively short interval on hot standby strategy.
3) Maintenance time during which the primary controller is needed to be maintained.

Thus, the SDN control platform should have a robust controller with a controller redundancy degree at least one, delay in the worst case no more than milliseconds as recommended maximum time delay for services. Failover time should be as low and close to zero.

C. Failover controller requirements

To support redundant controllers, to avoid single point of failure and to ensure that the standby controller has a sufficient computing resources for network infrastructure and data flows management in case of primary controller failure, the control platform must meet the following requirements:

1) There are to be at least two controllers and independent data store in each one of them or a shared data store, as shown in fig.5.
2) Similar hardware models and software versions.
3) A dedicated network from fiber, to increase the reliability, connects the two controllers with high bandwidth.
4) Alarm system with different level of warning from critical to normal.
5) A network connects the controllers to the data plane.
6) High resources including high processor with high cash and high memory to endure the heavy work.
7) The two controllers are placed in a separated sites to achieve the disaster backup technique.
8) Each controller should be equipped with an uninterruptible power supplies (UPS).
9) Electrical power from a station in different geographical locations.
10) The network adapters in each controller should be identical and has the same protocols, speed and duplex.
11) Each controller should be placed in a good server room with a good access control and a lockable door supported with fire stopper sleeves and a good cooling system.

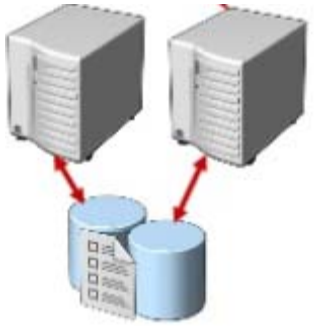

Fig. 5. Shared data store between the two controllers.

D. *Failover controller workflow*

To design a reliable and scalable system, using active/standby strategies, we should define the main states in the relation between the two controllers: a start state which occurs at the first to organize the order of launching the controllers, an operational state that a dedicated controller becomes the master controller and a standby controller waits and monitors by checking the connectivity of the primary controller and finally, failover state which a slave controller tries to replace the primary controller by taking over the control from the old primary control. In the start controller state, the primary controller and the standby controller run the configuration file for each one of them. In our case we built architecture based on ryu controller that depends on python language. It is a programming language that lets you work quickly and integrate system more effectively [17]. The configuration about the ordinary processes that make the switches forward the data from node to node in the source code for each controller is similar. The similar processes include how to deal with the incoming packets, how to deal with a new data forwarding switch or a new event in any switch connected to the controller and how to deal with errors. There are an echo process to ensure if the master controller is alive by sending hello message periodically to switches. The slave controller also follows this technique to ensure it is alive and ready to be the new master when the original master became down. Both of them has a process to check their role that can be equal or master or slave. Finally, the slave controller has an additional process that deal with its step when the master is down that enables the slave controller to be master. In the operational control state, or in other words, the normal steady state, the master controller, based on OpenFlow protocol, organize and control the OpenFlow switches and supervise the data flows. In the primary controller failure mode, the standby controller detect failure based on heartbeat messages. After timeout, the standby controller becomes the primary controller according to a particular rule define this issue. The major task of the temporary master controller is giving back the network services and applications and turn on the network interface that enable it to supervise the network devices. We propose for more optimization in SDN network to provide data store that share data to the primary controller and the standby controller by read only and repetitive snapshots in a shared network information base. The standby controller can read it by network operating system capabilities.

There are multiple messages or information should be exchanged or shared between the two controllers in a middleware to ensure the fast recovery from the failure of the master controller operations between of them. These agents include:

1) Breakdown message from the master if it would shutdowns abnormally to guarantee the real time synchronization between the two controllers.
2) Network parameters such as topological data that includes information about controller Ip address, link bandwidth, mac address of switches and host Ip addresses.
3) Information about QOS such as packet loss rate, packets transmitted, packets received and latency.
4) Heartbeat message to monitor the operational status of the controllers. It is sent periodically from one controller to another to guarantee it is not deed.

E. *Failover controller design*

In order to avoid single point of failures in the control platform we propose a design of the control platform, as shown in fig.6.

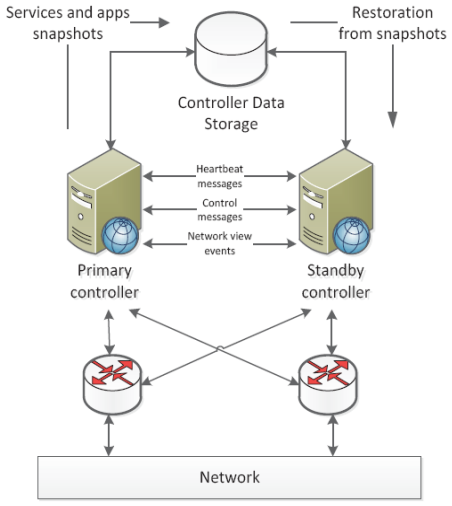

Fig.6. Failover SDN Controller

Two controllers in two different sites includes master controller and slave controller with a shared data storage and separated controller network from data network. We used Mininet to build a complex network virtually. To run each controller, you should run the python script for each controller, then run the Mininet script to build the network. You can test the network connectivity by pinging the different computers under the umbrella of master controller. When you disconnect the master controller by simply cancelling the application you can notice the role changing that make the slave controller to

be master controller and vice versa. If you ping between two computers successfully under the new supervision of the new master controller, it means your network in a good form.

IV. CONCLUSIONS

The analysis of the different topologies from central and distributed to hierarchical and our proposal of using failover technique showed that how complex to depend on one model only to achieve a reliable and scalable SDN network. Although failover guarantee a high availability that make the SDN is more reliable and recoverable in the minimum time, it is not the best scenario that fits a high scalability needing because it is in all times depending on one controller. The hierarchical architecture is a good competitor that guarantees a high scalable network but at the expense of the reliability. The core controller is single point of failure which degrades the scalability. To achieve the reliability and scalability together, we propose a framework to study in the future based on a hybrid model from hierarchical model with failover technique in the core layer to become one master core controller that owns group of data forwarding switches and a slave reference for the other data forwarding switches. The second master controller is master for the switches that has a slave role from the first master and is slave for the switches that has a slave reference from the first master. At any time, if one of them failed the other controller can be the central master for the whole data forwarding switches temporarily. The two core switches is connected directly so it must be configured with a spanning tree protocol to eliminate loop.

REFERENCES

[1] V. K. Jain and S. Kumar, "Big Data Analytic Using Cloud Computing," in IEEE. 2nd International Conf. Advances in Computing & Communication Engineering, Dehradun, India, 2015, pp. 667-672.

[2] D. Krutz, "Software-Defined Networking: A Comprehensive Survey," in *Proc. of the IEEE Volume: 103, Issue: 1, Jan. 2015,* pp 14-76

[3] Open Networking Foundation, "ONF SDN Evolution", *Version 1.0 ONF TR-535 2016*

[4] Ryu. [Online]. Available: *https://osrg.github.io/ryu/*

[5] N. Gude et al., "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review, vol. 38, no. 3*, pp. 105–110, 2008.

[6] R. Ahmed and B. Raouf, "Design considerations for managing wide area software defined networks," *IEEE Commun. Mag., vol. 52, no. 7*, pp. 116–123, 2014.

[7] A. R. Curtis et al., "Devoflow: scaling flow management for high performance networks," *ACM SIGCOMM Computer Communication Review,* vol. 41, no. 4, pp. 254–265, 2011.

[8] H. Yang, J. Ivey and G. F. Riley, "Scalability comparison of SDN control plane architectures based on simulations" in proc. 36th ANNU, Performance Computing and Communications Conf, San Diego, 2017, pp. 1-8

[9] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks*," ONF White Paper*, 2012.

[10] Open Networking Foundation, "OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01)," 2009.

[11] A. R. Curtis et al., "Devoflow: scaling flow management for highperformance networks," *ACM SIGCOMM Computer Communication Review*, 2011, pp. 254–265.

[12] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Commun. Mag., vol. 52,* 2014, pp. 116–123.

[13] S. H. Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in Proc. ACM Workshop on Hot Topics in Software Defined Networks (HotSDN'12), Helsinki, Finland, 2012, pp. 19–24

[14] https://netfpga.org/site/#/

[15] https://zookeeper.apache.org/doc/r3.2.2/zookeeperInternals.html

[16] Hemin Yang and George F. Riley, "Traffic engineering in the peer-to-peer SDN" *IEEE 2017 International Conference on Computing, Networking and Communications (ICNC)*

[17] https://www.python.org/

[18] Ebada Essam-Eldin ElDessouky, HasanDAĞ, Ayman M. Bahaa-Eldin, "Protecting Openflow Switches against Denial of Service Attacks" ,2017 12th International Conference on Computer Engineering and Systems (ICCES), 479-484, 2018, IEEE

[19] Mohammad Mousa, Mohmed Sobh, Ayman M. Bahaa-Eldin, "Software Defined Networking concepts and challenges", 2016 11th International Conference on Computer Engineering & Systems (ICCES), 79-90, 2016, IEEE

[20] Mohammad Mousa, Mohmed Sobh, Ayman M. Bahaa-Eldin, "Autonomic Management of MPLS Backbone Networks using SDNs", 2017 12th International Conference on Computer Engineering and Systems (ICCES), 169-174, 2018, IEEE

[21] Reham ElMaghraby, Nada Abd Elazim, Ayman M. Bahaa-Eldin, "A Survey on Deep Packet Inspection", 2017 12th International Conference on Computer Engineering and Systems (ICCES), 188-197, 2018, IEEE