International Journal on Emerging Technologies 10(2): 136-146(2019)

ISSN No. (Print) : 0975-8364
ISSN No. (Online) : 2249-3255

Comparison of Performance of Distributed Controller Fault Tolerance Model (DCFT) using Load Balancing in Software defined Networking in well-known Topologies

Gaurang Lakhani[1] and Amit Kothari[2]
[1]Research Scholar, Department of Computer Science and Information Technology, Gujarat Technological University, Ahmedabad, Gujarat, India.
[2]Sr. Software Engineer, Accenture India Bengaluru, India.

(Corresponding author: Gaurang Lakhani)
(Received 15 April 2019, Revised 27 June 2019 Accepted 19 July 2019)
(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: Lack of Flexibility, Centralized Control, and Cost are limitations of the traditional network. Software defined networking (SDN) adds flexibility and programmability in network management by separating the control plane from the data plane. Distributed controllers with SDN are logically centralized at control plane and physically distributed at data plane. They are deployed to improve the adeptness and accuracy of the control plane, which could isolate network into few subdomains with independent SDN controllers. Traffic is dynamic and configuration between switch and controller is static. If one of the controllers fails, load imbalance arises.To address this problem of fault tolerance in distributed controller DCFT (Distributed Controller Fault Tolerance) model is proposed in this paper. A novel switch migration method with coordinator controller in a distributed SDN controller is proposed for providing fault tolerance through load balancing. The system architecture of the proposed model with different modules such as coordinator controller election, loadcollection, decision taking, switch migration, Inter controller messenger designed. On failure of coordinator controller switch migration discussed. Implement DCFT model in Mininet, derived results, The results show that our proposal could achieve load balancing among distributed controllers while fault occurs, regardless network traffic variation and outperforms static binding controller system with communication overhead, controller load balance rate, and packet delay. We compare our model with CRD (controller redundancy decision), MUSM (maximum utilization switch migration) and ZSM (Zero switch migration) techniques. Simulation analysis performed on custom topology and two well-known topologies Abilene and Internet 2 OS3E from topology zoo. We compare packet delay, communication overhead and load balancing rate in a custom topology and other two well-known topologies with before and after migration of switches. It's revealed that the DCFT model produces better performance in fault tolerance.

Keywords. Software Defined Networking, Distributed controller, Fault Management, Switch Migration, coordinator Election, Load Balancing.

I. INTRODUCTION

Software-Defined Networking (SDN) is a new approach to network management and enable innovation in networking. Current traditional networks are complex and difficult to manage especially in light of changing routing and other quality of service demands of administrators. SDN separates the two main functions of a traditional networking device (switch or router) viz. packet switching and routing decision [1, 2]. The brain of the control plane is the SDN controller. The controller communicates with network devices through southbound Interface such as the OpenFlow protocol. The control plane disclosures some features and APIs through the Northbound Interfaces to network operators to design various management application exploiting such as a set of REST API [3]. East-West bound API used for inter-controller communication among multiple controllers. Network devices are dumb in SDN and worked as simple packet forwarding devices. Their control functionality centralized at the control plane. Forwarding decision depends on flow. Fig. 1 shows flow tables, set of packet field values to its related actions defined the flow. Forwarding devices provide the same service rules to all packets of the identical flow. The flow abstraction permits uniting the behavior of different types of network devices including routers, switches, firewalls, and middleboxes. Flow programming empowers unmatched flexibility, restricted only to the abilities of the applied flow tables. Well defined programming interface realized separation between the control plane and data plane by switches and SDN controller. One or more tables are available with OpenFlow switch for packet handling rules. Certain associated actions applied to rules in traffic. OpenFlow switch behaves like a router, switch firewall, etc based on the instructions received by the controller depending on the rules.

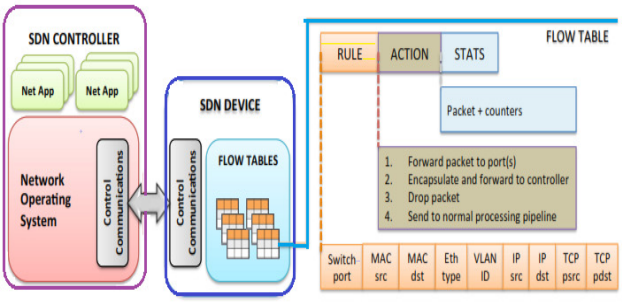

Fig. 1. Open flow enabled SDN device [14].

SDN adoption raises issues of scalability and reliability in centralized design [4]. That can be referenced with a physical delegation of the control plane. Such materially dispersed but logically centralized systems carry extra challenges. Logically centralized controllers liable for forwarding routing decisions, controller failure is a significant problem of SDN, a load of the futile controller should be spread between the rest of controllers. In this paper, we proposed a Distributed Controller Fault Tolerance model (DCFT) using load balancing in SDN.

Few previous papers [5-9] have explored switch migration to provide load balancing but existing projected algorithms can only work with load imbalance they cannot work with the event of controller failure. Control plane is distress from a lack of fault tolerance. For a distributed control plane coordinator election algorithm used to identify unique coordination between all SDN controllers.

The core involvement of this paper is described as follows

- A novel switch migration method with coordinator controller in a distributed SDN controller is proposed for providing fault tolerance through load balancing.
- The system architecture of the proposed model with different modules such as coordinator controller election, load collection, decision taking, switch migration, Inter controller messenger designed.
- On failure in coordinator controller, switch migration discussed.
- Implement the DCFT model in Mininet, derived results. The results show that our proposed model could achieve load balancing among distributed controllers while fault occurs, irrespective of network traffic variation and outperform motionless binding controller system with communication overhead, controller load balance rate, and packet delay. Verify the DCFT model on custom topology and the other two well-known topologies.

Rest of the paper is planned as follows. Section II demonstrate literature survey Section III shows proposed DCFT model system architecture with different modules. Section IV represents design and implementation with the proposed switch migration algorithm along with coordinator failure, ordinary controller failure, and load imbalance. Section V reports with simulation analysis. Section VI presented with a conclusion and section VII reports references.

II. LITERATURE SURVEY

A. The distributed control plane in SDN

SDN architecture divided into application plane,control plane and data plan, Many controllers are taken for distributed environment from different domains. Data plane and application plane discussion is out of scope of this paper. Focus given on the architecture of distributed SDN control plane, which is divided into flat SDN control architecture and hierarchical SDN control architecture.

(1) Flat architecture

Flat architecture implies horizontal partitioning of the network into different regions, each will be taken care by single controller in-charge dealing with a subgroup of SDN switches. ONOS [15], Onix [22], Hyper Flow [23] and Open Day Light [24] are flat SDN distributed controllers.

Each controller is statistically associated with certain switches and exclusively handles demands from them In ONOS [15]. In the interim to provide focal view and control among the network, the controllers intermittently synchronize organize data and direction with one other.

Onix [22] panels the NIB (Network Information Base) giving each controller instance duty for a subset of the NIB and it totals by making application decreases the fidelity of the information before sharing it between other Onix instance within the group. Hyperflow [23] used WheelFS as a distributed file system to form a global network view and each controller assumes responsibility for its system. The harmonization between controllers ought to be declared for certain occasion such as link status alterations that could affect the network view. The Open Day Light [24] controller jerks by building the data structure trees by utilizing the Yang modeling language and MD-SAL. They concentrated on necessary components to realize spread control plane, give a worldwide perspective of the network topology for higher level applications.

Weal *et al.*, [39] described LBFTFB (*load balancing to support fault tolerance using feedback control for SDNs),* model. It reduces the cascading failure problem effect. Compared with the Hyperflow [23] model LBFTFB outperforms by 16% in terms of packet loss and packet delay. Our model followed horizontal architecture. More details mentioned in the next sections.

(2) Hierarchical architecture

The hierarchical SDN control architecture accepts that the network control plane is vertically divided into a different dimension (layers) reliant upon the essential services.

Kandoo [32] expect progressive two layer control structure that segments control application into local and global. Contrast to Devoflow [33] and DIFANE [34], Kandoo suggests diminishing the general weight of the control plane without the need to alter OpenFlow switches. It set up two dimensions control plane where frequent events happening near the data path are handled by the bottom layer, and non-local events requiring network vide view dealt by the top layer.

B. Coordinator election algorithm

Esteban Hernandez *et al.*, [26] described a coordinator election algorithm using Raft consensus method to provide fault tolerance to the distributed control plane. Raft algorithm is the consensus algorithm for managing replicated logs. Raft algorithm permits a set of nodes or servers to collaborate as a unique comprehensible system that is able to handle disappointments of some of its nodes. It can be done by replicating state machine of the coordinator.

C. Switch migration algorithm

Dixit *et al.*, [5] work towards the utilization of controller resources using load balancing and reduce the power consumption by switching off under loaded controllers from the controller pool. Dixit, Advait Abhay, *et al.*, [6][12] proposed detailed and enhanced distributed control plane and switch migration protocol compare to their previous work viz. towards an elastic distributed SDN controller. They have proposed three properties to provide successful migration of a switch but fault tolerant mechanism and how to select a controller or switch to migrate were not discussed.

Liang, Ryota *et al.*, [7] proposed an architecture to balance the load among controllers. Controller with the role of coordinator calculates the load and take a decision for migration of switch. They have proposed a switch migration algorithm that can provide crash free migration. Yanyu Chentt, Qing Lit *et al.*, [8] proposed an elastic architecture that can change switch controller mapping as per the load condition. Cheng, Guozhen, *et al.*, [9] work towards Balance a load of control plane by switch

migration using parameters optimization of CPU, bandwidth and memory. Zhou, Yahoo*, et al.*, [10] work towards controller dynamic and adaptive load balancing algorithm for a distributed architecture. There is no centralized component. Each controller runs DALB (Dynamic adaptive load balancing) and collect a load of other controllers and make the decision to migrate switch. Yu, Jinee*, et al.*, [11] work towards load balancing in distributed controllers by switch migration. The focus of this work is to make a load balancing decision locally to reduce the migration time. Their algorithm can't work the event of controller failure.

Hu, yannan *et al.,* [21] referenced method of uneven burden problem in the distributed controller. The centralized node used for load balancing, a centralized controller is constrained by memory, CPU power, and bandwidth. Moreover, a centralized node gathers load information intermittently and it talks a lot of messages often with other controllers, which will prompt to performance reduction of the whole system. Aly *et* al. [18, 25] mentioned the selection of destination backup controller based on a span between switches and target controller, existing load and percentage of packet loss. The span between a switch and backup controller influence the packet response time. Which moves the network model efficiency. Our model considered the workload of the destination backup controller.

Katta *et al.*, [27] depicted Ravana, conveyed convention for fault tolerant SDN. Ravana forms the control posts transactionally and precisely once (at together the controllers and the switches). Ravana keeps up these certifications even with controller and switch crashes. The key understanding in Ravana is that reproduced state machines can be reached out with lightweight change side components to ensure accuracy, without including the switches in a detailed accord convention [16].

Botelho *et al.*, [28] mentioned replicated data store used as a central component of the design of this method. Data store implemented as fault-tolerant replicated state machine for storage and coordination operations. One controller configured as primary and other as backup. All controllers run the Lease management algorithm. The primary controller contains a cache of the data store.

Obadia *et al.,* [29] address problem of failover for distributed SDN controllers by proposing two strategies for neighbor dynamic controllers to assume the control of vagrant OpenFlow switches (1) greedy incorporation and (2) prepartioning among controllers. They utilized a model with distributed floodlight controllers to assess the techniques the outcome demonstrates that the failover term with the unstable methodology is corresponding to the no of vagrant switches while the pre-partitioning approach proposing a very little extra control traffic, empowers to respond faster in under 200 ms.

Fonseca *et al* [30] described resilience improvement in NOX controller through a primary-backup approach. Contrasting the distributed approach where the controller will want to gather information from each switch. Switch loss connection with a controller checked by probe message sent intermittently to the controller.

Hu tao *et al.,* [31] depicted distributed decision mechanism (DDM) built on switch migration in the multiple subdomain SDN networks. Through gathering network information, it develops distributed migration choice fields dependent on the controller load condition. Then migrating switches conferring to the selection chance, and the target controllers are dictated by integrating three network costs, including information accumulation, switch migration, and arranged switch movement. Lastly, set the migrating countdown to achieve the ordered switch migration. In this proposal no provision of controller disappointment or any adaption of failure activity discussed.

III. PROPOSED SYSTEM ARCHITECTURE OF THE DCFT MODEL

Different modules of the distributed controller are shown in Fig. 2. Modules described as follows.

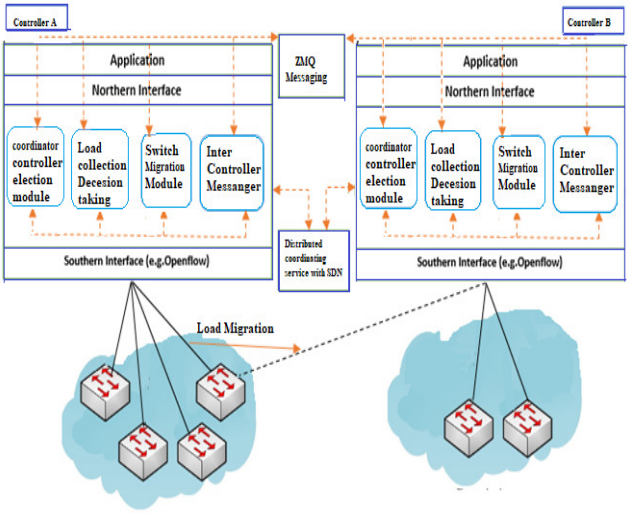

Fig. 2. Architecture of the distributed control plane with load balancing [10].

A. Coordinator controller Election module

Coordinator Controller of the system decided by this module. It will be available all the time in the cluster to take various coordination decisions in case of load imbalance as well as controller failure and to collect and calculate controller statistics. It stores each controller IP address, capacity, associated switches data. The controller's IP address recognizes each controller, while limit chooses whether the controller is equipped for overseeing more switches. The limit of the controller chosen by various streams every second that the controller can process if the load of the controller beyond the controller's threshold, the controller fails. The coordinator controller intermittently receives the current load of each controller and switches. Controller's current load defined by a load of the controller at a given time. Load of the controller defined with a number of flows per second that the controller receives from the switches. Coordinator controller checks intermittently the status of the controllers. To detect the failure of the controller, coordinator controller uses controller information. For every particular time coordinator controller checks the last refreshed time of the controller's current load. If the last refreshed time exceeds a certain threshold, the coordinator controller decided the given controller as a futile controller and take the next step to recuperate the controller disappointment.

The election module continuously running in the background, when it detects the failure of a current Coordinator it starts re-election and elects a new Coordinator. The election module can elect a new coordinator if and only if the 51% of the controllers are active, it's in order to ensure that there is at least one group which will produce a majority response to elect one coordinator. Otherwise, it sets the controller having id c_1 as the default Coordinator.

B. Internal controller messenger module

This module is responsible to provide all the updates of controllers of the cluster to each other. It synchronizes state between the controllers by letting all of them access updates published by all other modules in the controller. ZMQ, the asynchronous messaging service used for internal communication among controllers. Distributed coordination service such as zookeeper [17] glues cluster of the controllers to share the information about a link, topology, etc. it's used for updating the status of the controllers.

C. Load Calculation and decision taking module

In the load calculation module, all the controllers including Coordinator controller calculate its own load and send load information to the Coordinator controller. Load of the controller consists accumulation of load of the switches. With an enormous scale of flow table entries, the controller deals a big flow table and a load of the controller will be high. The bigger average message arrival rate of a switch shows this switch conveys more load to the controller. Propagation delay also impact factor. If the controller is overloaded, we choose to switch to migrate considering the following formula.

Load of the switches comprises a number of flow table entries (N), average message arrival rate (F) and propagation delay (D).

$$C_{Load} = w_1*N + w_2*F + w_3*D \quad (1)$$

Where w_1, w_2, and w_3 are weight coefficients and their sum is 1.0. Similarly, compute load of each switch based on their flow table entries, and compute the total load of the controllers depending on the number of switches.

Coordinator controller collects load information and stores it in the distributed database. Coordinator store load information as an array list sorted in ascending order. The first member of array list is a minimum loaded controller and the last member is maximum loaded controller without any duplicate entry. Later a quantified time interval of every 5 seconds, the load calculation module calculates the load and sends to Coordinator. The time interval can be adaptive or dynamic. The time interval can be set by the aggregate of the current load and previously calculated load balancing.

(i) Load Calculation Threshold

T=Tmax / (|CurrentLoad – PreviousLoad|+1)

Tmax = initially set interval
Current Load = Controller's Current Load
Previous Load = Controller's Previous Load

After receiving the load information Coordinator store load of each controller and aggregate load of all the controllers in a distributed data store.

(ii) Decision Taking Module

To balance the load of all the controller nodes, a threshold value C is decided to detect overload and under load condition. Based on this threshold value Coordinator decide to balance the load or not.

C= (Average of a load of all the controllers)/(a load of a maximum loaded controller)

$0 \leq C \leq 1$, C is the load balancing rate. If C will be close to 1 load is evenly distributed and if a load is close to 0 uneven load distribution is there. We have selected an initial load balancing rate is 0.7. If the value of C is less than 0.7 than load balancing is required. If the value of C is greater than 0.7 no need for load balancing [10].

(iii) Selection of Destination backup controller and switch to be migrated before migration, Coordinator must check that migrated switch should not overload the destination backup controller. Following formula used to check to an overload of destination controller on the migration of switch. If the migration can create an overload to destination Coordinator should choose another switch to migrate.

Load_of_Switch_to_Migrate ≤ CT – Load_of_Target

CT= Controller Capacity (packets/Sec)

Authors [18] mentioned a selection of destination backup controller based on the remoteness between switches and target controller, current load and percentage of packet loss. The span between a switch and backup controller affect the packet response time. Which influences the network model efficiency.

Our proposed switch migration algorithm (mentioned in section IV) to assigns switches to the adjacent standby controller with considering outstanding workload on the destination standby controller steps of assignment of the switch as follows.

1. Allocate each switch to n standby destination controller can be from sorted array list of the closest controller. array list stored at the distributed data store.
2. Each time span t, controller loads are processed based on eq. (1). The lightest loaded controller has selected whose load is less than the bellow capacity CT. The selection of switch to be migrated based on formulae of eq (1) as mentioned above.
3. Reorder switches the backup list according to the controller weight.
4. The maximum loaded switch should be select to migrate.
5. The coordinator controller found a failed controller, then it found the switches of the futile controller.
6. Repeat steps for the failed controller's associated changes of switches to check the standby list of the controller.
7. Check the accessibility of each standby controller in the standby controller list.
8. In the event of the first standby, the controller can endure the switch, the coordinator controller sends switch to the IP address of the controller.
9. On the off chance first backup controller can't endure the switch, the coordinator controller forms the succeeding accessible standby controller.
10. Steps 2 to 9 repeated until coordinator controller allots switch to an appropriate standby controller while the controller load variations over time.

IV DESIGN AND IMPLEMENTATION

- **Migration**

  Switch migration occurs in three situations. (1) Coordinator controller failure (2) ordinary controller failure (3) Load imbalance Pseudo code for three conditions as follows.

Algorithm: Switch migration process

/(a) Coordinator controller failure */*
Input: *c1… c2, cn controllers, coordinator controllers, threshold value*
Output: *Balanced distributed controllers*
1. Call coordinator controller election module for deciding new coordinator.
2. **if** all the switches migrated to the neighbor controller **then**

if (capacity of neighbor controller > threshold) **then**
a neighbor controller may be overloaded due to migration and crashed.
else
all the switches migrated and switch-controller mapping updated in a distributed database
endif
3. **if** all the switches migrated to other controllers equally **then**
check each controller capacity and switch-controller mapping updated in a distributed database
endif
4. **if** all switches migrated to the least loaded controller **then**
find least loaded controller from a distributed database and update switch controller mapping in a distributed database
endif
/(b) ordinary controller failure */*
5. **if an** ordinary controller failed **then**
coordinator controller select least loaded controller from distributed database
if (capacity of least loaded controller > threshold) **then**
call switch migration module and migrate switches
else
migrate few switches upto a limit of threshold and assign remaining switches to next least loaded controller
endif
endif

/ (c) Load Imbalance */*
6. **if** (capacity of ordinary controller >threshold) **then**
call switch migration module and migrate highest loaded switches to the least loaded controller
endif

Migration can be encounter in three cases, (a) coordinator controller failure (2) ordinary controller failure (3) load imbalance. In all cases, switch migration carried out.
Coordinator controller performs two roles, one is its ordinary role of routing incoming packets and second is a special role, Coordinator role, where it has to calculate the load of each controller of the cluster and information about the mapping of switch-controller and store it as an array list at the distributed database. All the controllers send its load information and switch information to the Coordinator controller. Coordinator controller calculates the aggregate load of all the controllers and stores it in the distributed database. Based on a load of the cluster, Coordinator controller takes the switch migration decision. Controllers can communicate with Coordinator using messaging services provided by ZMQ and SyncService of a floodlight. Each switch must be connected to one controller with a master role and with any no of controllers with a slave role.

A.Failover mechanism in the proposed system
The whole network divided into a logical cluster of controllers. All controllers of a cluster are assigned a controller id as per they joined the controller cluster viz. C1, C2…Cn. When cluster start, a controller having maximum controller id is elected as a coordinator controller using our election algorithm.
Failure in coordinator controller. The coordinator is the in-charge of the coordination of all the other controllers, controllers may have a different number of switches. Failure occurs in the coordinator node leads failure of a whole distributed control plane. Failure of coordinator can be detected by using separate function available with all the controllers in the cluster which will be synchronized with ZMQ and syncdb. Coordinator controller fails, aggregate load calculation stopped, a decision of load balancing cannot be taken, which leads towards the failure of an overloaded controller.
To overcome the failure of a coordinator controller we plan to run an election algorithm to elect a new coordinator on a failure of the current coordinator. Controller id decides priority among controllers. After a specified time interval, a check performed that elected coordinator is active or failed. If coordinator failed, the re-election starts. A controller having maximum controller id from the cluster, elected as a new coordinator of a distributed control plane. A new coordinator has to migrate switches of the failed controller to a lightest loaded controller by proposed switch migration. All the controllers may have a different number of switches. Fig. 3 shows failure in coordinator controller. C_{10} is a current coordinator, Switch of C_{10} migrated to C_7 (lightest loaded backup controller from the array list. C_9 becomes a new coordinator. Similarly array list from distributed data store updated at every time t seconds.
In our model, the coordinator controller periodically checks the status of the controllers, to perceive the failure of the controller, coordinator controller utilizes controller data, Every particular time coordinator controller checks last refreshed time of controllers If last refreshed time surpasses a certain threshold, coordinator controller think about this controller as failed and proceeds recovery steps.

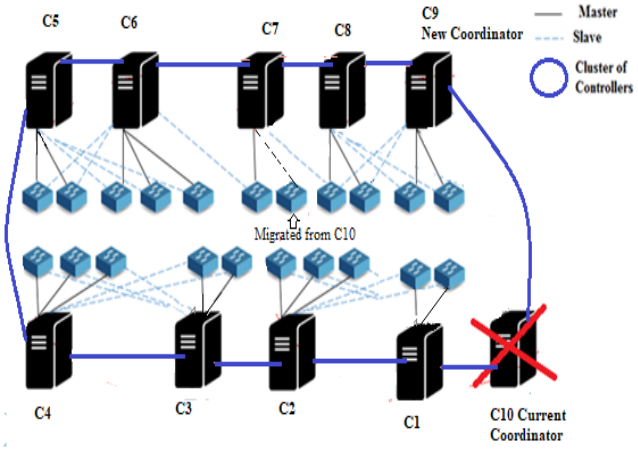

Fig. 3. Failure in coordinator controller, the election of new coordinator controller.

Failure in an ordinary controller. Coordinator controller manages the failure of an ordinary controller by using an array of least loaded controllers stored at the distributed database. On failure of any ordinary controller, its orphan switches will be migrated to the first least loaded controller, limited switches up to threshold value only migrated to the least loaded controller, rest switches if any migrated to next controller of the array.
Load Imbalance between controllers. Similarly, load imbalance occurs on overloading of a controller, the overloaded controller needs to migrate its highest loaded switches to the least loaded controller from an array of a distributed database.

B. The proposed Switch Migration process
Controllers having three roles master, slave and equal [10]. OpenFlow protocols 1.5.1 specification [19] included the capacity for a controller to set its part in the multi-controller condition. In OpenFlow protocols version 1.4 onwards the job status message empowers the switch to advise the controller about changes its part.

The default job of a controller is OFPCR_ROLE_EQUAL [19]. The controller has full read-write access to the switch equal to rest of the controller in the same role. As a matter of the course, the controller gets all the switch nonconcurrent messages (such as packet-in, flow-removed). The controller can direct controller-to-switch directions to alter the conditions of the switch. The switch does not do any intervention or asset distribution among controllers.

A controller can demand its job to be transformed into OFPCR_ROLE_SLAVE. In this job, the controller has read-only access to the switch. As a matter of course, the controller does not get switch asynchronous messages, aside from Port-position messages [19]. A controller can demand its job to be changed to OFPCR_ROLE_MASTER. This job is like to OFPCR_ROLE_EQUAL and has full admittance to the switch, the thing that matters is that the switch guarantees it is the main controller in this job. At the point when the controller changes its part to OFPCR_ROLE_MASTER, the switch transformed the present controller with the job OFPCR_ROLE_MASTER to have the job OFPCR_ROLE_SLAVE, yet does not influence controllers with job OFPCR_ROLE_EQUAL. At the point when the switch makes such job transformed, if a controller job is transformed from OFPCR_ROLE_MASTER to OFPCR_ROLE_SLAVE, the switch must produce a controller job status occasion for this controller educating it of its new state (much of the timing controller is never again reachable, and the switch will most likely to transmit that occasion).

Each controller may direct an OFPT_ROLE_REQUEST message to convey its job to the switch and the switch must recollect the job of each controller connection. A controller may alter its job whenever, gave the generation_id in the message is present [19].

The job demand message offers a lightweight system to enable the controller master decision process, the controllers design their job normally still need to facilitate among themselves. The switch cannot change the condition of a controller all alone, controller state is constantly transformed because of as a result of a solicitation from one of the controllers. Any Slave controller or Equal controller can choose itself, Master. A switch might be at the same time associated with different controllers in Equal state, multiple controllers in a Slave state, and at most one controller in Master state. The controller in Master state (assuming any) and every one the controllers in Equal state can completely change the switch state, there is no mechanism to implement partitioning of the switch between those controllers. On the off chance that the controller in Master job should be the main controllers ready to make changes on the switch, at that point, no controllers ought to be in Equal state and every single controller ought to be in a Slave state.

As shown in Fig. 4, Destination backup controller selected, switch decided to be migrated following steps performed for the switch migration process. All the handshakes in this protocol are using ZMQ [20]. Initially overloaded controller A connected as master with switch s and in slave role with controller B. Coordinator controller sends a switch migration request to selected destination controller. There is no need for the reply to this message.

1. After the receipt of the load migration request selected destination controller to send role change request (from slave to master) to the switch which needs to be migrated.
2. Switch replied configured destination underloaded controller as now master, from now original master no longer able to receive any packet-in message from a switch.
3. Destination controller sends End Migration message to the Coordinator. Coordinator update controller switches mapping in a Distributed Database.

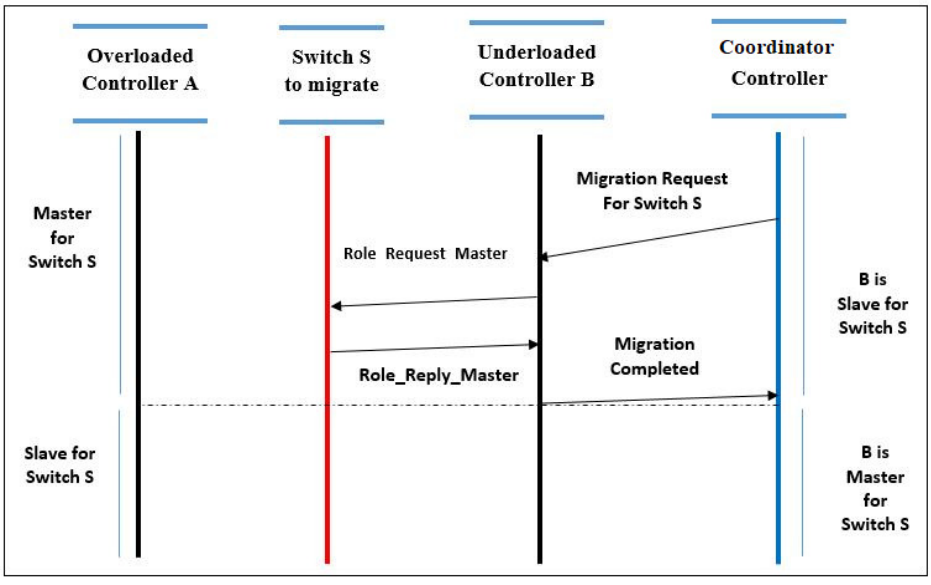

Fig. 4. Proposed switch migration process in the overloading of the controller.

V. SIMULATION ANALYSIS

We analyze the results of simulations of custom topology in our research paper [40]. In this paper, it is not discussed to save space. We use experimental testbed for simulation as mentioned in table 1. Physical devices contain ten machines with the configuration mentioned in table1. In the cluster, there is only one master controller, which enables programmed network management. We did many experiments to demonstrate the performance of the DCFT model. DCFT compared with some other mechanism such as Zero Switch Migration (ZSM), Controller Redundancy Decision (CRD) [6] and Maximum Utilization Switch

(MUSM) [13]. There is just one controller in ZSM. Overloaded controller randomly migrates switches to a nearby underloaded controller to resolve the load imbalance problem in CRD. In MUSM overloaded controller migrates switch into the controller that has a maximum outstanding capacity. DCFT model reduces packet delay, increased no of request processing by each controller, load balance rate and improve fault tolerance.

In our topology switch can be well-ordered by one master controller. A controller can control more than one switch. In the meantime, there are many slave controllers for the switches. the new master will be chosen from the slave controller in case of failure of the original master. We consider custom topology in Fig. 5. Traffic patterns are shown in Table 2 used for all simulations. We use two more well-known topologies for comparison of results.

In Hyperflow [23] controller fault tolerance technique directs the failed controller without considering the controller's current load. Which leads to packet loss, cascading failure and packet delay or latency. The proposed DCFT lessens the effect of these problems by allocating the controller's load among rest of controllers when a point of disappointment happens. It is performed by the coordinator controller. So DCFT model used for load balancing performance, topological adaptability and reveals fault tolerance.

Table 1: Simulation Testbed.

Software	Version	Function
Mininet[35]	2.2.1	Network Emulator tool
Floodlight[36]	1.2	SDN Controller
OpenFlow	1.5	Communication Protocol
Linux	Ubuntu 16.0.4 64 bit	An operating system on each virtual machine
RAM	8 GB	Main memory
Processor	Intel ® Core TM i3 2370 M CPU 2.4 GHz	Processing, coordinating all processes
Traffic	hping3	Traffic generator tool
Bandwidth	1000 Mbps	Between switch and hosts
Packet arrival rate	500 packets/s	Switch-controller

Table 2: Traffic designs used in the experiment.

Traffic sequence	Source	Destination
T1	H1	H4
T2	H8	H12
T3	H13	H18

We use hping3 to generate TCP flows to simulate the dispersal of network traffic the average flow requests The average packet arrival rate 500 packets/s. we use a floodlight controller to process packets received by the switch. To decrease the effect of packet delay and packet loss link bandwidth between switches and hosts to 1000Mbps. Packet in rate P = 30 Bytes/s. we set no of switches managed by one controller is from 2 to 10. All the simulations run for 12 Hours readings noted at every 20 minutes.

Consider the topology shown in Fig. 5. DCFT model takes interruption among switches and their associated controllers to curtail the response time.

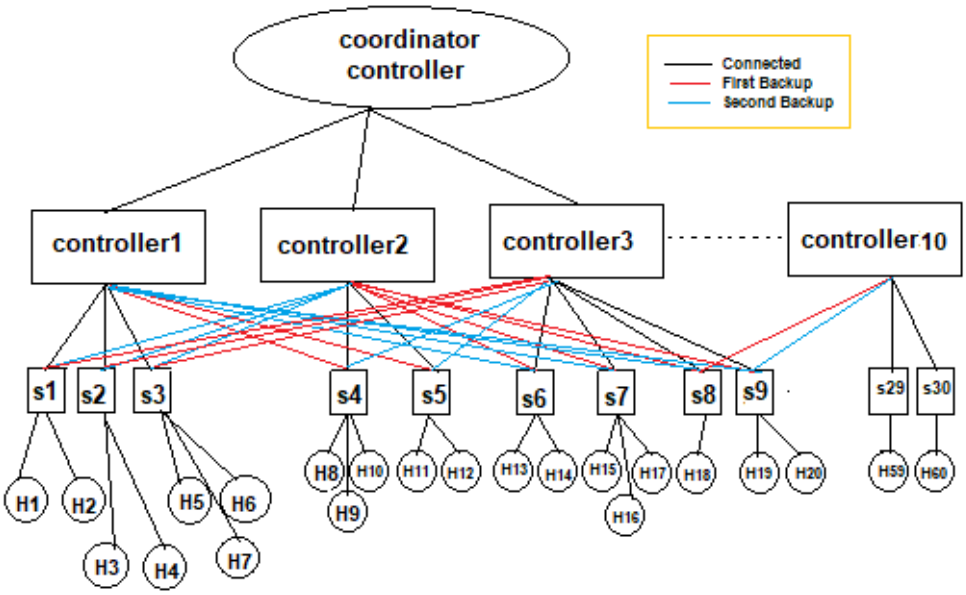

Fig. 5. The logical perspective of the topology used in a simulation.

A. Packet delay or latency

Consider traffic patterns T1, T2 and T3 of table 2. Traffic T1 generated from host H1 to host H4. Both are connected by controller C1. Simulation experiment starts with a packet delay of 12-14 ms for all traffics. After controller C2 falls flat at 18 seconds, coordinator controller manages controller C1 for it. Controller C1 assumes the responsibility of the switches related with controller C2 at 20 seconds because C1 is the nearest controller and lightest loaded compared to rest of the controllers. We assume that the CRD and MUSM mechanism takes the same recovery time as DCFT. Packet delay increases in traffic T1, T2 and not affected in traffic T3, because T3 not affected by switch migration.

Regarding DCFT, coordinator controller recoups the disappointment of controller by allocating the load of the failed controller C2 among C1 and rest of the controllers. This migration grounds an expanded number of solicitations to every controller then the blockage in this controller lead to the packet delay. The extreme packet delay for traffic T1 is 24.27 ms at 52 seconds, for traffic T2 is 27.32 ms at 57 seconds and for traffic, T3 is 17.2 ms at

55 seconds. Numerical results shown in Fig. 9 depicts the lowest packet delay by DCFT model compared to other methods of switch migrations. Packet delay reduced by our model is 21.63 %. Coordinator controller can't recover the controller C2 failure by migrating switches to least loaded controller C1 only, as it will be overloaded on migration.

Flow request coun t(Network traffic) of the custom topology, Abilene topology, Internet 2 OS3E topology shown in Fig. 6 (a), (b),(c) respectively. When the load imbalance occurs, packet delay (latency) increased, we change flow request count to overload controller and observe packet delay of the custom topology in figure 9. It is observed for all traffic sequences, packet delay for the custom topology and Internet 2 OS3E topology decreased but for only T3 in Abilene topology, it is increased in the DCFT model.

B. Communication overhead

The Communication overhead is created between switch-controller and between controller-controller. Rule installation in OpenFlow switches grounds wasteful network operation inferable from the high overhead potential on the OpenFlow controller. DCFT model demonstrated pursued routine with regards to fix a rule in the switches for a least flow entry in the network switches without damaging the network operation itself. Fig. 7 depicts a communication overhead for the custom topology and rest two well-known topologies.

Since ZSM just arranges single controller, the correspondence overhead among controllers is 0. The single controller is easily in the overloaded state since it needs to process all the flow demands. In this way, correspondence overheads among switches and controllers are most extreme in ZSM. CRD migrates switch to the nearest controller to streamline the choice of target controller, which brings down the overhead between controllers. On the other hand, closest migration is generating traffic congestion that increases communication overheads among switches and controllers. if many switches swarm into the nearest controller at the same time. MUSM lessens overhead by appending an additional controller and the communication overhead between switches and controller lowest.

DCFT model considers multiple costs and adopts a greedy algorithm to look for the ideal outcome. Design of the DCFT model reduces information interaction of irrelevant controllers by taken "first packet" of flow, which is sent to the controller for the purpose of flow acknowledgment and rule installation. The controller removes all the first packet payloads including VLAN id, source, and destination MAC addresses, IP addresses, ethertype, port and matches actions information so that the succeeding packets are hopped of the next switches as the first packet already holds and distribute forwarding information and reduces communication overhead. Communication overhead of the DCFT model is lowest among all other methods in controller-controller and switch-controller communication. Average communication overhead is reduced between switch-controller by 44.47% and controller-controller is reduced by 48.12% in custom topology, similarly, communication overhead between switch-controller and controller-controller is reduced by 47.09% and 38.47% in Abilene topology, and Internet 2 OS3E it is reduced by 43.09 % and 49.36% respectively as shown in Table 3.

C. Controller load balancing rate

We noted the number of requests processed by each controller and reflect the distribution of controller loads. ZSM has only one controller there is no load balancing. We relate the result of CRD, MUSM, and DCFT for three controllers in the given topology, which are shown in figure 8. CRD has a large difference in the number of requests handled by each controller. MUSM on second place and DCFT has a small variation. As CRD migrates the switch to the closest controller, switch migration frequently performed if neighbors of the overloaded controller receiving too many migrating switches. Controller load balancing rate increased by 17.81 % in custom topology, 18.54% in Abilene and 14.52% in Internet 2 OS3E topology as shown in Table 3.

Table 3: Maximum packet delay(ms), Communication overhead and load balancing rate in DCFT model (before/after switch migration.

	Before switch Migration								After switch Migration (At 15 second C2 failed)						
	Traffic Number	Maximum Packet delay (ms)	Communication Overhead(KB/s)		Load balancing rate (packets/s)				Maximum Packet delay (ms)	Communication overhead (KB/s)		Load balancing rate (packet/s)			
			Switch-controller	Controller-controller	C1	C2	C3	…C10		Switch-controller	Controller-controller	C1	C2	C3	…C10
Custom topology	**T1**	40.1	423	372	503	511	475	492	**24.27**	**237**	**193**	**496**	**521**	**489**	**510**
	T2	30.9	394	321	478	524	434	461	27.32	206	134	492	536	442	481
	T3	16.8	254	212	474	504	468	475	17.2	148	126	483	514	478	497
Abilene topology	**T1**	32.5	512	328	521	514	483	461	20.4	264	186	587	572	534	483
	T2	26.8	453	392	456	519	422	434	**14.8**	**217**	**158**	**587**	**624**	**537**	**501**
	T3	14.3	236	312	492	491	460	472	15.2	154	128	501	539	510	486
Internet 2 OS3E topology	**T1**	42.5	620	226	541	532	492	481	30.4	320	183	642	637	621	498
	T2	35.8	356	280	461	516	417	461	23.6	224	172	521	608	484	479
	T3	18.6	268	219	452	482	453	424	**18.2**	**164**	**134**	**524**	**556**	**526**	**438**

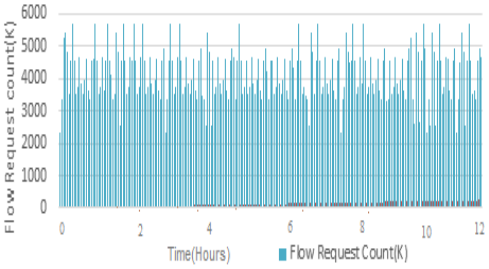

Fig. 6(a) Network traffic in custom topology.

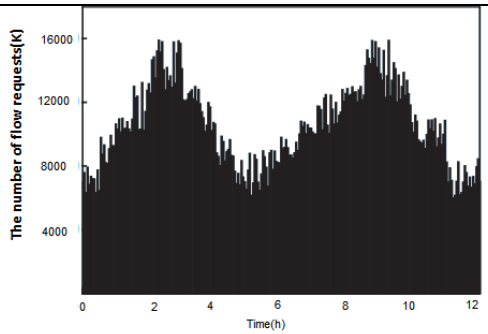

Fig. 6(b) Network traffic on Abilene topology[37].

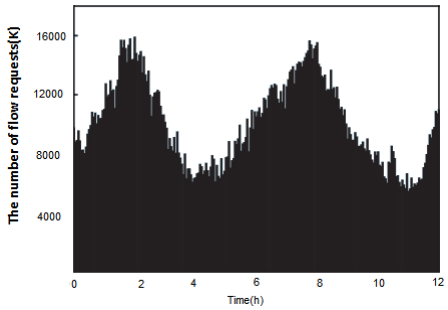

Fig. 6(c) Network traffic on Internet 2 OS3E topology [38].

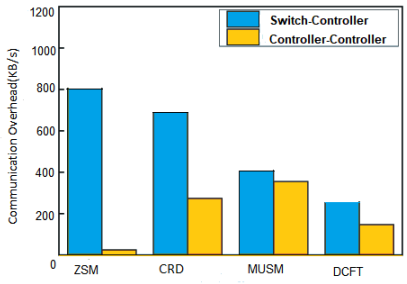

Fig. 7 (a) Communication overhead in custom topology.

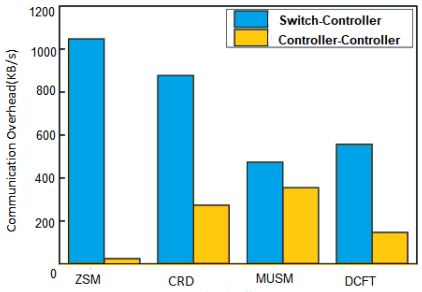

Fig. 7 (b) Communication overhead in Abilene.

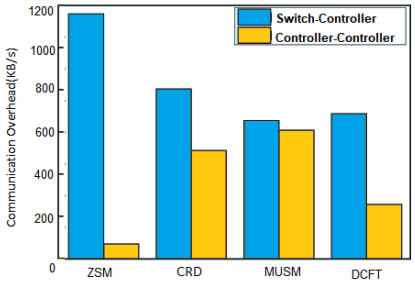

Fig. 7(c) Communication overhead in OS3E.

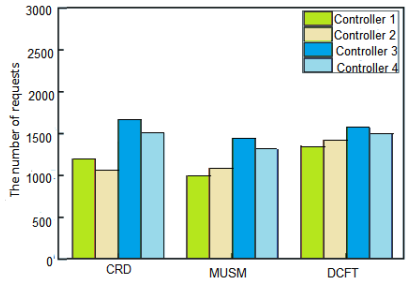

Fig. 8 (a) Request processed by each controller – custom topology.

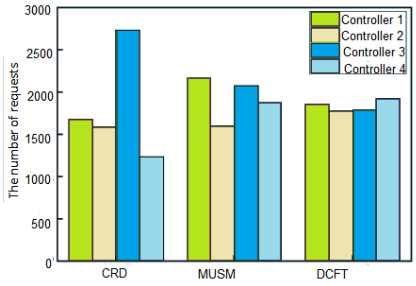

Fig. 8 (b) Request processed by each controller-Abilene.

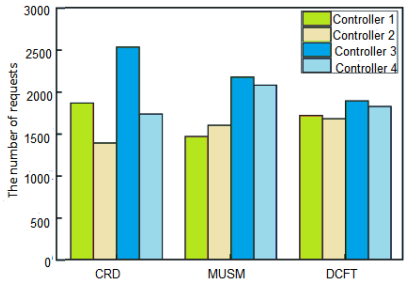

Fig. 8 (c) Request processed by each controller-OS3E.

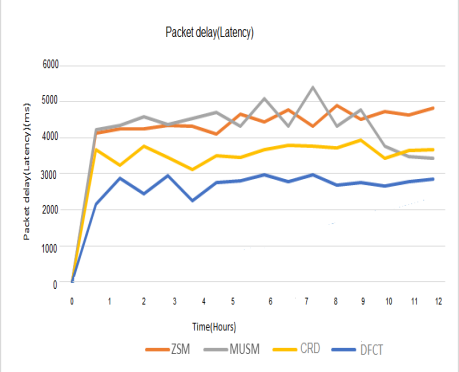

Fig 9(a) Packet delay(latency) in custom topology.

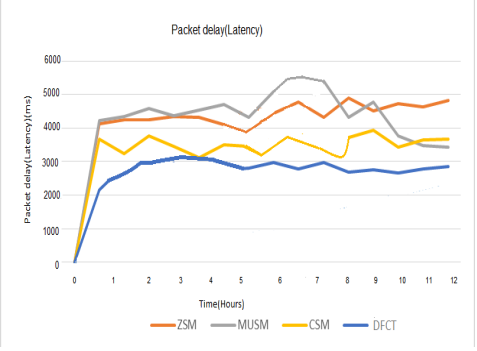

Fig 9(b) Packet delay(latency) in Abilene topology.

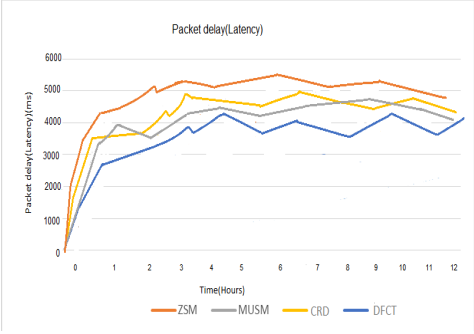

Fig 9(c) Packet delay(latency) in OS3E topology.

VI CONCLUSION

In this paper, we did a study of fault tolerance in the distributed controller with software defined networking. Paper introduced with the introduction of software defined networking with open flow devices. Distributed control plane with a flat SDN controller and hierarchical SDN controller discussed.

Simulation analysis performed with series of experiments performed using traffic patterns (Table 2), on custom topology, two well-known Abilene [37] Internet 2 OS3E [38] topologies ten controllers along with coordinator controller Communication overhead, controller load balance rate, packet delay used as evaluation indexes. It is found in Fig. 9 that DCFT model reduces packet delay by 21.63 %. From figure 7 Average communication overhead is reduced switch-controller by 44.47 % and controller-controller is by 48.12%. From figure 8 Controller load balancing rate is increased 17.81 % in custom topology, 18.54% in Abilene and 14.52% in Internet 2 OS3E topology. It is concluded that by reducing communication overhead, packet delay and increasing load balancing rate DCFT model contributes better in fault tolerance in the distributed control plane.

Our upcoming work focuses on an analysis of the failure of both switch and controller and finds a more refined technique to allocate a load of futile controller among other controllers founded on techniques of AI [41].

VII Acknowledgement and conflict of Interest.

We are very much thankful of Dr. Bhushan Trivedi, Director, GLS Institute of Computer application, Ahmedabad, Gujarat, India, Dr. Satyen Parikh, Dean, Faculty of Computer Application, Ganpat University, Kherava, Mehsana, India, all our collegues,friends and reviewers,editor of the International Journal of Emereging technology for their comments on our research work. We are hereby declare that we don't have any conflict of interest with any living person about the said research work mentioned in this paper.

REFERENCES

[1]. Yu, Y., Li, X., Leng, X., Song, L., Bu, K., Chen, Y., and Xiao, X. (2018). Fault management in software-defined networking: A survey. *IEEE Communications Surveys & Tutorials*, **21**(1), pp. 349-392.

[2]. Perešíni, P., Kuźniar, M., and Kostić, D. (2015). Monocle: Dynamic, fine-grained data plane monitoring. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies,* pp. 32.

[3]. Scott, C., Wundsam, A., Raghavan, B., Panda, A., Or, A., Lai, J., and Acharya, H.B. (2014). Troubleshooting blackbox SDN control software with minimal causal sequences. In *ACM SIGCOMM Computer Communication Review*, Vol**. 44**, No. 4, pp. 395-406.

[4]. Mahajan, K., Poddar, R., Dhawan, M., and Mann, V. (2016). Jury: Validating controller actions in software-defined networks. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 109-120.

[5]. Dixit, A., Hao, F., Mukherjee, S., Lakshman, T.V. and Kompella, R. (2013). Towards an elastic distributed SDN controller. *ACM SIGCOMM computer communication review*, **43**(4), pp. 7-12.

[6]. Dixit, A., Hao, F., Mukherjee, S., Lakshman, T.V., and Kompella, R.R. (2014). ElastiCon; an elastic distributed SDN controller. In *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 17-27.

[7]. Liang, C., Kawashima, R., and Matsuo, H. (2014). Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers. In *2014 Second International Symposium on Computing and Networking*, pp. 171-177.

[8]. Chen, Y., Li, Q., Yang, Y., Li, Q., Jiang, Y., and Xiao, X. (2015). Towards adaptive elastic distributed software defined networking. In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pp. 1-8.

[9]. Cheng, G., Chen, H., Hu, H., and Lan, J. (2016). Dynamic switch migration towards a scalable SDN control plane. *International Journal of Communication Systems*, **29**(9), pp. 1482-1499.

[10]. Zhou, Y., Zhu, M., Xiao, L., Ruan, L., Duan, W., Li, D., and Zhu, M. (2014). A load balancing strategy of sdn controller based on distributed decision. In *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 851-856.

[11]. Yu, J., Wang, Y., Pei, K., Zhang, S., and Li, J. (2016). A load balancing mechanism for multiple SDN controllers based on load informing strategy. In *2016 18th Asia-Pacific Network Operations and Management Symposium* (APNOMS), pp. 1-4.

[12]. Dixit, A., Hao, F., Mukherjee, S., Lakshman, T.V., and Kompella, R.R. (2014). Elasti Con; an elastic distributed SDN controller. In *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (ANCS), pp. 17-27.

[13]. Cheng, G., Chen, H., Wang, Z., and Chen, S. (2015, May). DHA: Distributed decisions on the switch migration toward a scalable SDN control plane. In *2015 IFIP Networking Conference* (IFIP Networking), pp. 1-9. IEEE.

[14]. Jarraya, Y., Madi, T., and Debbabi, M. (2014). A survey and a layered taxonomy of software-defined networking. *IEEE communications surveys & tutorials*, **16**(4), pp. 1955-1980.

[15]. Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., and Parulkar, G. (2014). ONOS: towards an open, distributed SDN OS. In Proceedings of the third workshop on Hot topics in software defined networking, pp. 1-6. ACM.

[16]. Mantas, A., and Ramos, F. (2016). Consistent and fault-tolerant SDN with unmodified switches. arXiv preprint arXiv:1602.04211.

[17]. Hunt, P., Konar, M., Junqueira, F.P., and Reed, B. (2010). ZooKeeper: Wait-free Coordination for Internet-scale Systems. In USENIX annual technical conference, Vol. **8**, No. 9.

[18]. Aly, W.H.F., and Al-anazi, A.M.A. (2018). Enhanced Controller Fault Tolerant (ECFT) Model for Software Defined Networking. In *2018 Fifth International Conference on Software Defined Systems* (*SDS*), pp. 217-222. IEEE.

[19]. OpenFlow switch specification 1.5.1, https://www.opennetworking.org/wp content/uploads/2014/10/openflow-switch-v1.5.1.pdf page no. 74, accessed online on 13th Feb 2019.

[20]. "ZeroMQ",http://www.zeromq.org, accessed online on 13th Feb 2019.

[21]. Hu, Y., Wang, W., Gong, X., Que, X., and Cheng, S. (2012). Balanceflow: controller load balancing for openflow networks. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, Vol. **2**, pp. 780-785.

[22]. Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., and Shenker, S. (2010). Onix: *A*

distributed control platform for large-scale production networks. In OSDI, Vol. **10**, pp. 1-6.
[23]. Tootoonchian, A., and Ganjali, Y. (2010). Hyperflow: A distributed control plane for openflow. *In Proceedings of the 2010 internet network management conference on Research on enterprise networking*, Vol. **3**.
[24]. Medved, J., Varga, R., Tkacik, A., and Gray, K. (2014). Opendaylight: Towards a model-driven sdn controller architecture. *In Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks* 2014, pp. 1-6.
[25]. Aly, W.H.F. (2017). A Novel Fault Tolerance Mechanism for Software Defined Networking. *In 2017 European Modelling Symposium* (*EMS*), pp. 233-239. IEEE
[26]. Hernandez Marulanda, E.S.T.E.B.A.N. (2016). Implementation and performance of a SDN cluster-controller based on the OpenDayLight framework.
[27]. Katta, N., Zhang, H., Freedman, M., and Rexford, J. (2015). Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research,* pp. 4.
[28]. Botelho, F.A., Bessani, A.N., Ramos, F.M., and Ferreira, P. (2014). On the Design of Practical Fault-Tolerant SDN Controllers. In EWSDN, pp. 73-78.
[29]. Obadia, M., Bouet, M., Leguay, J., Phemius, K., and Iannone, L. (2014). Failover mechanisms for distributed SDN controllers. In *2014 International Conference and Workshop on the Network of the Future* (NOF), pp. 1-6. IEEE.
[30]. Fonseca, P., Bennesby, R., Mota, E., and Passito, A. (2013). Resilience of sdns based on active and passive replication mechanisms. In *2013 IEEE Global Communications Conference* (GLOBECOM), pp. 2188-2193.
[31]. Hu, T., Yi, P., Zhang, J., and Lan, J. (2018). A distributed decision mechanism for controller load balancing based on switch migration in SDN. China Communications, **15**(10), pp. 129-142.
[32]. Hassas Yeganeh, S., and Ganjali, Y. (2012). Kandoo: a framework for efficient and scalable offloading of control applications. In Proceedings of the first workshop on Hot topics in software defined networks, pp. 19-24. ACM.
[33]. Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. (2011). DevoFlow: Scaling flow management for high-performance networks. *In ACM SIGCOMM Computer Communication Review* Vol. **41**, No. 4, pp. 254-265.
[34]. Yu, M., Rexford, J., Freedman, M.J., and Wang, J. (2011). Scalable flow-based networking with DIFANE. ACM SIGCOMM Computer Communication Review, **41**(4), pp. 351-362.
[35]. Lantz, B., Heller, B., and McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pp. 19.
[36]. "Project floodlight" [online], available: http://www.projectfloodlight.org/floodlight/", accessed on 08/10/2018).
[37]. Abilene Topology [online] available: "http://www.topology-zoo.org/files/Abilene.graphml" accessed online on 19/10/2018).
[38]. Internet 2 OS3E topology [online] available at: http://www.internet2.edu/media/medialibrary/files/.graphml" accessed online on 19/10/2018)
[39]. Aly, W.H.F. (2017). LBFTFB fault tolerance mechanism for software defined networking. In *2017 International Conference on Electrical and Computing Technologies and Applications* (ICECTA), pp. 1-5. IEEE.
[40]. Gaurang Lakhani, and Dr. Amit Kothari, (2019) Distributed Controller Fault ToleranceModel (DCFT) Using Load Balancing in Software Defined Networking. *International Journal of Computer Engineering and Technology*, **10**(2), pp. 215-233. Available: http://www.iaeme.com/IJCET/issues.asp?JType=IJCET&VType=10&IType=2.
[41]. Wu, Guowei, *et al*. A QoS and Cost Aware Fault Tolerant Scheme Insult-Controller SDNs. 2018 IEEE *Global Communications Conference* (GLOBECOM).

How to cite this article: Lakhani, Gaurang and Kothari Amit (2019). Comparison of Performance of Distributed Controller Fault Tolerance Model (DCFT) using Load Balancing in Software defined Networking in well-known Topologies. *International Journal on Emerging Technologies*, **10**(2): 136-146.