Hindawi
Journal of Computer Networks and Communications
Volume 2019, Article ID 6808693, 9 pages
https://doi.org/10.1155/2019/6808693

Research Article

Generic Controller Adaptive Load Balancing (GCALB) for SDN Networks

Wael Hosny Fouad Aly [1,2]

[1]*College of Engineering and Technology, American University of the Middle East, Kuwait*
[2]*College of Engineering and Technology, Arab Academy for Science Technology & Maritime Transport (on leave), Alexandria, Egypt*

Correspondence should be addressed to Wael Hosny Fouad Aly; drwaelaly@gmail.com

Received 28 March 2019; Accepted 31 October 2019; Published 1 December 2019

Academic Editor: Liansheng Tan

Fault tolerance is an important aspect of network resilience. Fault-tolerance mechanisms are required to ensure high availability and high reliability in different environments. The beginning of software-defined networking (SDN) has both presented new challenges and opened a new era to develop new strategies, standards, and architectures to support fault tolerance. In this paper, a study of fault tolerance is performed for two architectures: (1) a single master with multiple slave controllers and (2) multiple slave controllers. The proposed model is called a Generic Controller Adaptive Load Balancing (GCALB) model for SDNs. GCALB adapts the load among slave controllers based on a GCALB algorithm. Mininet simulation tool is utilized for the experimentation phase. Controllers are implemented using floodlights. Experiment results were conducted using GCALB when master controller is taking the responsibility of distributing switches among four and five slave controllers as a case study. Throughput and response time metrics are used to measure performance. GCALB is compared with two reference algorithms: (1) HyperFlow (Kreutz et al., 2012), and (2) Enhanced Controller Fault Tolerant (ECFT) (Aly and Al-anazi, 2018). Results are promising as the performance of GCALB increased by 15% and 12% when compared to HyperFlow and by 13% and 10% when compared to ECFT in terms of throughput and response time.

1. Introduction

Today's Internet suffers from not having software programmability, and hence, it is a challenging process to update and program networks. In traditional networks, there is not any underlying programming capability, and hence, distributed algorithms cannot promise any consistent behavior. In order to provide network programmability, software-defined networks (SDN) separate data and control planes. Although there is an amount of research conducted in the area of SDN research, most of the research performed so far focuses on exploring SDN as a programming-based technology, rather than studying the fault-tolerance aspects [1–4]. SDN is an interesting research topic, but there has been always a confusion regarding many SDN concepts such as SDN architecture, multiple SDN networking planes, and interaction between layers through interfaces.

Figure 1 shows the SDN architecture which is composed of several abstraction layers, interfaces, and well-defined planes.

The *data plane layer* is the layer that is responsible for handling data packets sent by the end user through network devices that are responsible for traffic forwarding. The forwarding table and medium access control are used for routers and switches. IP forwarding for the unlabeled packets is performed through the forwarding table in the data plane [5].

The *control plane layer* is the layer that is responsible for deciding on the way packets should be handled and forwarded at network devices to properly cross the network. The main purpose of the control plane is for synchronization/update of forwarding tables.

The *application plane layer* is the layer that is responsible for network applications and services. In SDN architectures,

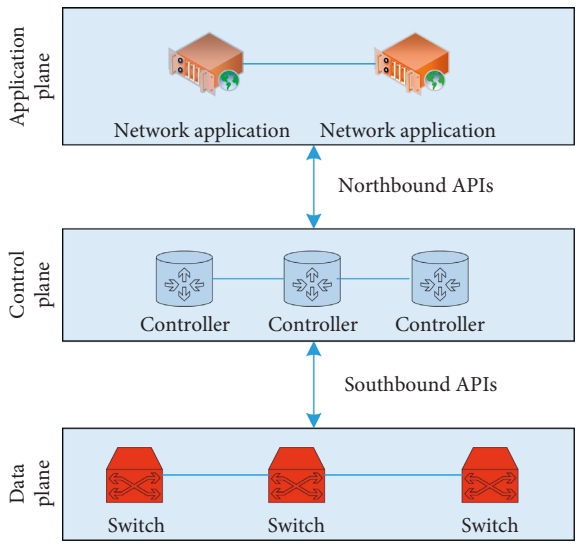

Figure 1: SDN architecture.

controllers are centralized components. Controllers are responsible for translating the SDN applications' requirements through Northbound Application Programming Interfaces to the SDN data layers. Controllers also are responsible for managing the packet flow processes, which facilitate programming capabilities of the network to be centrally controlled. This will ease the management of the entire network and its devices efficiently regardless of the complexity of the underlying infrastructure. SDN has a powerful programming characteristic. This eases the packet forwarding capabilities based on the network needs [1]. Management of the communication between the controllers and the switches is governed through OpenFlow standard [6, 7, 8]. OpenFlow manages and controls the communication flow between the controller and network entities [9]. SDN has many multicontroller architectures. Bilal et al. [6] described different types of SDN architecture and their associated implementations. The authors classify multicontroller SDN architecture in two broad categories: (1) logically centralized where there is a super controller and a set of slave controllers and (2) logically distributed architectures where there is a set of slave controllers that communicate through message passing. The authors did not offer optimal solutions to the SDN fault tolerance.

The SDN architectures have two main interfaces that use either APIs and/or protocols to enable communication among pairs of different SDN planes. The APIs are the Southbound and Northbound APIs [10]. The Southbound API is responsible for communication interfaces among data and control planes. OpenFlow is considered the de facto standard for this type of communication. The Northbound API is a communication interface between the control plane and the application plane. There is no standard to handle the Northbound API, and hence, the development of network applications for SDN has not been developed [11]. Most of the northbound implementations use REpresentational State Transfer- (REST-) based API due to its powerful platform efficient language [12]. Although a lot of research has been developed for SDN network, fault tolerance is still a challenging problem and still open for future investigation. Fault tolerance has been a very challenging problem, and having an optimal solution is an NP problem.

In this paper, a new fault tolerance algorithm is proposed. The algorithm is called *Generic Controller Adaptive Load Balancing* (GCALB). GCALB is tested for two different architectures. The first architecture is based on super controller and a set of slave controllers, and (2) the second architecture is based on a set of slave controllers that communicate through message passing. GCALB is compared with two reference models. The first reference model is based on HyperFlow. HyperFlow [13] is an application developed on the top of the NOX controller, to enable logically centralized multicontroller architectures. The HyperFlow-based network contains three components: (1) a control layer, (2) a forwarding layer, and (3) an application layer. The control layer contains multiple NOX controllers that are working cooperatively. In the forwarding layer, the switches are connected to the nearest controller. The second reference model has the Enhanced Controller Fault Tolerant (ECFT) [14]. ECFT uses only delays among switches along with their associated controllers in order to compute the load for each neighbor controller.

The paper is organized as follows: Section 2 has the fault-tolerance literature review. Section 3 has the reference models. Section 4 discusses the GCALB architecture. Section 6 has the conclusion and future work.

2. Fault-Tolerance Literature Review

This section has a short SDN fault-tolerance literature survey for different SDN architecture planes. The section starts with discussing fault tolerance for the data plane and then describing the fault tolerance for the control plane and then finalizes by explaining the fault tolerance for the application plane.

2.1. SDN Data Plane Fault Tolerance. SDN data plane fault tolerance inherits problems that already exist in current traditional network architectures. Due to the static nature of traditional networks, these approaches can achieve accepted performance in cases of link and node failures. In rapidly changing networks such as SDN, algorithms that are responsible for detection and recovery are redesigned to cope with the dynamic behavior of the SDNs. In the literature, fault-tolerance architectures are classified into (1) reactive and (2) proactive approaches [15]. In the reactive approaches, alternative paths are calculated after the fault occurs. In the proactive approaches, resources and backup paths are held in a programming fashion before the fault happens. If the fault occurs, the programmed logic starts to act immediately and recover the network. In case of approaches that deal with failure detection, the high availability of the data plane plays an essential role to maintain the required communication from source to destination. Resistance to failures is obtained in the data plane through the design and analysis of the topology when failures occur.

The design of an alternative path is performed. Failure recovery requirement could be to guarantee a recovery process within a certain amount of time [16]. Protection is a reactive approach while restoration is a proactive approach. In restoration, an alternative path is only established after the failure occurrence. Resources are not allocated before the occurrence of the failure, and the paths are dynamically preassigned. Protection process guarantees alternative paths and their allocation priori failure occurrence, which does not add more processing to recover from failure. In the restoration process, an additional processing is needed to recover from failure, which makes it not scalable.

2.2. SDN Control Plane Fault Tolerance. Control plane resistance to failure is a requirement for the proper operation of SDN. The controller is a crucial component that must be able to process all required traffic commands at all times. There are many alternatives to enhance the control plane in SDN control plane fault tolerance. An approach is to duplicate a controller on a different network. In the case of failure, the duplicated controller takes over to manage the situation. Another approach is to provide controllers with self-healing mechanism to immune controllers from possible attacks. In this approach, self-heal time should be relatively fast [17, 18]. The controller channel must be reliable at failures due to loss of switch connectivity, or error due to the communication protocol among the controller and the associated devices. In SDN networks, these problems could affect the network and might lead to several failures. In order to cope with these issues, controller redundancy [19, 20] and path backup are required to be utilized. Controller placement and assignment are two major problems when it comes to controllers within an SDN domain [21]. Improper controller assignment might lead to two main problems: overutilization and underutilization of controllers. Overutilization occurs when a small number of controllers are utilized to handle a higher volume of traffic compared to their capacities. Controllers are hence overloaded, and an increase downtime could occur that could affect the network performance. Underutilization of controller problem occurs when more controllers are available to deal with less load. In this case, controllers are not sufficiently used. Optimal controller placement algorithms are used to deal with the controller placement issue [22]. To avoid a single point of failure, multicontroller architecture approaches are always recommended to increase resiliency [23]. Controllers' resiliency should not conflict with controllers' consistency [24].

2.3. SDN Application Plane Fault-Tolerance Support. The application plane is the layer that has the network applications and services as well to make requests for network functions offered by both the control and the data planes. In classical networks, application layer holds security, management, and monitoring facilities. The application layer allows commercial applications to modify/update the network behavior to provide customer services. APIs are used for third-party developers to use network applications for network operator [25].

In order to develop reliable SDN applications, debugging and testing tools help in fixing software bugs as service evolves. To ensure the quality of software network troubleshooting, debugging and testing are essential.

3. SDN Architecture Fault-Tolerance Issues

In this section, first, SDN fault-tolerance highlights and issues are explored. State-of-the-art research efforts focusing on such fault-tolerance issues in SDN is also provided.

3.1. OpenFlow Fault-Tolerance Support in SDN. OpenFlow fault tolerance usually provides faster recovery when it comes to service failure [9]. OpenFlow was developed to support communication among forwarding tables. The OpenFlow protocol provides an abstraction of forwarding tables through the OpenFlow group table concept. OpenFlow protocol communicates with the controller, which modifies packets forwarding policies that facilitate forwarding table programmability. A proper controller functionality manages the control logic of its associated switches. The latest versions of OpenFlow protocols support a master-slave configuration at the control layer in order to increase the overall resiliency [6, 7, 8].

3.2. Fault Tolerance in HyperFlow Reference Model. The research studies discussed by Amin and et al. [13, 26] use HyperFlow to provide control plane resiliency. HyperFlow is a distributed event-based control plane, which is logically centralized but physically distributed. HyperFlow enables scalability, and it ensures centralized network control. Amin et al. argue that HyperFlow offers a scalable solution for control plane resilience in SDN-enabled network.

3.3. Fault Tolerance in ECFT Reference Model. Aly et al. [27–29] proposed a feedback control theoretic techniques and Petri net modeling to implement fault tolerance for controllers. The work gave promising results, but the feedback control theoretic techniques have put extra burden on the controllers. The ECFT [14] introduced load balancing at controller's failure, and the proposed ECFT model focuses on balancing the load among other neighboring controllers. The proposed ECFT uses only delay among switches and their associated controllers in order to compute the load for each neighbor controller and sort the slave controllers accordingly.

4. Proposed Model Design

In this section, two architectures are discussed. The first architecture is based on a single master controller that controls a set of subclusters. Each subcluster has a slave controller where a set of switches are connected to it. The second architecture is group of peer controllers that communicate with each other through message passing. In this

section, the proposed *Generic Controller Adaptive Load Balancing Technique for Fault Tolerance (GCALB)* model is discussed for both these architectures.

4.1. Master-Slave Controller Architecture. In the first architecture, the *GCALB* aims at balancing the load among multiple controllers in a multicontroller SDN paradigm based on throughput and response time metrics. *GCALB* takes into consideration different metrics such as data loss, utilization, and delay among different slave controllers and their associated switches in order to improve the overall response time and throughput. In this architecture, GCALB assumes that there is a master controller and n slave controllers connected to it. As depicted in Figure 2, the master controller is connected to three slave controllers where each slave controller is connected to four switches.

Figure 3 shows a master-slave architecture with three slave controllers. Master controller is connected to four slave controllers. Each slave controller has a set of switches associated with it. Each controller within a cluster has a unique ID called CTRL_ID. The controller with CTRL_ID = 0 is considered the master controller. The master controller is responsible for updates the switch-controller forwarding table located at the master controller, routes incoming packets, periodically collects the load of each controller at the cluster, and assigns the failed controller's switches to a suitable backup controller. Each controller within a cluster updates the master controller current load, and the number switches associated with the salve controller. The master controller decides where the switch is assigned based on GCALB strategy.

4.2. Slave-Slave Controller Architecture. In the second architecture, there is no master controller. Only slave controllers are there to conduct message passing to send and receive information about their current load and utilization in addition to data loss.

Figure 4 has three slave controllers in a slave-slave controller architecture. Each slave controller has four switches allocated to it.

Figure 5 has four slave controllers connected to each other in a slave-slave architecture. Each controller is assumed to have four switches connected to it. In the proposed GCALB model, slave controllers have the capability to periodically monitor the master controller to make sure that the master controller is still alive. If a slave controller senses that the master controller is down, then a slave controller is elected from the list of tentative master controller list. The master controller tentative list is sorted based on the slave controller current status metric. In the GCALB, the current status metric measured at each slave controller is defined by the current remaining capacity and rate of data losses. Figure 6 contains the GCALB algorithm. The GCALB algorithm does the following operations: (1) computes the summation of the overload load, (2) gets the desired ratio for each available slave controller choice, and (3) computes the minimum of the difference between computed current load ratio and the desired load ratio for every slave controller. The slave controller that is corresponding to the minimum value is chosen to associate the switch load to it. This process is repeated for the overall switch allocation procedure.

5. Simulation Testbed and Results

The simulation testbed used is as follows:

(i) Simulator: Mininet version 2.2.2
(ii) Controller: Floodlight version 1.2
(iii) Switch: OpenFlow version 2.0.2
(iv) OS: Ubuntu version 14.04
(v) RAM: 32 GB
(vi) Processor: Intel® Core™ i7 5500U CPU 2.4 GHz
(vii) Traffic: hping3

Simulation is conducted to test two different scenarios using the GCALB algorithm discussed in Figure 6. (1) The first scenario uses the custom topology depicted in where there are four controllers in different domains: master-controller domain and slave-slave domain as shown in Figures 2 and 4. (2) The second scenario uses similar fashion to the first scenario but in five slave controllers' context as shown in Figures 3 and 5.

5.1. GCALB Distributing among Four Slave Controllers. A set of experiments has been conducted for distributing the load among four slave controllers (shown in Figures 2 and 4) with various possible required ratios as shown in Table 1.

Four switches are allocated to each slave controller. It is assumed that there is one host allocated for each switch. Switches form a network of connectively. All the slave controllers are connected to all other switches. Master controller provides fault tolerance to the control plane. That is it, when the current master controller fails, it synchronizes the state among the slave controllers through allowing all of them to access updates published by all other modules in the controller efficiently. This is implemented through accessing the next master controller.

In addition, it runs a master controller election process, in order to enable modules to perform role-based programming in a distributed system with multiple controllers. Simulation is used to compare the GCALB model with the HyperFlow [26] reference model. The HyperFlow fault tolerance technique directs the failed controller switches to the closest controller without considering the controller's current workload. This leads to different network problems such as the cascading failure problem, packet loss, and packet delay. The proposed GCALB reduces the effect of these problems by distributing the controller's load among other controllers whenever a failure occurs. The super controller considers a controllers' capacity before assigning the switches to it to avoid a cascading failure. Four different floodlight controllers are utilized in four terminal windows with the same IP address and different ports on port no 6000, 6001, 6002, and 6003. Default master controller will be CTRL_ID = 1 but after starting 2 slave controllers. While performing experiments, we tested floodlight controller by

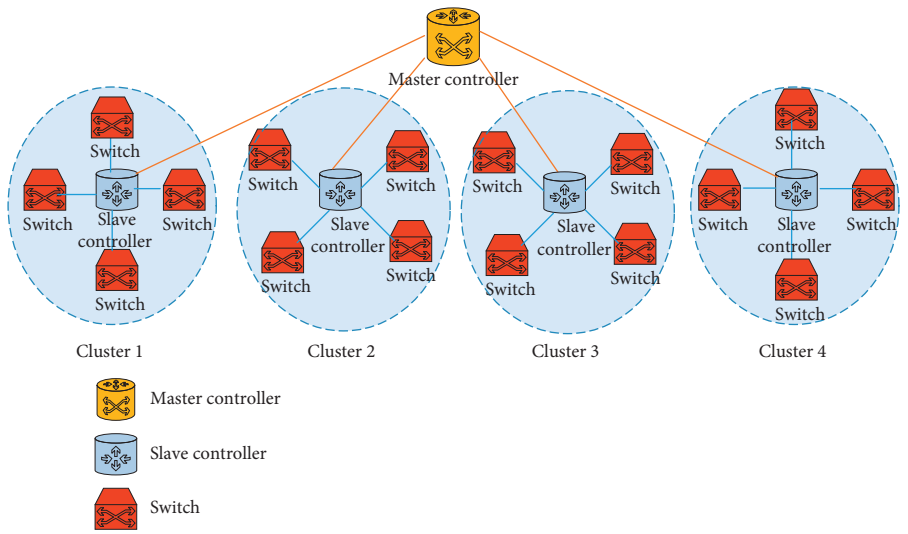

Figure 2: Four slave controllers connected to a master controller in a master-slave architecture.

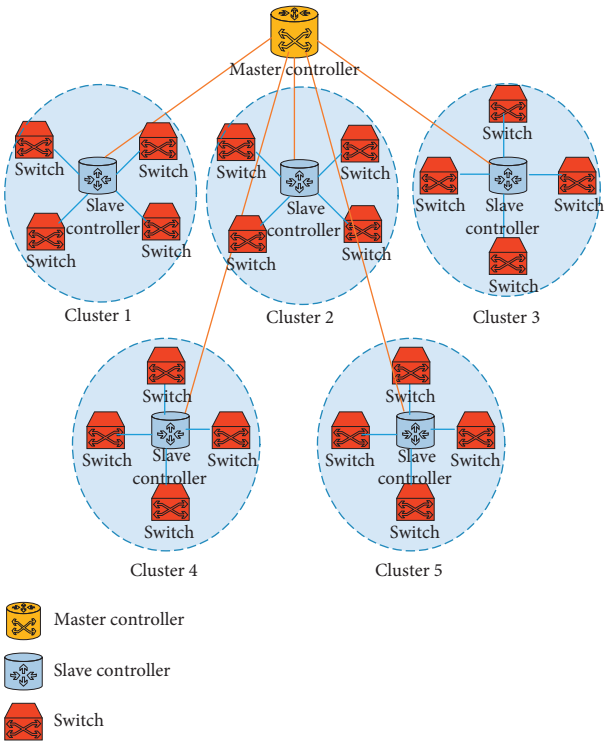

Figure 3: Five slave controllers connected to a master controller in a master-slave architecture.

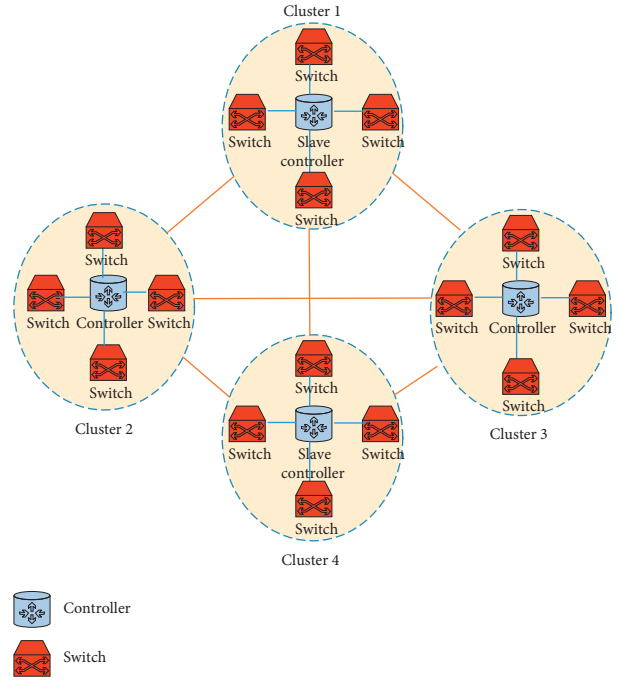

Figure 4: Four slave controllers connected in slave-slave architecture.

varying the number of controller nodes and packet arrival rates.

Floodlight is tested in both the mode of cbench through the throughput and latency as well to get throughput and response time, respectively. Figures 7–11 show the results of applying GCALB load distribution algorithm that is used by the master controller through assigning workload to slave controllers with different ratios.

In this section, a discussion about distributing the load among five slave controllers for the topologies discussed in Figures 3 and 5. Another set of experiments is conducted for distributing load among three controllers as shown in Table 2.

Results are promising since the required goal is achieved to obtain the ratios requested by the master controller. Results for the second scenario implementing GCALB algorithm are shown in Figures 12–16.

Concerning the throughput, packets are sent by varying packet arrival rate and CIDs from 1 to 4 in one set of experiments and then the other set of experiments using CTRL_ID from 1 to 5. GCALB is compared with two reference algorithms, the HyperFlow algorithm and the ECFT.

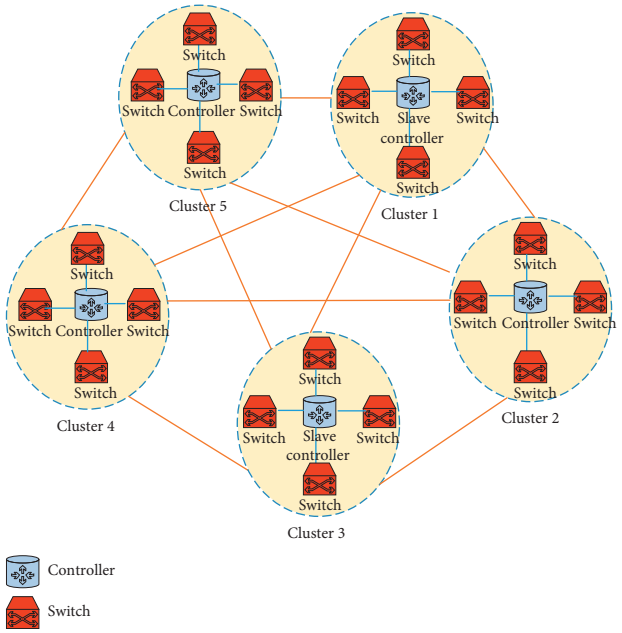

Figure 5: Five slave controllers connected in slave-slave architecture.

```
int GCALBN (int N)
{
    //Assume N is the total number of
    available choices

    TotalRatio=Σ_i^N CurrentRatio_i

    for (i=0; i<choices; i++)
    {
        DesiredRatio_i=Ratio_i/TotalRatio

        ComputedRatio_i=
            [CurrentRatio_i+1]/[TotalRatio+1]

        For each available choice "i",
                compute the minimum "Value" for

        Value=ComputedRatio_i–DesiredRatio_i
    }
    Choice=i;//i that gave the min Value
    return Choice;
}
```

Figure 6: GCALB load distribution algorithm.

Table 1: Experiments among four controllers.

Exp	X	Y	Z	W
1	1	2	3	4
2	1	3	5	7
3	1	4	7	9
4	2	4	6	8
5	3	6	9	12

GCALB outperforms the reference algorithms by 15% when it was compared to the HyperFlow reference model and by 13% when it was compared to ECFT reference model in

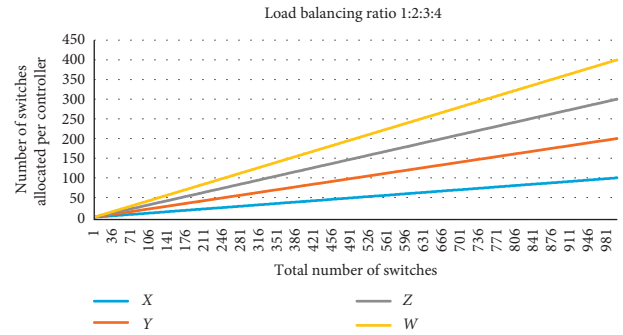

Figure 7: Load balancing for four slave controllers with ratios 1 : 2 : 3 : 4.

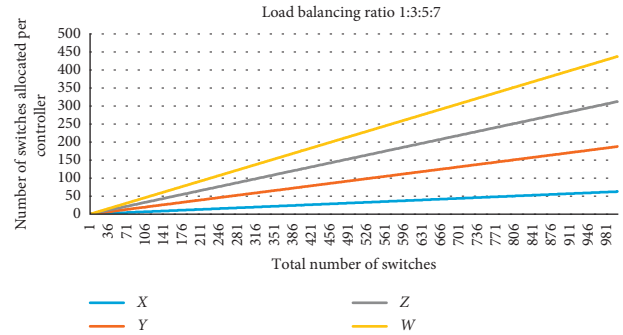

Figure 8: Load balancing for four slave controllers with ratios 1 : 3 : 5 : 7.

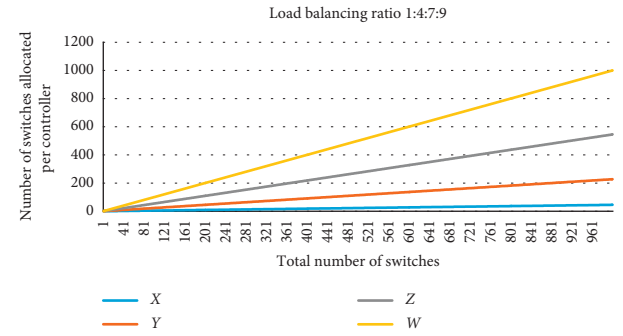

Figure 9: Load balancing for four slave controllers with ratios 1 : 4 : 7 : 9.

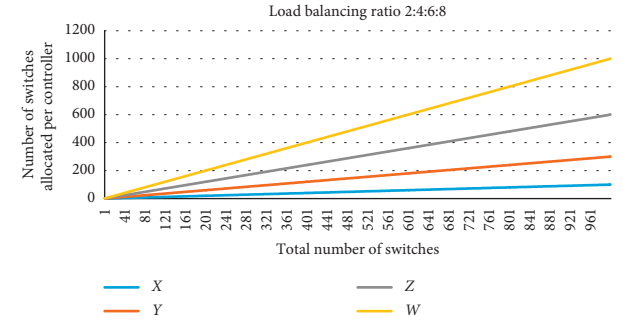

Figure 10: Load balancing for four slave controllers with ratios 2 : 4 : 6 : 8.

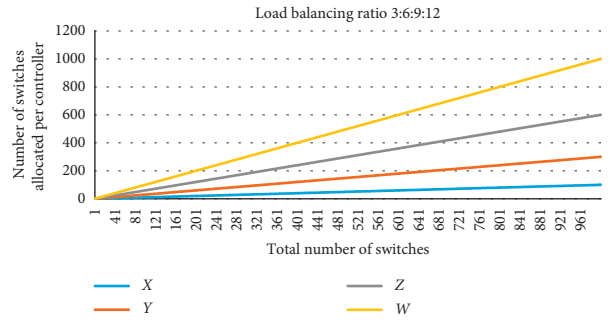

Figure 11: Load balancing for four slave controllers with ratios 3 : 6 : 9 : 12.

Table 2: Experiments among five controllers.

Exp	X	Y	Z	W	Q
1	1	2	3	4	5
2	1	3	5	7	9
3	2	4	6	8	10
4	1	4	7	10	13
5	1	5	8	14	19

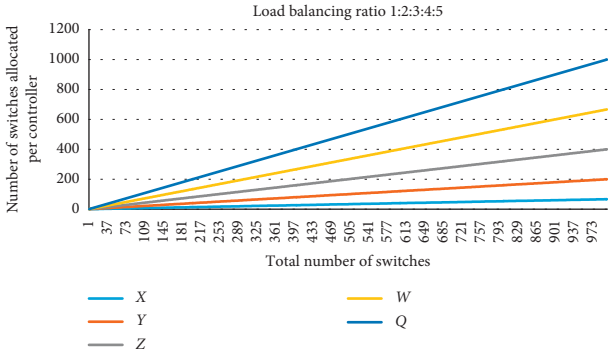

Figure 12: Load balancing for five slave controllers with ratios: 1 : 2 : 3 : 4 : 5.

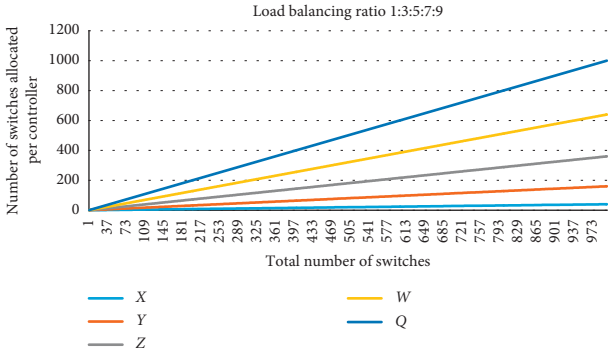

Figure 13: Load balancing for five slave controllers with ratios 1 : 3 : 5 : 7 : 9.

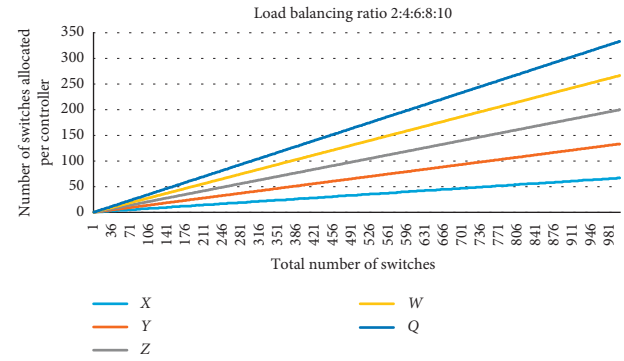

Figure 14: Load balancing for five slave controllers with ratios 2 : 4 : 6 : 8 : 10.

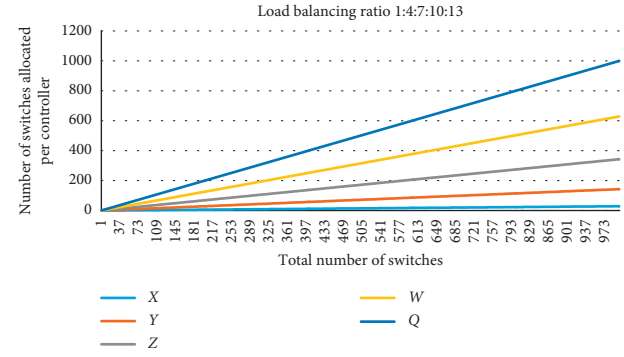

Figure 15: Load balancing for five slave controllers with ratios 1 : 4 : 7 : 10 : 13.

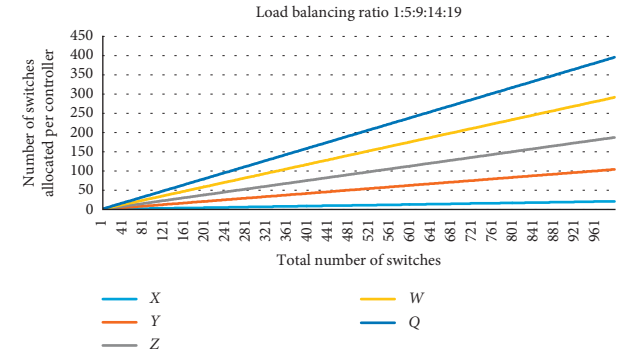

Figure 16: Load balancing for five slave controllers with ratios 1 : 5 : 9 : 14 : 19.

terms of throughput in four slave controller scenario as depicted in Figure 17 and five slave controller scenario in Figure 18.

Concerning the response time, packets are sent to floodlight controllers by varying its packet arrival rates from 200 packets/sec to 5000 packets/sec. Packet arrival rate is observed to increase while a decrease in the response time was observed. The overall response time for the GCALB is outperforming the response time for the HyperFlow by 10% as depicted in Figure 19, and the GCALB outperformed the ECFT by 12% as depicted in Figure 20.

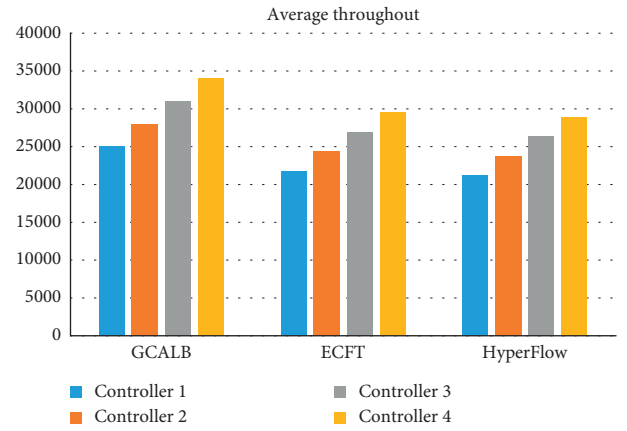

Figure 17: Comparison among average throughputs for GCALB, ECFT, and HyperFlow for four slave controller model.

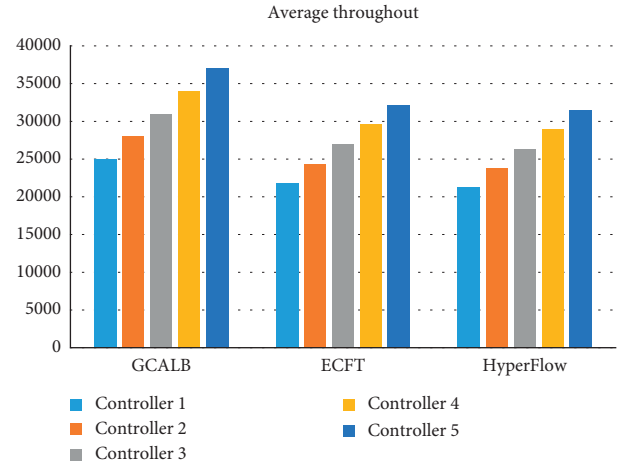

Figure 18: Comparison among average throughputs for GCALB, ECFT, and HyperFlow for five slave controller model.

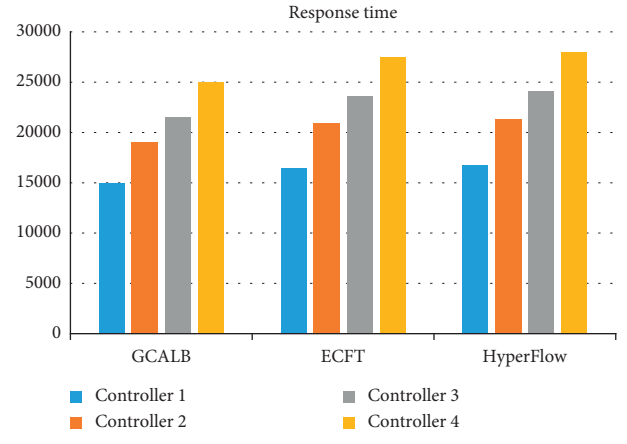

Figure 19: Average response time for GCALB, ECFT, and HyperFlow among four controllers.

6. Conclusion and Future Work

The paper proposes a generic controller adaptive based on load balancing mode. The proposed model is called a Generic Controller Adaptive Load Balancing (GCALB) model for SDNs. In the proposed GCALB model, there are two main operations. (1) The first operation is that the slave controllers have the capability to periodically monitor the master controller to make sure that the master controller is still alive. If a slave controller senses that the master controller is down, then a slave controller is elected from the list of tentative master controller list. The master controller tentative list is sorted based on the slave controller current status metric. In the GCALB, the current status metric measured at each slave controller is defined by the current remaining capacity and rate of data losses. (2) The second operation is to maintain a fault-tolerance mechanism. Two architectures were studied: (1) a single master with multiple slave controllers and (2) multiple slave controllers. GCALB adapts the load among slave controllers based on a GCALB algorithm. Mininet simulation tool is utilized for the experimentation phase. Experiment results were conducted using GCALB when master controller is taking the responsibility of distributing switches among four and five slave controllers as two case studies. Throughput and response time metrics are used to measure performance. GCALB is compared with two reference algorithms: (1) HyperFlow [13] and (2) Enhanced Controller Fault Tolerant (ECFT) [14]. Results are very promising as the performance of GCALB increased by 15% and 12% when compared to HyperFlow and by 13% and 10% when compared to ECFT in terms of throughput and response time.

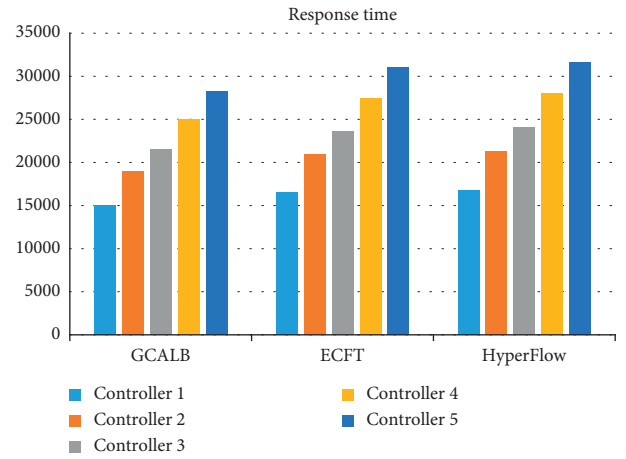

Figure 20: Average response time for GCALB, ECFT, and HyperFlow among five controllers.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The author declares that there are no conflicts of interest regarding the publication of this paper.

References

[1] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking:

past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: from concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[4] T. Bakhshi, "State of the art and recent research advances in software defined networking," *Wireless Communications and Mobile Computing*, vol. 2017, Article ID 7191647, 35 pages, 2017.

[5] G. Warnock and A. Nathoo, *Alcatel-Lucent Network Routing Specialist II (NRS II) Self-Study Guide: Preparing for the NRS II Certification Exams*, John Wiley & Sons, Hoboken, NJ, USA, 2011.

[6] O. Bilal, M. B. Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *Computer Networks and Communications Journal*, vol. 2016, Article ID 9396525, 8 pages, 2016.

[7] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[8] W. Braun and M. Menth, "Software-defined networking using openflow: protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[9] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Openflow: meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, 2013.

[10] E. Haleplidis, J. H. Salim, J. M. Halpern et al., "Network programmability with forces," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1423–1440, 2015.

[11] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013.

[12] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, Addison-Wesley Professional, Boston, MA, USA, 1st edition, 2015.

[13] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, February 2013.

[14] W. H. F. Aly and A. M. A. Al-anazi, "Enhanced CONTROLLER Fault Tolerant (ECFT) model for software defined networking," in *Proceedings of the 5th IEEE International Conference on Software Defined Systems (SDS)*, Barcelona, Spain, April 2018.

[15] J. Chen, J. Chen, F. Xu, M. Yin, and W. Zhang, "When software defined networks meet fault tolerance: a survey," in *Algorithms and Architectures for Parallel Processing*, G. Wang, A. Zomaya, G. Martinez et al., Eds., pp. 351–368, Springer International Publishing, Cham, Switzerland, 2015.

[16] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: meeting carrier grade requirements," in *Proceedings of the 2011 18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*, pp. 1–6, Chapel Hill, NC, USA, October 2011.

[17] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "A proposal for fault tolerant and self-healing hybrid SDN control network," in *Proceedings of the 23rd Annual Conference on Professional Information Resources*, pp. 47–52, Prague, Czech Republic, May 2017.

[18] C.-Y. Hong, S. Kandula, R. Mahajan et al., "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 15–26, ACM, Hong Kong, China, August 2013.

[19] L. Sidki, Y. Ben-Shimol, and A. Sadovski, "Fault tolerant mechanisms for SDN controllers," in *Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 173–178, Palo Alto, CA, USA, November 2016.

[20] K. Kuroki, N. Matsumoto, and M. Hayashi, "Scalable openflow controller redundancy tackling local and global recoveries," in *Proceedings of the Fifth International Conference on Advances in Future Internet*, pp. 25–31, Barcelona, Spain, 2013.

[21] T. Yuan, X. Huang, M. Ma, and J. Yuan, "Balance-based sdn controller placement and assignment with minimum weight matching," in *Proceedings of the 2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, Anchorage, Alaska, USA, May 2018.

[22] Y. Jiménez, C. Cervelló-Pastor, and A. J. García, "On the controller placement for designing a distributed SDN control layer," in *Proceedings of the 2014 IFIP Networking Conference*, pp. 1–9, Trondheim, Norway, June 2014.

[23] T. Koponen, M. Casado, N. Gude et al., "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'10*, pp. 351–364, USENIX Association, Berkeley, CA, USA, 2010.

[24] A. S. Muqaddas, P. Giaccone, A. Bianco, and G. Maier, "Inter-controller traffic to support consistency in onos clusters," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1018–1031, 2017.

[25] C. Trois, M. D. Del Fabro, L. C. E. de Bona, and M. Martinello, "A survey on SDN programming languages: toward a taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2687–2712, 2016.

[26] T. Amin and G. Yasher, "Hyperflow: a distributed control plan for openflow," in *Proceeding of the Internet Network Management Conference on Research on Enterprise Networking*, San Jose, CA, USA, 2010.

[27] W. H. F. Aly, "LBFTFB fault tolerance mechanism for software defined networking," in *Proceedings of the International Conference on Electrical and Computing Technologies and Applications, 2017 (ICECTA'2017)*, Aurak, UAE, November 2017.

[28] W. H. F. Aly, "A novel fault tolerance mechanism for software defined networking," in *Proceedings of the European Modelling Symposium on Mathematical Modelling and Computer Simulation*, Manchester, UK, November 2017.

[29] W. H. F. Aly and Y. Kotb, "Towards SDN fault tolerance using petri-nets," in *Proceedings of the 28th International Telecommunication Networks and Application Conference (ITNAC 2018)*, Sydney, Australia, November, 2018.