Controller Adaptive Load Balancing for SDN Networks

Wael Hosny Fouad Aly[#1]
[1]College of Engineering and Technology
American University of the Middle East (AUM)
Egaila, Kuwait
Arab Academy for Science and Technology (on leave)
Alexandria, Egypt
e-mail: wael.aly@aum.edu.kw

Abstract- Software Defined Network (SDN) is a novel network architecture which decouples the control plane from the data plane. One drawback of SDNs is the inability to survive when facing failures, in particular in large scale data-center networks. Since SDN is programmable, flexible dynamic mechanisms could be designed to achieve fault tolerance. The paper studies the single master multiple slave controllers architecture to study fault tolerance. The paper proposes a *Controller Adaptive Load Balancing* (CALB) model for SDNs. CALB dynamically adapt the load of the master controller based on the current load at each slave controller. Mininet simulation tool is for the experimentation phase. Floodlight is used to implement the controllers. The key contribution of the paper is proposing a load balancing algorithm that the CALB is using to distribute the load among slave controllers based on a desired ratio. Experiments results were conducted using CALB in two scenarios: (1) Master controller is distributing switches among two slave controllers, and (2) master controller is distributing switches among three slave controllers. The paper uses throughput and response time for performance measurement. Results are promising as the performance increased 12% in terms of the throughput and 9% in terms of the response time when compared to a reference model Enhanced Controller Fault Tolerant fault tolerance technique (ECFT).

Keywords: Software Defined Network, Fault tolerance, OpenFlow, load balancing, switch migration.

I. Introduction

Software Defined Network (SDN) is an emerging and promising network architecture that decouples the control plane from the data plane. The control plan is responsible for the routing decision while the data plane is responsible for data forwarding. One of the limitations for an SDN is that it can hardly survive when facing failure in large-scale data-center networks. Due to the programmability of SDN, mechanisms are designed to achieve fault tolerance. In this paper, the paper discusses the fault tolerance issue and review the existing models towards this problem. SDN shifts the controlling tasks to a logically centralized controller while the switches handle the forwarding tasks. Controllers are responsible for the forwarding decisions of the switches [1][2][3]. SDN refers to a network architecture where the forwarding state in the data plane is managed by a controlled plane that is decoupled from the switches

The problem occurs when a controller fails or a controller is overloaded (i.e., its load exceeded a certain threshold). The goal of this paper is to propose a new load-balancing model to provide fault tolerance for the SDN control plane. The proposed model is called *Controller Adaptive Load Balancing model (CALB)*. The paper considers consistency improvement when it comes to controllers' synchronization in a distributed fashion. The proposed model defines a weighted load balancing as the process of controlling the load of the total system switches among a set of slave controllers. Fault tolerant system is active in three cases, (1) overloading of controllers, (2) under-loading of controllers, and (3) failure of the controllers. Research have explored various techniques for switch migration to provide load balancing. The proposed protocols can only deal with balancing an unbalanced load. Proposed algorithms cannot deal with controller failure. Control plane is suffering from lack of fault tolerance. The paper proposes the CALB algorithm that can provide fault tolerance to the control plane in the event of controller failure. CALB has a switch migration feature to provide load balancing in the event of load imbalance in addition to controller failure as well as master controller failure [8][9].

The paper is organized as follows: Section II has the related work, section III has the proposed CALB model design. Section IV has the simulation results. Section V has the conclusion and the future work

II. Related Work

Scott D. Stoller et al. [13] have used modified Bully Algorithm for master controller election in which separate failure detection module is required. Ricardo Macedo et al. [12] implemented master controller election which is based on controller's performance. Controller with the highest performance is elected as a master controller. Periodic measurement of the controllers is used to determine the possible master controllers. Dongeun Suh, et al. [11] have used raft consensus to improve fault tolerance of controller. Esteban Hernandez [10] used a master controller election using a consensus algorithm to provide fault tolerance to the controllers in the control plane. Esteban Hernandez [1][10] have used a master controller election using raft consensus algorithm to provide fault tolerance to the control plane. Aly et al. [4][5][6] proposed a feedback control theoretic techniques and petri-nets modeling to implement fault tolerance for controllers. The work is giving good results, but

the feedback control theoretic techniques have put extra burden on the controllers. In the aforementioned work, controllers have performed additional task for electing the master controller. This adds extra overhead for the control plane.

Dixit et al. [14] have implemented a dynamic mapping of switches and controllers through focusing on primary architecture of distributed control plane. This work works on migration of switch from overloaded controller to underloaded controller. This work uses a controller pool to assign controllers based on controller utilization information. The information is used for load balancing purposes and to reduce the power consumption by migrating switches from overloaded controllers to underloaded ones. Authors in [14] have developed algorithms to provide migration of switches. This work has not dealt with controller's fault tolerance problem. In addition, the work has not covered an efficient controller's selection mechanisms.

III. Controller Adaptive Load Balancing Technique (*CALB)* Model Design

In this section, the proposed Controller Adaptive Load Balancing Technique for Fault Tolerance (CALB) model is discussed. CALB aims to balance the load among multiple controllers in a multi-controller SDN paradigm. CALB takes into consideration the delay between the switches and their associated slave controllers to minimize the response time. CALB assumes that there is a master controller and *n* slave controllers connected to it. Figure 1 has master controller is connected to two slave controllers.

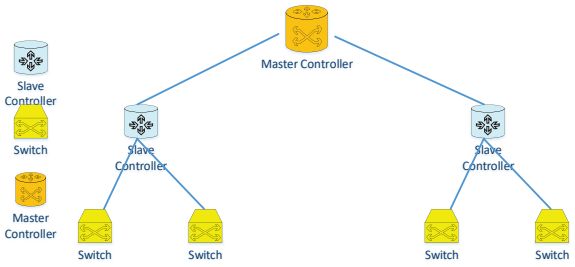

Figure 1: Two slave controllers connected to a Master controller

Figure 2 shows the CALB where the master controller is connected to three slave controllers. Each slave controller has set of switches associated with it.

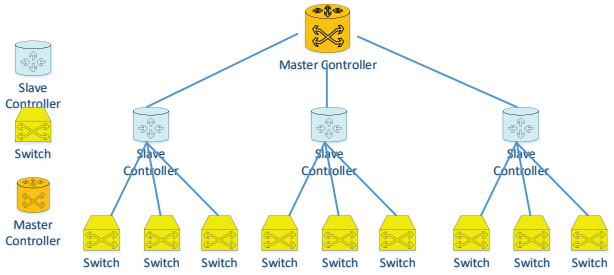

Figure 2: Three slave controllers connected to a Master controller

Each controller within a cluster has a unique ID (CID). The controller with CID equals to zero is initially the master controller of the cluster. The master controller performs the following tasks (1) Routing incoming packets, (2) periodically collects the load of each controller of the cluster, and (3) Update the switch-controller mapping table located at the master controller. Each controller within a cluster updates the master controller with two pieces of information: (1) current workload, and (2) current number of associated switch. The master controller takes the switch migration decision based on the overall workload level. Floodlight facilitates message passing services through ZMQ. Each switch is assumed to be connected to a master controller and a set of slave controllers.The master controller is responsible of assigning the failed controller's switches to a suitable backup controller. A set of experiments has been conducted for distributing the load among two and three controllers with different weights as shown in Table 1. In this paper, it is assumed that the ratios to distribute the request among different slave controllers depend on different parameters such as the workload on each slave controller, the drop ratio, and the response time. The weights below are selected to test the different scenarios and make sure that the master controller is distributing the load successfully to the desired slave controller.

Table 1: Experiments among two controllers

Experiment	Controller 1	Controller 2
1	1	1
2	1	2
3	1	3
4	1	4

Another set of experiments is conducted for distributing load among three controllers as shown in Table 2 based on the selection criteria discussed above.

Table 2: Experiments among three controllers

Exp	Controller 1	Controller 2	Controller 3
1	1	2	7
2	1	3	6
3	5	3	3
4	8	1	1

If the master controller fails, there should be a backup scenario to handle this situation. In the proposed CALB model, each slave controller has a monitoring service to monitor the failure of the master controller. The monitoring module polls the master controller to test if still-alive within a time interval T which is assumed to be 10 seconds. If the master controller is sensed to be down, the controller with the highest availability will be elected as the new master controller. CALB assumes that the highest availability refers to the slave controller with the minimum load and high throughput at the time the master controller fails. Each controller has to forward its current status and the associated switches to the other controllers within the cluster. Floodlight facilitates the message passing operation through ZMQ

asynchronous messaging service. Two conditions upon which the switches association has to migrate from one controller to another: (1) Unbalanced load on controllers, and/or (2) controller failure. If the failure occurs to the master controller, migration is achieved by sending the switches to slave controllers based on a predetermined ratio. The ratio is determined on many factors, such as the current workload and the response time. Algorithm 1 shows how the master controller distributes the load among the associated two available slave controllers based on a given predetermined ratio DRatioX and DRatioY. The master controller gives these values according to the workload list. Both algorithm 1 and algorithm 2 use 100 requests to test the load balancing mechanism and make sure the load is evenly distributed according to the desired ratio. The same algorithm still holds for larger number of requests. The algorithms are tested for 1,000 and 10,000 requests. Extensive results with larger number of requests are not included in the paper due to lack of space.

Algorithm 1: CALB4TWO distributing load among two controllers:

```
void CALB4TWO()
{
        xplusy=x+y;
        DRratioX=x/xplusy;
        DRratioY=y/xplusy;

        x=0;
        y=0;
        while(z>0)
        {
        if |((x+1)/(xplusy+1)-DRratioX|
        <|(y+1)/(xplusy+1)-DRratioY|
        x++;
        else y++;

        xplusy=x+y;
        printf("\n%d\t%d", x, y);
        z--;
        }
}
```

Algorithm 1 and Algorithm 2 show how the master controller distributes the workload among the associated three slave controllers based on a given predetermined ratio. The algorithm calls a function to get the rank of minimum of three values. Algorithm 2 computes the DRratioX, which is the percentage of the workload at a given switch to the overall workload. The two other switches are working in the same fashion. The slave controller that minimizes the difference of its ratio with the target ratio takes the pending workload. This is repeated until all the switches are assigned to the appropriate slave controller to rebalance. Slave controllers send their workload periodically to the master controller every time interval T. Master controller sorts the slave controllers list in an ascending order according to their workloads.

***Algorithm 2*:** CALB4THREE distributing load among three controllers

```
int getmin (float a, float b, float c)
{
  if ((a<=b) && (a<=c)) return 1;
  else if ((b<=a) && (b<=c)) return 2;
  else if ((c<=a) && (c<=b)) return 3;
}
void CALB4THREE()
{
        all=x+y+z;
        DRratioX=x/all;
        DRratioY=y/all;
        DRratioZ=z/all;
        x=0; y=0; z=0;
        while(counter>0)
        {
        selection=getmin
        (|((x+1)/(all+1)-DRratioX|,
         |((y+1)/(all+1)-DRratioY|,
         |((z+1)/(all+1)-DRratioZ|);
        if (selection==1) x++;
        else if (selection==2) y++;
        else if (selection==3) z++;
        all=x+y+z;
        counter--;
        }
}
```

The workload percentage is computed according to equation (1) .

$$\text{Workload Percentage} = \frac{\text{Current workload}}{\text{Total workload}} \qquad (1)$$

In order to decide whether a slave controller is eligible to take the load of a failed slave controller, the estimated workload is computed for each slave controller based on equation (2). Estimated workloads are sorted in an ascending order at the master controller.

$$\text{Estimated Workload} = \frac{\text{Current workload} + \text{additional workload}}{\text{Total workload}} \qquad (2)$$

IV. Simulation Results

Simulation testbed used in our experiments uses Mininet Simulator version 2.2.2, Floodlight controller version 1.2, OpenFlow switch version 2.0.2, the operating system used is Ubuntu version 14.04, RAM of 32 GB is used, the processor is Intel ® Core ™ i7 5500U CPU 2.4GHz, and hping3 is used as the traffic source.

The work in this paper has created a custom topology where two different scenarios were adopted. The following subsection A describes the custom topology discussed in Figure 1 while applying Algorithm 1 for load balancing.

Similarly, subsection B describes distributing the load among three controllers while applying algorithm 2.

A. Distributing Among Two Slave Controllers

Each slave controller has two switches associated to it. Each switch is connected to exactly one host. All the switches are connected to each other to form a network. All the controllers are connected to all other switches with slave role except its own switch to which they have connected with master role. Master controller provides fault tolerance to the control plane when current master controller get failure and at multiple instances of the controller are run. It uses ISyncService in order to store the data. It synchronizes state between the controllers by letting all of them access updates published by all other modules in the controller in an efficient manner. 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. Simulation compared CALB model to the reference architecture, which is the Enhanced Controller Fault Tolerant fault tolerance technique (ECFT)[7] reference model.

The way the ECFT works is that it sends the switches associated with the failed slave controller to the nearest controller not taking into consideration the workload of the current workload. Consequently, problems such as the cascading failure problem, packet loss, and packet delay are not addressed. The proposed CALB reduces the effect of such problems through distributing the load among other controllers based on determined ratios whenever a failure occurs. The master controller considers the controllers' workload and current response time before assigning the switches to it. This is to avoid a cascading failure problem. However, we compared the CALB to ECFT regarding the packet delay, the packet loss, and the controller workload after the failure occurs.

Four different floodlight controllers are used in four terminal windows with same IP address and different ports on port no `4000`, `4001`, `4002`, `4003`. Default Master controller is ControllerID1 but after starting two slave controllers. while performing experiments we tested floodlight controller by varying the number of controller nodes and packet arrival rate. We have tested floodlight in both the mode of `cbench` through the throughput and latency as well to get throughput and response time respectively. Figure 3, Figure 4, Figure 5, and Figure 6 have the results of applying Algorithm 1 that is used by the master controller through assigning workload to slave controllers with different ratios. As seen in the figures, the required ratios of distributing the switches to the slave controllers are obtained successfully. We have experimented with the following ratios to test the Algorithm 1 1:1, 1:2, 1:3, and 1:4. Results are very promising as we see that the master controller is able to send the appropriate switch to the appropriate slave controller. We have experimented with 100 switches in total.

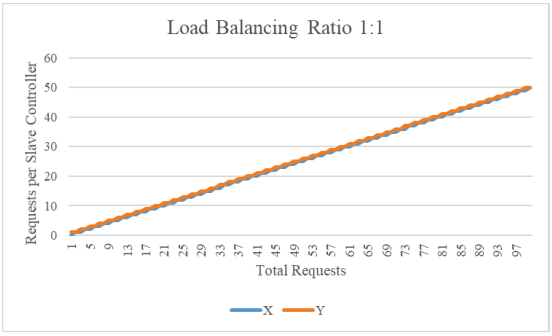

Figure 3: Load balancing for two slave controllers with ratio 1:1

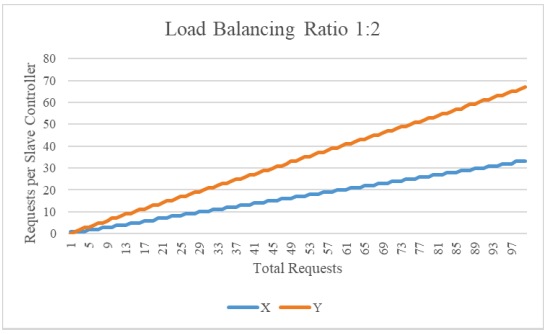

Figure 4: Load balancing for two slave controllers with ratio 1:2

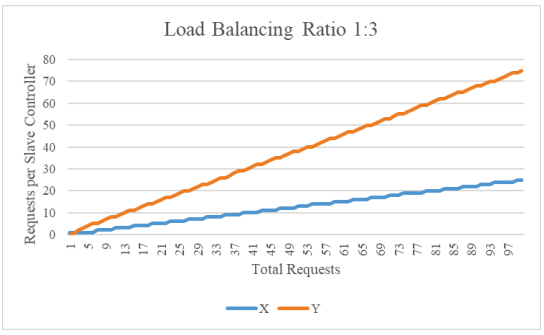

Figure 5: Load balancing for two slave controllers with ratio 1:3

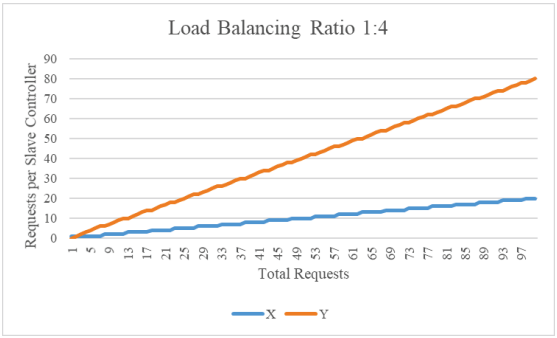

Figure 6: Load balancing for two slave controllers with ratio 1:4

B. *Distributing Among Three Slave Controllers*

The second scenario is using the topology discussed in Figure 2 along with the algorithm stated in Algorithm 2. Floodlight controllers are used. In this scenario, the master controller is distributing the load among three slave controllers based on a determined ratio. We conducted several experiments to test Algorithm 2. We experimented with different ratios such as 1:2:7, 1:3:6, 5:3:2, and 8:1:1. As seen in in Figure 7, Figure 8, Figure 9, and Figure 10, the load of switches per slave controller is obtained successfully. Results are so promising since the required output behavior is obtained as the master controller desires.

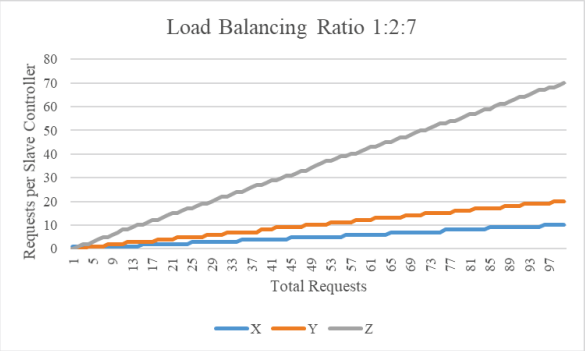

Figure 7: Load balancing for three slave controllers with ratio 1:2:7

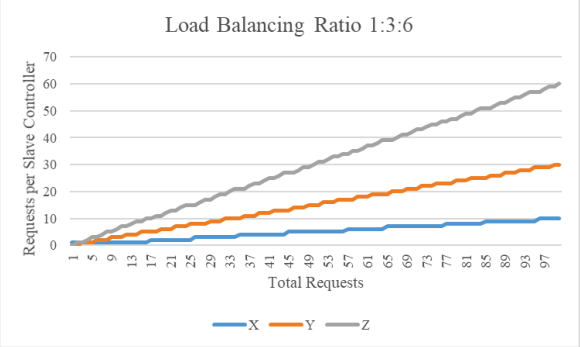

Figure 8: Load balancing for three slave controllers with ratio 1:3:6

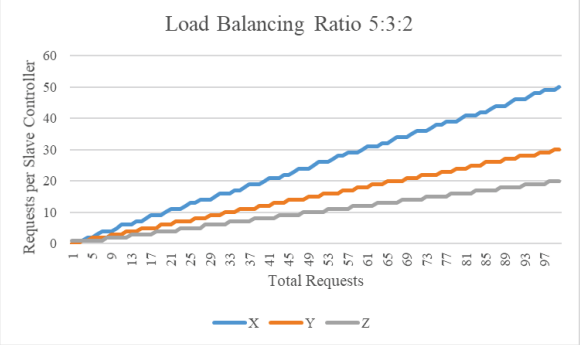

Figure 9: Load balancing for three slave controllers with ratio 5:3:2

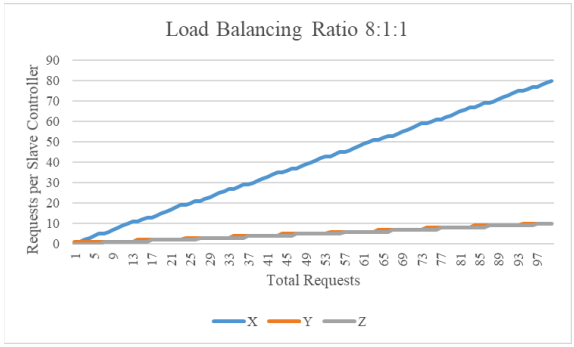

Figure 10: Load balancing for three slave controllers with ration 8:1:1

C. *Throughput and Response Time*

Concerning the Throughput, packets are sent by varying packet arrival rate and CIDs from 1 to 2 in one set of experiments, then the other set of experiments using CID from 1 to 3, we observe in the result that adding controller nodes increase the throughput. Figure 11 shows the comparison between the proposed CALB algorithm and the ECFT in a two slave controller environment. Figure 12 repeats the experiment wen it has 3 slave controllers. It is clear that the proposed model (CALB) outperforms the ECFT model by around 12%.

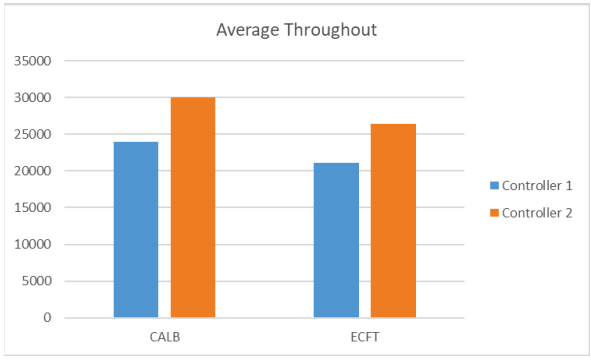

Figure 11: Average Throughput for CALB and ECFT for two slave controller model

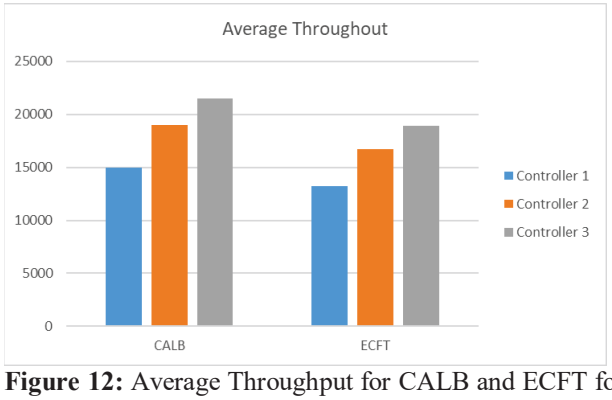

Figure 12: Average Throughput for CALB and ECFT for three slave controller model

Concerning the response time, packets are sent to floodlight controllers by varying its packet arrival rates from 250 packets/sec to 2000 packets/sec. We have observed that as packet arrival rate increases while the response time decreases. The overall response time for the CALB is outperforming the response time for the ECFT by around 9% on average as shown in Figure 13.

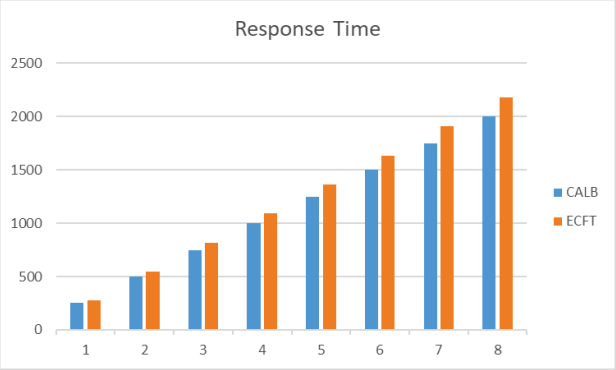

Figure 13: Average Response Time for CALB and ECFT

V. Conclusion and Future Work

This paper proposes a new architecture of fault tolerant distributed control plane using master controller election algorithm and switch migration technique. A new model called Adaptive Controller Load Balancing is proposed to provide Fault Tolerance (CALB). The key contribution of the paper is proposing a load balancing algorithm that the CALB is using to distribute the load among slave controllers based on a desired ratio. CALB assumes that migration occurs under two conditions either due to unbalanced load, or master controller failure condition. In existing static mapping between switch and controller affects the performance of network in case of load imbalance or controller failure. Implementation of master controller election module verified with controller throughput and controller response time. As the number of controller increases throughput increases sequentially when packet arrival rate greater than the capacity of floodlight controller throughput increases dramatically. Two algorithms are discussed in the paper to deal with two different scenarios. Master controller can distribute the switches based on predetermined ratio according to the workloads list stored at the master controller. Results are promising as the proposed model, improves by 12% in terms of the throughput and 9% in terms of the response time.

References

[1] Wolfgang Braun, Michael Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Future Internet journal*, vol. 6, pp. 302-336, Jan 2014

[2] Othmane Blial, Mouad B. Mamoun, RedouaneBenaini, "An Overview on SDN Architectures with Multiple Controllers," Computer Networks and Communications Journal, vol.2016, pp.8, 2016

[3] Wenfeng Xia, Yonggang Wen, Chuan H. Foh, DusitNiyato, HaiyongXie, "A survey on Software Defined Networking,", *in IEEE communication Surveys & Tutorials journal*, vol.17, pp. 27-51, June 2015

[4] Wael Hosny Fouad Aly, "LBFTFB Fault Tolerance Mechanism for Software Defined Networking", The International Conference on Electrical and Computing Technologies and Applications, 2017 (ICECTA'2017), AURAK, UAE, Nov 2017.

[5] Wael Hosny Fouad Aly, "A Novel Fault Tolerance Mechanism for Software Defined Networking", European Modelling Symposium on Mathematical Modelling and Computer Simulation, Manchester, UK, Nov 2017.

[6] Wael Hosny Fouad Aly, Yehia Kotb, "Towards SDN Fault Tolerance using Petri-Nets," The 28th International Telecommunication Networks and Application Conference (ITNAC 2018), Sydney, Australia, 21-23 Nov, 2018.

[7] Wael Hosny Fouad Aly, Abeer Mohammad Ali Al-anazi, "Enhanced Controller Fault Tolerant (ECFT) Model for Software Defined Networking," The 5th IEEE International Conference on Software Defined Systems (SDS), Barcelona, Spain, April 2018.

[8] Cheng, Guozhen, Hongchang Chen, Hongchao Hu, and Julong Lan."Dynamic switch migration towards a scalable SDN control plane." International Journal of Communication Systems 29, no. 9 2016: Page no.1482-1499.

[9] Zhou, Yuanhao, Mingfa Zhu, Limin Xiao, Li Ruan, Wenbo Duan, Deguo Li, Rui Liu, and Mingming Zhu. "A load balancing strategy of sdn controller based on distributed decision." In Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on, pp. 851-856. IEEE, 2014

[10] Hernandez Marulanda. "Implementation and performance of a SDN cluster-controller based on the OpenDayLight framework." (2016).

[11] Suh, Dongeun, Seokwon Jang, Sol Han, Sangheon Pack, Taehong Kim, and Jiyoung Kwak. "On performance of OpenDaylight clustering." In NetSoft Conference and Workshops (NetSoft), 2016 IEEE, pp. 407-410. IEEE, 2016.

[12] Macedo, Ricardo, Rafael de Castro, Aldri Santos, Yacine Ghamri-Doudane, and Michele Nogueira. "Self-Organized SDN Controller Cluster Conformations against DDoS Attacks Effects." In Global Communications Conference GLOBECOM), 2016 IEEE, pp. 1-6. IEEE, 2016.

[13] Stoller, Scott D. "Master controller election in distributed systems with crash failures." Technical report, Indiana University, April 1997, pg.169.

[14] Dixit Advait. "Towards an elastic distributed SDN controller." ACM SIGCOMM Computer Communication Review. Vol. 43. No. 4. ACM, 2013