Indonesian Journal of Electrical Engineering and Computer Science
Vol. 31, No. 1, July 2023, pp. 378~385
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v31.i1.pp378-385

Fault tolerant and load balancing model for software defined networking controllers

Ihssane Choukri, Mohammed Ouzzif, Khalid Bouragba
Department of Computer Engineering, Faculty of Science, Laboratory of Computer Science and Smart Systems (C3S), Higher School of Technology (ESTC), University Hassan II, Casablanca, Morocco

Article Info

Article history:

Received Dec 8, 2022
Revised Mar 11, 2023
Accepted Mar 12, 2023

Keywords:

Controllers
Fault tolerance
Leader election
Load balancing
Software defined networking

ABSTRACT

Previous years have seen increased interest in a new network paradigm, Software defined networking (SDN). The basic idea behind this new concept consists of removing the smart parts of the connectivity components and moving them to a seul control point known as a controller. This centralized network view makes the network maintenance and management easier and facilitates the creation of new services. Despite many advantages of SDN, the concentration of network intelligence in a single controller raises serious challenges that impact SDN scalability, performance, and fault tolerance. One of the main problems in SDN is controller failure. In this article, we develop a fault tolerant model called fault-tolerant load balancing (FTLBC) for SDN controllers. To reduce the cascading failure problem, the proposed model requires the load of the failing controller to be shared among other controllers. In the case of a controller failure, The FTLBC model concentrates on distributing the load among the remaining controllers based on the load of the orphans' switches and the load of the remaining controllers.

Corresponding Author:

Ihssane Choukri
Department of Computer Engineering, Faculty of Science
Laboratory of Computer Science and Smart Systems (C3S), Higher School of Technology (ESTC)
University Hassan II
Casablanca, Morocco
Email: choukriihssan@gmail.com

1. INTRODUCTION

Software-defined networking (SDN) is a novel architecture in the networking domain that has recently appeared. The core concept of SDN is to combine the intelligent features of the connectivity equipment and direct them toward a single control point known as a controller [1]. The latter provides a single network view that simplifies network design and facilitates the creation of new services due to the programmability provided by the SDN [2]. With the northbound interface (NBI), the controller makes some functionality available to network operators to create a variety of applications using, for instance, a set of representational state transfer (REST) application programming interfaces (APIs) [3].

The SDN, therefore, offers several advantages, however, there are challenging issues that can hamper its performance in large data center networks. For instance, concentrating all network load in a single controller causes serious problems that affect the performance, security, and reliability, in addition to the SDN fault tolerance which leads to a single point of failure (SPOF) [4], [5]. In SDN networks, fault tolerance techniques are necessary to guarantee availability and dependability [2]. For the SDN architecture that recommends having one controller, the SPOF issue is more prominent. If the controller fails, SDN networks consistently lose packet transmission. Therefore, using multiple controllers might be effective to complete this challenge. While the multi-controller architecture avoids a SPOF and thus improves the SDN reliability, the failure of a controller could cause network parallelization. When a controller crashes, the connected switches need to be migrated to a different controller. The charge of the failing controller must be split among other controllers to overcome the problem of cascading failure [6]. The SDN Load balancing while failure is an important point. Developing highly reliable, fault-tolerant, and load-balancing SDN controllers was the major challenge [7]. Load balancing takes intelligent actions and ensures better performance and reliability [8].

In this work, we address the challenge of fault tolerance and discuss the models used to solve it with an emphasis on fault tolerance in SDN architecture with multiple controllers. This paper's objective is to present a new fault-tolerant model that considers the load balancing between the SDN controllers. The proposed model is called fault-tolerant load balancing (FTLBC) for SDN controllers. In the FTLBC model, the load of the failing controller must be distributed to active controllers to reduce the cascading failure problem. If a controller has failed, the FTLBC model concentrates on distributing the load across the active controllers considering two metrics: the load of the orphan switches and the load of the active controllers. Compared to traditional SDN architecture, this FTLBC model can provide enhanced fault tolerance functionalities. The document is structured as shown in: section 2 outlines the proposed method along with the topology that was chosen for our FTLBC model, section 3 presents the method used for the recovery of our system, section 4 discusses the results, and section 5 concludes the paper.

2. THE PROPOSED METHOD

2.1. Related work

To address the issue of SDN controller failures, the authors propose different solutions. As mentioned in the previous article [9], there are two types of solutions in the literature: solutions proposed for a single controller SDN architecture, and other solutions proposed for a multiple controller SDN architecture, in this article we will concentrate on the solutions proposed for a multiple controller SDN architecture. A comparison between the proposed approach and the existing approaches with respect to fault tolerance, and load balancing is shown in Table 1.

Table 1. Comparison of the proposed approach with existing approaches

Method	Year	Assignment of switches	Metrics to select the candidate controller	Fault tolerance	Cascading failure problem-solving	Load balancing	Consider the switches load in the recovery process
Hyperflow [10]	2010	Local	Distance	✓	×	×	×
RDSDN [11]	2018	Global	Distance Controller'load = Number of links and switches. Topology failure of links.	✓	×	×	×
FT-SDN [12]	2020	Local	Controller' load latency	✓	✓	×	×
LBFTFB [13]	2017	Local	Assignment cost	✓	✓	✓	×
ECFT [14]	2018	Global	Distance Controller' load= number of flows per second Packet loss	✓	✓	✓	×
CALB [15]	2019	Global	Controller' load Throughput	✓	✓	✓	×
FTLBC (proposed method)	-	Local + Global	Switch' Load= total number of flow table entries + Controller' Load =accumulation of the Switches Load	✓	✓	✓	✓

Hyperflow [10] is a widely used fault-tolerant control model. Every controller manages the switches in its network and communicates with other controllers from other networks to synchronize the overall status of the network. This design for the control plane is logically centralized. In this model, each switch is connected to the nearest controller. In case of failure, a switch can be migrated to another controller. When a controller fails, the switches attached to this controller should be adjusted to connect to the closest controller by considering the distance metric. The switch-controller reassignment is done locally. In Hyperflow, since it blindly selects a neighboring controller in case of failure, it does not offer a solution if the chosen controller is not operational which might lead to the cascading failure problem.

Another approach improving the SDN fault tolerance is reliable distributed SDN (RDSDN) [11]. This solution involves adopting a distributed architecture with multiple controllers, where each controller is the master of one subnet and a slave of other subnetworks for the purpose of recovery. In case of RDSDN, the controller failure is identified by the coordinator, who decides on the alternate controller best suited to take over the subnetwork whose controller is failing. Each controller receives a reliability rate, which is shared across several controllers. The coordinator is determined by the controller with the highest reliability value. This approach is regarded as one of the most appropriate methods since it considers differents metrics to select the best controller to support the subnetwork whose controller has failed. However, assigning globally all the orphan switches to one controller, even if it is the most reliable controller, can make it overloaded, which can cause the problem of cascading failure.

Although Hyperflow [10], and RDSDN [11] methods address the issue of SDN controller fault tolerance, otherwise, affecting all the orphaned switches to the nearby controller or even the most reliable controller can cause the problem of cascading failure. So these two methods can result in a cascading failure. To address the problem of cascading failure, several solutions were suggested in the literature. The objective of these solutions is to find a method that manages the failures of a controller without causing the failures of the others controllers. This can be achieved with load balancing between controllers. e.g. if a controller fails, the switches associated with this controller must be distributed among other controllers.

fault-tolerant distributed architecture for software defined networks (FT-SDN) [12] is an architecture suggested by the authors to overcome the problem of fault tolerance in SDN. In the FT-SDN architecture, a switch is connected to multiple controllers, with one of the controllers acting as a master controller while the others remain as slave controllers. Once the master controller fails, one of the slave controllers assumes control of the switch and updates the network information base (NIB). This solution minimizes the problem of cascading failure since it distributes the switches of the crashed controller to the remaining controllers taking into account the load of the controllers and the latency.

Load balancing to support fault tolerance using feedback control for SDNs (LBFTFB) [13] is another method proposed in the literature to provide fault tolerance in SDN networks. For each switch, all slave controllers are sorted in ascending order in terms of the assignment cost, the first ranked controller is selected as its master. All others are considered slaves. In case of failure of the master, the first slave (i.e. having the lowest assignment cost) is automatically used as a master. LBFTFB provides load balancing for several levels of backup controllers. The LBFTFB model also reduces the cascading failure problem effect.

Enhanced controller fault tolerant (ECFT) [14] is a new method for managing controller failures. When a controller fails, the ECFT model concentrates on load balancing among other nearby controllers. The load of each neighboring controller is determined by ECFT using the delay between switches and the related controllers. ECFT chooses a master controller that is in charge of reassigning the orphans' switches to an appropriate backup controller. Since the charge of the failing controller is shared between others controllers, this avoids the problem of cascading failure.

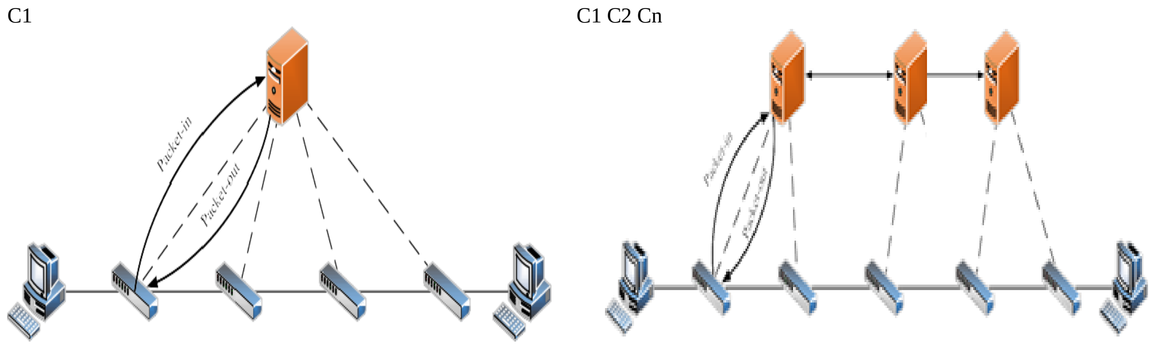
Controller adaptive load balancing model (CALB) [15] is a new approach for load balancing that provides fault tolerance for the SDN controllers in case of failure. This method proposes a load balancing algorithm used to share the charge among the slave controllers. Acording to CALB, the load balancing consists for distributing the charge of the orphaned switches among the slave controllers.

Since the objective of our FTLBC model is to find an efficient method that can handle controller failures and at the same time does not cause the cascading failure problem. for instance, in RDSDN [11] the authors propose a reliable method for controllers fault tolerance, but the fact of re-assigning all the orphaned switches to one controller, even if it is reliable can cause the problem of cascading failure. That's why we have proposed in our model a recovery method that also takes into consideration the load balancing between controllers. In this context, some works considers the load balancing between controllers in the failure recovery stage, for instance, the solutions FT-SDN [12], LBFTFB [13], ECFT [14], and CALB [15] minimize the problem of cascading failure since it distributes the orphaned switches to the active controllers taking into account the controllers' load. All these approaches redistribute the orphans' switches considering only the controllers'load, but none of these methods consider the load of the switches in the recovery process. In our proposed FTLBC model we take into account in addition to the controllers'load, the switches'load. Two criteria are used in our model: controllers'load and switches'load to migrate the orphan switches to the active controllers. It is possible to optimize this reassignment locally or globally. In the local reassignment, switches are migrated individually, which increases latency. On the other hand, the global reassignment results when all the orphan switches can be migrated simultaneously to a new controller. In our FTLBC model, switch-controller migration is done locally and also globally. i.e. we can reassign at the same time one or more switches from the failing controller to other controllers.

2.2. Topology used for FTLBC model

In SDN implementation there exist two types of architectures a centralized controller architecture and a multiple controller architecture as shown in Figure 1. The centralized controller architecture can be seen in Figure 1(a) is simpler to manage but inherently unreliable [16]. Concentrating all network intelligence into a single controller raises significant issues that have an impact on scalability, performance, and the fault tolerance of the SDN, leading to a SPOF. When an SDN controller breaks, the switches lose the capacity to forward new packets and then the network crashes.

The multiple controller architecture as shown in Figure 1(b) promises to solve the single SDN controller's shortcomings [17]. The basic concept is to have different controllers that can share the load equally across the network. Also, one controller can relay another controller if it fails [18]. Moreover, this solution is known to improve reliability and fault tolerance. In short, we can state that a multi-controller architecture is necessary while fault tolerance or availibility is the problem. However, the limitation of this solution is that it requires the synchronization overhead between the controllers for consistency reasons, the controllers must share their network information through the east or westbound interfaces. Since the objective of our FTLBC model is to enhance fault tolerance in controllers, we chose to use a multiple SDN controller architecture.



(a) (b)

Figure 1. The centralized controller architecture; (a) with single controller and (b) the multiple controller architecture

In a multiple SDN controller architecture, controller disposition is either hierarchical (vertical disposition) or fully distributed (flat disposition) [18] as shown in Figure 2. In a hierarchical model as shown in Figure 2(a), just the roots controller possesses the status of the global network and controls it. Some examples of hierarchical models are Kandoo [19], improved Kandoo [20], [21], and FlowBroker [22]. This architecture may not tolerate failures since it uses a root controller wish can be a SPOF.

Conversely, in a fully-distributed model as shown in Figure 2(b), each controller controls a subnet of the network. Some examples of fully-distributed models are Onix [23], Open Network Operating System (ONOS) [24], distributed multi-domain SDN controllers (DISCO) [25], and ElastiCon [26]. In this architecture, the global network state is stored and maintained by the SDN controllers in a data store, which may result in a significant cost of synchronization to add the controllers, but it remains the most appropriate solution for improving fault tolerance and reliability. In our FTLBC model, we have chosen a flat architecture, since the hierarchical architecture can't help us to manage the controller's failures.

2.3. Synchronization between controllers

To create a global network state, the controllers must share topology information between them. The East or Westbound interfaces are introduced for SDN distributed architecture [27]. This API provides connectivity across controllers for the purpose of synchronization. For example, SDN controllers may use this API to communicate their local network status and create a global network status.

When one of the SDN controllers breaks, active controllers, need to be prepared to assume the role of defaulting controller. Using the East or Westbound interfaces, the FTLBC model will transmit the essential information, including the name and the availability between controllers. Furthermore, an abstract overview of the network topology of each controller, comprising controller' load and switch' load, can be transmitted through this interface to offer a global view of the network.

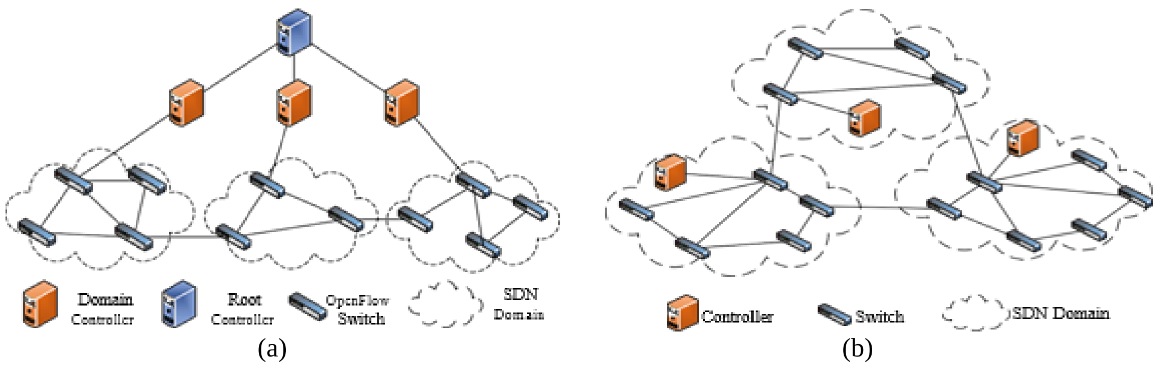


Figure 2. A view of distributed controller architecture; (a) hierarchical controller(root) and (b) fully distributed controller (flat)

3. METHOD

The FTLBC model distributes the failing controller's switches to the other controllers, taking into account two metrics: Switches ' load and Controllers' load.

With a large scale of flow inputs, the controller handles a large flow table and the controller’s load will be high. The switches' load includes a total number of flow table inputs, and the controllers' load is an accumulation of the load of the switches. According to the reciprocal relationship a shown in Table 2, the least loaded switches are mapped to the most heavily loaded controls and the most heavily loaded switches are mapped to the least loaded controls.

Table 2. Reciprocal relation between the switch’s load and the controllers' load

Metric 1	Relation	Metric 2
Switch Load	=>	Controller Load
Switch Load	=>	Controller Load

To explain the efficiency of our recovery method, we will take the example of the architecture shown in Figure 3. In this architecture we consider three controllers: C1, C2, and C3. C1 is the master controller of network1, C2 is the master controller of network2 and C3 is the master controller of network3. we assume that the network controlled by C1 (network1) is located in an industrial area; in this area, the load of the switches will be too high because the switches send a lot of requests to the controller C1 and also the activity of this controller will be high; we consider that the network controlled by C3 (network3) is located in a rural area; in this area, the load of the switches will be reduced because they send only a few requests to the C3 controller and also the activity of this controller will be reduced; and we assume that the network of the C2 controller (network2) is located in a normal area (between an industrial area and a rural area) where the load of the switches and the controller is normal (neither too high nor too low).

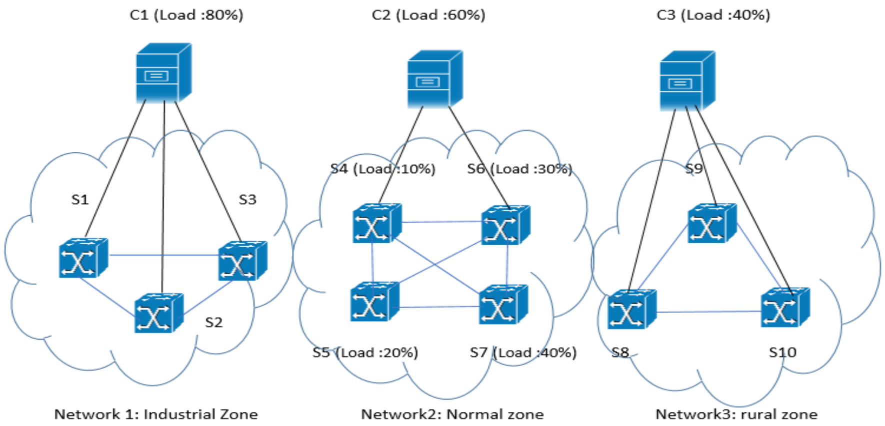


Figure 3. The architecture used for the recovery process in FTLBC model

Considering that controller C2 has failed, the controller chosen as the leader (C1 or C3) will redistribute the load of C2 (the switches of C2) among other controllers, taking into consideration the load on C2's switches and the load on other controllers (C1 and C3). As it is shown in Figure 3, we assume that the load of C1, C2, and C3 are successively 80%, 60%, 40%, and the load of switches of C2: S4, S5, S6, and S7 are successively 10%, 20%, 30%, 40%. if C2 has failed, we will assign the less loaded switches of C2 to the more loaded controller; that is to say in this case we will assign S4 and S5 to C1, and we will assign the more loaded switches of C2 to the less loaded controller; that is to say, assign S6 and S7 to C3. So, in this case, if C2 has failed the switches S4 and S5 will be assigned to C1, and the switches S6 and S7 will be assigned to C3.

4. RESULTS AND DISCUSSION

As a result, in this section we discuss the recovery algorithm in Algorithm 1 against controller failures. This approach is described in:

- Firstly, the leader controller must store the active and unloaded controllers in a list L, and after order these controllers in decreasing order (lines 1–6 of Algorithm 1).
- Afterward, the leader ordered the orphans' switches in a list T, in increasing order (lines 7–10 as shown in Algorithm 1).
- Then assigns the variable q to the number of functioning controllers, and the variable step to integer division of the number of the orphans' switches n / on the number of active controller’s q (line 11 of Algorithm 1).
- After, the leader checks if the number of the orphans' switches is lower than the number of active controllers, then assigns the variable step to 1 and modifies q (the size of the active controllers) to the number of the orphans' switches. In this case, we will assign the switches just to the first controllers (lines 12-14 of Algorithm 1).
- Afterward, go through the two lists: the list of active controllers (L) and the list of the orphans' switches (T), and assign the switches to the controllers respecting the variable step that represents the number of switches to assign to each controller (line 15-21 of Algorithm 1).
- The test:!(T. isEmpty () (line 16 of Algorithm 1) concerns the case when the number of switches n is higher than the number of controllers q. In this case, the remaining switch (s) in the list (T) will be assigned to the last controller.

Algorithm 1. The recovery algorithm

```
Input: -Number of network controllers (i = 1 to m), the Controllers designed by Ci
 -Number of the orphans' switches (j = 1 to n), the Switches designed by Si
 -L= []: an empty list to fill with controllers in a decreasing order
 -T= []: an empty list to fill with the orphans' switches in increasing order.
Procedure:
1: For i:1 to m do
2: IF Ci isActive () && load(Ci) < capacity(Ci) Then
3 : L.add (Ci, load(Ci))
4: End IF
5: End For
6: sortDescending (L)
7: For j:1 to n do
8: T.add (Si, load(Si))
9: End For
10: sortAscending (T)
11: k=1; q= length(L); step=n/q ; p=step ; //q is the number of active controllers
12: IF(step==0) Then
13: step=1; p=step; q= n; //numbers of switches (n)< numbers of controllers (q)
14: End IF
15: For i:1 to q do
16: For j: k to step and !(T. isEmpty ()) do
17: affecter T[j] à L[i] ;
18: End For //end For 1
19: k=j;
20: step=k+p;
21: End For //end For 2
```

To implement our recovery algorithm, we'll look at the topology in Figure 3 that has three SDN controllers in charge of various subnetwork topologies at internet protocol (IP) addresses C1=10.10.10.1, C2=10.10.10.2, and C3=10.10.10.3. To simulate the network, we use the network emulation tool Mininet [28]. The three controllers are emulated with the Floodlight controller [29]. Floodlight is a Java OpenFlow controller [30] widely used by developers. It provides a framework that can be used to easily enhance SDNs.

In this study, we install the Mininet tool and Floodlight controller in a virtual machine. The final results of our implementation are currently being processed. All experiments were performed on a computer with an Intel Core i5, 8250U 1.80 GHz CPU, and 16 GB RAM. Table 3 summarizes the tools used for our FTLBC model.

Table 3. The simulation testbed' tools

Tool	Feature	Version
Mininet [28]	Network emulator	2.2.1
Floodlight [29]	SDN controller	1.2
OpenFlow [30]	Programmable network protocol	1.5
Windows	Operating system on PC	Windows 10 64bit
Oracle VM virtual box	Virtualizing software	6.0
Linux	Virtual machine's operating system	Ubuntu 16.04 64bit
Hping3	Traffic	Version 3

5. CONCLUSION

In this research, we design a new model called fault-tolerant load balancing for SDN controllers. In our approach, the load of the failing controller has to be shared across the remaining controllers to reduce the cascading failure problem. In case of a controller failure, the FTLBC model concentrates on distributing the load across remaining controllers taking into account two metrics: the load of orphans' switches and the load of other controllers. Compared to the traditional SDN architecture, the FTLBC model offers enhanced fault tolerance functionalities and better management of independent controllers. As further research for this work, we will test the reliability of our FTLBC model compared to other models that exist in the literature. And also, we would like to develop a more sophisticated model that uses artificial intelligence techniques.

REFERENCES

[1] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, "Software-defined networking: challenges and research opportunities for future internet," *Computer Networks*, vol. 75, no. PartA, pp. 453–471, Dec. 2014, doi: 10.1016/j.comnet.2014.10.015.

[2] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "Fault-tolerance in the scope of software-defined networking (SDN)," *IEEE Access*, vol. 7, pp. 124474–124490, 2019, doi: 10.1109/ACCESS.2019.2939115.

[3] L. Li, W. Chou, W. Zhou, and M. Luo, "Design patterns and extensibility of REST API for networking applications," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 154–167, Mar. 2016, doi: 10.1109/TNSM.2016.2516946.

[4] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, 2020, doi: 10.1016/j.jnca.2020.102563.

[5] S. Hamid, N. Zakaria, and J. Ahmed, "ReCSDN: Resilient controller for software defined networks," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 8, 2017, doi: 10.14569/IJACSA.2017.080826.

[6] G. Yao, J. Bi, and L. Guo, "On the cascading failures of multi-controllers in software defined networks," in *Proceedings - International Conference on Network Protocols, ICNP*, Oct. 2013, pp. 1–2, doi: 10.1109/ICNP.2013.6733624.

[7] M. Shailly, "A critical review based on fault tolerance in software defined networks," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 2, pp. 456–461, Apr. 2021, doi: 10.17762/turcomat.v12i2.849.

[8] O. M. A. Alssaheli, Z. Z. Abidin, N. A. Zakaria, and Z. A. Abas, "Software defined network based load balancing for network performance evaluation," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, pp. 117–124, 2022, doi: 10.14569/IJACSA.2022.0130414.

[9] I. Choukri, M. Ouzzif, and K. Bouragba, "Fault tolerance in SDN controllers," in *AI2SD 2020: Advanced Intelligent Systems for Sustainable Development (AI2SD'2020)*, 2022, pp. 1–13.

[10] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *2010 Internet Network Management Workshop / Workshop on Research on Enterprise Networking, INM/WREN 2010*, 2010, p. 3.

[11] S. Moazzeni, M. R. Khayyambashi, N. Movahhedinia, and F. Callegati, "On reliability improvement of software-defined networks," *Computer Networks*, vol. 133, pp. 195–211, Mar. 2018, doi: 10.1016/j.comnet.2018.01.023.

[12] R. K. Das, F. H. Pohrmen, A. K. Maji, and G. Saha, "FT-SDN: A fault-tolerant distributed architecture for software defined network," *Wireless Personal Communications*, vol. 114, no. 2, pp. 1045–1066, Sep. 2020, doi: 10.1007/s11277-020-07407-x.

[13] W. H. F. Aly, "A novel fault tolerance mechanism for software defined networking," in *Proceedings - UKSim-AMSS 11th European Modelling Symposium on Computer Modelling and Simulation, EMS 2017*, Nov. 2017, pp. 233–239, doi: 10.1109/EMS.2017.47.

[14] W. H. F. Aly and A. M. A. Al-Anazi, "Enhanced controller fault tolerant (ECFT) model for software defined networking," in *2018 5th International Conference on Software Defined Systems, SDS 2018*, Apr. 2018, pp. 217–222, doi: 10.1109/SDS.2018.8370446.

[15] W. H. F. Aly, "Controller adaptive load balancing for SDN networks," in *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2019, vol. 2019-July, pp. 514–519, doi: 10.1109/ICUFN.2019.8805922.

[16] P. Lin *et al.*, "A west-east bridge based SDN inter-domain testbed," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 190–197, Feb. 2015, doi: 10.1109/MCOM.2015.7045408.

[17] O. Blial, M. B. Mamoun, and R. Benaini, "An overview on SDN architectures with multiple controllers," *Journal of Computer Networks and Communications*, vol. 2016, pp. 1–8, 2016, doi: 10.1155/2016/9396525.

[18] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: A survey on design choice," *Computer Networks*, vol. 121, pp. 100–111, Jul. 2017, doi: 10.1016/j.comnet.2017.04.038.

[19] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks*, Aug. 2012, pp. 19–24, doi: 10.1145/2342441.2342446.

[20] Z. Jingjing, C. Di, W. Weiming, J. Rong, and W. Xiaochun, “The deployment of routing protocols in distributed control plane of SDN,” *The Scientific World Journal*, vol. 2014, pp. 1–8, 2014, doi: 10.1155/2014/918536.

[21] A. Koshibe, A. Baid, and I. Seskar, “Towards distributed hierarchical SDN control plane,” in *2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC)*, Oct. 2014, pp. 1–5, doi: 10.1109/MoNeTeC.2014.6995587.

[22] D. Marconett and S. J. B. Yoo, “FlowBroker: A software-defined network controller architecture for multi-domain brokering and reputation,” *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 328–359, Apr. 2015, doi: 10.1007/s10922-014-9325-5.

[23] T. Koponen *et al.*, “Onix: A distributed control platform for large-scale production networks,” in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010*, 2019, pp. 351–364.

[24] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, “Inter-controller traffic in ONOS clusters for SDN networks,” in *2016 IEEE International Conference on Communications, ICC 2016*, May 2016, pp. 1–6, doi: 10.1109/ICC.2016.7511034.

[25] K. Phemius, M. Bouet, and J. Leguay, “DISCO: Distributed multi-domain SDN controllers,” in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–4, doi: 10.1109/NOMS.2014.6838330.

[26] A. Dixi, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, “ElastiCon: An elastic distributed SDN controller,” in *ANCS 2014 - 10th 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Oct. 2014, pp. 17–27, doi: 10.1145/2658260.2658261.

[27] H. Yin, H. Xie, T. Tsou, P. Aranda, D. Lopez, and R.Sidi, “SDNi: A message exchange protocol for software defined networks (SDNS) across,” *Internet Research Task Force*, pp. 1–14, 2012, [Online]. Available: https://tools.ietf.org/pdf/draft-yin-sdn-sdni-00.pdf.

[28] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Oct. 2010, pp. 1–6, doi: 10.1145/1868447.1868466.

[29] F. Anónimo, “Floodlight controller - confluence,” *Project Floodlight*, 2020, [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview.

[30] P. Göransson, C. Black, and T. Culver, “The OpenFlow specification,” in *Software Defined Networks*, Elsevier, 2017, pp. 89–136, doi: 10.1016/B978-0-12-804555-8.00005-3.

BIOGRAPHIES OF AUTHORS

Ihssane Choukri was born in Casablanca, Morocco, 1993. She received her Master’s degree in computer and internet engineering (3I) from the faculty of science Ain Chock, Hassan II University, Casablanca, Morocco. Currently a Ph.D. student in the research laboratory in networks, computing, telecommunications, and multimedia (C3S) of the High School of Technology (ESTC), in Hassan II University of Casablanca, Morocco. Her research interests include computer networks, distributed computing, software-defined networking, fault tolerance, and artificial intelligence. She can be contacted at email: choukriihssan@gmail.com.

Mohammed Ouzzif joined the Higher School of Technology in 1996 as assistant professor. He got his postgraduate doctorate in 1999 at the faculty of science of Rabat. In 2005, he defended his Ph.D. in the national doctorate at the National School of Computer Science and Systems Analysis. His principals contributions was in the dynamic management of resource sharing in a collaborative system with distributed algorithms allowing the extension of static distributed mutual exclusion algorithms for the support of new joins and departures dynamically. Currently, Mr. Mohammed OUZZIF's research interests concern distributed systems and networks. He has published a considerable number of scientific articles in various journals, conferences and workshops. He is a steering committee member of the SYSCO-BIOTS conference, and he served on the technical program committees of several international conferences and workshops, most of which were related to wireless network and distributed systems. In 2021, he founded the C3S laboratory (computer science & smart systems). He is currently the head of this laboratory and he is a professional member of the IEEE Computer Society. He can be contacted at email: ouzzif@est-uh2c.ac.ma.

Khalid Bouragba received the DESS in network and telecom from Chouaib Doukkali University of El Jadida, Morocco, in 2002, and a Ph.D. degree in informatics (computer science) from CED ENSEM in Hassan II University of Casablanca, Morocco, in 2012. Currently, he is a PH professor in the department of computer engineering at the High School of Technology in Hassan II University of Casablanca, Morocco. He has served as a TPC member in several leading conferences in the field of mobile and wireless communications and collaborative system workflow. He has co-authored several papers published in international journals and conference proceedings. His research interests include distributed computing, workflow interoperability, network and systems management, requirement engineering, and sensor networks. He is currently research committee coordinator in High School of Technology. He is a member of the C3S (computer science and smart systems) research laboratory. He can be contacted at email: khalid.bouragba@univh2c.ma.